Česká společnost uživatelů otevřených systémů EurOpen.CZ
Czech Open System Users' Group
www.europen.cz

54. konference
Sborník příspěvků

Balónový hotel a pivovar, Radešín
28. května – 1. června 2022

Programový výbor:
Petr Švenda (předseda, Masarykova univerzita),
Milan Brož (Masarykova univerzita),
Vít Bukač (HERE Technologies),
Jan Hajný (Vysoké učení technické v Brně),
Matúš Jókay (FEI STU),
Jan Krhovják (Invasys),
Marek Kumpošt (Oracle NetSuite),
Václav Lorenc (HERE Technologies),
Vašek Matyáš (Masarykova univerzita),
Zdeněk Říha (Masarykova univerzita),
Marek Sýs (Masarykova univerzita)

Sborník příspěvků z 54. konference EurOpen.CZ, 28. 5.–1. 6. 2022
©EurOpen.CZ, Univerzitní 8, 306 14 Plzeň
Plzeň 2022. První vydání.
Editor: Zdeněk Šustr
Sazba a grafická úprava: Zdeněk Šustr, vydavatelství, Praha
e-mail: `vydavatelstvi@sustr.net`

Tisk: Typos, tiskařské závody, s.r.o.
Podnikatelská 1160/14, Plzeň

Upozornění: Všechna práva vyhrazena. Rozmnožování a šíření této publikace jakýmkoliv způsobem bez výslovného písemného svolení vydavatele je trestné.

Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-34-1

Obsah

PTWEBDISCOVER: Nástroj pro efektivní mapování webových aplikací během penetračního testování

Willi Lazarov, Zdeněk Martinásek, Roman Kümmel

E-mail: {xlazar15,martinasek}@vut.cz,
r.kummel@hacker-consulting.cz

Abstrakt

Článek popisuje nástroj ptwebdiscover, který je určen k mapování webových aplikací během penetračního testování. Hlavní rozdíl od současných dostupných nástrojů představuje běh ve více vláknech a rozšiřující možnosti testování. Příkladem těchto možností je aplikování metody hrubé síly v situacích, kdy je nutné hledat kratší názvy souborů nebo možnost označit v testovaném URL konkrétní místo, na které bude program vkládat ověřované řetězce. Článek dále popisuje vlastní vícevláknovou implementaci nástroje, inteligentní vyhledávání záloh a porovnání s konkurenčními nástroji pro penetrační testování webových aplikací.

Klíčová slova: Penterep; ptwebdiscover; penetrační testování; webové aplikace; OWASP

1 Úvod

Kybernetické útoky na informační a komunikační systémy státní správy, komerčních firem i běžných koncových uživatelů představují v dnešní době reálnou hrozbu a jejich četnost se neustále zvyšuje. Jednou z účinných možností ochrany je pravidelná realizace a vyhodnocení penetračních testů, během kterých lze odhalit možné slabiny a zranitelnosti dříve, než je odhalí a zneužijí potenciální útočníci. Díky tomuto preventivnímu opatření může provozovatel testované aplikace nalezené nedostatky odstranit a předejít tak velice nepříjemným následkům kybernetických útoků [1]. Nástroj popsaný v tomto článku se zaměřuje na penetrační testování webových aplikací, které jsou z důvodu jejich snadné dostupnosti častým cílem útoků, a představují tak vstupní bránu pro eskalaci dalších útoků.

Každý penetrační test webové aplikace by měl začínat průzkumem prostředí a důkladným zmapováním testované aplikace [2]. [1] Během mapování může tester odhalit nejen všechny veřejně dostupné zdroje, na které aplikace odkazuje, nebo je načítá, ale i zdroje, na které žádný odkaz nesměřuje, a nejsou proto indexovány. Takovými zdroji jsou například soubory s logy, administrace nebo často také zálohy. Dále se například u nalezených adresářů testuje, zda není povolen výpis jejich obsahu. Podobné bezpečnostní nedostatky mohou mít vážné dopady, a proto je jejich odhalení i přes značnou časovou náročnost velice důležité. Je třeba si uvědomit, že pokud je útočník dostatečně motivován, tak pro něj není čas hlavním limitujícím faktorem. Tato skutečnost je ve velkém kontrastu s penetračním testováním, kde je naopak čas strávený testováním zásadní.

1.1 Současný stav

V současné době patří mezi nejpoužívanější nástroje pro mapování webových aplikací `dirb`, `dirbuster`, `dirsearch` a `dirstalk`. Uvedené nástroje mají ovšem několik zásadních nevýhod, které limitují jejich použití. Mezi nejpodstatnější nevýhody z pohledu penetračního testovaní patří nemožnost vyhledávání subdomén a zdrojů v libovolné části URL (Uniform Resource Locator), chybějící podpora vícevláknového zpracování, nebo neefektivní vyhledávání záloh. Podrobný seznam funkcionalit nástrojů pro mapování webových aplikací je uveden v podkapitole 3.

1.2 Vlastní přínos

Jedním z výstupů aplikovaného výzkumu, jehož cílem bylo vyvinout platformu pro penetrační testování s názvem Penterep[2], je nový nástroj `ptwebdiscover`. Nástroj byl vyvinut s hlavním cílem eliminovat výše uvedené nedostatky a zefektivnit tak práci penetračních testerů. Výsledný nástroj

[1] Penetrační test by měl následovat kroky metodologie OWASP ASVS (Application Security Verification Standard).

[2] Oficiální webová prezentace platformy je dostupná na adrese `https://www.penterep.com`.

`ptwebdiscover` je svým použitím velice podobný známému nástroji `dirb`, který je implicitně obsažen v linuxové distribuci Kali Linux a testeři jsou na jeho použití zvyklí [3]. Hlavní rozdíl od nástroje `dirb` představuje běh ve více vláknech, díky kterému může být nástroj `ptwebdiscover` mnohonásobně rychlejší, čímž se důkladnost a rychlost penetračních testů výrazně zvyšuje. Další výhodou nástroje je možnost aplikování metody hrubé síly v situacích, kdy je nutné hledat kratší názvy souborů, nebo pokud je potřeba odhalit pouze část jejich názvu (např. po identifikaci krátkých názvů útokem IIS tilde enumeration). Nástroj `ptwebdiscover` poskytuje také možnost označit v testovaném URL konkrétní místo, na které bude skript vkládat ověřované řetězce. Stejně tak lze označit i subdoménu, nebo konkrétní HTTP(S) GET parametr, díky čemuž se nástroj hodí i k fuzzingu. Mimo to nabízí nástroj možnost použití speciálních slovníků, které umožňují identifikaci použitých technologií na základě pro ně typických souborů nebo adresářů. Nástroj `ptwebdiscover` řeší efektivním způsobem také vyhledávání záloh, pomocí kterého se již na veřejně dostupných webových stránkách podařilo dohledat jejich velké množství.

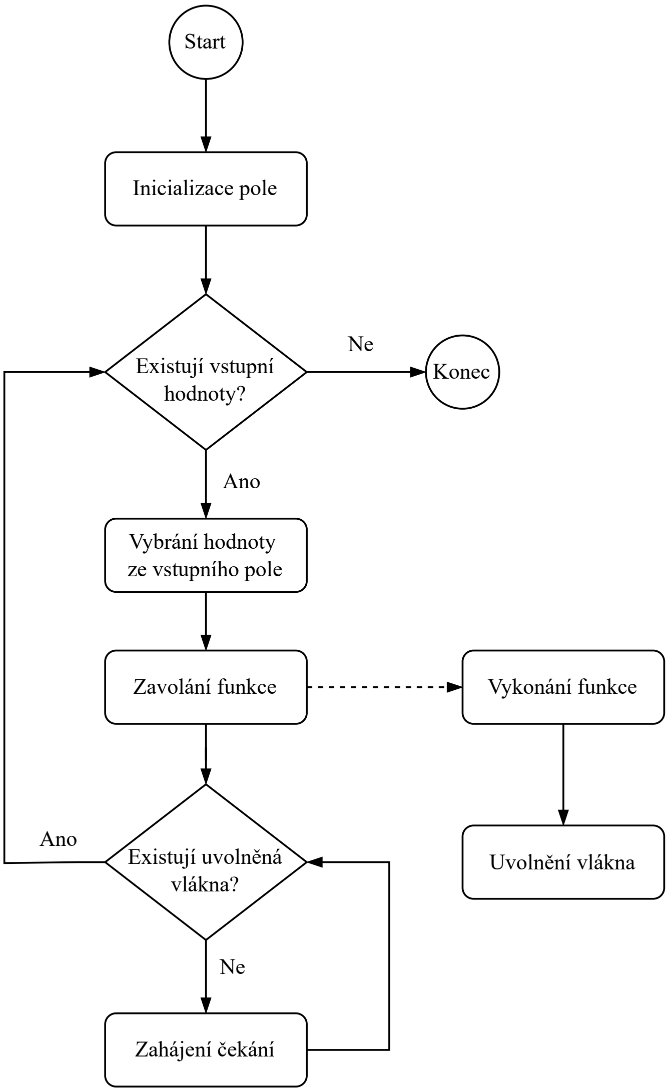
2 Návrh a implementace nástroje

Pro zvýšení efektivity nástroje `ptwebdiscover` byla vyvinuta vlastní třída s názvem `ptthreads`[3]. Vývoj této třídy byl uskutečněn z toho důvodu, aby bylo možné její použití z různých nástrojů spolu s volbou počtu vláken, ve kterých se má spouštět požadovaná funkce. Instance třídy pomocí stejnojmenné funkce přijímá následující argumenty:

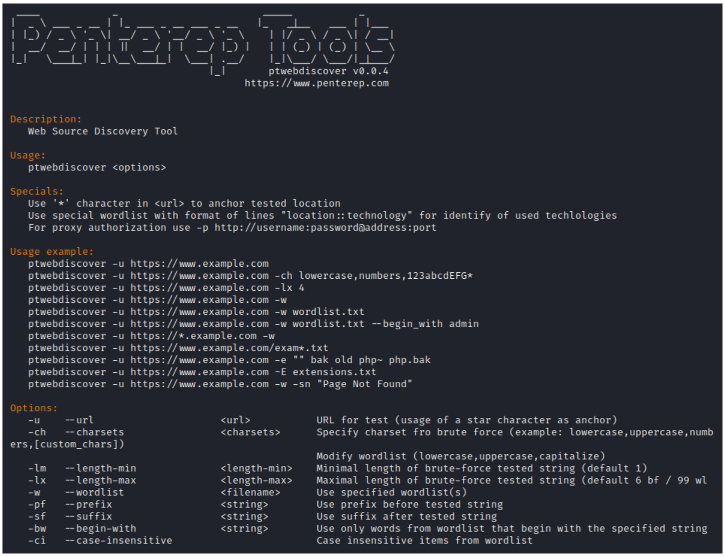
- pole hodnot, které se mají projít ve více vláknech,
- volaná funkce v každém vlákně,
- celkový počet vláken.

Volané funkci jsou ve vláknech předávány jednotlivé položky vstupního pole. Po vykonání funkce se postupně uvolňují rezervovaná vlákna pro další iteraci. Jakmile jsou vybrány všechny vstupní hodnoty, tak je běh programu ukončen. Postup při implementaci je zobrazen na následujícím obrázku 1.

[3]Třída ptthreads je pod otevřenou licencí GPLv3+ dostupná ke stažení na adrese `https://pypi.org/project/ptthreads/`.



Obrázek 1: Vykonání funkce ve více vláknech pomocí třídy ptthreads



```
 ____            _                         _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|       ptwebdiscover v0.0.4
                                       https://www.penterep.com


Description:
   Web Source Discovery Tool

Usage:
   ptwebdiscover <options>

Specials:
   Use '*' character in <url> to anchor tested location
   Use special wordlist with format of lines "location::technology" for identify of used techlologies
   For proxy authorization use -p http://username:password@address:port

Usage example:
   ptwebdiscover -u https://www.example.com
   ptwebdiscover -u https://www.example.com -ch lowercase,numbers,123abcdEFG*
   ptwebdiscover -u https://www.example.com -lx 4
   ptwebdiscover -u https://www.example.com -w
   ptwebdiscover -u https://www.example.com -w wordlist.txt
   ptwebdiscover -u https://www.example.com -w wordlist.txt --begin_with admin
   ptwebdiscover -u https://*.example.com -w
   ptwebdiscover -u https://www.example.com/exam*.txt
   ptwebdiscover -u https://www.example.com -e "" bak old php~ php.bak
   ptwebdiscover -u https://www.example.com -E extensions.txt
   ptwebdiscover -u https://www.example.com -w -sn "Page Not Found"

Options:
   -u    --url                   <url>          URL for test (usage of a star character as anchor)
   -ch   --charsets              <charsets>     Specify charset fro brute force (example: lowercase,uppercase,numb
ers,[custom_chars])
                                                Modify wordlist (lowercase,uppercase,capitalize)
   -lm   --length-min            <length-min>   Minimal length of brute-force tested string (default 1)
   -lx   --length-max            <length-max>   Maximal length of brute-force tested string (default 6 bf / 99 wl
   -w    --wordlist              <filename>     Use specified wordlist(s)
   -pf   --prefix                <string>       Use prefix before tested string
   -sf   --suffix                <string>       Use suffix after tested string
   -bw   --begin-with            <string>       Use only words from wordlist that begin with the specified string
   -ci   --case-insensitive                     Case insensitive items from wordlist
```

Obrázek 2: Ukázka výpisu nástroje v terminálu Kali Linux

3 Porovnání s konkurenčními nástroji

Všechny podporované funkcionality jsou uvedeny v tabulce 1, ve které je nástroj `ptwebdiscover` porovnán s dalšími nástroji určenými pro penetrační testování webových aplikací. Porovnány byly nástroje `ptwebdiscover` [4], `dirstalk` [5], `dirb` [3], `dirbuster` [6] a `dirsearch` [7]. Nástroj `ptwebdiscover` je volně dostupný ke stažení jako svobodný software pod licencí GPLv3+ na adrese `https://pypi.org/project/ptwebdiscover/` a ke dni 15. 4. 2022 má již 1950 stažení [4]. Rozhraní nástroje je zobrazeno na obrázku 2, kde lze vidět i částečný výčet funkcí nástroje.

Z výsledku porovnání v tabulce 1 je patrné, že nejvíce možností testování nabízí nástroj `ptwebdiscover`. Chybějící funkce tohoto nástroje jsou pouze výstup do XML formátu a podpora klientských certifikátů, které naopak nabízí nástroj `dirbuster`. Jednou z hlavních předností nástroje `ptwebdiscover` je inteligentní vyhledávání záloh a více nabízených metod pro samotné vyhledávání. Zatímco ostatní nástroje s výjimkou ná-

[4]Statistiky celkového počtu stažení nástroje jsou dostupné na `https://pepy.tech/project/ptwebdiscover`.

Tabulka 1: Porovnání vlastností nástrojů

Funkcionalita	ptwebdiscover	dirstalk	dirb	dirbuster	dirsearch
Hledání souborů	✓	✓	✓	✓	✓
Hledání adresářů	✓	✓	✓	✓	✓
Hledání subdomén	✓	✗	✗	✗	✗
Fuzzing parametrů	✓	✗	✗	✗	✗
Hledání v libovolné části URL	✓	✗	✗	✗	✗
Použití hrubé síly	✓	✗	✗	✓	✗
Použití slovníků	✓	✓	✓	✓	✓
Filtrování slovníků	✓	✗	✗	✗	✗
Testování přípon	✓	✗	✓	✓	✓
Možnost načtení přípon ze souboru	✓	✗	✓	✗	✓
Nastavení minimální délky řetězce	✓	✗	✗	✓	✗
Nastavení maximální délky řetězce	✓	✗	✗	✓	✗
Použití prefixů a sufixů	✓	✗	✗	✗	✓
Identifikace použitých technologií	✓	✗	✗	✗	✗
Inteligentní vyhledávání záloh	✓	✗	✗	✗	✗
Rekurzivní procházení adresářů	✓	✗	✓	✓	✓
Upozorňování na directory listing	✓	✗	✓	✗	✓
Načtení URL adres pro otestování	✓	✗	✗	✗	✓
Použití libovolné HTTP metody	✓	✓	✗	✓	✓
Nastavení cookies	✓	✓	✓	✓	✓
Nastavení hlavičky User-Agent	✓	✓	✓	✓	✓
Nastavení HTTP request hlaviček	✓	✓	✓	✓	✓
Použití proxy serveru	✓	✓	✓	✓	✓
Běh ve více vláknech	✓	✓	✗	✓	✓
Nastavení prodlevy mezi requesty	✓	✓	✓	✗	✓
Pozitivní hledání řetězce v odpovědi	✓	✗	✗	✗	✗
Negativní hledání řetězce v odpovědi	✓	✗	✗	✗	✗
Výstup do konzole	✓	✓	✓	✓	✓
Výstup do souboru	✓	✓	✓	✓	✓
Výstup v JSON formátu	✓	✗	✗	✗	✓
Výstup v XML formátu	✗	✗	✗	✗	✓
HTTP autentizace	✓	✓	✓	✓	✓
Podpora klientských certifikátů	✗	✗	✓	✗	✓
PROXY autentizace	✓	✗	✓	✓	✓
Parsování robots.txt	✓	✗	✗	✓	✗
Parsování sitemap.xml	✓	✗	✗	✗	✗
Stažení nalezeného obsahu	✓	✗	✗	✗	✗
Přidávání znaku „/“ na konce řetězců	✓	✗	✓	✗	✓
Možnost zahrnutí GET parametrů	✓	✗	✗	✗	✓
Možnost výstupu s/bez domény	✓	✗	✗	✗	✗
Zahrnutí výsledků z dir. listingu	✓	✗	✗	✓	✗
Case-sensitivita u slovníků	✓	✗	✓	✓	✓
Kopírovatelný výstup (seznam)	✓	✗	✗	✓	✓
Automatické dohledávání záloh	✓	✗	✗	✗	✗
Vypsání stromové struktury	✓	✗	✗	✓	✗

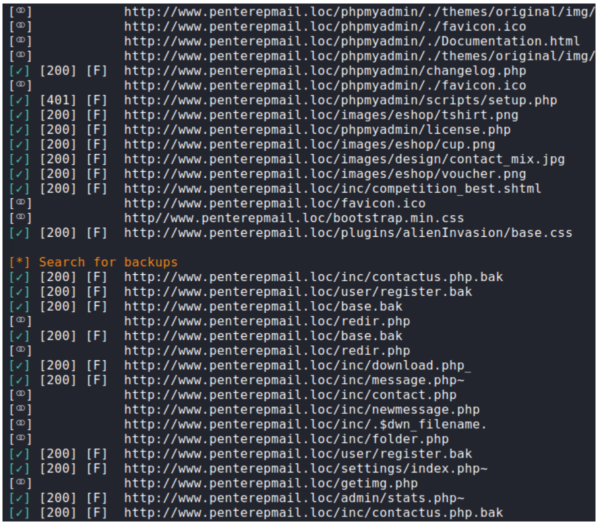
stroje `dirbuster` umožňují vyhledávat pouze na základě slovníku, nástroj `ptwebdiscover` podporuje také vyhledávání hrubou silou nebo parsováním obsahu.

3.1 Vyhledávání záloh

Veřejně přístupné zálohy jsou častým příkladem zranitelnosti webové aplikace. Jednou z metod jejich vyhledávání je použití hrubé síly, které se ale vzhledem k časové náročnosti příliš nepoužívá. Namísto toho se využívá rychlejší metoda pomocí slovníku. Způsoby vyhledávání záloh jsou u jednotlivých nástrojů různé. Základním postupem je použití slovníků, který obsahuje názvy souborů včetně všech možných přípon. Některé nástroje naopak umožňují univerzální použití slovníku bez přípon, které jsou nástroji předány až na vstupu.

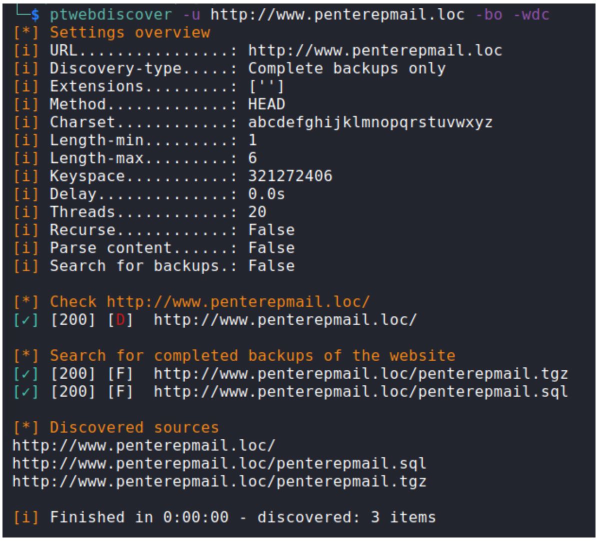
Při použití nástroje `dirstalk` je nutné nejprve vygenerovat slovník obsahující hledané názvy souborů bez možnosti vlastní specifikace přípon. Efektivnější jsou v tomto ohledu nástroje `dirb`, `dirbuster` a `dirsearch`, které nabízí možnost specifikovat přípony, které mají být společně s výrazy ze slovníku použity. Nástroj `ptwebdiscover` nabízí kromě specifikace přípon navíc také explicitní parametry pro vyhledávání záloh, při jejichž použití nejprve zjišťuje existenci daného souboru a až následně začne vyhledávat možné existující zálohy. Díky tomu je mnohem efektivnější a rychlejší než konkurenční nástroje.

Další přidanou funkcí nástroje `ptwebdiscover` je inteligentní vyhledávání kompletních záloh webové aplikace nebo databáze, při kterém se používá pouze doména testovaného cíle. Pro zadanou doménu jsou zkoušeny všechny možné přípony (např. penterep.com.zip, penterep_com.zip apod.), díky čemuž lze rychle vyhledat zálohy bez nutnosti použít slovník. Oba režimy vyhledávání záloh jsou zachyceny na obrázku 3



```
[∞]             http://www.penterepmail.loc/phpmyadmin/./themes/original/img/
[∞]             http://www.penterepmail.loc/phpmyadmin/./favicon.ico
[∞]             http://www.penterepmail.loc/phpmyadmin/./Documentation.html
[∞]             http://www.penterepmail.loc/phpmyadmin/./themes/original/img/
[✓] [200] [F]   http://www.penterepmail.loc/phpmyadmin/changelog.php
[∞]             http://www.penterepmail.loc/phpmyadmin/./favicon.ico
[✓] [401] [F]   http://www.penterepmail.loc/phpmyadmin/scripts/setup.php
[✓] [200] [F]   http://www.penterepmail.loc/images/eshop/tshirt.png
[✓] [200] [F]   http://www.penterepmail.loc/phpmyadmin/license.php
[✓] [200] [F]   http://www.penterepmail.loc/images/eshop/cup.png
[✓] [200] [F]   http://www.penterepmail.loc/images/design/contact_mix.jpg
[✓] [200] [F]   http://www.penterepmail.loc/images/eshop/voucher.png
[✓] [200] [F]   http://www.penterepmail.loc/inc/competition_best.shtml
[∞]             http://www.penterepmail.loc/favicon.ico
[∞]             http//www.penterepmail.loc/bootstrap.min.css
[✓] [200] [F]   http://www.penterepmail.loc/plugins/alienInvasion/base.css

[*] Search for backups
[✓] [200] [F]   http://www.penterepmail.loc/inc/contactus.php.bak
[✓] [200] [F]   http://www.penterepmail.loc/user/register.bak
[✓] [200] [F]   http://www.penterepmail.loc/base.bak
[∞]             http://www.penterepmail.loc/redir.php
[✓] [200] [F]   http://www.penterepmail.loc/base.bak
[∞]             http://www.penterepmail.loc/redir.php
[✓] [200] [F]   http://www.penterepmail.loc/inc/download.php_
[✓] [200] [F]   http://www.penterepmail.loc/inc/message.php~
[∞]             http://www.penterepmail.loc/inc/contact.php
[∞]             http://www.penterepmail.loc/inc/newmessage.php
[∞]             http://www.penterepmail.loc/inc/.$dwn_filename.
[∞]             http://www.penterepmail.loc/inc/folder.php
[✓] [200] [F]   http://www.penterepmail.loc/user/register.bak
[✓] [200] [F]   http://www.penterepmail.loc/settings/index.php~
[∞]             http://www.penterepmail.loc/getimg.php
[✓] [200] [F]   http://www.penterepmail.loc/admin/stats.php~
[✓] [200] [F]   http://www.penterepmail.loc/inc/contactus.php.bak
```

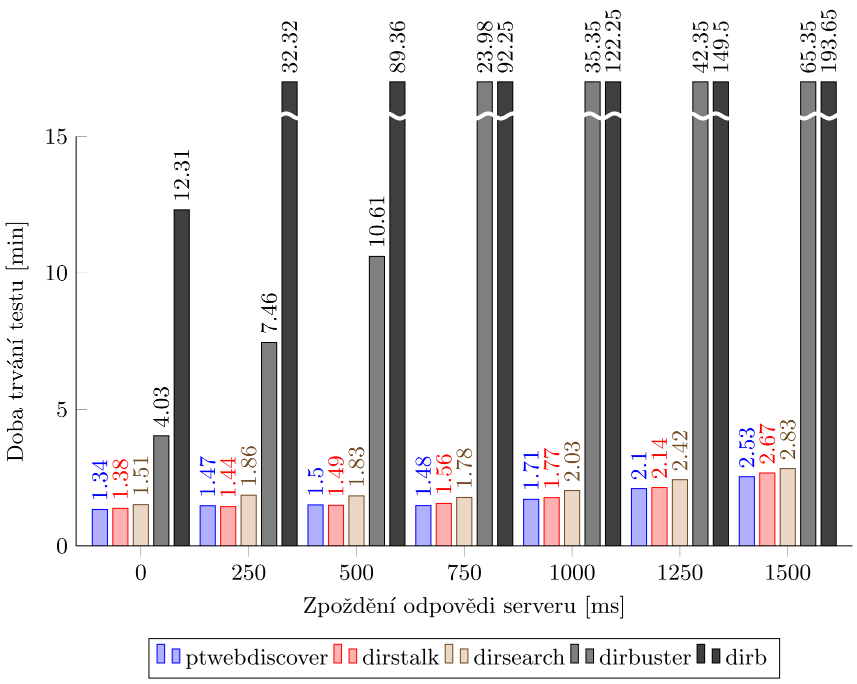
(a) Použití slovníku



```
└─$ ptwebdiscover -u http://www.penterepmail.loc -bo -wdc
[*] Settings overview
[i] URL...............: http://www.penterepmail.loc
[i] Discovery-type.....: Complete backups only
[i] Extensions.........: ['']
[i] Method.............: HEAD
[i] Charset............: abcdefghijklmnopqrstuvwxyz
[i] Length-min.........: 1
[i] Length-max.........: 6
[i] Keyspace...........: 321272406
[i] Delay..............: 0.0s
[i] Threads............: 20
[i] Recurse............: False
[i] Parse content......: False
[i] Search for backups.: False

[*] Check http://www.penterepmail.loc/
[✓] [200] [D]  http://www.penterepmail.loc/

[*] Search for completed backups of the website
[✓] [200] [F]  http://www.penterepmail.loc/penterepmail.tgz
[✓] [200] [F]  http://www.penterepmail.loc/penterepmail.sql

[*] Discovered sources
http://www.penterepmail.loc/
http://www.penterepmail.loc/penterepmail.sql
http://www.penterepmail.loc/penterepmail.tgz

[i] Finished in 0:00:00 - discovered: 3 items
```

(b) Inteligentní vyhledávání

Obrázek 3: Vyhledávání záloh nástrojem ptwebdiscover



Obrázek 4: Porovnání rychlosti jednotlivých nástrojů

3.2 Porovnání rychlosti

Jednotlivé nástroje pro mapování webových aplikací byly dále porovnány v celkové době trvání testu. Porovnání rychlosti nástrojů bylo provedeno za použití virtuálních strojů s linuxovou distribucí Kali Linux (zdroj testu) a Debian (cíl testu). Cílem testu byl webový server Apache verze 2.4, na kterém běžela zranitelná webová aplikace, ve které byly postupně všemi nástroji vyhledávány soubory a adresáře za použití slovníku o celkové velikosti 4164 slov. Při testování byl u nástrojů podporujících běh ve více vláknech nastaven konstantní počet vláken na hodnotu 100. Pro každý nástroj bylo provedeno celkem 7 testů, ve kterých se zvyšovala prodleva odpovědi serveru v rozsahu 0–1500 ms. Výsledky všech testů jsou uvedeny v grafu 3.2.

Z grafu je patrné, že doba trvání testů je zásadně ovlivněna tím, zda nástroj běžel ve více vláknech. Nástroje `ptwebdiscover`, `dirstalk` a `dirsearch` proto dosahují velmi podobných výsledků, přičemž vliv zvyšující se odpovědi serveru byl na dobu trvání testu těchto nástrojů minimální. Přestože nástroj `dirbuster` umožňuje běh ve více vláknech, tak je ve srovnání s předchozími nástroji výrazně pomalejší a se zvyšujícím se zpožděním odpovědi serveru narůstá zároveň doba trvání celého testu. Nejpomalejším byl při testování `dirb`, který neumožňuje běh ve více vláknech a výsledky testů jsou tak při porovnání s ostatními nástroji znatelně horší.

4 Závěr

Cílem článku bylo blíže představit nový nástroj `ptwebdiscover` pro penetrační testování webových aplikací. Nástroj se při testování ukázal jako velmi efektivní a v porovnání s ostatními nástroji používanými pro mapování webových aplikací nabízí `ptwebdiscover` nejvíce možností. Mezi jeho hlavní přednosti patří zejména běh nástroje ve více vláknech, inteligentní vyhledávání záloh, vyhledávání hrubou silou a parsováním obsahu.

Nástroj vznikl v rámci řešení projektu aplikovaného výstupu, jehož výstupem jsou i další nástroje určené pro penetrační testování webových aplikací, jako je například `ptinsearcher` určený k extrakci informací z webových zdrojů, `ptmultiviews` pro ověření zranitelnosti MultiViews, nebo `ptiistild` pro identifikaci zranitelnosti IIS Tilde Enumeration. Všechny tyto nástroje jsou integrovány do platformy Penterep, která je určena pro komplexní realizaci a správu penetračního testování. Jednotlivé nástroje je také možné vhodně kombinovat pro dosažení optimálních výsledků testování.

Poděkování

Výzkum a vývoj popsaný v tomto článku byl spolufinancován se státní podporou Technologické agentury ČR v rámci Programu Zéta (č. projektu TJ04000456).

Odkazy

[1] Aileen Bacudio et al. „An Overview of Penetration Testing“. In: *International Journal of Network Security & Its Applications* 3 (lis. 2011), s. 19–38. DOI: `10.5121/ijnsa.2011.3602`.

[2] A. Van der Stock et al. *OWASP Application Security Verification Standard 4.0.3.* 2021.

[3] Kali Linux. *Dirb.* URL: `https://www.kali.org/tools/dirb/`.

[4] PTWEBDISCOVER. *PyPI.* URL: `https://pypi.org/project/ptwebdiscover/`.

[5] Kali Linux. *Dirstalk.* URL: `https://github.com/stefanoj3/dirstalk`.

[6] Kali Linux. *Dirbuster.* URL: `https://www.kali.org/tools/dirbuster/`.

[7] Kali Linux. *Dirsearch.* URL: `https://www.kali.org/tools/dirsearch/`.

Bug Bounty Hunting

Mgr. Jan Kvapil

Abstract

Looking for vulnerabilities in software—previously, often practices resulting in illegal activities, now, an (extra)ordinary occupation. How to look for those vulnerabilities legally and help to secure the software for others as well? Where and how to submit the vulnerability disclosure report and corresponding proof of concept? Apart from the answers, we will take a look at some publicly disclosed reports. Next, we will see some of the available tools and also wonder, how vulnerability disclosures fit into the open-source world and whether you should consider starting a bug bounty program.

Disclaimer

This text is based on my (admittedly short) personal experience in bug bounty hunting and vulnerability research. The area of interest is quite broad and would benefit from more formal research. Maybe I will come to that some day in the future. As such, take this paper as light and introductory material into the interesting world of bug bounty hunting.

The following text is educational and the author does not take any responsibility for any acts the reader might decide to partake in.

1 Introduction: an ethical hacker

Following precisely how the word *hacker* has been used and perceived by various groups and the general public would probably suffice as a topic for small research on its own. In my dictionary, this word carries less and less of a negative connotation as time progresses. While I have yet to finish the book Hackers: Heroes of the Computer Revolution by Steven Levy[1] I can already recommend it in order to help the reader with rebuilding the intuition behind the mindset of *a hacker.* A hacker is strongly driven to understand and explore all the *nooks and crannies* of a given system—even, or especially, if the authors of the system, its documentation and other sources assure that everything works perfectly. Adding *ethical* in front of the word hacker makes it transparent to the public (and the hacker as well) that the hacker has an inner moral compass that he or she follows.

The phrase *bug bounty hunter* might also have its connotations. Namely, that the motif of a bug bounty hunter is solely a financial gain. While such motifs can be present, there are others as well. And if seen as an occupation, bug bounty hunting does not differ from the others.

Now, I would like to walk the reader through the topics and processes related to bug bounty hunting and ethical hacking.

1.1 Platforms and programs

A hacker spends his time exploring, testing and analysing various pieces of software. However, such practice can lead to a significant crossing of legal boundaries. Consider for example the Computer Fraud and Abuse Act—*Whoever having knowingly accessed a computer without authorization or exceeding authorized access...* has quite probably committed a criminal offence under this act. Interestingly, the evolution in the security field has lead to the creation of *platforms* that help to legally connect hackers with companies. Those platforms brought also a certain notion of gamification to the process. While the details of the security vulnerabilities are disclosed only under certain conditions (e.g. after mutual agreement between the hacker and the company) some information can be public, such as the reputation and impact of the hacker.

Both hackers and companies can register on the platforms. The company creates a Bug Bounty Program (a *program* in the following text). The programs can be *public*, i.e. anyone can see them. For example, this is the U.S. Department of Defense's program. Another option is a *private* program, whose visibility is limited only to certain hackers. The incentives to have a private program can vary, one can imagine that making the program public could result in too many reports that a small company won't be able to handle. Access to a private program can be given after gaining a certain amount of reputation.

[1]ISBN-13: 978-1449388393

In general, a company can host Bug Bounty Program directly. However, there could be a non-trivial amount of legal and financial paperwork (payouts for bounties going all over the world). Similarly to other business processes, bounties are subject to international laws and also sanctions, which is quite relevant at the time of writing this text due to the ongoing Russian invasion to Ukraine. The response of one of the platforms is worth reading.

Some of the known platforms are HackerOne, Bugcrowd, Intigriti (however, there many more). Understandably, the tech *giants* are capable of hosting their own programs, such as Microsoft, Google. One might ask: Can the platforms themselves be a target of an attack? I'll answer by quoting the last platform's website:

> After something like 2 years of virtual blood, sweat, and tears to build this site, it took a bug hunter only a few hours to discover our private APIs.

It is not a surprise that the well-known software companies have their own program, however, there are also other programs worth noting. Namely, The Internet Bug Bounty program (sponsored by multiple companies), whose scope covers more and more open-source projects (e.g. The Ruby Programming Language, Ruby on Rails, OpensSSL and others) as time progresses. Another interesting program is the one ran by the European Commission (hosted on Intigriti). Finally, it might be of interest that Google's program also spreads outside of Google's own projects to some open-source projecs (e.g. OSS-Fuzz projects among many other) and also to security research in general.

2 Bug hunting

I will now show the steps leading to submitting a vulnerability report in order to provide a better insight into it. I am most familiar with HackerOne, therefore the following scenario is based around this platform.

2.1 Where to start

There is a multitude of tools (most of which I haven't tried) that can probably help with finding a vulnerability. That said, relying on tools and skipping over fundamental understanding is not something I would recommend. For example, a report that consists of the output from a static analysis tool won't be accepted—a clear impact has to be shown and discussed at a minimum. Information security is broad, therefore I would suggest starting small and in an area, one is familiar with. However, hacking is a constant process of refining one's skills and learning about new technologies, practices, etc. One of the first programs that I have been reviewing was a web application written in PHP. I have dived into PHP, and its documentation and started looking for the weak spots. The ability to learn quickly and navigate through lots of materials at once is beneficial—and soon interesting things can pop up (such as "types juggling" in PHP).

To start safely, HackerOne provides a nice sandbox consisting of multiple vulnerable applications of various levels of difficulty. In case the reader starts finding the first issues in a matter of hours or days he can try starting hunting on a real program. It is worth reiterating that bug bounty hunting is *a delicate practice* from the legal perspective and my suggestion is to **read** the platform's Terms and Conditions (ToC) carefully this time.

2.2 Finding a program

In general, I see two approaches, on one hand, seasoned hackers are able to work on multiple projects at once, especially if they prefer automation over a manual review. On the other hand, some stick to a single program for months or years. The first group benefits from automation and the ability to catch the low-hanging fruits, the second one builds a much deeper understanding of a given service—and the understanding pays off because it is easier to showcase the impact of a vulnerability. Finding vulnerabilities is not an easy task and starting with software or app that one is familiar with gives a head start. There are multiple well-known public programs on HackerOne, for example GitLab. The downside of public programs (from the point of view of a hacker) is that many other hackers are working on them already, which lowers the chances of finding an original bug (because most programs do not pay off bounties for duplicate reports).

Once a program is chosen *things* start to get exciting—looking for a security vulnerability can bring an adrenaline rush. However, similarly to reading ToC, it is **a necessity** to carefully go through the program's policy and scope. The assets in scope can be targeted and hacked by the attacker. The programs can whitelist the assets (i.e. the "In scope" section) in multiple ways. For example, Curl only mentions a single asset in scope and that is its repository on GitHub. Others provide a much broader scope—consisting of raw IP ranges, links to binaries, etc. Programs can tag the scopes with the technologies (at least the main ones) that the asset is built upon, presumably to help the hacker direct the attention. Similarly, some assets are explicitly out of scope and hacking them is prohibited.

Following the policy and staying within the scope is essential to remaining an ethical hacker. For example, this is an excerpt from GitLab's policy:

> Any activities conducted in a manner consistent with this policy will be considered authorized conduct and we will not initiate legal action against you. If legal action is initiated by a third party against you in connection with activities conducted under this policy, *we will take steps to make it known that your actions were conducted in compliance with this policy.*[2]

A policy can also contain slightly surprising clauses, such as this one from PayPal's program:

> As a condition of participation in the PayPal Bug Bounty Program, you hereby grant PayPal, its subsidiaries, affiliates and customers a perpetual, irrevocable, worldwide, royalty-free, transferrable, sublicensable (through multiple tiers) and non-exclusive license to use, reproduce, adapt, modify, publish, distribute, publicly perform, create derivative work from, make, use, sell, offer for sale and import the Submission, as well as any materials submitted to PayPal in connection therewith, for any purpose. You should not send us any Submission that you do not wish to license to us.

While understandable when viewed from the program's side, similar statements made me question, whether I would submit a report to such a program.

2.3 Looking for a bug

After agreeing to the policy and understanding the scope one can finally start hacking. The first thing is to get familiar with the program—is it a

[2]Emphasis mine

web application? Then go ahead, register and start using it. The platforms often maintain `<username>@<platforms-domain>` e-mail that is meant to be used for such registration. The program can then easily monitor and filter those accounts from the ones of the real users. Some programs provide test account credentials at request or as a part of their policy. It is important to understand that a bug in one program can fall into the *won't fix* category, but be a security vulnerability in a different context. The goal of a hacker is to find a vulnerability and show its **impact** if exploited. Of course, there is no single path how to find a bug. There are many tools and resources that can help out and the pentesting, hacking and security community comes with new ones often. For example:

- BurpSuite: well-known tool for web security and penetration testing,
- Payload All the Things: a list of various payloads such as Cross Site Scripting payloads,
- Amass: a tool for network mapping of attack surfaces,
- sqlmap: tool for detecting SQL injections,
- HackerOne resources: a list of other tools categorized by their area of interest.

While such tools can be of great help, one needs to be careful to understand first what the tools do and how (again, complying with the policy and scope is crucial). However, there are more basic tools that are enough for starting out. Fluency in some kind of a shell program is at the base level. Often, the first tool I use is the command-line `curl`—it is great at testing out APIs. Next, the ability to script in languages like Python goes well alongside that (combined with libraries such as pwntools). Most of my proof of concepts take advantage of a simple Python script. The necessary tool set depends on the target, but basic DevOps tactics, for example spinning up multiple Docker containers, do come in handy. Testing locally denial of service attacks is much more doable if a successful attack crashes only a container and not the host system.

In general, the hacker does not need to be a coder in order to find issues (e.g. information disclosure vulnerabilities can sometimes be found by simply using the website and *clicking* around), but it empowers the hacker greatly. That said, the current browsers already provide enough information through developer tools that can be used to start investigating the web traffic and look for weaknesses.

At this point, it is good to articulate the common phrase that *the attacker only needs to find a one way in.* I want to stress this point because while there are rules, one can still find quite unorthodox ways of gathering meaningful information. Once I needed to learn, whether and how a target uses a particular piece of software. Its own assets and domains did not tell me much, but a quick Open Source Intelligence (OSINT) provided me with an hour-long podcast from one of the lead developers of the target. The podcast was meant to be interesting for other developers, but at the same time, it has given me valuable information on how that particular software is being utilized. A hacker needs to be creative and unorthodox in his ways and the application of his skills. A piece of information is often not harmful on its own but becomes when put into a mosaic of others.

2.4 Submitting a report

Once a bug is found the vulnerability disclosure report can be created. Depending on the platform it has to contain the vulnerable asset (e.g. a domain), type of weakness (e.g. buffer overflow) and its severity—which can either be set directly (e.g. High) or calculated using the CVSS calculator. Then the most valuable parts come, the proof of concept (PoC) and the impact.

Depending on the program the reports are first read and triaged by someone from the platform (in order to avoid spam) and only then handed over to someone from the program or processed by the program directly. Writing a report and communicating clearly with the program is another skill set worth having for a hacker. It is not unheard of that the communication within a report does not go smoothly. Also, the triager might lack the particular skill set required for successfully exploiting the vulnerability. Keeping the proof of concept succinct and clear is wise. Ideally, I try to provide a single `curl` command that showcases the vulnerability or a simple Python script. If possible (e.g. for some OSS projects), I add a container with the required provisioning to minimize the work for the triager. Some reporters create a screencast to give proof of the vulnerability (and also provide the commands)—the triager then does not need to necessarily recreate the issue but can triage it anyway. Giving the right amount of details is non-trivial and the communication is lacking, because waiting for a response several days or weeks is normal.

At this point, it is apparent where open-source projects benefit. Since the hacker has access to the source code he can reference it directly. Or even better, actually provide a working fix through the means of a patch. The disclosure process can therefore be quicker. Worth mentioning are also GitHub's Security Advisories that allow collaboration of the maintainers and the reporter on the fix together in a private branch.

A hacker has to be careful about his expectations. The program owners can update the severity (e.g. lowering it from High to Low) or discard the report completely by marking it as Informative. Such reports do not result in any bounty, but at least do not harm the hacker's reputation. Also, the triage results vary across the programs. One of the vulnerabilities I have reported was triaged as High, Low or disregarded as Informative when reported to three different programs—one has to avoid frustration and understand that the program owner is in charge. Sometimes, the frustration pushes the hacker to try harder and escalate the previous issue into something more severe, consequently, forcing the program owner to reconsider the previous decision.

If the report is triaged, all communication goes well and the bug is fixed one can expect to receive a bounty (after filling in the required paperwork on the platform). The time between submitting the report and the payout can be several weeks (though some programs pay an initial bounty right after the triage). The overall experience heavily depends on the program and can be a reason to stick to only a few programs or to move on to another one.

2.5 Is bug bounty hunting worth it?

There are several positives to bug bounty hunting. The hacker can choose her target. If there is a new technology that interests her, she can find a program that uses it and start learning. Looking for vulnerabilities gives a new perspective—quite different from the point of view of an ordinary developer. The goal is to find a way in, misuse anything that is available, and break things. There is no need to worry about the future deployment of the tools, and scripts that are created along the way. It is a technical and creative work that provides unbounded learning opportunities. Depending on a program the bounties can be of a significant amount. Also, some programs reward hackers with *a swag* or a Pro version of their services.

There are negatives as well. Staying *on toes*, and noting down any suspicious behaviour all the time is tiring after a while. Finding a good program that one enjoys exploring is not straightforward. Spending weeks diving into a piece of software and not finding anything is not an experience one can handle over and over again. A hacker is paid based on the value she brings to the program (higher severity bugs are rewarded a bigger bounty) not by hours—so, it is in the hands of the hacker, whether bug bounty hunting is sustainable long-term.

Looking from the other side, is it worth it for a company to have a bug bounty program? I can't comment on the experience of a program owner, especially concerning the time, effort and investment it takes to run it. I can comment as someone with an experience of trying to build something and also trying to break something. The inner incentives and motifs are quite different. The developer is driven to make *things work*. Finding an issue is a burden, not an achievement. Contrary, anything out of the ordinary fuels the hacker's curiosity. I can imagine a company simulating the bug bounty environment internally—reward tickets that report a security vulnerability, train developers in security and give them dedicated time to go and break the company's apps (preferably built by a different team). Also, a company can organize a hackathon, where the sole goal is to break its systems or create a team and participate in Capture the Flag style of events. That said, internal employees are easily in conflict of interests and therefore having a program hosted on a platform is a viable option.

However, the least a company can do is to clearly specify on the homepage or in the project's `README`, how is a hacker supposed to contact the owners in case they discover a security vulnerability. Surprisingly, even big projects are lacking security contact for reporting security vulnerabilities. Failing to provide such contact can either result in an unwanted public disclosure or no disclosure at all.

Resources

The hacking community produces a lot of materials—reports, videos, podcasts or blog posts. The amount of content can be quite overwhelming, but reading and understanding the published reports and techniques is a a great way how to hone one's hacking skills.

- HackerOne hactivity: a list of all the disclosed reports on the platform, to narrow it down take a look at @vakzz's work on GitLab, which I might revisit during the talk,

- Capture the Flag writeups: another interesting list of techniques used to circumvent the security and exploit a vulnerability.
- John Hammond, LiveOverflow and STÖK: hackers, educators and content creators on various things related to hacking and bug hunting,
- Bug Bounty Reports explained: a YouTube channel concerned with explaining the disclosed reports,
- The Bug Hunter's Methodology and here: to keep up with growing number of programs one benefits from a better methodology,
- Bugcrowd's LevelUp series: online conference targeting hackers and security researchers,
- dayzerosec: a podcast discussing the latest disclosed reports from all the platforms.

Bungle in the jungle: Analysing the security certifications landscape

Adam Janovsky, Petr Svenda, Jan Jancar, Jiri Michalik, Stanislav Bobon

Abstract

Timely notification of end-users about a vulnerability found and the corresponding fix typically relies on a vendor or public sources like vulnerability databases. Such notifications may fail even for highly sensitive and certified devices, as demonstrated, e.g., by that of the Estonian government about critical ROCA cryptographic vulnerability [1].

To facilitate an automatic vulnerability notification of (potentially) impacted certified composite products, we build a dependency graph of more than 10 thousand items certified under the Common Criteria (CC) and FIPS 140-2/3 schemes. Furthermore, we trained a classifier ($\approx 90\%$ accuracy) for pairing certified products with the corresponding entries in the CVE vulnerability database. Both tools work in an unsupervised manner, allowing for predictions on newly introduced vulnerabilities and certificates. Our dataset is further qualitatively evaluated using five high-profile past vulnerabilities – demonstrating our ability to quickly identify the impacted certificates (including previously omitted cases like Estonia's eID) and possibility to proactively identify set of certificates that may affect a composite product in the future.

While CC certification is primarily focused on the evaluation of security claims, it shall also improve the practical product's security. By correlating the achieved security assurance levels with the severity and time of past vulnerabilities, we identify that some security assurance requirements of the certificate are strong predictors for the expected number and severity of the vulnerabilities the certificate is likely to suffer from in the future. Among others, `AVA_VAN` (assumed attacker potential) and `ALC_TAT` (tools and techniques) have the highest inverse correlation with the number and severity of the expected vulnerabilities. We also reveal that majority of certificate maintenance updates that are subsequent to exposed vulnerabilities do not address these vulnerabilities explicitly. Based on the issues encountered in processing certificates issued over the past 25 years, we *create a list of recommendations on how to improve the newly issued certificates. The open tools and database of continuously updated results are available at* ***https://seccrets.org.***

1 Introduction

The security certification frameworks Common Criteria (CC) [2] and FIPS-140 [3] were introduced to evaluate the security of products and provide increased assurance to customers. The schemes typically offer multiple certification levels (e.g., Evaluation Assurance Level for CC) with increasing requirements and corresponding scrutiny for the claims made about the certified item. The increased scrutiny comes at a non-trivial cost, both in time to obtain the certificate (typically months or even years) and financial cost (hundreds of thousands of dollars or more) [4].

Despite such effort, serious security vulnerabilities were discovered later in the products certified even to high levels [5, 1, 6, 7], showing a gap between the intended and actual security. Due to the high certification costs, a vulnerability missed by the certification process tends to persist for number of years as vendor is less motivated to make changes in the product which would require re-certification – contributing to increased impact on end-users. Finally, notification of the users about a vulnerability found is typically left up to the vendor of the certified item. This is unnecessary as the item is quite precisely specified and identified (including its components, if being composite) during the certification process (e.g., Target of Evaluation in Security Target documents in CC). An automated notification would be possible if the certificate and referenced parts are uniquely identified and if such labeling is consistently used also in vulnerability database(s) – unfortunately, no such complete mapping exists today. This paper provides such mapping and performs related analyses.

Additionally, a simple question like *Is the certification improving the security of an actual product in correlation to the certification level?* seems to be difficult to answer – partially due to the complex task of measuring the security in general, but also due to burdensome to process, insufficient or missing input data for such analysis. But without the understanding of the certification efficiency, the certification procedure may become a burden preventing innovation from smaller companies due to its high cost rather than security improvement, and the well-meant improvements are ad-hoc rather than data-based.

Previous works unveiling the real situation of certification schemes were based on individual studies (e.g., products affected by specific vulnerability) [8], manual analysis on the small fraction of certificates [9] or qualitative analysis of the actual standards [10]. We instead focus on the extraction of the analytical information directly from the public documents of all the certified products and their correlation with other relevant sources of information like vulnerability databases.

We leverage the resulting processed dataset to provide insight into the state of the ecosystem, starting from the general trends in the certification that runs for more than two decades, over empirical evidence of the security improvement achieved (when measured by the number and severity of the vulnerabilities found) to providing tools for vulnerability impact assessment and early notification of the product end-users.

While attempted, a reliable and complete mapping between certified products and records in other sources like CVE vulnerability database is not fully available, making it difficult to establish which of the publicly disclosed vulnerabilities are relevant to a particular certified product. For example, ROCA vulnerability (CVE-2017-15361) in RSA keypair generation directly affecting more than a hundred CC certificates is listed in the CVE database as a vulnerability in Trusted Platform Module firmware with 130 affected CPE configurations (mostly desktop and laptops), but none of it being the certified CC item. Yet, we show that many certified devices are, in fact, affected.

In this paper, we answer the following research questions: **RQ1**: Can certified items automatically be mapped to records in CVE vulnerability database? **RQ2**: What is the practical impact of increased security assurance levels on the expected number and severity of the products' vulnerabilities? **RQ3**: How are certified items referencing and affecting each other?

We address these problems, delivering the following contributions:

- Systematic analysis of inter-certificates references in all certificates issued under Common Criteria and FIPS 140 schemes using a newly developed open-source framework. The resulting dependency graph provides means for automated vulnerability notification and insight into the certification ecosystems for the past 25 years.

- Quantitative analysis of the real-world security impact of the different certification security levels based on the automatic pairing of certified devices with the corresponding records in CVE vulnerability database.

- Qualitative study of improvements in vulnerability assessment for security researchers gained by the developed framework based on the five real-world noteworthy vulnerabilities found in certified products.

The open-source toolchain, manually labeled training datasets, vulnerability notification feature and all results continuously updated with newly issued certificates are available at `https://seccrets.org`. Note that the Europen paper merely provides the introduction, state of the art, and the methodology. The complete results, as well as the practical demonstration of our work, will be part of the presentation and are not included in the printed version of the paper, for they are still partially a work in progress.

2 Related work

We start with a short description of the Common Criteria certification framework. After doing so, we turn our attention to the existing work that is most relevant for their analysis.

2.1 CC framework description

In the Common Criteria certification scheme, any security-related product can be certified. The primary focus of the certification is the production process. The assumption is that if the process is right, the products will likely be secure as well. Prior to certification, the applicant chooses security functional requirements (SFR) and security assurance requirements (SAR) to cohere to. Preferably, the applicant chooses from an already established list of protection profiles (PP) that bind typical use-cases (e.g., smart cards) to their SFR and SAR specifications. During the certification, an independent evaluator party verifies the conformance of the device to this specification. Depending on the chosen protection profile of security requirements, the device can be certified on multiple evaluation assurance levels (EAL). The following documents that accompany the certification process and/or are publicly available were used for our analysis:

- Security Target document – provided by the vendor (or on behalf) to the evaluation facility, specifies the certified product.

- Certification Report – issued by certification authority member (e.g., French ANSSI), after checks by an accredited evaluation facility/lab (e.g., Serma Technologies).
- Maintenance Report(s) – documenting smaller changes in an already certified product that do not require full re-certification.
- Protection Profiles documents – template for specific functionality, provided by a single vendor or collaborative.
- CSV/HTML pages with some additional metadata, summary documents.

2.2 Certification frameworks analysis

We divide the studied papers into three different categories: *(i)* papers that identify problems in certification schemes, often suggesting immediate improvements; *(ii)* papers that try to make sense of the certification process, often case-study driven; *(iii)* more recent papers that work towards automated and transparent certification, or automated processing of security-related documents.

Most of the research on CC and FIPS certification schemes builds on individual case studies that expose the perks of these schemes. This also limits the possible findings, as the authors are mostly steered by intuition, lacking the support of large-scale data analysis. An exception to this is a study [9] that exploits data-driven analysis of CC scheme, parsing metadata from its website. This work, however, omits the analysis of the pdf documents and resorts to manual labour when examining vulnerabilities, being limited only to a small portion of the dataset.

Perks and problems of certification

In [11], Harn exposes the problems that the CC had already in 2004. Eight years later, Murdoch et al. [12] discuss why both CC and FIPS-140 fail, pointing to the lack of transparency, a problem that prevails. More of a political perspective is taken in [13], describing the weaknesses of international standards that rely on national alliances, which may not last forever. Work [10] studies the intriguing fact that CC requires formal models to achieve high EALs, however, formal verification of the implementation is not required, despite the recent advances in the field.

Many problems to be articulated in our work are also mentioned in user study [14] of 29 users experienced in work with NIST cryptographic standards and validation programs.

Understanding certification

To many, employed certification schemes may appear so complex and full of nontransparent limitations that it is difficult to understand how one can obtain such certification, and what the implications for the certified device are. This is tackled by several papers that explain the process of certification [15] or provide guidance on how to approach it [16, 17, 18, 19, 20]. Some works treat separate product domains for which they provide support (e.g., IoT devices) [21, 22, 23, 24]. Also, an interesting study [25] dissects the Windows version that acquired EAL4 certification in 2004. A novel piece of research recently studied the costs introduced by securing software, focusing on Common Criteria as well [26].

Automated processing of cryptography-related documents

Here, we limit our attention to recent natural language processing (NLP) studies that could be translated to the domain of security certificates. Additionally, we mention some papers devoted to linking computer systems to known vulnerabilities from National Vulnerability Database (NVD).

We believe that much precious information is hidden in the certification reports. More powerful approach than regular expressions could be leveraged to extract the context from these documents. For instance, it would be possible to approach recent advances in NLP as already suggested for FIPS-140 by [27], or performed on privacy policy documents in [28]. A study from 2014 [29] attempts to automatically identify security-related sentences and security requirements from natural language artifacts using machine learning.

In our work, we assumed that a properly processed CPE database and perfect matching between CVE and CPE database is achieved by NIST. But even this is an active area of research, as demonstrated by works that try to make this process more efficient, less erroneous, and more complete [30, 31, 32].

3 Data processing and methodology

In this section, we describe the methodology and tooling of our analytical work. To answer our research questions, we devise surrogate metrics that we collect from certification documents and afterward evaluate. We strive to make our methods fully automated, at the cost of possible false positives and false negatives. We hope that this is positively outweighed by scaling our research to all available certificates. Wherever possible, we manually evaluate the precision of our methods, always report the most conservative numbers, and carefully examine the positive findings.

3.1 Tooling

As a part of our research, we release an open-source tool [1] written in Python for analysis of both CC and FIPS 140 certificates. We are also continuously processing all available data from both CC and FIPS 140 schemes, generating daily snapshots of their landscape. These temporal trends, as well as information about individual certificates, are displayed on our website `seccerts.org`, where one can browse through our daily updated datasets and analyses. The users can also opt-in for receiving notifications about certificates that are subject of their interest, e.g., when a new vulnerability is identified that likely affects a given certificate.

3.2 Dataset processing

The tooling outlined above is used to parse html source of CC [33] and FIPS [34] websites. These webpages are crawled for the lists of certificates containing various metadata (category of product, security level, etc.), which are subsequently downloaded and processed. From these lists, hyperlinks to all pdf certification documents are recovered and the respective files downloaded. We then convert these documents to text files and apply three processing steps:

1. We leverage regular expressions to extract features from the documents. Among others, we search for security assurance requirements, connections to other certified products, clues of existing vulnerabilities, evaluation laboratories, or employed cryptographic primitives.

[1] `github.com/crocs-muni/sec-certs`

2. We correlate our dataset with the National Vulnerability Database (NVD) to identify potentially vulnerable certificates.

3. We normalize the existing certificate IDs to build a graph of dependencies between the certified devices (even across the schemes).

The exhaustive list of extracted features is depicted in Table 1. The CC dataset is also enhanced with processed maintenance updates and protection profiles. In total, we have analyzed 4 170 CC certificates and 4 937 FIPS-140 certificates. Notably, the certification documents were not designed and written with automated processing in mind. Instead, they are manually written by humans for humans. This complicated the feature extraction, as we faced many inconsistencies along the way. The rest of this section conveys the technical facets of vulnerability matching, feature extraction, and dependency graphs.

3.3 Matching certificates to National Vulnerability Database

We designed a classifier that maps each of the certificates to a list of vulnerabilities that likely affect it. In prior work [9], some small fraction of certificates was manually assigned with CVEs and their characteristics were further studied. In contrast, our method is fully automated and works with previously unseen certificates or vulnerabilities. We deploy the matching on our website to perform daily scans for new vulnerabilities. Our main goal is to study the relationship between exposure to vulnerabilities and the process of certification. In other words, what aspects of certification (e.g., security level, the extent of automated tests) have an influence on the number of vulnerabilities (and their severity) the certificate suffers from?

Classifier design

Each vulnerability record in the NVD database is associated with a list of vulnerable product configurations. These configurations are shaped as Common Platform Enumeration (CPE) records. Human experts craft the lists of vulnerable CPEs for all vulnerabilities. Assuming that these records are created flawlessly, knowing CPE of a product, one can traverse all vulnerabilities to obtain those that affect the product. Further in the text, we assume that the mapping between CPEs and CVEs is perfect, i.e., that each CVE record contains all (but no more) CPE records that correspond to the vulnerable configurations. For more information about CVE and CPE syntax and semantics, we refer the reader to [35].

Leveraging this assumption, we create a classifier that assigns CPEs to the certified products, and CVEs are then reconstructed deterministically. CPE is a flat data structure, a list of fields related to the given platform. Among others, it contains the self-explanatory fields `<vendor>`, `<product>`, `<version>`. Apart from those, the CPE dataset released by NIST [36] also often contains the human-readable title of the product. To match the certified devices to CPEs, we measure the string similarity between a certificate heading (i.e., name) and candidate CPE records that are carefully pre-selected. To measure the string similarity, we utilize Levensthein distance. The threshold for declaring a match is set dynamically to obtain results that achieve 90% precision, where a manually labeled subset of the data is used for external evaluation. While higher precision is possible, it asymmetrically decreases the recall, i.e., the number of matched CPE records. Before including a CPE record into a list of candidates to match a given certificate (the string similarity is computed only on promising candidates), we impose the following set of conditions that must hold:

- The CPE record must be at least 5 characters long (short CPEs lead to 100% string similarity)
- The vendor of the certificate must match the vendor CPE
- The version of the certificate must be contained in the version string of the CPE or vice versa. E.g., matching a certified product of version 5.1.2 to CPE record of version 5.1. is allowed.

Classifier evaluation

Once the model was trained, we randomly sampled 200 certificates and manually labeled the obtained CPE matches. This task was done independently by a pair of experts, and the conflicting instances were revisited until a consensus was reached.The required similarity score was then increased to 100% to achieve $\approx 90\%$ precision of classification. Given a certificate and the matched CPE records, precision denotes the probability that the CPE record is a true positive, i.e., that it corresponds to the certificate. Note that we could not evaluate the recall, as it is difficult to manually obtain the complete list of CPE records for each of the certificates. For that reason, our estimates about the number of vulnerabilities form a conservative lower bound, with probably more CPEs and CVEs related to the certified devices.

Caveats and limitations

Along the way, we identified aspects that can be improved to allow for simple, unambiguous mapping between certificates and vulnerabilities. The main message is that each of the certified products should be assigned with its own CPE (or some other unambiguous identifier) on creation. This would solve numerous problems. For instance, we needed to normalize vendor names of both certificates and CPEs, as some are given in the form of abbreviation (e.g., CPE records contain *Hewlett Packard* as a vendor, while certificates merely use *HP*). The same problem also concerns product names (e.g., Internet Explorer, IE). Also, many CPEs listed in the NVD are actually missing from the official CPE dataset, and some are missing a version identifier. Furthermore, a small portion (6.8% as of September 2021) of vulnerabilities does not list any vulnerable CPEs. A manual inspection confirmed that these represent rather exotic vulnerabilities unlikely to affect the certificate devices.

We attempted to advance our approach by matching certificate names directly to the vulnerability descriptions to bypass these problems, but this fell short of its promise, delivering deteriorated performance.

3.4 Certificate feature extraction

Using expert knowledge, we have defined 400 regular expressions to extract natural text keywords from the certification documents. These features can be divided into several categories, as listed in Table 1. The complete list of regular expressions can be examined directly in the source code [37]. In CC, we traversed both the certification report and the security target documents to match these expressions.

3.5 Extraction of certificate references

The list of other certified products referenced from the specific certificate is vital information allowing us to assess the products impacted with newly found vulnerability. For example, electronic identity cards utilizing a platform built atop the certified integrated circuit with fresh vulnerability shall be assessed for impact. The complete graph of all references among all

Feature category	Framework
Certificate IDs (17 distinct authorities)	CC
Employed protection profiles	CC
Referenced standards (e.g., ISO-27001)	CC, FIPS
Achieved security level	CC, FIPS
SAR, FAR	CC
CC Claims (objectives, threats, ...)	CC
Javacard features (e.g., Java version)	CC, FIPS
Employed cryptographic algorithms	CC, FIPS
Cryptographic libraries (e.g., GnuTLS)	CC, FIPS
Applied defenses (e.g., timing attacks)	CC, FIPS
Vulnerabilities (CVE-XXXX-YYYY)	CC, FIPS

Table 1: Features extracted from the certification documents.

certified products would allow to automatically identify all other affecting directly or transitively the specified one, all affected by the specified one, and provide insight into the dependency complexity of such graph. But as the current certification procedures require no explicit software and hardware "bill of material (BOM)", such a graph is not existing and has to be built from the publicly available certification documents.

Every certificate is assigned its own supposedly unique but structurally different number under the national issuing scheme (e.g., BSI-DSZ-CC-1169-2021, NSCIB-CC-15-66461, or Rapport de certification 2013/42). The certificate id always contains a component with an incremental number (may reset between years), mostly year of issuance, sometimes additional modifiers for the certification or maintenance reports (CR/MR), version, or country-specific strings (BSI, NSCIB, ANSSI). Importantly, CC is *not* even providing authoritative pairing of certified product and its id in public documents, nor a list of all referenced certificates. Additionally, the references are human-written and frequently contain errors (around 5% of all references) – missing structural parts (e.g., a year or a version), containing typos or referencing via product name instead. To build a pairing and list of all referenced certificates, one must parse the free-form documents and attempt to correct and normalize possible typos.

We build the graph of all references using the following steps:

1. Extraction of candidate certificate id using set of regular expressions manually built based on the naming structures for all current and past certificate authoring members.
2. Parsing of the certificate front page (if present) to establish the id of a given certificate. If not present, other heuristics (most frequent candidate, occurrence in certificate file name) are used.
3. All extracted certificate ids are corrected for the typical errors and typos occurring for given naming scheme and transformed to a normalized form using a set of manually built, predefined rules and all ids collected so far, allowing to completion of missing elements like "BSI-DSZ-CC-1137-V2" to "BSI-DSZ-CC-1137-V2-2021".

In total, we found 6166 certificate identifiers for the Common Criteria of 16 certificate issuing countries, following 24 different structural patterns (some countries changed the numbering scheme over time).

3.6 Limitations

Our analysis comes with several limitations, mostly due to incomplete and noisy input data used. Below, we list the most important ones and give short discussion about the impact of limitations on the presented results.

- **Noisy base data.** The certification documents are written primarily by humans for other humans and without automatic processing in mind, resulting in an unstructured, ambiguous, and inconsistent primary source of the data. Typos are found even in the critical elements like certificate identification. We attempt to heuristically correct some using redundant input sources (e.g., pdf and csv), expected data structure (e.g., year in certificate id) and manually investigating inconsistencies (e.g., same certificate id assigned to multiple certificates). As a result, some certificates may be incorrectly classified and matched. But based on the manual investigation of a randomly selected subset of certificates, we believe the errors are generally acceptably low and do not impact overall conclusions.
- **Only public data is processed.** Additional documents are created during the certification process, which remains confidential between a vendor, evaluation facility, or certification body. Our analysis is missing such additional metadata, resulting in fewer connections

in the graph of references and vulnerabilities evaluated during the (re-)certification process.

- **Only part of the product is certified.** A security target document specifies the boundaries of the evaluated product (Target of Evaluation, ToE), which typically does not cover the whole product or all configuration options. A vulnerability found in the parts excluded from ToE will likely be paired to the certificate, while formally not relevant to the certified sub-part, introducing inaccuracies in the results presented. While clearly classification error when only ToE is strictly considered, we believe that imprecise such result generally matches the expectations of ordinary end-users using the certified product. Especially given the anecdotally evidence that some vendors incrementally limit the ToE scope in response to vulnerability found during the evaluation instead of fixing it. Additionally, one can expect that somewhat comparable level of quality of product parts despite some being excluded from the ToE.
- **Not all vulnerabilities are reported in CVE database.** Not all vulnerabilities found are listed in CVE database and/or properly referenced to the product's platform CPE. As a result, more vulnerabilities likely exist for every certified product than matched by our analysis. We conjecture that the rate of unreported vulnerabilities is comparable among the products certified in the same certification category and do not compare between the categories.

References

[1] Matús Nemec et al. „The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 1631–1648. doi: 10.1145/3133956.3133969. url: https://doi.org/10.1145/3133956.3133969.

[2] *Common Criteria for Information Technology Security Evaluation.* version 3.1, revision 5. [cit. 2020-09-03]. Available from https://www.commoncriteriaportal.org/cc/. 2020.

[3] National Institute of Standards and Technology (NIST). *Security Requirements for Cryptographic Modules.* [cit. 2020-09-03]. Available from `https://csrc.nist.gov/publications/detail/fips/140/2/final`. 2001.

[4] Lightship security. *FAQ on Common Criteria.* Available from `https://lightshipsec.com/common-criteria/`. 2022.

[5] Gunnar Alendal, Stefan Axelsson, and Geir Olav Dyrkolbotn. „Chip chop — smashing the mobile phone secure chip for fun and digital forensics“. en. In: *Forensic Science International: Digital Investigation* 37 (July 2021), p. 301191. ISSN: 26662817. DOI: `10.1016/j.fsidi.2021.301191`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S2666281721000998` (visited on 12/15/2021).

[6] Jan Jancar et al. „Minerva: The curse of ECDSA nonces; Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces“. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.4 (2020), pp. 281–308. DOI: `10.13154/tches.v2020.i4.281-308`. URL: `https://doi.org/10.13154/tches.v2020.i4.281-308`.

[7] Daniel Moghimi et al. „TPM-FAIL: TPM meets Timing and Lattice Attacks“. In: *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020.* Ed. by Srdjan Capkun and Franziska Roesner. USENIX Association, 2020, pp. 2057–2073. URL: `https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm`.

[8] Shaanan N. Cohney, Matthew D. Green, and Nadia Heninger. „Practical State Recovery Attacks against Legacy RNG Implementations“. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 265–280. ISBN: 9781450356930. DOI: `10.1145/3243734.3243756`. URL: `https://doi.org/10.1145/3243734.3243756`.

[9] Samuel Paul Kaluvuri, Michele Bezzi, and Yves Roudier. „A Quantitative Analysis of Common Criteria Certification Practice“. In: *Trust, Privacy, and Security in Digital Business.* Vol. 8647. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 132–143. ISBN: 978-3-319-09769-5 978-3-319-09770-1. DOI: `10.1007/978-3-319-09770-1_12`. URL: `http://link.springer.com/10.1007/978-3-319-09770-1_12` (visited on 12/14/2021).

[10] Bernhard Beckert, Daniel Grahl, and Sarah Grebing. „Mind the Gap: Formal Verification and the Common Criteria (Discussion Paper)“. In: *VERIFY@IJCAR*. 2010.

[11] J. Hearn. „Does the common criteria paradigm have a future?“ en. In: *IEEE Security & Privacy Magazine* 2.1 (Jan. 2004), pp. 64–65. issn: 1540-7993. doi: 10.1109/MSECP.2004.1264857. url: http://ieeexplore.ieee.org/document/1264857/ (visited on 12/14/2021).

[12] Steven Murdoch, Mike Bond, and Ross J. Anderson. „How Certification Systems Fail: Lessons from the Ware Report“. In: *IEEE Security & Privacy Magazine* (2012), pp. 1–1. issn: 1540-7993. doi: 10.1109/MSP.2012.89. url: http://ieeexplore.ieee.org/document/6231616/ (visited on 12/14/2021).

[13] Jan Kallberg. „The Common Criteria Meets Realpolitik: Trust, Alliances, and Potential Betrayal“. In: *IEEE Security & Privacy Magazine* 10.4 (July 2012), pp. 50–53. issn: 1540-7993. doi: 10.1109/MSP.2012.29. url: http://ieeexplore.ieee.org/document/6148206/ (visited on 12/16/2021).

[14] Julie Haney et al. *Organizational views of NIST cryptographic standards and testing and validation programs*. Tech. rep. NIST IR 8241. Gaithersburg, MD: National Institute of Standards and Technology, Dec. 2018, NIST IR 8241. doi: 10.6028/NIST.IR.8241. url: https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8241.pdf (visited on 12/15/2021).

[15] John Tierney and Tony Boswell. „Common Criteria: Origins and Overview“. In: *Smart Cards, Tokens, Security and Applications*. Cham: Springer International Publishing, 2017, pp. 193–216. isbn: 978-3-319-50500-8. doi: 10.1007/978-3-319-50500-8_8. url: https://doi.org/10.1007/978-3-319-50500-8_8.

[16] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. „A Common Criteria based security requirements engineering process for the development of secure information systems“. en. In: *Computer Standards & Interfaces* 29.2 (Feb. 2007), pp. 244–253. issn: 09205489. doi: 10.1016/j.csi.2006.04.002. url: https://linkinghub.elsevier.com/retrieve/pii/S0920548906000511 (visited on 12/16/2021).

[17] Debra S Herrmann. *Using the Common Criteria for IT security evaluation*. English. Boca Raton: Auerbach Publications, 2003. ISBN: 978-0-8493-1404-9 978-1-4200-3142-3.

[18] M. Razzazi et al. „Common Criteria Security Evaluation: A Time and Cost Effective Approach". In: *2006 2nd International Conference on Information & Communication Technologies*. Vol. 2. Damascus, Syria: IEEE, 2006, pp. 3287–3292. ISBN: 978-0-7803-9521-3. DOI: `10.1109/ICTTA.2006.1684943`. URL: `https://ieeexplore.ieee.org/document/1684943/` (visited on 12/16/2021).

[19] Monika Vetterling, Guido Wimmel, and Alexander Wisspeintner. „Secure systems development based on the common criteria: the PalME project". en. In: *ACM SIGSOFT Software Engineering Notes* 27.6 (Nov. 2002), pp. 129–138. ISSN: 0163-5948. DOI: `10.1145/605466.605486`. URL: `https://dl.acm.org/doi/10.1145/605466.605486` (visited on 12/16/2021).

[20] Andrzej Białas. „Patterns Improving the Common Criteria Compliant IT Security Development Process". In: *Dependable Computer Systems*. Vol. 97. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–16. ISBN: 978-3-642-21392-2 978-3-642-21393-9. DOI: `10.1007/978-3-642-21393-9_1`. URL: `http://link.springer.com/10.1007/978-3-642-21393-9_1` (visited on 12/16/2021).

[21] Dariusz Rogowski. „Software Support for Common Criteria Security Development Process on the Example of a Data Diode". In: *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30 – July 4, 2014, Brunów, Poland*. Cham: Springer International Publishing, 2014, pp. 363–372.

[22] Andrzej Bialas. „Common Criteria Related Security Design Patterns—Validation on the Intelligent Sensor Example Designed for Mine Environment". en. In: *Sensors* 10.5 (Apr. 2010), pp. 4456–4496. ISSN: 1424-8220. DOI: `10.3390/s100504456`. URL: `http://www.mdpi.com/1424-8220/10/5/4456` (visited on 12/16/2021).

[23] Samuel Paul Kaluvuri, Michele Bezzi, and Yves Roudier. „Bringing Common Criteria Certification to Web Services". In: *2013 IEEE Ninth World Congress on Services*. Santa Clara, CA, USA: IEEE, June 2013, pp. 98–102. ISBN: 978-0-7695-5024-4. DOI: `10.1109/SERVICES.2013.17`. URL: `http://ieeexplore.ieee.org/document/6655681/` (visited on 12/16/2021).

[24] Sooyoung Kang and Seungjoo Kim. „How to Obtain Common Criteria Certification of Smart TV for Home IoT Security and Reliability“. en. In: *Symmetry* 9.10 (Oct. 2017), p. 233. ISSN: 2073-8994. DOI: 10.3390/sym9100233. URL: http://www.mdpi.com/2073-8994/9/10/233 (visited on 12/16/2021).

[25] J.S. Shapiro. „Understanding the windows EAL4 evaluation“. en. In: *Computer* 36.2 (Feb. 2003), pp. 103–105. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1178059. URL: http://ieeexplore.ieee.org/document/1178059/ (visited on 12/16/2021).

[26] Elaine Venson et al. „Costing Secure Software Development: A Systematic Mapping Study“. en. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security.* Canterbury CA United Kingdom: ACM, Aug. 2019, pp. 1–11. ISBN: 978-1-4503-7164-3. DOI: 10.1145/3339252.3339263. URL: https://dl.acm.org/doi/10.1145/3339252.3339263 (visited on 12/16/2021).

[27] Apostol Vassilev. „BowTie-A deep learning feedforward neural network for sentiment analysis“. In: *International Conference on Machine Learning, Optimization, and Data Science.* Springer. 2019, pp. 360–371.

[28] Hamza Harkous et al. „Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning“. In: *27th USENIX Security Symposium (USENIX Security 18).* Baltimore, MD: USENIX Association, Aug. 2018, pp. 531–548. ISBN: 978-1-939133-04-5. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/harkous.

[29] Maria Riaz et al. „Hidden in plain sight: Automatically identifying security requirements from natural language artifacts“. In: *2014 IEEE 22nd International Requirements Engineering Conference (RE).* 2014, pp. 183–192. DOI: 10.1109/RE.2014.6912260.

[30] Daniel Tovarňák, Lukáš Sadlek, and Pavel Čeleda. „Graph-Based CPE Matching for Identification of Vulnerable Asset Configurations“. In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM).* 2021, pp. 986–991.

[31] Emil Wåreus and Martin Hell. „Automated CPE Labeling of CVE Summaries with Machine Learning“. In: *Detection of Intrusions and Malware, and Vulnerability Assessment.* Ed. by Clémentine Maurice et al. Cham: Springer International Publishing, 2020, pp. 3–22. ISBN: 978-3-030-52683-2. DOI: 0.1007/978-3-030-52683-2_1.

[32] Luis Alberto Benthin Sanguino and Rafael Uetz. „Software Vulnerability Analysis Using CPE and CVE“. In: *arXiv:1705.05347 [cs]* (May 2017). arXiv: 1705.05347. URL: `http://arxiv.org/abs/1705.05347` (visited on 12/15/2021).

[33] CC. *Common Criteria Portal.* Available from `commoncriteriaportal.org/`. 2022.

[34] NIST. *Computer security resource center.* Available from `https://csrc.nist.gov`. 2022.

[35] David Waltermire et al. *The technical specification for the security content automation protocol (SCAP) version 1.3.* Tech. rep. NIST SP 800-126r3. Gaithersburg, MD: National Institute of Standards and Technology, Feb. 2018, NIST SP 800–126r3. DOI: `10.6028/NIST.SP.800-126r3`. URL: `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-126r3.pdf` (visited on 12/16/2021).

[36] NIST. *National Vulnerability Database - Data Feeds.* Available from `https://nvd.nist.gov/vuln/data-feeds`. 2022.

[37] Adam Janovsky et al. *Sec-certs GitHub repository: Examined regular expressions.* Available from `https://github.com/crocs-muni/sec-certs/blob/b84b03a57c81636bc617eccc1d03f9b6ad6a15ac/sec_certs/cert_rules.py`. 2022.

Vládní certifikace a otevřený software

Jaroslav Řezník

„Kybernetická bezpečnost“ se v poslední době stává velkým tématem, zejména po mnoha (nedávných) bezpečnostních incidentech nejen v dodavatelských řetězcích a s nárůstem zranitelností v softwaru a hardwaru se nacházíme v bezprecedentní době, kdy je téměř každý systém nějakým způsobem zranitelný (a máme štěstí, když jsou zranitelnosti známé a v ideálním případě již opravené). Existuje také mnoho geopolitických důsledků počítačové bezpečnosti, které můžeme pozorovat téměř v reálném čase v boji mezi „hodnými“ a „zlými“ hochy a vlády si samozřejmě chtějí uchovat svá tajemství (ale rády by znaly tajemství ostatních). Pro ochranu těchto informací existuje mnoho norem a certifikací. V tomhle článku se zaměříme především na severoamerické normy a certifikace (i když se může jednat o mezinárodní standard) a to především z praktického hlediska projektového řízení.

Totéž platí pro open source – v dnešní době téměř každý produkt obsahuje alespoň část open source kódu a mnoho komerčně dostupných produktů je dokonce založeno zcela na open source. Vlády přijímají open source každým rokem více a více vzhledem k jeho flexibilitě, cenovým výhodám a možnosti review kódu.

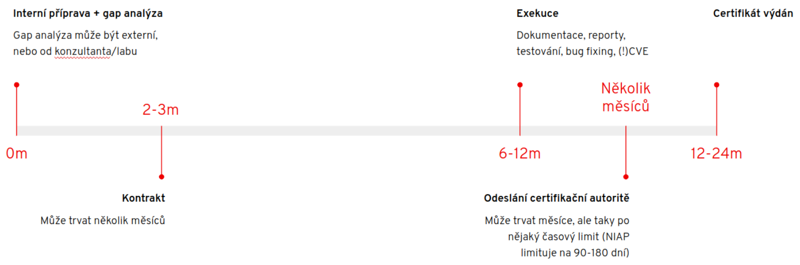
Co se ale stane, když tyto dvě věci spojíme? Jsou vládní standardy, certifikace, bezpečnost a open source kompatibilní, nebo ne? Odpověď zní ano. Ale jako vždy „s ale“ …

Začneme odpovědí na otázku jaké certifikace máme a jaké jsou problémy při certifikaci open source softwaru (i když je distribuován jako součást komerčního enterprise řešení).

1 Bezpečnostní (vládní) certifikace

Mezi nejznámější bezpečnostní certifikace patří:

- NIST SP FIPS 140-2 a nástupnický FIPS 140-3
- Common Criteria for Information Technology Security Evaluation


Interní příprava + gap analýza
Gap analýza může být externí, nebo od konzultanta/labu
Exekuce
Dokumentace, reporty, testování, bug fixing, (!)CVE
Certifikát výdán
Několik měsíců
2-3m
0m
6-12m
12-24m
Kontrakt
Může trvat několik měsíců
Odeslání certifikační autoritě
Může trvat měsíce, ale taky po nějaký časový limit (NIAP limituje na 90-180 dní)

Obrázek 1: Typický průběh certifikací

1.1 Proces

Ačkoliv se jednotlivé certifikace liší (především v cílech a terminologii), proces je obvykle velmi podobný.

1. Výběr akreditované laboratoře
2. „Gap“ analýza, aneb prvotní review vlastností produktu vůči standardu. Pokud produkt požadavky nesplňuje, je nutné nejprve nedostatky napravit, nebo (pokud to jde) změnit rozsah/cíl.
3. Dokumentace, především v případě FIPS veřejné dokumenty Security Policy a pro Common Criteria Security Target, ale také neveřejné dokumenty popisující design produktu (low/high level), entropii, reporty atd.
4. Testování produktu vůči požadavkům certifikace a případné opravy chyb
5. Analýza CVE a jejich případné opravy
6. Odeslání „balíčku“ certifikační autoritě
7. Reakce na připomínky certifikační autority, případné opravy nebo i re-testování
8. Certifikát je vydán.

1.2 Výzvy

Jak je zřejmé z ilustračního obrázku výše, certifikace jsou obvykle velmi pomalé a časově a finančně náročné a v případě, že gap analýza odhalí nedostatky, může se takový proces protáhnout o několik měsíců až let. Jedním z velkých problémů jsou také časté změny ve standardech, které mají buď velmi krátké přechodné období, nebo mohou mít i okamžitou platnost jako jsou například Technical Decisions NIAPu. Tohle se samozřejmě projeví na ceně jak vývoje, tak samotné certifikace. Bavíme se od vyšších desítek tisíc dolarů, až po miliony dolarů.

V případě open source je pak důležitý i přístup upstream projektů. Některé projekty jsou k certifikacím přátelské, jako například OpenSSL – možná nejčastěji FIPS 140 validovaný modul, kde upstream při vývoji na FIPS přímo myslí. Naopak některé open source projekty (či spíše jedinci) jsou k certifikacím i vyloženě nepřátelské. Z pohledu open source se ale dá tenhle přístup pochopit – kvůli nákladům jsou certifikace přístupné jen velkým komerčním společnostem, které využívají open source a samotným open source projektům jsou tak prakticky zapovězeny.

Open source samozřejmě přináší i výhody. Code reviews jsou výrazně jednodušší, stačí odkázat laboratoře na repozitář s kódem a není třeba řešit sdílení kódu proprietární aplikace. Do open source projektů pak mohou s podporou certifikací přispívat všichni, kterým na podpoře záleží a v komerční sféře jsou navzájem konkurenty. Otevřený kód ale v některých případech vede až ke stavu, že se část napsaná komunitou schválně přehlíží (jakoby neexistovala), neboť v open source komunitách často neexistuje žádná kontrola vývoje a naopak se velmi tlačí na procesy v době, kdy se kód převezme do downstreamu.

2 FIPS 140-2 a FIPS 140-3

FIPS je zkratka pro Federal Information Processing Standards – tedy sada standardů pro zpracování informací (především) v amerických vládních organizacích a regulovaných odvětvích. FIPS 140-2 a jeho nástupnický FIPS 140-3 se zabývají validací kryptografických modulů (Cryptographic Module), kdy modul může být jako softwarový, tak hardwarový různé úrovně zabezpečení (Security Level). Nejnižší je úroveň 1 – většinou softwarové moduly, nejvyšší pak úroveň 4. Za FIPS 140 stojí National Institute of Science and Technology (NIST) a jeho dva programy CAVP (Cryptographic Algorithm Validation Program) a CMVP (Cryptographic Module

Validation Program). Primárním cílem FIPS 140 je ochrana vládních dat - jejich šifrování a bezpečné ukládání.

V současné době je aktuální verze FIPS 140-3, která je zároveň založena na mezinárodním standardu ISO/IEC 19790:2012. FIPS 140-3 je platný od září 2021 a plně nahradil FIPS 140-2. Původní certifikáty zůstávají platné (platnost FIPS 140 certifikátů je obvykle pět let), ale není možné již získat FIPS 140-2 certifikát. Jediné povolené změny jsou takové, které neprodlužují jeho platnost (tedy například scénáře 1SUB, 3ASUB, 3BSUB – viz níže).

Z technických požadavků FIPSu je třeba zmínit self testy, kdy se moduly při prvním spuštění musí otestovat jestli fungují správně a omezení nepovolených algoritmů. V některých případech může omezení být pouze „papírové“ (a tedy dokumentované v Security Policy), ale některé moduly naopak mohou přestat fungovat. Velmi striktní požadavky jsou pak i pro generátory náhodných čísel. Povolení FIPS módu tak často přináší nečekané problémy, které mohou být složité na nalezení a opravu.

2.1 Proces

Validaci modulů provádí akreditovaná laboratoř (např. Atsec), součástí je testování vůči požadavkům a dokumentace. Nejdůležitějším dokumentem je veřejně dostupný Security Policy dokument, který popisuje jak samostatný modul, jeho kryptografickou část (cryptographic boundary), ale taky požadavky na to, jak modul správně používat. V případě kryptografických knihoven může například omezovat které API je možné využít a obsahuje seznam schválených a neschválených algoritmů, které modul implementuje. Náročnost validace pak záleží na jednotlivých scénářích.

Mezi základní scénáře patří

- 1SUB (nově 1VI, 1VA, 1UP atd.) – jiné než bezpečnostní změny (non-security relevant) ve formě aktualizace dokumentace, neplatí se NIST recovery fee, změny se projeví v existujícím certifikátu
- 3SUB (nově 1MU, 3MC) – security-relevant změny do 30 procent kódu, platí se NIST recovery fee, NIST CMVP vydá nový certifikát, vyžaduje testování, podvarianty jsou
  - 3ASUB (nově 3CVESI) – zrychlený proces pro revalidaci v případě CVE, žádné jiné security-relevant změny nejsou dovoleny
  - 3BSUB – aktualizace kvůli změně standardu
- 5SUB (nově 5FS) – nové moduly, nebo změny, které nespadají pod žádný jiný scénář, obvykle víc jak třicet procent security-relevant kódu

Pro každý schválený algoritmus je pak třeba mít algoritmové certifikáty. Jedná se o Known Answer Test (KAT), kdy laboratoř vyžádá ze serveru ACVP testovací vektory, ty se spustí a výsledky se odešlou na server pro konfirmaci výsledků.

2.2 Problémy specifické pro FIPS 140

Aktuálně (duben 2022) je největším problémem doba trvání validace na straně NISTu. Jedním z důvodů je přechod mezi verzemi standardu FIPS 140-2 a FIPS 140-3. Není neobvyklé, že moduly čekají na certifikát víc jak deset měsíců. Po tuhle dobu jsou uveřejněné na Modules In Process (MIP) listu. Důsledkem je pak nejen to, že zákazníci musí čekat na nové verze modulů, často i na end-of-life nepodporovaných verzích, ale taky že čerstvě vydané certifikáty jsou na verze, které obsahují bezpečnostní problémy (CVE). Vede to tak ke kuriózní situaci, kdy si uživatelé musí vybrat, jestli chtějí být striktně compliant, nebo používat verzi bez známých zranitelností.

3 Common Criteria

Common Criteria je mezinárodní standard ISO/IEC 15408 pro certifikaci počítačové bezpečnosti, celým názvem Common Criteria for Information Technology Security Evaluation, běžně se zkracuje jako CC. Certifikáty udělují národní certifikační autority (schémata). V USA se tak jedná o NIAP (National Information Assurance Partnership), v Německu pak BSI (Bundesamtes für Sicherheit in der Informationstechnik) atd. Stejně jako v případě FIPS 140 provádí certifikaci akreditovaná laboratoř (znovu například Atsec).

COMMON CRITERIA

3.1 Proces

Před každou Common Criteria certifikací je potřeba si ujasnit si cíle a dle toho zvolit laboratoř a schéma. Ne každá laboratoř je akreditovaná ve všech schématech atd. Klíčové jsou požadavky zákazníků a zemí, ve kterých se produkt bude dodávat vládním agenturám. Ve Spojených státech tak primárním cílem bude evaluace vůči NIAPem schváleným protekčním profilům (Protection Profile, PP). Protekční profil specifikuje sadu Security Functional Requirements (SFRs), které produkt musí splnit z pohledu bezpečnostní funkcionality. Jen takový produkt je pak přidán na Product Compliant List (PCL). V jiných zemích může být naopak požadavek na EAL (Evaluation Assurance Level), kdy každá úroveň přidává množinu SARs (Security Assurance Requirements), tedy procesních požadavků jako jsou zdrojové kódy ve verzovacím systému, tzv. site visits, kdy se provádí audit fyzické bezpečnosti jako je přístup do budov, k serverům atd.

V případě, že neexistuje protekční profil pro danou technologii (a vytvoření a schválení nového PP je na roky), nebo produkt nesplňuje striktně PP, tak jedinou možností je EAL certifikace s vlastním Security Targetem.

Vzhledem k odlišnosti procesů se jako příklad zaměříme na proces NIAPu.

Prvním krokem je sepsání Security Targetu (ST) podle vybraného protekčního profilu (PP). Pro operační systém je to General Purpose Operating System Protection Profile (PP_OS), nyní ve verzi 4.2.1. Pro tzv. check-in, nebo-li spuštění oficiální procesu s NIAPem, je pak třeba ještě Entropy Assessment Report. Jelikož po oficiálním check-inu běží lhůta 180 dnů na dokončení certifikace (a tím je myšleno i vydání certifikátu!), je vhodné mít alespoň předběžně otestovaný produkt vůči požadavkům, nebo v ideálním případě mít testování kompletně hotové. Testování produktu provádí v případě PP laboratoř (naopak v EAL evaluacích musí i dodavatel předvést výsledky jeho testování jako součást evidence).. Takový produkt je pak publikován jako In Evaluation na webu NIAPu. Po dodání všech reportů a tzv. check-outu NIAP začne oficiální hodnocení a pokud je vše v pořádku, vydá certifkát. Z reportů stojí za povšimnutí AVA_VAN, tedy vulnerability assessment report – v produktu, který dostane Common Criteria nesmí být žádné známé CVE, které není opravené, vyargumentované, nebo nějakou formou mitigované.

3.2 Problémy specifické pro Common Criteria

Ačkoliv jsou Common Criteria mezinárodní standard, tak přístup jednotlivých signatářských zemí k tomuhle standardu je velmi odlišný. Severoamerické země preferují PP certifikace, evropské země pak EAL. Kvůli změně v Common Criteria Recognition Agreement (CCRA) se mezinárodně uznávají EAL certifikace jen do úrovně EAL 2.

Zajímavou výzvou jsou pak i CVE v již zmíněném AVA_VAN reportu. Jelikož tenhle report nesmí být starší třiceti dnů a to od doby vydání certifikátu, jedná se tak o skoro nekonečný boj s časem a novýma CVE. V praxi se nejlépe osvědčují pravidelné měsíční aktualizace AVA_VAN a tedy mít report vždy připraven.

4 Další certifikace

Zběžně zmíním i některé z dalších vládních certifikací, které se snažíme získat pro naše produkty.

USGv6 Test Program (U.S. Government IPv6 Test Program) jak již název napovídá, je program na testování správné implementace IPv6 protokolu – jak konformity s IPv6 standardem, tak interoperability. Jedná se o testování vůči sadě standardů NISP SP 500. Testování provádí University of New Hampshire Interoperability Lab.

SCAPVP (Security Content Automation Protocol Validation Program) je program (opět od NISTu – SP 800-126 rev 3) pro validaci SCAP nástrojů jako je například OpenSCAP. Poslední verze je SCAP 1.3.

FedRAMP (Federal Risk and Authorization Management Program) je program pro bezpečnostní hodnocení, autorizaci a monitoring cloudových produktů. Jedním z požadavků pro FedRAMP je FIPS 140.

Standardizace v oblasti kryptografie

Jan Dušátko

E-mail: https://cryptosession.cz

Abstrakt

O standardizaci v oblasti kryptografie patrně každý slyšel. Ale jaký účel plní? Jakou má historii, jak se vyvíjela a vyvíjí? Jaký vliv na ni mají zákony a normy? Jak se tyto standardy vyvíjely? Jaká je blízká budoucnost a co to znamená z pohledu IT? Uvedené otázky mohou poskytnout zajímavý pohled na současné snahy o zajištění bezpečnosti. Stejně tak je potřeba si uvědomit, k čemu kryptografie slouží a s jakými hrozbami se potkává.

1 Úvod

Kryptografie byla vždy nástrojem pro utajení obsahu komunikace před nepovolanými osobami. Kdy vznikla, je sporné, ale jisté je použití těchto technologií pro válečné účely. Tedy pro koordinaci aktivit a utajení těchto informací před protivníkem. Pro masivní rozšíření je ale nutné takový systém standardizovat a zjednodušit. A zároveň pochopit principy, které kryptografie vyžaduje, tedy jak s ní správně pracovat. Ty byly poprvé formulovány Auguste Kerckhoffem.

2 Využití kryptografie a omezující podmínky

Využití kryptografie je omezeno několika faktory. Patrně největší omezení je technologické, mimo něj se ale nesmí zapomínat na ekonomickou stránku věci, lokální zákonná omezení a případná organizační pravidla. Uvedené podmínky ale nejsou kompletní. Otázkou je i kdo kryptografii používá.

Zda státní organizace, soukromá organizace nebo soukromá osoba. V neposlední řadě tu je otázka, zda se jedná o vývoj, nákup, prodej nebo užití této technologie. Z praktického pohledu je ale situace výrazně jednodušší. Většina aplikací řeší technologická omezení, tj. co je možné nastavit a použít, ekonomická omezení a omezení daná kryptopolitikou, která je použita v rámci zákonných podmínek a naplňuje požadavky organizace.

Po ukončení II. světové války a v průběhu studené války začala být kryptografie používána jako zbraň. Čtení cizích dopisů sice není práce gentlemanů, ale schopnost vidět do hlavy protivníka, znát jeho tajemství a díky tomu získat včas jeho rozhodnutí (s dobrou analytikou je i předjímat) umožňuje přípravu a dovolí střet na místě, které si ten, kdo má tuto schopnost, může zvolit. Proto byla snaha omezit sdílení těchto strategických informací nejprve pomocí organizace COCOMM, následně pomocí mezinárodní smlouvy Wassenaar Arrangement. Veškeré restrikce ale budí i odpor. Jeho důsledkem byly soudní spory, které následně umožnily použití silné kryptografie i soukromým osobám. Zvlášť v současnosti se jedná o důležitou možnost, ať kvůli ochraně před represivními režimy, nebo ochraně před perzekucí kultury rušení, různých pozitivních diskriminací, nebo „jenom“ o ochranu svobody slova. V případě soukromých a státních organizací se pak jedná o ochranu strategického rozhodování a klasifikované komunikace. Chyby v ochraně mají závažné dopady, bez této ochrany soukromá firma ztrácí konkurenceschopnost a zaniká, státní pak ztrácí důvěru občanů. Vzájemná provázanost tak v dobrém i špatném vzájemné ohrožuje další navázané organizace.

3 Současný vývoj a standardizace

Po vzniku algoritmů DES v USA a obdobného standardu Magma v bývalém Sovětském Svazu v sedmdesátých letech minulého století došlo k postupnému rozvoji kryptografie. Hlavní zásluhu na tom měl jak rozvoj počítačů a potřeba soukromé komunikace po internetu, tak několik soudních sporů. V současnosti se tak zájem v oblasti kryptografie dělí do několika směrů. Symetrická kryptografie, asymetrická kryptografie, hash funkce, generátory náhodnosti a MCP (multiparty computation), která zasahuje např. do oblasti anonymizace, zero-knowledge protokolů, ale i do algoritmů pro balancování zátěže v rámci clusterů.

Díky prvotnímu vývoji bylo možné v polovině devadesátých let začít používat alespoň nějakou metodu ochrany transportního kanálu. Během několika let se ukázalo, že je nedostatečná, a tak se na základě zkušeností začaly používat dočasné klíče jako forma Perfect Forward Secrecy a následovalo i šifrování na aplikační vrstvě (End-to-End Encryption). Jejich použití ale potřebuje společnou technologickou základnu, kterou zajišťuje právě standardizace v posledních letech. Proto byly vytvořeny veřejné soutěže, které měly za cíl spojit schopnosti a znalosti co největšího množství profesionálních i amatérských kryptologů. Cílem je zajistit tvorbu prověřených standardů, které odolají útokům.

První vlaštovkou byla soutěž o nový kryptografický standard AES. Byl určen jako náhrada již zastaralého algoritmu DES. Tuto soutěž vyhrál algoritmus Rijndael. Postup veřejné soutěže byl inspirací i pro klání Nessie a CryptRec. Zatímco CryptRec byl v Japonsku prohlášen za úspěšný, Nessie v EU prokázala slabiny dostupných proudových šifer a ve finále vedla k soutěži eStream. Další ze soutěží byl výběr nového hashovacího standardu SHA3 (NIST). Mimo oficiálních aktivit byly i podobné aktivity v rámci kryptologické komunity, kde se jednalo o soutěže CAESAR a PHC. První zmiňovaná měla za účel najít nové metody, zajišťující autentizované šifrování, druhá speciální třídu pomalých hash funkcí, použitelnou pro ukládání hesel. V nedávné době se pak jednalo o další oficiální klání, to konkrétně lehká kryptografie (NIST LWC, tedy LightWeigh Cryptography), která by měla zabezpečit komunikaci v případě IoT, RFC či SmartCard. V současné době je ještě stále aktivní soutěž NIST PQC (Post Quantum Cryptography). Tato soutěž je velice důležitá z důvodu současného rozvoje kvantových počítačů, které se mohou v horizontu let stát hrozbou pro klasickou asymetrickou kryptografii.

Země, jako je Rusko nebo Čína jdou vlastním směrem. Rusko používá národní standardy GOST (государственный стандарт), které jsou vyvíjeny na základě specifikací. Obdobně je na tom Čína, která používá rodinu SM (Shang-Mi) algoritmů, ani zde se nejedná o veřejné soutěže.

Díky těmto krokům máme k dispozici algoritmy, na kterých se mohou obě strany snadno domluvit. Asymetrické algoritmy zajišťující digitální podpis nebo domluvu na sdíleném tajemství. Máme symetrické algoritmy pro šifrování dat, jak proudové, tak i blokové, dále módy operací blokových algoritmů. Máme i standardizované hash funkce. To je základ, ale mimo uvedené máme možnost použít stovky funkcí, které standardizované nejsou.

Ale mimo šifrovacích algoritmů je nutné standardizovat ještě jednu důležitou komponentu, generátory náhodnosti. V současnosti dochází v této oblasti k masivní obměně, kdy jsou staré mechanismy nahrazovány novými, tentokrát už standardizovanými. Slouží pro generování inicializačních vektorů, solí, nonce a dalších nepředvídatelných bloků dat. Přesto stále některé operační systémy obsahují zastaralé nebo i chybné generátory, které je možné se „správnou“ konfigurací zneužít.

Z praktického hlediska se nejčastěji používají výše zmíněné algoritmy při zajištění SSL/TLS/DTLS přenosů obsahu webu nebo pro zajištění přenosu e-mailů. Nejedná se o jedinou vrstvu, kryptografie se používá i jinde (IPSec, nativní součást NTP, SNMP, SSH a další), ale pro účely uživatelů se jedná o vrstvu nejzásadnější. Současné implementace tak postupně sledují trend přechodu k bezpečnějším, široce rozšířeným mechanismům. Standardizace je důležitou součástí, zajišťující technologickou spolupráci mezi jednotlivými datovými rozhraními. Postupná obměna za standardy, které mají požadované vlastnosti, tak zajišťuje vyšší úroveň bezpečnosti, než byla u většiny těch předchozích.

4 Současná a budoucí rizika

Kryptografie slouží k zajištění ochrany důvěrnosti a integrity dat, nezajišťuje jejich dostupnost (přesněji zajišťuje jejich nedostupnost). Je oslabována neznalostí uživatelů, ale i chybami implementace (rozdíl mezi kryptografickým protokolem a sadou logických podmínek). To, co je ale největší problém, je ignorování požadavků, kladených matematickou konstrukcí. Díky tomu ještě po 40 letech nacházíme zásadní problémy v implementaci relativně jednoduchých mechanismů. Většina současné asymetrické kryptografie je matematika téměř na úrovni střední školy.

Zde je i největší problém pro blízkou budoucnost. Post-kvantová kryptografie používá vyšší matematiku, která je výrazně složitější a výrazně náročnější na znalosti. Jak dlouho bude trvat odstranění implementačních problémů u těchto algoritmů? Standard v této oblasti ještě nějakou dobu mít nebudeme, a i kdybychom ho měli, dalších 40 let si nemůžeme dovolit. V současnosti ještě stále probíhá výběr nadějných algoritmů. Potřebujeme čas na jejich implementaci a hledání chyb. Nutně musíme mít i ochrannou lhůtu pro zajištění důvěrnosti dat. Závod mezi kvantovými počítači a asymetrickými algoritmy proti těmto počítačům odolnými začal. A vítěz je stále nejistý.

Lesk a bída šifrování disků

Milan Brož

E-mail: xbroz@fi.muni.cz

Abstrakt

Šifrování dat na úrovní disku je jedna z nejstarších, a zdánlivé triviálních, cest k dosažení důvěrnosti dat (nebo alespoň cesty pro splnění položky osobní data jsou šifrována *v IT auditní zprávě).*

Zkusme si projít historii různých implementací, včetně dobrých nápadů, ale i omylů, počínaje TrueCryptem (respektive VeraCryptem), BitLockerem, LUKS a FileVault a jejich přístupem.

1 Úvod

Šifrování dat na úložných zařízeních (*data-at-rest storage encryption*) se stalo již běžným opatřením chránící data v případě ztráty či odcizení zařízení. Přestože má softwarové šifrování disku některé nevýhody vůči šifrování přímo v souborovém systému (či aplikaci), je stále masivně používáno prakticky na všech platformách včetně mobilních (jako jsou chytré telefony).

Pokud však potřebujeme přenášet data mezi různými operačními systémy (jako je Windows, Linux či macOS) a nechceme-li použít cloud služeb, setkáme se s problémem, jaké šifrování použít, aby se zařízení dalo bez problémů odemknout na všech těchto systémech. Tento problém byl jednou z motivací, proč implementovat v Linuxu přístup k jiným formátům šifrovaných disků než je nativní LUKS.

Implementace open-source podpory vyžaduje dvě základní věci – dostatečnou znalost formátu (bez nutnosti používat proprietární dokumentaci) a dostatečně flexibilní architekturu pro konfiguraci. Svobodná dokumentace proprietárních formátů (jako je *BitLocker* či *FileVault*) není sice dokonalá, ale je z velké části dostupná díky snaze o forenzní analýzu těchto formátů v open-source prostředí [1][2]. Zároveň integrace do základních systémových utilit a Linuxového jádra přináší tuto funkcionalitu uživatelům prakticky všech distribucí bez nutnosti cokoliv dalšího instalovat.

Konfigurace šifrovaných disků v Linuxu je velmi flexibilní díky dvěma základním stavebním blokům, které si stručně popíšeme.

1.1 Linuxové jádro a dm-crypt

Prvním blokem je jaderný ovladač *dm-crypt*, který umožňuje transparentně šifrovat blokové zařízení on-line, podle dostupných parametrů. *Dm-crypt* neřeší žádné uložení klíčů ani jejich správu, je tedy jen výkonným ovladačem, který se stará o šifrování sektorů. Na jeho konfiguraci je třeba znát konkrétní parametry šifrování (blokovou šifru, šifrovací mód, použitý inicializační vektor a pochopitelně klíč) a parametry jako je offset na blokovém zařízení nebo velikost sektoru. Zároveň *dm-crypt* interně neimplementuje šifrovací algoritmy (s výjimkou speciálních inicializačních vektorů), ale jen používá rozhraní k již implementovaným šifrovacím algoritmům v jádře. To automaticky umožňuje použít hardwarovou akceleraci, pokud ji daný ovladač poskytuje (například instrukce AES-NI). Na konfiguraci *dm-cryptu* je tedy potřeba znát šifrovací klíč, kterým se přímo šifrují sektory na disku, obvykle označovaný jako *Media Encryption Key* (MEK) nebo *Volume Key.*

1.2 Knihovna libcryptsetup

Druhým stavebním blokem je knihovna *libcryptsetup* (kterou využívá nástroj cryptsetup, se kterým budeme v příkladech pracovat). Tato knihovna zpracovává metadata jednotlivých formátů uložených na disku, implementuje tedy správu klíčů. Základní vlastností je, že veškeré zpracování metadat probíhá jako dočasný uživatelský proces, jádro systému pak jen obdrží finální konfiguraci. To umožňuje efektivně oddělit vlastní zpracování metadat od aktivního procesu šifrování sektorů.

Nad aktivním *dm-crypt* zařízením je pak možné použít libovolný souborový systém, který má ovladače v jádře. Tento ovladač (či aplikace přistupující přímo k blokovému zařízení) pak o vlastním šifrování disku prakticky neví.

Data přístupná přes *dm-crypt* je možné jak číst, tak zapisovat (pokud to ovladač souborového systému podporuje). Metadata proprietárních formátů disků jsou však vždy používána jen pro čtení – pomocí *libcryptsetup* není možné je vytvořit či modifikovat (například změnit heslo); pochopitelně s výjimkou nativního *LUKS* formátu. Toto je záměrné omezení, neboť s omezenou znalostí metadat není možné garantovat plnou funkčnost v proprietárním operačním systému. Toto řešení tedy umožňuje používat existující zařízení, ale naformátované musí být nativními nástroji daného operačního systému.

2 TrueCrypt a VeraCrypt

TrueCrypt byl multiplatformním nástrojem pro šifrování disků, dostupným pro Windows, Linux a macOS. Přestože je to nástroj s dostupným zdrojovým kódem, původní licence nebyla kompatibilní se svobodnými licencemi. Po ukončení vývoje se vytvořilo několik následovníků, ale v současné době je jediným rozumně udržovaným *VeraCrypt*. *VeraCrypt* se snaží řešit problémy (včetně podpory nových operačních systémů, EFI bootu apod), ale stále obsahuje kód, který je licencovaný původními autory.

Původní implementace formátu pro *libcrypsetup* byla částečně studií kryptografických algoritmů reálně použitých *TrueCryptem* [3], ale ukázala se natolik použitelnou, že je nyní součástí knihovny *libcryptsetup*. Na podporu *VeraCrypt* zařízení tedy není nutné nic navíc instalovat.

2.1 Metadata

Metadata formátu *TrueCrypt* jsou velmi jednoduchá a jako jediný z popsaných formátů celá metadata šifruje (bez znalosti hesla celý disk vypadá jako náhodná data – přítomnost šifrování by nemělo jít bez hesla prokázat).

Metadata (parametry a klíče) jsou uloženy v jednom šifrovaném sektoru (se sekundárním záložním sektorem). Při zadání hesla se pak zkouší všechny varianty algoritmů, dokud v dešifrovaném metadata sektoru není detekován magický řetězec (*TRUE* pro *TrueCrypt* a *VERA* pro *VeraCrypt*). Hlavička je chráněna CRC32 algoritmem proti náhodnému poškození. *VeraCrypt* používá identickou strukturu metadat, jediným rozdílem je jiný magický řetězec a implementace nových variant algoritmů.

Aplikace umožňuje použití skrytého disku (data jsou uložená v nepoužívaném prostoru primárního disku). Z pohledu metadat je skrytý disk používán stejně jako primární, jen je metadata sektor uložen na jiném offsetu.

Autoři poměrně striktně odmítají používání *Trusted Platform Module* (TPM), lze však použít SmartCard. Z pohledu metadat je to však jen externí úložiště pro heslo. Aplikace také umožňuje použít svázání se specifickými soubory, kde obsah části těchto souborů (keyfiles) je přimíchán do hesla.

Šifrovací algoritmy pro data prošly dlouhým vývojem, mnoho algoritmů se přidávalo a odebíralo dle objevených zranitelností. Lze také použít zřetězené šifrování (několik různých algoritmů aplikovaných za sebou).

K podporovaným symetrickým algoritmům patří AES, Serpent, Twofish, Cammelia a trochu překvapivě i ruský Kuzniechik. Historicky byly podporovány algoritmy Blowfish, CAST5, 3DES a velmi krátce IDEA, ale jejich podpora byla z aplikace odstraněna. Šifrovacím módem byl z počátku CBC (s velmi zvláštní modifikací pro zřetězené šifry a s použitím triviálního whiteningu). CBC mód byl po problémech se špatně implementovaným inicializačním vektorem odstraněn (existuje útok, který umožňoval ve specifických případech odhalit skrytý disk). Krátce byl nahrazen šifrovacím módem LRW, aby se následně přešlo na mód XTS, který se používá dodnes.

Zajímavým problémem je nutnost přeformátovat celé zařízení (*To avoid hinting whether your volumes contain a hidden volume or not*), pokud používá skrytý disk a pokud bylo vytvořeno ve verzi starší než *VeraCrypt 1.18*. Není jasné, na jakém principu hinting funguje, ale zřejmě je možné rozpoznat výplňová data (pokud skrytý disk není použit) od reálně uloženého šifrovaného metadata sektoru skrytého disku.

Pro odvození klíče z hesla (kterým se pak dešifruje metadata sektor) je vždy použit algoritmus PBKDF2, liší se však v parametrech. Podporované hashovací algoritmy byly RIPEMD160, Whirlpool, SHA1, SHA256, SHA512 a ruský Streebog512. *VeraCrypt* odstranil některé algoritmy (SHA1, Whirlpool) a některé varianty (pro bootovací disk se používal menší počet iterací). *VeraCrypt* se snaží řešit problém odolnosti algoritmu PBKDF2 vůči slovníkovým útokům zvýšením počtu iterací (PBKDF2 je velmi efektivně paralelizovatelný). Nad rámec řádového zvýšení počtu defaultních iterací (například pro SHA512 *TrueCrypt* používal 1000, zatímco *VeraCrypt* 500000) a navíc zavádí *Personal Iterations Multiplier* (PIM), což je číslo, které řádově zvýší počet iterací (a vyhne se defaultnímu počtu).

Zvýšení počtu iterací je však pouze částečným řešením, které navíc přenáší část problémů na uživatele – nejen, že odemčení disku nyní může trvat velmi dlouho (pokud je použitý algoritmus až na konci seznamu, je nutné zkusit všechny přetím), ale v případě použití PIM si uživatel musí zapamatovat nejen heslo, ale i PIM (který nejde odhadnout).

2.2 Použití s pomocí cryptsetup

Implementace v *libcryptsetupu* umožňuje namapovat jakékoliv (i historické, z původní aplikace již odstraněné) kombinace algoritmů. Výjimkou jsou specifické nestandardní šifry, které jádro neimplementuje (Blowfish v little-endian variantě a zřetězené CBC módy, neboť modifikují standardní CBC). Lze však zobrazit metadata a ověřit, že heslo je platné (*libcryptsetup* umí tyto nestandardní módy pro metadata sektor naemulovat, lze jej tak použít pro kontrolu, že nekompatibilita je na straně VeraCryptu, který daný mód již odstranil). Pro použití ruských algoritmů je nutné doinstalovat jaderné ovladače (v Ubuntu je to balík *gost-crypto-dkms*).

Vzhledem k tomu, že i *VeraCrypt* aplikace na Linuxu používá *dm-crypt* (a ne interní implementaci algoritmů jako na Windows), výkonnostní srovnání různých Linuxových magazínů mezi LUKS a *VeraCrypt* postrádají smysl – šifrovací ovladač je identický. *VeraCrypt* také implementuje ve Windows šifrování klíče v paměti (pro zabránění jednoduchého extrahovaní klíče ze snímku RAM), ale toto řešení je poměrně často zpochybňováno jako neefektivní [4] a navíc na Linuxu není nikdy použito.

Příklad použití s nástrojem cryptsetup je na obrázcích 1 a 2.

3 BitLocker

BitLocker je nativním řešením šifrování disků v operačním systému Windows a umožňuje poměrně komplexní nastavení. Zde se zaměříme jen na část, která je dostupná i přes implementaci v *libcryptsetup*, zejména podporu přenositelných disků (označovaných jako *BitLocker to Go*).

Zajímavou epizodou v historii *BitLockeru* byla přímá podpora self-encrypted disků (SED; *BitLocker* pak pouze uložil metadata o disku, neprováděl žádné šifrování), ale tato podpora byla po několika objevených zranitelnostech víceméně odstraněna [5] (resp. lze ji zapnout pouze zásahem administrátora).

3.1 Metadata

Metadata *BitLockeru* jsou inspirována metadaty souborového systému NTFS; zjednodušeně lze říci, že jsou navržena tak, aby šla implantovat do existujícího NTFS souborového systému na místa, která NTFS nepoužívá pro vlastní metadata (toto platí i když BitLocker podporuje formáty FAT a exFAT).

```
# cryptsetup open --type tcrypt --veracrypt test.img test
Enter passphrase for test.img:
# mount /dev/mapper/test /mnt/tst
...

# cryptsetup status test
/dev/mapper/test is active.
  type:    TCRYPT
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: dm-crypt
  device:  /dev/loop12
  loop:    test.img
  sector size:  512
  offset:  256 sectors
  size:    130560 sectors
  skipped: 256 sectors
  mode:    read/write
```

Obrázek 1: Příklad otevření VeraCrypt kontejneru

```
# cryptsetup tcryptDump test.img
Enter passphrase for test.img:
VERACRYPT header information for test.img
Version:        5
Driver req.:    1.b
Sector size:    512
MK offset:      131072
PBKDF2 hash:    sha512
Cipher chain:   aes
Cipher mode:    xts-plain64
MK bits:        512
```

Obrázek 2: Příklad zobrazení metadat VeraCrypt kontejneru

Metadata obsahují základní hlavičku s viditelným řetězcem (signaturou) a třemi bloky (kopiemi) metadat (FVE záznamy), které obsahují variabilní počet key-value záznamů. Tyto bloky se volně prolínají s šifrovanými daty (obdobně jako kopie NTFS metadat). *BitLocker* tedy neobsahuje jednu spojitou oblast pro data, ale je nutné tyto metadata bloky při odemčení maskovat. Ne všechny typy key-value záznamů jsou volně zdokumentované, ale pro základní práci se zařízením jsou důležité záznamy, které obsahují klíče.

BitLocker používá terminologii *Full Volume Encryption Key* (FVEK, obdoba MEK), a *Volume Master Key* (VMK, obdoba KEK), kterých může být více a umožní tak odemčení zařízení různými způsoby (heslem, kombinací s TPM, recovery heslem apod.) Další typy záznamů jsou například identifikátory disku nebo záznamy o datu a jménu stroje, kde byl disk zformátován.

BitLocker vždy podporoval jen algoritmus AES, ale použité šifrovací módy prošly vývojem. Původně byl použit mód CBC, volitelně s přidaným difuzér algoritmem *Elephant*. Tento algoritmus částečně řešil problémy CBC módu, (šlo v zásadě o mixování dat před šifrováním s pomocí rotace a sekundárního klíče). Tato operace byla za cenu použití nestandardní modifikace CBC (a drobného snížení výkonu). V nových verzích Windows byla tato volba kompletně opuštěna a šifrovací mód byl nahrazen módem XTS.

Pro odvození klíče *BitLocker* používá vlastní algoritmus založený na iteraci SHA256 hashe. Šifrovaný metadata záznam s klíči je pak chráněn algoritmem AES-CCM (je tedy chráněna jeho integrita).

3.2 Použití s pomocí cryptsetup

Implementace s pomocí *libcryptsetup* umožňuje otevřít všechny typy *BitLocker* zařízení [6] (jak systémový disk, tak varianta to Go) včetně CBC módů s difuzérem (zde bylo potřeba implementace tohoto difuzéru v *dm-cryptu*). Pro odemčení jsou podporovány pouze hesla a recovery heslo (TPM a ostatní typy nejsou v *libcryptsetup* podporovány). Mapování aktivního zařízení využívá flexibility device-mapper jaderného ovladače, který umožňuje vymaskovat bloky s metadaty *BitLockeru* (souborovému systému se jeví jako prázdné sektory; nemá tedy přístup k *BitLocker* metadatům).

Příklad použití s nástrojem cryptsetup je na obrázcích 3 a 4.

```
# cryptsetup open --type bitlk /dev/sdb test
Enter passphrase for /dev/sdb:
# mount /dev/mapper/test /mnt/tst
...

# cryptsetup status test
/dev/mapper/test is active and is in use.
  type:    BITLK
  cipher:  aes-xts-plain64
  keysize: 256 bits
  key location: dm-crypt
  device:  /dev/sdb
  sector size:  512
  offset:  16 sectors
  size:    15974400 sectors
  skipped: 16 sectors
  mode:    read/write
```

Obrázek 3: Příklad otevření BitLocker to Go flashdisku

4 FileVault2

FileVault2 je nativním řešením šifrování disků pro operační systémy macOS. Původní FileVault (podle aplikace pro analýzu formátu pejorativně přejmenovaný na VileFault) byl nahrazen verzí *FileVault2*, které se zde budeme věnovat.

Nové verze macOS přechází na šifrování na úrovni souborového systému s pomocí *Apple File System* (APFS), ale zařízení (zejména ta přenosná) lze stále formátovat jako *FileVault2*. Toto je důležité zejména proto, že zatím neexistuje dostupná stabilní implementace APFS pro Linux, zatímco souborové systémy používané nad *FileVault2* (HFS či FAT) jsou v jádře Linuxu podporovány.

4.1 Metadata

Formátování disku v macOS s sebou nese poměrně komplikovanou strukturu particií a dalších metadat; pro přístup k šifrované části nás však

```
# cryptsetup bitlkDump /dev/sdb
Info for BITLK device /dev/sdb.
Version:        2
GUID:           fccc81d1-91b3-4947-85c2-0cb98cd17950
Sector size:    512 [bytes]
Created:        Sat Apr 23 16:24:22 2022
Description:    DESKTOP-71L8959 TEST 23.04.2022
Cipher name:    aes
Cipher mode:    xts-plain64
Cipher key:     256 bits

Keyslots:
 0: VMK
        GUID:           a78c6964-0619-4ac7-a63d-b2a56f5c9e49
        Protection:     VMK protected with passphrase
        Salt:           1cdd33ae1dabe5ccfda16e6c0bad4403
        Key data size: 44 [bytes]
 1: VMK
        GUID:           23a8c064-5de6-46f1-98be-91ebb5c09fd4
        Protection:     VMK protected with recovery passphrase
        Salt:           95d386c7016f62884a13d211321eea7e
        Key data size: 44 [bytes]
 2: FVEK
        Key data size: 44 [bytes]

Metadata segments:
 0: FVE metadata area
        Offset:         34603008 [bytes]
        Size:           65536 [bytes]
 1: FVE metadata area
        Offset:         1108344832 [bytes]
        Size:           65536 [bytes]
 2: FVE metadata area
        Offset:         2182086656 [bytes]
        Size:           65536 [bytes]
 3: Volume header
        Offset:         34668544 [bytes]
        Size:           8192 [bytes]
        Cipher:         aes-xts-plain64
```

Obrázek 4: Příklad zobrazení metadat BitLocker to Go flashdisku zajímá pouze datová particie, na které jsou uloženy jak metadata, tak i šifrovaná data.

Metadata formátu *FileVault2* jsou uložena na počátku disku, následovaná oblastí s šifrovanými daty. Metadata jsou poměrně komplexní (v terminologii Apple jde o Core Storage), ale pro funkční mapování stačí pouze omezená znalost metadat, již dokumentovaná v otevřené formě [2].

FileVault2 obsahuje počáteční sektor s viditelnou hlavičkou (lze tedy detekovat existenci formátu bez znalosti hesla). Dále obsahuje blok s metadaty a další blok, který je uložen v šifrované formě (klíč je však uložen v prvním metadata bloku). Tato šifrovaná oblast pak obsahuje variabilní počet metadata záznamů uložených ve formátu XML, binární data jsou zde uložena v Base64 kódování. Pro zpřístupnění mapování však potřebujeme znát pouze záznamy, které obsahují klíče a základní informace o datové oblasti, jako je offset a délka. Integrita dat ve všech metadata oblastech je chráněna pomocí CRC32.

FileVault2 používá blokovou šifru AES a mód XTS, jak pro uživatelská data, tak pro šifrování metadata (XML) oblasti.

Klíče jsou uložený zapouzdřeny (key wrap). Pro dešifrování se používá klíč odvozený z hesla pomocí algoritmu PBKDF2 (s hash algoritmem SHA256 a počtem iterací uloženým v metadatech). Zajímavostí je, že druhý klíč pro XTS mód není v metadatech uložen jako nezávislý, ale odvozuje se s pomocí hash algoritmu SHA256 a *Family UUID* z metadat.

4.2 Použití s pomocí cryptsetup

Implementace formátu *FileVault2* v *libcryptsetup* je zatím v experimentální fázi, následující popis vychází z experimentů z dosud nezačleněného rozšíření popsaného v rámci diplomové práce [7] a volně dostupného jako merge request v projektu cryptsetup [8].

Příklad použití s nástrojem cryptsetup je na obrázcích 5 a 6.

5 Závěr a srovnání s LUKS

Pro Linux je nativním formátem šifrování disků *Linux Unified Key Setup* (LUKS), a to jak ve verzi LUKS1 (který používá binární formát metadat), tak v nové verzi LUKS2, který řeší některé problémy a umožňuje ukládat variabilní metadata ve formátu JSON [9].

Ve srovnání s ostatními systémy (nejen s těmi popsanými výše) je zřejmé, že stejné problémy lze řešit různě. Použití formátů jako XML či

```
# cryptsetup open --type fvault2 /dev/sdb2 test
Enter passphrase for /dev/sdb2:
# mount /dev/mapper/test /mnt/tst
...

# cryptsetup status test
/dev/mapper/test is active and is in use.
  type:    FVAULT2
  cipher:  aes-xts-plain64
  keysize: 256 bits
  key location: dm-crypt
  device:  /dev/sdb2
  sector size:  512
  offset:  131072 sectors
  size:    6586368 sectors
  mode:    read/write
```

Obrázek 5: Příklad otevření FileVault2 flashdisku

```
# cryptsetup fvault2Dump /dev/sdb2
Info for FVAULT2 device /dev/sdb2.
Physical volume UUID    047a90a3d7b94b62974917d0fde502d7
Logical volume offset:  67108864
Logical volume size:    3372220416
Cipher:                 aes
Cipher mode:            xts-plain64
PBKDF2 iterations:      160780
PBKDF2 salt:            50ab6b6aeb18661f7b660ed762fa5fb0
Family UUID:            ca2f817c39ca467f86095ccd5cb377c9
```

Obrázek 6: Příklad zobrazení metadat FileVault2 flashdisku

JSON se může zdát problematické, ale ve srovnání s nutnosti zachovat kompatibilitu při zavádění nových vlastností jde o velmi flexibilní nástroj.

Pro šifrovací mód se většina v současnosti používaných formátů shodne na AES-XTS, alternativa je snad jen speciální konstrukce Adiantum [10] pro Google systémy (LUKS jej umí použít také).

Různý přístup je při nastavení několika způsobů odemčení zařízení (ať již pro záložní metody odemčení s recovery heslem, nebo pro víceuživatelský přístup v případě keyslotů v LUKS).

Co je ale podstatné z pohledu uživatelské přívětivosti je integrace do Linuxových distribucí. Pro všechny výše uvedené formáty existují alternativní nástroje, které si můžete instalovat a používat, ale jen plná integrace do systému umožní obyčejnému uživateli jednoduše sdílet data mezi operačními systémy bez nutnosti další instalace a nastavování.

Nejde jen o *libcryptsetup* (případně o nástroje nad systemd pro aktivaci disků, které jsou nad *libcryptsetup* postaveny), ale i podpora v *blkid* na automatickou detekci metadat po vložení zařízení (například připojení USB flashdisku) a zejména podpora v desktop systémech s pomocí *udisk* a podobných nadstaveb.

Pokud máte moderní Linuxový desktop, zkuste si vložit *BitLocker to Go* flashdisk naformátovaný ve Windows – většina distribucí jej dnes bez problému automaticky rozpozná a po zadání hesla zpřístupní uživateli. A to je cílem integrace popsané v tomto článku.

Odkazy

[1] J. Metz. *Library and tools to access the BitLocker Drive Encryption (BDE) encrypted volumes.* URL: `https://github.com/libyal/libfvde/`.

[2] J. Metz. *Library and tools to access FileVault Drive Encryption (FVDE) encrypted volumes.* URL: `https://github.com/libyal/libbde/`.

[3] Milan Brož a Vashek Matyáš. „The TrueCrypt On-Disk Format–An Independent View“. In: *IEEE Security Privacy* 12.3 (2014), s. 74–77. DOI: `10.1109/MSP.2014.60`.

[4] *Security Evaluation of VeraCrypt.* URL: `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Veracrypt/Veracrypt.pdf`.

[5] Microsoft. *Guidance for configuring BitLocker to enforce software encryption*. URL: https://msrc.microsoft.com/update-guide/en-us/vulnerability/ADV180028.

[6] V. Trefný. *Bitlocker šifrování disku v Linuxovém prostředí*. URL: https://digilib.k.utb.cz/bitstream/handle/10563/44511/trefn%5C%C3%5C%BD_2019_dp.pdf.

[7] P. Tobiáš. *FileVault Disk Encryption in Linux Environment.*

[8] Cryptsetup project. *FileVault2 support*. URL: https://gitlab.com/cryptsetup/cryptsetup/-/merge_requests/298.

[9] M. Brož. *LUKS2 format documentation*. URL: https://gitlab.com/cryptsetup/LUKS2-docs.

[10] Paul Crowley a Eric Biggers. „Adiantum: length-preserving encryption for entry-level processors“. In: *IACR Transactions on Symmetric Cryptology* 2018.4 (pros. 2018), s. 39–61. DOI: 10.13154/tosc.v2018.i4.39-61. URL: https://tosc.iacr.org/index.php/ToSC/article/view/7360.

Certificate Validation

Tomas Weinfurt

Digital certificated become part of our daily lives – often without us knowing. When we go to browse Internet, many sites now transparently redirect to "secure" instances. Software packages are digitally signed and that is crucial part of update process. And probably most visibly, we can send and receive digitally signed email. In most cases standard software takes care of all of it for us. Dealing with certificate and validation can be more complicated for application developers as they may need to perform some validation in their code. There are also aspects of certificate itself that can impact validation and possibly security.

To start the discussion, let's look at a simple X509 Certificate.

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 14652970177198592840 (0xcb59cebdcc509348)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=localhost
        Validity
            Not Before: Jan 28 22:57:27 2019 GMT
            Not After : Jun 11 22:57:27 2020 GMT
        Subject: localhost
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:91:b6:b6:ef:5b:82:ab:43:b0:c3:d3:37:94:47:
                    ...
                    78:33
                Exponent: 65537 (0x10001)
```

Figure 1: Simple certificate

Before validating cryptographic operation, the certificate is used for (TLS handshake, software update, etc) we need to also verify the certificate.

1 Cryptographic verification

The cryptographic part is straight forward. Signature is essentially hash computed over certificate metadata and the public key signed with Issuer's private key. Assuming the receiver has access to the Issuer's public key, they can compute the hash themselves, decrypt the presented signature with the public key and compare both. If someone tampered with the certificate hash computed locally and the hash encrypted with the private key would differ. If that happens, nothing in the certificate can be trusted. This part is most expensive from a CPU utilization prospective. It is heavily impacted by choice of algorithms and key sizes and unsurprisingly there is tradeoff between performance and security. In many cases performance of the handshake does not matter as many modern protocols (like Http 1.1 persistent connection or HTTP/2 multiplexing) reuse single TLS for multiple transactions. But it can be significant if a large number of transactions is needed, and high CPU utilization can make the systems more vulnerable to DenialOfService attacks.

Following table shows impact of choices for TLS handshake.[1]

```
|                         Method | protocol |        Mean |      Error |     StdDev |
|------------------------------- |--------- |------------:|-----------:|-----------:|
|      HandshakeECDSA256CertAsync |    Tls12 | 1,543.22 us |  25.475 us |  22.583 us |
|      HandshakeECDSA512CertAsync |    Tls12 | 3,257.17 us | 139.793 us | 160.985 us |
|       HandshakeRSA1024CertAsync |    Tls12 | 1,482.57 us |  53.269 us |  61.344 us |
|       HandshakeRSA2048CertAsync |    Tls12 | 1,989.16 us |  76.070 us |  87.603 us |
|       HandshakeRSA4096CertAsync |    Tls12 | 5,060.03 us |  93.271 us |  87.246 us |
|      HandshakeECDSA256CertAsync |    Tls13 | 1,689.33 us |  66.739 us |  76.857 us |
|      HandshakeECDSA512CertAsync |    Tls13 | 3,159.07 us |  39.501 us |  32.985 us |
|       HandshakeRSA2048CertAsync |    Tls13 | 2,024.73 us |  72.737 us |  83.763 us |
|       HandshakeRSA4096CertAsync |    Tls13 | 5,019.56 us | 128.195 us | 147.629 us |
```

While it may be tempting, weak algorithms should not be used. RSA 1024 and Sha1 are not safe and unfortunately the Internet is still full of old examples using them. Probably best choice is use of Elliptic Curves with 256 key size and Sha2 family algorithm for hashing. It typically offers better performance and smaller data size compared to RSA. RSA may be good choice if EC is concern for compatibly. Not every client may support the ciphers and RSA is probably best choice for interoperability. While 2048 RSA or 256 EC are considered secure for now, certificates with validity beyond the year 2030 should use RSA 4096 or 512 EC. This is because an attacker has a longer time for brute force attacks and there

[1] `https://github.com/dotnet/performance/tree/main/src/benchmarks/micro/libraries/System.Net.Security`

being a long history of algorithms that are broken by increasing CPU power.

2 Certificate metadata

When the signature on a certificate is verified, we can continue looking into additional aspects. This is where we can see more variations and where software often pushes final decisions to the end users. That may not be viable in certain scenarios as well as it may be very difficult for the end user to make a informed decision.

2.1 Time check

valid time. Each certificate has assigned time range of validity and it will be deemed invalid outside of that range. This is one of the rare cases when end user can make reasonable decision. Let say you go to your favorite web site, and it shows the certificate expired yesterday. Is that sign of malicious activity or did the privacy get compromised? Probably not, but it is not trivial to do this automatically. If the expiration is less the year 2050, UTCTime is used. Software components should not forget to convert local time to UTC before verification. For certificates with longer expiration, Generalized Time should be used as it allows communication with the time zone. Adjustments for time are generally not recommended unless it is clear that the local device may not have a valid time. On rare occasions, the certificate may be presented before it's validity period starts. This can happen for example when certificates are created by some automation but there is some drift across networking devices. When this happen, I would recommended to adjust the certificate validly to "now – safeOffset" instead of fiddling with validation logic.

2.2 Certificate names

X509 certificates were designed as part of X.5xx standards. That makes them somewhat difficult to use on Internet where completely different naming conventions are used. Both Issuer and Subject are lists of name and value pairs. For example, Figure 2 shows "C=US, ST=California, L=San Francisco, O=GitHub, Inc., CN=github.com" as Subject name. Most of that text is irrelevant to validation. It is the CommonName (CN) part that maters and it typically contains FQDN name or mail

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            05:18:9a:54:eb:e8:c7:e9:03:e0:ab:0d:92:55:45:de
    Signature Algorithm: ecdsa-with-SHA384
        Issuer: C=US, O=DigiCert Inc, CN=DigiCert TLS Hybrid ECC SHA384 2020 CA1
        Validity
            Not Before: Mar 15 00:00:00 2022 GMT
            Not After : Mar 15 23:59:59 2023 GMT
        Subject: C=US, ST=California, L=San Francisco, O=GitHub, Inc., CN=github.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:4a:b0:93:71:85:21:ec:62:3f:cb:74:c0:46:c8:
                    e7:00:dc:27:4a:32:4b:8a:d5:51:83:08:11:23:52:
                    65:c5:9d:64:75:94:10:9f:99:6d:3f:7b:fb:29:3b:
                    58:b8:37:54:78:4b:b7:3d:1c:77:7e:90:dd:bb:67:
                    23:32:5c:80:d1
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:0A:BC:08:29:17:8C:A5:39:6D:7A:0E:CE:33:C7:2E:B3:ED:FB:C3:7A
            X509v3 Subject Key Identifier:
                78:AA:72:C6:71:69:68:14:B5:59:B1:9E:8B:6E:2B:40:87:42:3B:1E
            X509v3 Subject Alternative Name:
                DNS:github.com, DNS:www.github.com
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl3.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crl
                Full Name:
                  URI:http://crl4.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crl
            X509v3 Certificate Policies:
                Policy: 2.23.140.1.2.2
                  CPS: http://www.digicert.com/CPS
            Authority Information Access:
                OCSP - URI:http://ocsp.digicert.com
                CA Issuers - URI:http://cacerts.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crt
            X509v3 Basic Constraints:
                CA:FALSE
            1.3.6.1.4.1.11129.2.4.2: ......
    Signature Algorithm: ecdsa-with-SHA384
```

Figure 2: Certificate with extended attributes

address. When name is verified, exact match is used e.g. “github.com” would not be valid for “www.github.com” or “images.github.com”. But it is typically on the application to determine what expected name is. In the first certificate example the CommonName was “localhost”. Connecting to “127.0.0.1” would typically fail even if “localhost” may (or may not) resolve to 127.0.0.1.

This become limiting as single web site can often be accessible via different names (for example with or without leading www) or there could be subdomains for various reasons. To make that easier, it is possible to use wildcard matching like CN=*.azurewebsites.net. However, that can add significant cost comparing to string comparisons and specially crafted malicious strings can lead to DoS. To make that smaller problem, RFC6125 from 2011 forbid certain patterns like "sub.*.foo.com""*.*.foo.com" or "*". That basically enforces valid domain and allows simplified matching (like comparing last N characters) instead of building full state machine. With the restrictions above, wildcard certificates are quite common.

2.3 Application logic

Applications are permitted to apply any addition validation logic. For example, certificates may be issued to particular devices. It is not uncommon to put additional information like device serial number (different from certificate serial number) to either Subject as some additional element in the list or add it as private extension. That allows to bring application or business logic to validation and reject certificates that do not meet the additional check.

While this may work from a security point of view it usually has poor useability. For example, when TLS handshake fails, user may get some cryptic error from browser or application but there is no good channel to render user friendly error. For that reason, if such verification is desirable, it is done after initial certificate validation and failures are handled at application layer.

3 Extended attributes

To make certificates more usable, version 3 of the specification added extensible framework for extensions. Since the certificates are generally encoded in ASN1 notation in TLV, known extensions have assigned OID and they can vary in internal structure. That allows you to add new extensions easily and they can be optionally ignored by implementations that do not understand them. In common case, they are set by CA when certificate request is signed. Public CAs generally always follow best practices so following the section is mostly for deployments where PKI is managed privately.

3.1 Extended name check

As described above, even with wildcard it is not always easy to manage single certificates that would cover multiple names, or IP addresses. SubjectAlternativeName is extension that allows you to add list of DNS names, email or IP addresses or URIs. The certificate should be considered valid as long as any of the variants matches the expected name. That allows single certificates to server multiple sites or subdomains without forcing them to have common domain.

3.2 Certificate usage

To prevent certificate use even more, there are several additional properties that can be set.

BasicConstrains is flag showing if given certificate is intended as CA certificate or not. That should always be set to false for certificates presented by web site or email client.

KeyUsage added digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly primitives. Basic principle is to use only minimal flags to minimize possibility of misuse. While this may not be obvious, digitalSignature should be sufficient for HTTPS web servers.

To make the usage somewhat more useable for users and align better with particular workloads, ExtendedKeyUssage added serverAuth, clientAuth, codesigning, emailProtection. (and few more). In typical scenario, certificate would have exactly one use e.g. certificate for user/client would never be used to prove server's identity.

Current recommendation is to always set all three extensions above for leaf certificates. BasicConstrains and KeyUssage should be marked as critical extensions.

4 Trust

Sections above were so far focused on specific certificate instances. But that does not cover an important aspect of PKI and that is trust. Malicious attackers can forge any site certificate and create all the recommended extension, but they still cannot act as the original site because their certificates are not trusted. Certificates can be valid and not trusted or vice versa. The following section will talk more about establishing trust.

4.1 Certificate chains

We already talk about different certificate roles. Let's look at the certificate for https://github.com/

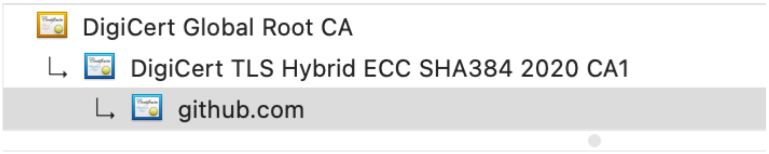

DigiCert Global Root CA
↳ DigiCert TLS Hybrid ECC SHA384 2020 CA1
↳ github.com

Figure 3: github.com certificate chain

It all starts with Root certificate. Root certificate is a special self-signed certificate (issuer and subjects are identical) that is somehow trusted by OS or application. That CA can sign intermediate certificate and that can either sign another subordinate CA or specific leaf certificate. Because root CA is trusted, the intermediate certificate is trusted and therefore the leaf certificate is trusted. When all the previous checks succeed, the receiver can assume that the certified is not forged by an imposter.

This relation is called a certificate chain, and in most cases, they can be represented by a tree graph. When a leaf certificate is presented, the sequence is opposite(reversed?). The validation logic will need to find who signed given certificate, find CAs certificate, verify CA's certificate properties, and if not trusted, repeat the process to parent certificate. This process can have several outcomes. If this process finds trusted CA certificate (root or any intermediate anchor) and all other checks pass, the certificate is deemed valid and trusted. Information coming across can be viewed as coming from a trusted source (most of the time) and depending on usage, privacy can be counted on. This process may lead to root CA that is not trusted or this process may fail to find intermediate certificate. In that case the leaf certificate will not be trusted, and verification will fail. In general, there is no distinction between the cases.

While there is convenience in trusting small set of CA certificates to gain access to thousands of sites, this is crucial and often problematic part of the validation.

4.2 OS Trusted certificates

Based on OS and version, there would be 50-150 certificates pre-selected by your OS vendor as trusted roots. In general, anything signed by them (or their subordinates) will be trusted by the OS or applications. Each leaf certificates can be signed by one (or more) trusted CA and they would all be treated equally. That creates a problem if you do not trust your OS default selection. In the old days, one could possibly opt-in only for CA applicable to particular country or jurisdiction but that does not really work well in the global market and modern Internet days. If an application wants to, it is usually possibly to detach it from OS trust and use custom root certificates. Some browsers did that in the past. While that approach may look reasonable and limiting trust is usually good, it should be used with caution. For example, there were deployments in the past when some small, embedded device would trust only particular issuer. But when the signing change (or public CA) goes out of business, it is very difficult to update the trust. That also makes validation less predictable. Accessing particular sites may work from a browser when using OS trust but it may fail when custom trust is used by an application.

The general recommendation is to leave the OS trust intact unless there is good reason for change. It most cases, somebody may want to add additional trust to support private PKI. That can be done either via various management tools or it could be part of the application.

5 Additional checks and online processing

In addition to all processing above, endpoints processing certificates can use additional strategies to improve validation.

5.1 Fetching missing certificates

Section 4.1 described the process of building a certificate chain. According to RFC, TLS servers SHOULD send intermediate certificates during handshake. But often they don't – either because of some misconfiguration or intentionally to decrease data exchanged during handshake. But this is only a recommendation not a strict requirement. To improve chances of successful validation, Authority Information Access (AIA) extension was added. It allows you to specify how CA certificate can be retrieved in case it is not available during verification (TLS handshake or signature verification). When the given intermediate certificate is missing and AIA is present, validators can choose to fetch it online. While it is convenient it may have unexpected consequences. It can make the verification unpredictable (as it may or may not work depending on network condition and security policies) as well as there may be significant performance drawbacks. Let's consider a server application that retrieves requests and needs some data from another server in a data cluster. If there is problem with the certificate chain, it may try to fetch it externally and that would introduce significant delay that can lead to service failure. While this typically impacts only clients and middle-ware applications, it can possibly impact standalone servers as well if they use client certificates. Since the client certificate is untrusted, malicious attackers can force server to do IO to server at attacker's command.

Depending on AIA fetching implementation, there could be additional concerns beyond reliability and performance. When requesting some information via HTTP, the server can send a HTTP redirect. Most HTTP clients would honor this and they would try to fetch data from the updated URL. That can possibly lead to longer delay, redirect loops and possibly to protocol changes.

It is strongly recommended to put some bounds on AIA process – in terms of duration, CPU consumption and general IO management.

5.2 Revocation status

Additionally, to all check above, X509 certificates also have a concept of revocation e.g. marking certificate invalid/untrusted after they were issued and possibly deployed in production. That allows recovery in cases when leaf or intermediate CA was compromised.

While that is a great improvement from a security perspective, it has the disadvantages of additional IO described in section above. Most browsers would do this as optional check e.g. use it when available, but they would not fail the handshake if not available.

The initial mechanism is Certificate Revocation List (CRL). It is essentially a list of certificate serial numbers that CA no longer sees as trusted. It is sealed with CA signature to prevent malicious DoS on valid certificates. Depending on the CA, the CRL list can have significant size. For example, `http://crl3.digicert.com/DigiCertTLSRSASHA2562020CA1-4.crl` is ˜ 2MB as of spring 2021. Fetching that on slow links during TLS handshake can cause significant delay and possibly cost more if used on networks charging per transferred data.

To avoid issues with increasing CRL size, RFC 6960 describes online protocol allowing to fetch status for specific certificate. While the relative overhead is much bigger, it allows you to do quick and relatively simple validation. Similar to CRL, responses are signed by CA certificate to eliminate forging.

6 Conclusion

Certificate validation is nontrivial process. Not only certificates need to be check in isolation for properties and cryptographic aspects, they also need to be verified together in chains of trust and together with intended use. While some techniques may improve validation process, they can also lead to service outages and possible DoS scenarios. When possible standard solutions should be used as they tend to be more complete and accurate.

7 .NET issues related to certificate validation

This is of course not complete list, but it links specific validation issue with full details.

- RemoteCertificateValidationCallback cannot detect future certificates
  `https://github.com/dotnet/runtime/issues/63830`
- Https against invalid domain names (containing underscores) behaves differently on Linux than Windows and macOS
  `https://github.com/dotnet/runtime/issues/58722`
- SSL Connection Error with Wildcard Certificates and Underscores
  `https://github.com/dotnet/runtime/issues/35880`
- HttpClient rejects valid certificates for dot-appended FQDNs
  `https://github.com/dotnet/runtime/issues/57792`
- SSL RemoteCertificateNameMismatch on MacOS Catalina
  `https://github.com/dotnet/runtime/issues/666`
- Unexpected HttpClient cert validation & timeout failures
  `https://github.com/dotnet/runtime/issues/805`
- DotNet HttpClient doesn't give SSL Error While curl and Java fails
  `https://github.com/dotnet/runtime/issues/55322`
- Ubuntu18.04 net5.0 HttpClient SendAsync ignores timeout (via token), hangs for some time and eventually crash the app
  `https://github.com/dotnet/runtime/issues/45010`
- SSLStream should support taking a pre-validated immutable full certificate chain
  `https://github.com/dotnet/runtime/issues/35844`

- SSLStream should support cancelling certificate chain building with cancellation token
  `https://github.com/dotnet/runtime/issues/35839`
- High native memory usage in certificate revocation checking
  `https://github.com/dotnet/runtime/issues/52577`
- Kestrel stops serving https (http2) requests after reboot under load
  `https://github.com/dotnet/aspnetcore/issues/21183`

MeeSign: Prahové podepisování pro správu elektronických důkazů

Antonín Dufka, Jakub Janků, Jiří Gavenda, Petr Švenda

Abstrakt

Přístup k elektronickým důkazům typicky vyžaduje schválení od více osob za účelem zajištění odpovědnosti. Toto schvalování se v digitálním prostředí zpravidla realizuje digitálními podpisy, nicméně, řada stávajících nástrojů pro práci s dokumenty neumožňuje ověřit či vytvořit více digitálních podpisů stejného dokumentu.

Prahová kryptografie řeší tento problém tím, že zprostředkuje konstrukci digitálních podpisů, které pro vytvoření vyžadují souhlas více stran, ale jsou nerozlišitelné od standardních digitálních podpisů a tedy i kompatibilní se stávajícími nástroji. Tyto techniky používáme v návrhu systému MeeSign, otevřené platformy pro vícestranné podepisování dokumentů.

1 Úvod

V rámci procesu důkazního řízení je typicky nutné zaznamenávat přístup k zajištěnému důkaznímu materiálu, průběžně vznikajícím analýzám a dalším dokumentům, čehož lze v elektronickém prostředí dosáhnout tvorbou digitálních podpisů. Digitální podpisy nepopiratelně zaznamenají přístup dané osoby k danému dokumentu pomocí jejího kryptografického klíče a slouží tak pro zajištění odpovědnosti. Některé procesy vyžadují podpis více stran, avšak běžně používané nástroje tuto funkcionalitu mnohdy nepodporují – ať už se jedná o jejich vytváření v rámci jednoho dokumentu nebo následné ověřování. Takovéto nástroje obvykle umožňují vytvořit a ověřit nanejvýš jeden podpis na dokumentu. Nicméně, vzniklé podpisy je potřeba verifikovat všechny a často opakovaně, automatizovaně a s využitím existujících standardních softwarových prostředků.

Myšlenka prahové kryptografie [1] byla navržena jako způsob reflektování společenských struktur v digitálním světě. Prahová kryptografie se zabývá návrhem protokolů, které umožňují provádět některé operace pouze při účasti dostatečného množství z vybraných účastníků. Například, s pomocí protokolů prahové kryptografie lze vytvářet digitální podpisy pouze pokud se podepisování zúčastní dostatečně velká podmnožina oprávněných podepisujících. Takto vzniklé podpisy mohou být navíc nerozlišitelné od standardních jednostranných digitálních podpisů a tudíž verifikovatelné standardními nástroji.

Z tohoto důvodu je prahová kryptografie vhodným řešením výše popsaného problému a využíváme ji v nástroji MeeSign – systému pro vícestranné podepisování elektronických dokumentů. V této práci prezentujeme architekturu systému MeeSign a jeho otevřenou implementaci dostupnou v našich GitHub repozitářích[1][2]. Aktuálně nástroj umožňuje tvorbu ECDSA podpisů PDF dokumentů skupinami libovolné velikosti. Takto vzniklé podpisy jsou pak ověřitelné standardními nástroji pro prohlížení dokumentů PDF.

2 Prahové podpisy

V této kapitole stručně popisujeme historii prahové kryptografie a shrnujeme základní vlastnosti vícestranných prahových protokolů pro podepisování.

V roce 1987 Yvo Desmedt navrhl, že by kryptografie měla odrážet strukturu naší společnosti a zavedl myšlenku prahových podpisů [1]. Myšlenka byla dále rozpracována a následně došlo k návrhu prvních prahových podepisovacích protokolů založených na kryptosystému ElGamal [2] v roce 1989. V roce 1991 došlo k publikování prahových protokolů pro široce rozšířený kryptosystém RSA [3]. Následoval návrh prahových protokolů i pro kryptosystém (EC)DSA od Gennara et al. v roce 1996 s předpokladem kompromitace méně než poloviny zúčastněných stran [4]. V roce 2001 pak MacKenzie a Reiter navrhli ECDSA protokol speciálně pro případ 2-ze-2 [5], který nemohl být realizován dřívějším protokolem Gennara et al. V roce 2016 došlo k vytvoření obecného protokolu k-z-n pro ECDSA s využitím tzv. Paillierova kryptosystému [6]. V rychlém sledu došlo k publikování dalších variant [7, 8, 9, 10, 11] představených po roce 2017. Všechny tyto

[1] `https://github.com/crocs-muni/meesign-server`
[2] `https://github.com/crocs-muni/meesign-client`

varianty již nepředpokládaly nadpoloviční většinu nekompromitovaných stran a snižovaly výpočetní a komunikační složitost, díky čemuž jsou již prakticky implementovatelné na běžných zařízeních jako jsou notebooky a mobilní telefony.

Zvýšená aktivita ve výzkumu a implementaci prahových podpisů, zejména ECDSA, byla v poslední době způsobena především jejich nasazením pro zlepšení bezpečnosti a flexibility kryptoměnových peněženek. Tyto technologie se však již využívají i v dalších oblastech například v DNSSec nebo v bankovnictví.

2.1 Základní vlastnosti prahových protokolů

Ve schématu prahového digitálního podpisu má každá z n stran (konkrétní hodnota n je volitelným parametrem pro danou instanci) podíl soukromého klíče a definuje podmnožinu (kvórum) stran, která je požadována pro vytvoření výsledného podpisu.

Schéma prahového podpisu typu k-z-n je sada protokolů s následujícími vlastnostmi:

1. Všech n stran se účastní generování společného veřejného klíče. Každá ze stran získá po dokončení generování svou vlastní část soukromého klíče.

2. Jakákoli podmnožina stran o velikosti alespoň k může vytvořit validní digitální podpis.

3. Žádná podmnožina o velikosti menší než k nemůže vytvořit validní digitální podpis.

4. Během vytváření podpisu nikdy nedochází k rekonstrukci celého podepisovacího klíče na jednom místě.

5. Žádná ze stran nemůže být oklamána k podpisu jiné zprávy, než předpokládá na základě podepisovacího protokolu.

Prahové podpisy mohou být založeny na různých kryptografických primitivech (například RSA, eliptické křivky, bilineární párování nebo kryptosystémy založené na mřížkách). Pro účely praktické aplikace se však zaměřujeme primárně na standardní, široce používané kryptosystémy typu RSA nebo ECDSA.

Z hlediska potřeby dlouhodobého uchovávání některých artefaktů důkazního řízení jsou relevantní i tzv. post-kvantové kryptografické systémy poskytující bezpečnost vůči dostatečně velkým kvantovým počítačům. Zároveň však ještě není dokončena standardizace těchto algoritmů a v dohledné době tedy nelze očekávat jejich širší rozšíření a zpětnou kompatibilitu s existujícími systémy.

3 Systém MeeSign

Systém MeeSign (Multi-party electronic evidence Signing) má za cíl poskytnout platformu pro digitální podepisování dokumentů několika samostatnými uživateli, kteří tak vyjadřují svůj souhlas s obsahem dokumentu. Takto vzniklý podpis je nerozlišitelný od standardního jednostranného podpisu, a tedy je kompatibilní se standardními nástroji pro ověřování digitálních podpisů, avšak vyjadřuje akci více osob.

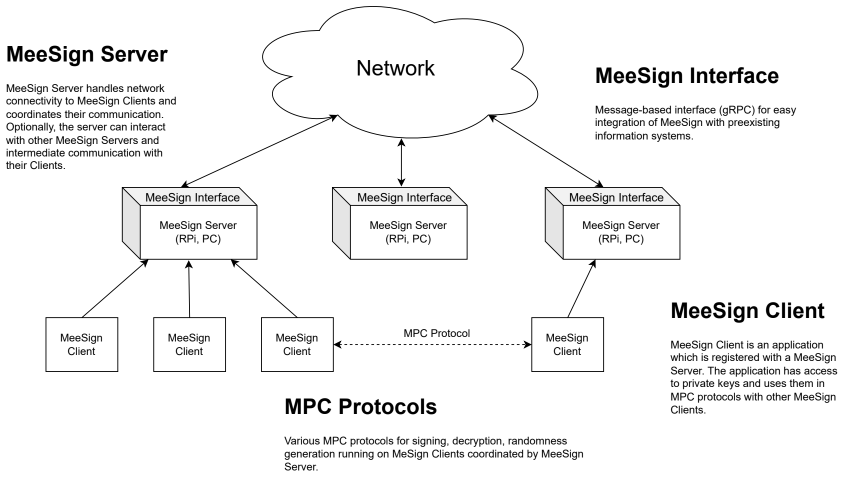
3.1 Použití platformy

Uživatelská zařízení si vygenerují svůj vlastní identitní klíčový pár (privátní a veřejný klíč) a pomocí svého veřejného klíče jako identifikátoru jsou registrováni do systému MeeSign. Během této registrace by mělo proběhnout ověření identity vlastníka zařízení a doplnění jeho identifikačních údajů, podobně jako v případě registrace u certifikační autority.

Registrovaná zařízení mohou být organizována do tzv. podepisovacích skupin, které realizují vícestranné podepisování. Při vytváření těchto skupin je potřeba zadat identifikační informace skupiny, zvolit seznam účastnících se zařízení a nastavit minimální hranici (práh) stran, které se musí účastnit podepisování, aby vznikl validní podpis.

Při vytváření skupiny dochází ke generování skupinového klíče, pod kterým bude skupina vydávat podpisy. Toto generování vyžaduje vícekolový interaktivní protokol, který je prováděn zařízeními jednotlivých uživatelů, ti však musí svou účast v protokolu (a tedy i v dané skupině) nejprve potvrdit. Jakmile všechny strany potvrdí svou účast, protokol proběhne transparentně a účastníci jsou informování o úspěšném vytvoření skupiny.

Skupinám mohou být přiřazovány dokumenty k podepsání. Po přiřazení se tyto dokumenty zobrazí všem členům skupiny, kteří mohou potvrdit či zamítnout účast v podepisování daného dokumentu, čímž vyjadřují souhlas či nesouhlas s jeho obsahem. Jakmile alespoň daná hranice uživatelů potvrdí svůj souhlas s vytvořením podpisu (vynuceno na úrovni kryptografického protokolu), inicializuje se protokol vícestranného podepisování. Tento protokol využívá privátní klíče uložené v zařízeních, která


MeeSign Server
MeeSign Server handles network connectivity to MeeSign Clients and coordinates their communication. Optionally, the server can interact with other MeeSign Servers and intermediate communication with their Clients.
Network
MeeSign Interface
Message-based interface (gRPC) for easy integration of MeeSign with preexisting information systems.
MeeSign Interface
MeeSign Server (RPi, PC)
MeeSign Client
MPC Protocol
MeeSign Client
MeeSign Client is an application which is registered with a MeeSign Server. The application has access to private keys and uses them in MPC protocols with other MeeSign Clients.
MPC Protocols
Various MPC protocols for signing, decryption, randomness generation running on MeeSign Clients coordinated by MeeSign Server.

Obrázek 1: Základní architektura aplikace MeeSign umožňující zapojení různých typů zařízení a koordinaci vytvoření vícestranného podpisu.

souhlasila s vydáním podpisu. Po dokončení jsou všichni uživatelé informováni o úspěšném vytvoření podpisu dokumentu a tento dokument je společně s podpisem uložen. Korektnost takto vzniklého podpisu lze ověřit standardními nástroji pro ověřování podpisů PDF dokumentů.

V dalších iteracích se plánuje přidání podpory pro podepisování jiných typů dokumentů, nastavení politik pro podepisování, možnost využití privátních klíčů v kryptografických čipových kartách zprostředkovaných mobilním zařízením, zálohování pomocí vícestranného dešifrování, a propojení zařízení napříč více samostatnými MeeSign servery.

3.2 Architektura

Návrh systému (Obrázek 1) se skládá ze serverové aplikace a instancí klientských aplikací pro chytré mobilní telefony.

Serverová aplikace

Serverová aplikace poskytuje následující funkcionalitu:

1. ukládá informace o registrovaných zařízeních, skupinách, stavech protokolů a prováděných úlohách;
2. poskytuje rozhraní pro konfiguraci, registraci nových zařízení, správu skupin a zadávání úloh;
3. zprostředkovává a koordinuje komunikaci mezi jednotlivými zařízeními.

Tato služba je implementována v jazyce Rust. Rozhraní služby využívá gRPC protokol pro snadnou integraci s jinými systémy.

Klientská aplikace

Klientská aplikace je hlavním přístupovým bodem uživatelů do systému MeeSign. Tato aplikace má za cíl:

1. poskytnout uživatelské rozhraní pro interakci se systémem MeeSign – registraci zařízení, vytváření skupin, účast ve skupinách, zadávání a vykonávání podepisovacích úloh;
2. bezpečně uchovávat privátní klíče identity zařízení, a i klíče pro účast ve skupinách;
3. realizovat síťovou komunikaci se serverem a přijímat notifikace v reálném čase;
4. vykonávat kryptografické operace pomocí privátních klíčů při účasti ve vícestranných protokolech.

Implementace je provedena v jazyce Dart s frameworkem Flutter, který umožňuje jednotně vytvářet multiplatformní aplikace pro chytré telefony. Pro realizaci kryptografické funkcionality se využívá prahového ECDSA protokolu dle návrhu Gennara a Goldfedera [10] implementovaného v jazyce Rust v knihovně ZenGo X [12].

Implementace

Nynější verze systému umožňuje vytvářet ECDSA podpisy PDF dokumentů skupinami s nastavitelným počtem stran – např. 3-ze-3 (tři uživatelé, všichni se musejí podílet na vytvoření podpisu) nebo 2-ze-3 (tři uživatelé, libovolní dva mohou vytvořit digitální podpis). Zdrojové kódy jsou dostupné pod otevřenou licencí v git repozitářích na adresách `https://github.com/crocs-muni/meesign-server` a `https://github.com/crocs-muni/meesign-client`.

4 Závěr

Prahová kryptografie je vhodným nástrojem pro konstrukci digitálních podpisů vyžadujících součinnost více stran, které jsou zároveň nerozlišitelné od standardních jednostranných podpisů. Díky tomu mohou být takto vzniklé podpisy přímo integrovány do stávajících technických řešení a tedy i kompatibilní s běžnými verifikačními nástroji. Z těchto důvodů využíváme prahové kryptografie v platformě MeeSign – systému pro vícestranné podepisování elektronických dokumentů.

Stávající implementace poskytuje základní funkčnost demonstrující definici parametrů vícestranného protokolu, kolaborativní generování klíče a vytvoření digitálního podpisu dokumentu ve formátu PDF. Pro budoucí iterace aplikace MeeSign je plánováno postupné začlenění nových protokolů, podpora dalších typů dokumentů, rozšíření stávajících rozhraní a umožnění interakce více MeeSign serverů a jimi spravovaných zařízení.

Poděkování

Tento článek vychází z průběžné zprávy Identifikace požadavků a návrh technického řešení systému pro bezpečnou správu el. důkazů za rok 2021 [13] z projektu MVČR Impakt VJ01010084 Elektronické důkazy v trestním řízení.

Odkazy

[1] Yvo Desmedt. „Society and group oriented cryptography: A new concept“. In: *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1987, s. 120–127.

[2] Yvo Desmedt a Yair Frankel. „Threshold cryptosystems“. In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, s. 307–315.

[3] Yvo Desmedt a Yair Frankel. „Shared generation of authenticators and signatures“. In: *Annual International Cryptology Conference*. Springer. 1991, s. 457–469.

[4] Rosario Gennaro et al. „Robust threshold DSS signatures“. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, s. 354–371.

[5] Philip MacKenzie a Michael K Reiter. „Two-party generation of DSA signatures“. In: *Annual International Cryptology Conference*. Springer. 2001, s. 137–154.

[6] Rosario Gennaro, Steven Goldfeder a Arvind Narayanan. „Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security“. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2016, s. 156–174.

[7] Yehuda Lindell. „Fast secure two-party ECDSA signing“. In: *Annual International Cryptology Conference*. Springer. 2017, s. 613–644.

[8] Jack Doerner et al. „Secure two-party threshold ECDSA from ECDSA assumptions“. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, s. 980–997.

[9] Yehuda Lindell a Ariel Nof. „Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody“. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, s. 1837–1854.

[10] Rosario Gennaro a Steven Goldfeder. „Fast multiparty threshold ECDSA with fast trustless setup“. In: *Proceedings of the 2018 ACM SIGSAC*. 2018, s. 1179–1194.

[11] Rosario Gennaro a Steven Goldfeder. „One round threshold ECDSA with identifiable abort“. In: *Cryptology ePrint Archive* (2020).

[12] ZenGo X. *Multi-party ECDSA*. `https://github.com/ZenGo-X/multi-party-ecdsa`. [cit. 22. duben 2022]. 2019.

[13] Petr Švenda et al. *Identifikace požadavků a návrh technického řešení systému pro bezpečnou správu el. důkazů*. Zpráva k řešení výzkumného pilíře Technologie (Bezpečné vícestranné digitální podepisování). 2021.

Binary Obfuscation using the LLVM Framework

Roman Oravec

Abstract

In the context of cybersecurity, obfuscation is a method of manipulating a computer program with the intention to obscure its inner workings. Various obfuscation techniques have found their use as a means to protect intellectual property and to prevent code tampering, as well as to achieve malicious purposes, such as creating malware that can circumvent detection mechanisms. In this work, we examine multiple commonly used obfuscating techniques and freely available tools that implement them. Furthermore, we have implemented several obfuscating transformations, leveraging the LLVM Pass Framework. The quality of the implemented transformations has been tested and evaluated with respect to selected metrics.

1 Introduction

As shown by Barak et al. [1], it is not possible to create a perfect obfuscation and with enough time and resources, any program can be deobfuscated. However, the goal of obfuscation is to discourage a reverse engineer from analyzing or tampering the code, by transforming a program in a way that any such attempts would be infeasible. This can provide useful as a means to harden parts of a program that perform sensitive operations, such as cryptographic functions.

Obfuscation is also widely used by malicious actors, who aim to produce malware that can bypass automated analysis tools, e.g. an antivirus software. Furthermore, their goal is to confuse a reverse engineer so that it would take more time and resources to analyze the malware and develop countermeasures. Both malicious and benign parties keep trying to come up with more advanced obfuscating transformations as well as new reverse engineering techniques, and the result is a form of a cat-and-mouse game that accelerates the research in this area.

In a pivotal paper published in 1997 [2], Collberg et al. proposed a variety of obfuscating transformations, together with recommendations on how to categorize and evaluate them. Many techniques used in the present are based on the ones described in this paper. Most of them can be sorted into two categories – control obfuscation and data obfuscation. Techniques in the first category manipulate the flow of control in the program, either by breaking up or merging computations, randomizing their order, or adding redundant code. The latter category includes transformations that obscure data such as character strings and constants, for example by changing their encoding or making the program compute them during runtime, and thus not storing them statically as a part of the executable file.

2 Obfuscating transformations

Collberg et. al [2] formally defines an *obfuscating transformation* as follows:

Definition 1 (Obfuscating transformation).
Let $P \xrightarrow{\mathrm{T}} P'$ be a transformation of a source program P into a target program P'. $P \xrightarrow{\mathrm{T}} P'$ is an *obfuscating transformation*, if P and P' have the same *observable behavior*[1]. More precisely, in order for $P \xrightarrow{\mathrm{T}} P'$ to be a legal obfuscating transformation, the following conditions must hold:

- If P fails to to terminate or terminates with an error condition, then P' may or may not terminate.
- Otherwise, P' must terminate and produce the same output as P.

This definition states that applying the transformations should preserve the input-output behavior of the program, but allows the obfuscated version of the program to produce some additional behavior, for example performing calls to the operating system or creating new files.

The transformations may, and usually do, increase the computational complexity and memory requirements of the program.

[1]Regarding the program inputs, corresponding outputs, and program termination.

2.1 Opaque predicates

Opaque predicates are common but effective constructs used for obfuscation. They are typically used to enhance other obfuscation methods, for example, dead code insertion, or control-flow graph flattening. Constructing an opaque predicate consists of transforming a simple boolean expression into a complex one.

Definition 2 (Opaque predicate)**.** A predicate P is opaque at point p in a program, if its outcome is known at obfuscation time. We write P_p^F (P_p^T) if P always evaluates to `False` (`True`) at p, and $P_p^?$ if P may sometimes evaluate to `True` and sometimes to `False` [3].

Based on the possible values of an opaque predicate, we can distinguish two types:

Invariant opaque predicates. The value of the predicate is fixed, i.e. the obfuscator knows whether it evaluates to `True` or `False`, but its outcome is hard to deduce for the reverse engineer performing static analysis.

Two-way opaque predicates. This type of predicates can evaluate either to `True` or `False` for all possible inputs. Collberg et al. [2] suggested using two-way opaque predicates as a branching point to two functionally identical branches, which can be created by cloning a single branch and applying different obfuscations on both of them. This makes the static analysis of the program harder while preserving the original functionality.

2.2 Mixed Boolean-Arithmetic

A Mixed Boolean-arithmetic expression (MBA) is composed of integer arithmetic operations on n-bit words $(+, -, \times, \div)$ and bitwise operations $(\wedge, \vee, \oplus, \neg)$. Zhou et al. [4] present a method to generate an unlimited supply of MBA transforms based on MBA expressions, MBA identities, and invertible functions, which can be used to obscure secret constants, intermediate values, and algorithms, while preserving the original functionality. Together with opaque predicates, they can be considered building blocks for creating or enhancing various obfuscating transformations. Below is an example of constructing an MBA expression to obfuscate a constant integer in the program:

Let:

- f be an invertible polynomial over $\mathbb{Z}/(2^n\mathbb{Z})$

- $g = f^{-1}$
- E an MBA expression non-trivially equal to zero, for example $E = x + y - (x \vee y) - (\neg x \vee y) + (\neg x)$
- C a constant to be obfuscated.

To obfuscate C, we can encode it as $C = g(E + f(C))$. The following example will show how the process works in practice. Suppose we are performing calculations in $\mathbb{Z}/(2^n)$ with 32-bit words[2], f is a linear function with coefficient a being an odd integer, so that it is invertible $\mod 2^{32}$ and b is an arbitrary integer, while all integer values are 32 bits large:

$$\begin{aligned}
C &= 123456 \\
a &= 1337 \\
b &= 42 \\
x &= 1307300694 \\
y &= 2583472541 \\
f(x) &= ax + b = 1337x + 42 \\
g(x) &= a^{-1}x + (-a^{-1}b) \\
&= 1337^{-1}x + (-1337^{-1} \cdot 42) \\
&= 1185372425x + 1753965702 \\
E &= x + y - (x \vee y) - (\neg x \vee y) + (\neg x)
\end{aligned}$$

Then we can encode C as:

$$\begin{aligned}
f(C) &= 1337 \cdot 123456 + 42 = 165060672 + 42 \\
E &= 1307300694 + 2583472541 - 3724540895 \\
&- 3153898941 + 2987666601 = 0 \\
C &= g(x + y - (x \vee y) - (\neg x \vee y) + (\neg x) + f(C)) \\
&= g(0 + 165060672 + 42) \\
&= 1185372425 \cdot 165060714 + 2541001594 \\
&= 123456 \mod 2^{32}
\end{aligned}$$

[2]All calculations are $\mod 2^{32}$.

The intermediate results of the functions f, g, and the MBA expression E are calculated during runtime, which makes it much harder to obtain the value of C during static analysis. The values of a, b, x, y can be either randomly generated during compilation (obfuscation) time, or by injecting a call to a random number generator that executes during runtime. The third option is to use suitable program inputs or function arguments, which would hinder deobfuscation techniques based on taint analysis[3].

Guinet et al. presented a tool called `arybo` [5], which analyzes the operations performed by MBA expressions at the bit-level. The tool is able to significantly simplify MBA expressions, thus presenting a possibility to circumvent this type of obfuscation.

2.3 Instructions Substitution

Instructions substitution is one of the most simple obfuscation techniques. The principle of this method is to replace instructions containing binary arithmetic operations, such as addition and subtraction, and binary boolean operations, such as logical AND or XOR, with more complicated sequences of code, which yield the same result.

Despite its simplicity, this technique is still used in current obfuscation tools. For example, *Obfuscator-LLVM* uses simple substitutions using expressions composed exclusively of arithmetic operations to substitute addition and subtraction, and expressions composed of boolean operations to substitute boolean XOR, AND, and OR. Below are some of the substitutions implemented in *Obfuscator-LLVM*:

$$\begin{aligned}
&x + y \rightarrow -(-x + (-y))\\
&x - y \rightarrow x + (-y)\\
&x \vee y \rightarrow (x \wedge y) \vee (x \oplus y)\\
&x \oplus y \rightarrow (\neg x \wedge y) \vee (x \wedge \neg y)
\end{aligned}$$

It is also important to note that the compiler can optimize out this kind of transformation, therefore it should not be used for source-level obfuscation. It also implies that it should be run after the optimization passes if we are obfuscating at the intermediate representation level.

A reverse engineer could use an optimizer on the obfuscated binary file to get rid of this transformation and reduce the complexity of the code

[3]Tracking flow of values in the program and identifying values and variables that influence program's outputs, as well as the control flow of the program.

for easier analysis. However, increased diversification of the code can still be useful, for example, to bypass an antivirus engine that is performing a static analysis of a program based on its signature, e.g., looking for known malicious patterns and sequences of instructions.

2.4 Garbage Code Insertion

This technique consists of inserting arbitrary instructions into the program without making an impact on the execution of the program. The total number of different programs possible through garbage insertion is limited from above by the total number of free bits available for program space, which is limited only by available memory for program storage and is clearly enormous [6].

Similarly to instructions substitution, the inserted code can be optimized out, therefore it should be applied after optimizing the program and it might get easily removed by a reverse engineer who is analyzing the program.

This technique could be used to bypass simple automated malware analysis engines, as it breaks the signature of the program. The garbage instructions get executed, which adds complexity while performing dynamic analysis of the program, but on the other hand, it might impact the performance.

A simple example of this technique is inserting `NOP` instruction into the assembly code. It does not have an impact on the program execution, but it's still reachable by the control flow of the program. A more advanced way to apply this method, described in [6], is to insert spurious calls to the operating system, which could lead the attacker to analyze a large amount of garbage code and increase the time needed to perform the analysis.

Yadegari et al. [7] proposed a generic automated approach for deobfuscation of executable code based on taint analysis, which tracks the flow of values from the program's inputs to its outputs. This method can identify instructions that do not affect the execution of the program and remove the garbage instructions from the code.

2.5 Dead code insertion

Dead code insertion is a technique similar to garbage code insertion. The main difference is that the dead code adds a branch to the control flow of the program, but this branch is never taken during the execution of the program.

This method was first introduced by Collberg et al. [2]. The paper suggests, that there is a strong correlation between the perceived complexity of a piece of code and the number of predicates it contains. This technique could be further enhanced by using opaque predicates (2.1) – for example, adding a condition with an opaque predicate, which creates a branching point between a valid and a dead branch. The predicate would always evaluate to `True`, making it impossible for the control flow of the program to reach the redundant branch.

Another way to further confuse the reverse engineer, described in [2], is to add dummy blocks of code to the redundant branches. For example, one can clone a sequence of instructions from a valid block of code, introduce a bug into it and place it into the redundant (dead) branch.

This transformation can be removed by utilizing optimization features including in modern compilers, as well as performing the automated deobfuscation approach proposed by Yadegari. et al. [7]. An attack proposed by Salem and Bansescu [8], which is based on machine learning and pattern recognition algorithms to identify obfuscating transformations in the program, might also prove useful to a reverse engineer trying to remove this type of obfuscation.

2.6 Control Flow Flattening

This transformation was first described by Wang et al. in [9]. The goal of this technique is to obscure targets of the branches between the basic blocks and thus to make the analysis of a program more difficult.

First, the basic blocks of the function are put on the same nesting level, preceded by a new block, usually referred to as the *dispatcher*. The dispatcher contains code that works as a `switch` statement, used to determine which basic block is going to be executed next. In addition to the dispatcher, a routing variable also needs to be created. Each time one of the original basic blocks terminates its execution, the routing variable is updated and the flow of control is transferred back to the dispatcher, which forwards the control flow to the next basic block, in accordance with the value stored in the routing variable.

The main issue with control flow flattening is finding a way to make the information about the dispatcher, the routing variable, and its updating, difficult to analyze. In its naive implementation, where the routing values

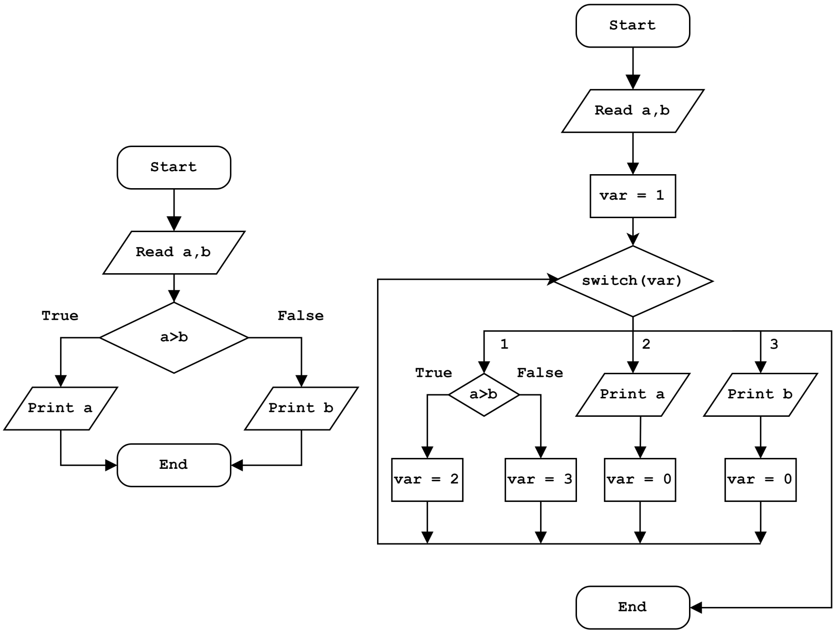


Figure 1: Flowchart of a simple program returning a maximum of two numbers, before and after flattening. Notice that the comparison and print statements are on the same level of nesting after being flattened.

are hardcoded during the obfuscation (as in Figure 1), it's easy to analyze the code of the basic blocks and reconstruct the original control flow graph.

In [9], Wang et. al suggest the use of global pointers, as in some cases, analysis of pointers can be proven to be NP-hard [10]. In contrast, the authors of [11] propose using one-way functions, which are always hard to analyze.

Another issue with this obfuscation method is the computational overhead it introduces due to additional operations performed by the dispatcher. Johansson et. al [12] proposed a novel method using lightweight dispatchers, which present a similar level of complexity for the reverse engineer as analyzing a flattened program augmented with cryptographic hash functions, while reducing the overhead by one or more orders of magnitude.

2.7 Evaluating Obfuscating Transformations

Determining the usefulness of an obfuscating transformation is a complex task. When designing and evaluating an obfuscating transformation, one needs to consider multiple criteria, such as how hard would it be for an adversary (e.g., a reverse engineer) to understand the functionality of an obfuscated program P', how hard would it be to construct a deobfuscator, or how much resources would a deobfuscator need to reconstruct the original program P, given P' as an input. Unless the deobfuscation process is fully automated, these criteria will never be fully objective, since they will always, at least partly, depend on the cognitive abilities of the attacker.

Collberg et al. proposed some metrics [2], which can be used to quantify and approximate the quality of obfuscation methods. They have defined the following three criteria:

- *Potency* consists of various metrics which were originally designed to be used to measure software complexity in the field of software engineering, for example, the number of operators and operands in P, number of predicates (cyclomatic complexity) in a function, or a nesting level of conditional statements in a function.

- *Resilience* is measured on a four-point scale, ranging from *trivial* to *one-way*. The value of resilience depends on two parameters – *programmer effort* – how much time would a programmer need to construct a deobfuscator to reduce the *potency* of a transformation, and *deobfuscator effort* – time and space complexity of a deobfuscator which can reduce the *potency* of a transformation. *Programmer effort* is based on the scope of the transformation, from local to inter-process, while *deobfuscator effort* could be either polynomial or exponential.

- *Cost* of a transformation measures how much execution time and space overhead would a transformation introduce to the obfuscated program. This value is also measured on a four-point scale, ranging from *free* (transformation adds a constant overhead) to *dear* (P' requires exponentially more resources than P).

Mohsen and Pinto [13] proposed using Kolmogorov complexity [14] to measure the quality of obfuscation. Kolmogorov complexity can be described as the shortest length of a program, which can produce a given object (e.g., a binary string, or an obfuscated program). Due to the undecidability of the halting problem, exact Kolmogorov complexity can not be computed. However, it can be estimated using compression algorithms. More specifically, Kolmogorov complexity is the lower bound of a compression algorithm.

The idea to use Kolmogorov complexity to quantify the quality of obfuscation is based on an assumption that obfuscation produces irregularities in the obfuscated code (e.g., by inserting opaque predicates, or cloning and diversifying basic blocks), thus making it less comprehensible for the adversary. A program that contains more regular patterns can be compressed with a higher rate, and thus has a lower Kolmogorov complexity. In contrast, an obfuscated program would have higher Kolmogorov complexity, which implies that it is a useful metric for evaluating obfuscation, as shown in [14].

3 Open-source obfuscators based on LLVM

3.1 Obfuscator-LLVM

Obfuscator-LLVM aims to provide increased software security through code obfuscation and tamper-proofing[4]. The project is a fork of the LLVM compilation suite, therefore it works with the LLVM IR, utilizing LLVM's possibility of writing custom transformation passes. It supports all programming languages and target platforms that are currently supported by LLVM.

The open-source version[5] of the project implements three obfuscating transformations:

1. *Instructions Substitution* is the most simple obfuscating transformation included in the project. It replaces simple binary operations with more complex ones. Obfuscator-LLVM supports the substitution of integer additions and subtractions and the Boolean operators AND (&), OR (|) , and XOR (^). For example, the expression $a = b \land c$ is substituted as $a = (b \oplus \neg c) \land b$. Some of the operations have multiple substitution candidates, which are chosen randomly to increase code diversity. A full list of the implemented substitutions can be

[4]Preventing a user from modifying the software against the manufacturer's wishes.

[5]The authors also developed a commercial version, providing additional transformations.

found in [15]. The substitutions are rather simple and according to the authors, this transformation can easily be circumvented by re-optimizing the generated code.

2. *Bogus Control Flow* modifies the control flow graph by inserting a new basic block before an existing one. The new basic block ends with an opaque predicate (2.1), which always evaluates to `True`, making a conditional jump to the original basic block. The original basic block is also cloned, randomly filled up with various junk instructions, and inserted to the `False` branch leading from the new basic block. This cloned block is never reached, because of the invariant opaque predicate. The weakness in the implementation of this transformation is that it uses just a single opaque predicate:

   $$(y < 10 \,||\, x \cdot (x - 1) \mod 2 = 0)$$

   The two global variables, x and y, which are declared to construct this predicate, can also give a hint on where the opaque predicates are, which would make it easier for a reverse engineer to overcome this transformation.

3. *Control Flow Flattening* is implemented in a naive way, as described in Section 2.6 – by hard-coding the routing values during the obfuscation process. This implementation provides some resilience against automated analysis tools, e.g. signature-based malware scanners, but it does not propose a significant challenge for a reverse engineer or automated tools performing a more complex static analysis.

Apart from the aforementioned obfuscating transformations, this tool also implements a transformation pass for basic block splitting. This pass introduces additional complexity to the transformations manipulating the control flow.

The authors also added a feature to tag specific functions, which are supposed to be obfuscated, in the source code of the program. This way, the developers can reduce the negative performance impacts on the program they are obfuscating, by omitting non-crucial functions from the obfuscation process.

The GitHub repository, containing the open-source version, is not being maintained since then as well. The authors of the project apparently founded a startup named Strong.Codes, which has been later acquired by Snap Inc. [16].

3.2 Armariris

This tool extends the Obfuscator-LLVM project by adding a string obfuscation pass. By examining the source code of the project, we have found out that this transformation is rather simple. Strings are just being XOR-ed with a randomly generated integer (see the code snippet below), and a simple decoding function is inserted at the beginning of the basic block containing the encoded string. This function is then executed during runtime.

This method prevents the strings from being viewed in plain by a simple static analysis methods, such as using the `strings` GNU tool, but it does not require a significant effort to decode them, since we just need to find a right value for the XOR operation to obtain the original text.

```
cur->key = llvm::cryptoutils->get_uint8_t();
char *encr = const_cast<char *>(orig);
for (unsigned i = 0; i != len; ++i) {
encr[i] = orig[i]^cur->key;
}
dynGV->setInitializer(initializer);
gv->replaceAllUsesWith(dynGV);
```

3.3 Hikari

This tool is also based on the Obfuscator-LLVM project. However, it is modified to support obfuscation of Objective-C projects. It also introduces multiple new transformations:

- *AntiClassDump*, used to combat the `class-dump`[6] utility. However, the documentation states that it is not yet fully implemented.
- Improved version of the *Bogus Control Flow* pass from Obfuscator-LLVM, with dynamically-generated opaque predicates, instead of the original hard-coded one.
- *FunctionCallObfuscate* works by scanning all CallSites that refer to a function outside of the current translation unit then replaces then with `dlopen`[7] and `dlsym`[8] calls.

[6]https://github.com/nygard/class-dump

[7]https://man7.org/linux/man-pages/man3/dlopen.3.html

[8]https://man7.org/linux/man-pages/man3/dlsym.3.html

- *FunctionWrapper* simply creates dummy functions that wraps around the actual function call.
- *IndirectBranching* Branching Instructions are replaced with indirect branching, which is converted into register-based indirect calls on supported backend. This makes disassemblers fail to predict the complete control flow from static analysis.
- *StringEncryption* pass, which encodes the strings in a similar fashion as Armariris. However, the key generation is more sophisticated (see the code snippet below) and it also provides support for obfuscating strings in Objective-C.

```
vector<uint8_t> keys;
vector<uint8_t> encry;
for (unsigned i = 0; i < CDS->getNumElements(); i++) {
uint8_t K~= cryptoutils->get_uint8_t();
uint64_t V~= CDS->getElementAsInteger(i);
keys.push_back(K);
encry.push_back(K ^ V);
}
```

4 Custom implementation and testing

This section describes our own implementation of obfuscating transformations using the LLVM framework. The implementation also builds upon the foundations from Obfuscator-LLVM, while improving its currently available transformations and writing some new ones from scratch. Thanks to the modularity of the LLVM pass framework, each of the transformation passes can be applied independently, or combined in a sequence.

4.1 Design

Selected transformations are have been designed with regard to their resilience against reverse engineering and impact on performance. Below is a summary of the implemented passes:

- *Instructions Substitution* – Based on the method for generating MBA identities proposed by Zhou et al. [4], Eyrolles has generated a list of all rewrite rules composed of three boolean expressions [17]. We have decided to implement a selection of those rules in the obfuscator. For each binary operation[9], the rewrite rule is picked randomly out of three possibilities. to increase code complexity and diversity.

- *Opaque Constants* – Constant integers are being encoded using the MBA expressions, as described in 2.2.
- *Bogus Control Flow* – We have replaced the original, hard-coded predicate with two types of more advanced opaque predicates. The first type is similar to the original implementation, but the pass picks from a set of multiple arithmetic expressions, and it uses input arguments of the obfuscated function for the variables x and y to construct the expression. Since the formulas are always true for an arbitrary integer, the predicate will always lead to a correct block. The second type of predicates, referred to as *Symbolic memory opaque predicates*, is based on a method described in [18].
- *String Obfuscation* – During compilation (obfuscation), the strings can be encoded using an arbitrary function that transforms them. Then, a decoding function is injected into the LLVM IR module. Using the LLVM *ExecutionEngine*, the encoding function can be loaded from an external C source file, therefore this transformation is not limited only to XOR-ing with a key, as in the case of other available tools.

4.2 Testing

The test programs consist of a C++ implementation of SHA-512 hash function[10], a C++ implementation of AES-CBC[11] with 128-bit key, and an implementation of QuickSort[12] algorithm in C.

4.3 Potency

For testing potency, we have computed the software complexity metrics of the obfuscated bitcode, and compared it to the non-obfuscated version.

[9]Namely addition, subtraction, AND, OR, XOR.
[10]`https://github.com/martynafford/sha-2`
[11]`https://github.com/kkAyataka/plusaes`
[12]`https://www.programiz.com/dsa/quick-sort`

Except for the Bogus Control Flow pass, the transformations do not significantly change the Control Flow Graph of the program, since the passes obfuscating constants and substituting instruction operate only in the scope of individual basic blocks, while the pass obfuscating strings manipulates global variables in the bitcode. It also injects a call to decode the string, but this operation does not change the CFG in a significant way. Therefore, we are going to use instruction count (μ_1) and Kolmogorov complexity (μ_2) as metrics to evaluate the potency of obfuscations, and avoid other metrics which are mostly influenced by the changes of the CFG.

To obtain the values, we use the `obfuscation-metrics` tool[13], which includes an implementation of a simple LLVM analysis pass that parses the IR module and outputs the metrics. To compute Kolmogorov complexity, the tool uses the `zlib::compress()` function, which is included in the LLVM libraries.

Table 1 shows the changes in software complexity metrics after applying obfuscation, which allows us to estimate the potency of the transformations. Values in the table show a ratio of metrics of an obfuscated and a non-obfuscated program. Substitution ×2 shows the effects of applying the Instruction Substitution pass twice. String obfuscation 1 and 2 refer to the two different string encoding functions – based on bit rotation and bitwise XOR, respectively. The last four rows show the changes of the metrics after applying multiple passes sequentially. The ordering of the passes is discussed in detail in Section 4.3.

Combining the passes

After evaluating each pass individually, we have determined the following sequence in which we apply all of the passes:

String obfuscation[14] → *Bogus* → *Opaque constants* → *Substitution.*

The reasoning behind choosing this order of the passes is following:

- The String obfuscation pass injects new functions, which are not obfuscated, therefore it is reasonable to apply this pass first, so the subsequent passes can obscure these functions.

[13]`https://github.com/b-mueller/obfuscation-metrics`

[14]We have used the encoding and decoding functions based on bitwise XOR, since they have proved to have a higher potency.

- Bogus Control Flow pass creates clones of the basic blocks, which would be easily identifiable without subsequent obfuscation. The opaque predicates are hardened and diversified by applying the substitution pass.

- Expressions for hiding the constants also benefit from being applied before the substitution pass, same as the opaque predicates.

- Instructions substitution pass has basically the smallest scope – transforming single instructions. Since it might improve the resilience and potency of all the other passes, it is applied last.

Table 1 shows that a combination of the passes results in significantly higher potency. We also present the values of μ_1 and μ_2 when the passes have been applied in a reverse order, where it is clearly visible how the particular ordering of the passes can influence the potency. In this case, the values are significantly lower than the original sequence.

Due to the randomness of the obfuscation (e.g., randomly selecting the instruction substitution, randomly generating values for opaque constants), there is a slight variance in the resulting metrics when comparing obfuscated codes originating from the same program, after applying the same transformation in multiple independent runs. However, the variance was only around 4%, therefore we do not include the individual results of multiple independent applications of the transformations in the table.

There is a noticeable pattern among all of the results in Table 1. The number of instructions (μ_1) always increases more than Kolmogorov complexity (μ_2). This implies that the transformations have a larger influence on the size of the program, but they do not add irregularities to the program at the same rate. We suppose that this ratio can be made more even, if we add more diversity to the transformations, for example by implementing more substitution candidates for the instructions, creating arrays of various sizes for the symbolic memory opaque predicates, or by inserting random sequences of junk instructions into the cloned blocks in the Bogus Control Flow pass.

Performance impact

We have tested impact on the performance of the selected programs, and their obfuscated versions, by generating random inputs of various sizes

Program	SHA-512		AES		QuickSort	
Metric	μ_1	μ_2	μ_1	μ_2	μ_1	μ_2
Substitution	1.37	1.07	1.09	1.04	1.23	1.03
Substitution ×2	2.25	1.26	1.34	1.13	1.82	1.11
Opaque constants	2.66	1.86	3.29	2.40	3.76	1.96
String obfuscation 1	1.05	1.04	1.01	1.02	1.19	1.12
String obfuscation 2	1.09	1.05	1.02	1.03	1.35	1.16
Bogus Control Flow	3.52	2.23	4.28	3.49	7.18	3.40
Str 2 + Bogus	3.72	2.35	4.40	3.64	10.64	4.44
Str 2 + Bogus + Const	14.26	8.08	19.16	16.23	32.43	12.27
All	26.98	11.03	36.05	23.12	57.88	16.23
Reverse order	8.10	4.24	9.50	7.05	21.57	7.84

Table 1: Changes in software complexity metrics after applying obfuscation. Values show a ratio of metrics of an obfuscated and a non-obfuscated program.

and measuring the total number of CPU seconds that the process spent in user mode[15].

An interesting observation we have noticed after evaluating the results is that without the Bogus Control Flow pass, applying the passes in the order *String obfuscation – Opaque constants – Instructions substitution* resulted in better performance of the obfuscated program, than when this order has been reversed. We have tested the same combinations of passes (omitting the Bogus Control Flow pass) and the results were similar – worse performance in case of the reverse order. However, the metrics μ_1 and μ_2 have been slightly higher ($\sim 15\%$) for the programs obfuscated with the original ordering of the transformations. This leads us to a conclusion that the cost of a transformation does not always grow proportionally to its potency.

Another important finding from measuring the computation time is that the performance impact of all tested transformations is *constant* in regard to the input size, which can be seen in the Figures 2, 3 and 4.

[15]Using the Bash `time`[16] built-in utility

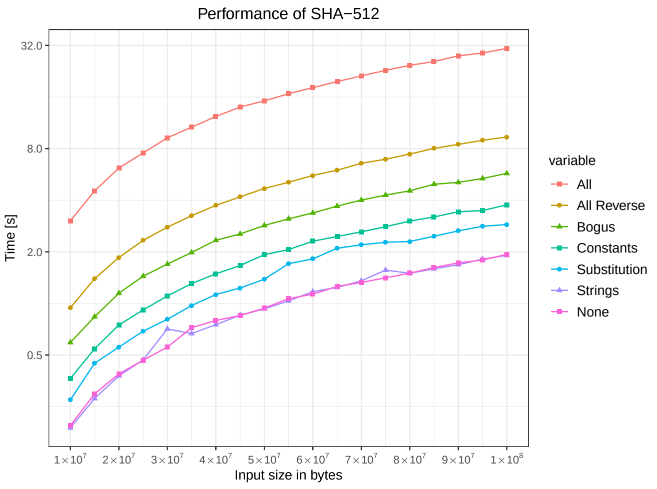


Figure 2: Performance of a program computing the SHA-512 hash function. The passes plotted as *All* and *All Reverse* have been applied in the order described in Section 4.3.

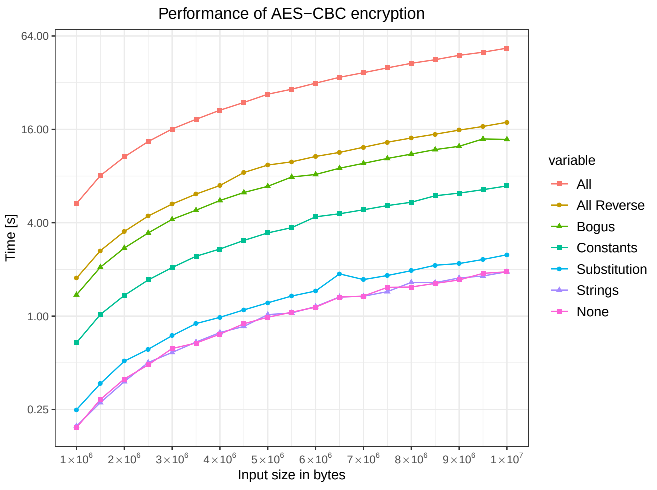


Figure 3: Performance of a program performing AES encryption with a 128-bit key in CBC mode. The passes plotted as *All* and *All Reverse* have been applied in the order described in Section 4.3.

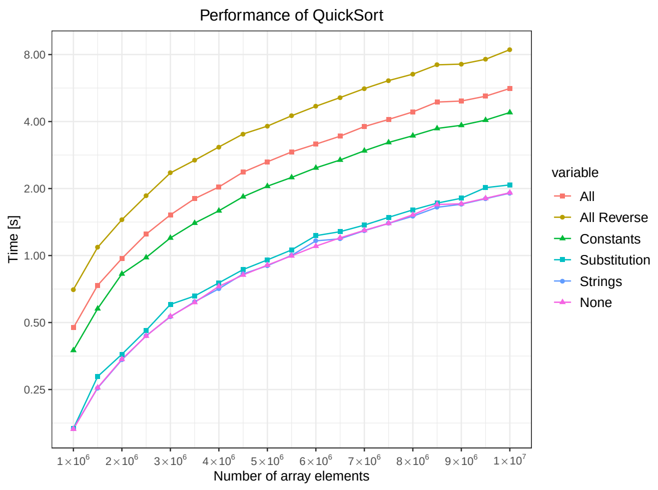


Figure 4: Performance of a QuickSort algorithm. The passes plotted as *All* and *All Reverse* have been applied in the order described in Section 4.3, except for Bogus Control Flow pass, which has been omitted.

5 Conclusion

In this work, we have explored obfuscation techniques that can be used to increase the security of compiled code against reverse engineering and other methods of analysis. Currently, a large amount of obfuscating transformations exists. We believe that the transformations described in this thesis are among the most used ones, as they (or their variants) can be found both in obfuscation tools we have analyzed, as well as in multiple research papers that deal with obfuscation or the countermeasures against it.

We conclude that the use of Mixed Boolean-Arithmetic expressions is relevant and has a big potential for designing obfuscating transformations. Regarding manipulation of the control flow of a program, the most important element is the design of resilient opaque predicates. Reverse engineers currently tend to leverage the capabilities of various tools based on symbolic execution and taint analysis, which aid them with understanding an obfuscated program. Therefore, the main focus of the research in the area of obfuscation is shifting towards countermeasures against such tools.

The results show that an increase in the size of the program (in terms of instructions count) does not always increase its perceived complexity[17], therefore measures to increase the code diversity, such as the use of randomness, are crucial. Based on the results, we also conclude that applying the transformations in a different order influences the resulting program in a significant way, i.e. different ordering of the passes results in different degrees of potency and performance impact. Since high potency does not always correlate with high performance overhead, we suggest that the ordering of the passes should always be based either on experimenting, or specific heuristics with respect to the nature of the transformations, to achieve optimal results. The cost of the implemented transformations has been proved to be constant – the size of the input increases the computation time of the obfuscated program proportionally to the increase with the non-obfuscated version.

We reckon that there exist multiple commercially available obfuscating tools that provide more advanced transformations than the ones described and implemented in this work. However, we believe that this thesis can serve as a good starting point, or a reference, for other researchers, developers, and reverse engineers interested in this area. Together with the implementation details, it can aid with developing and experimenting with new obfuscating transformations, and help to spark innovative ideas.

References

[1] Boaz Barak et al. „On the (Im)possibility of Obfuscating Programs“. In: *IACR Cryptology ePrint Archive* 2001 (Jan. 2001), p. 69. DOI: 10.1145/2160158.2160159.

[2] C. Collberg, C. Thomborson, and Douglas Low. „A Taxonomy of Obfuscating Transformations“. In: 1997.

[3] Christian Collberg, Clark Thomborson, and Douglas Low. „Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs“. In: 1998.

[4] Yongxin Zhou et al. „Information Hiding in Software with Mixed Boolean-Arithmetic Transforms“. In: 2007.

[17]In other words, how it complicates the task of reverse-engineering the program.

[5] Adrien Guinet, Ninon Eyrolles, and Marion Videau. „Arybo: Manipulation, Canonicalization and Identification of Mixed Boolean-Arithmetic Symbolic Expressions“. In: *GreHack 2016*. Proceedings of GreHack 2016. Grenoble, France, Nov. 2016. URL: `https://hal.archives-ouvertes.fr/hal-01390528`.

[6] Frederick B. Cohen. „Operating System Protection through Program Evolution“. In: *Comput. Secur.* 12.6 (Oct. 1993), pp. 565–584. ISSN: 0167-4048. DOI: `10.1016/0167-4048(93)90054-9`. URL: `https://doi.org/10.1016/0167-4048(93)90054-9`.

[7] Babak Yadegari et al. „A Generic Approach to Automatic Deobfuscation of Executable Code“. In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 674–691. DOI: `10.1109/SP.2015.47`.

[8] Aleieldin Salem and Sebastian Banescu. „Metadata Recovery from Obfuscated Programs Using Machine Learning“. In: 2016.

[9] Chenxi Wang et al. „Protection of software-based survivability mechanisms“. In: *2001 International Conference on Dependable Systems and Networks*. IEEE. 2001, pp. 193–202.

[10] William Landi. „Undecidability of Static Analysis“. In: 1.4 (Dec. 1992), pp. 323–337. ISSN: 1057-4514. DOI: `10.1145/161494.161501`. URL: `https://doi.org/10.1145/161494.161501`.

[11] Jan Cappaert and Bart Preneel. „A general model for hiding control flow“. In: *Proceedings of the tenth annual ACM workshop on Digital rights management*. 2010, pp. 35–42.

[12] Björn Johansson, Patrik Lantz, and Michael Liljenstam. „Lightweight dispatcher constructions for control flow flattening“. In: *Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop*. 2017, pp. 1–12.

[13] Rabih Mohsen and Alexandre Miranda Pinto. „Evaluating obfuscation security: A quantitative approach“. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9482 (2016), pp. 174–192. ISSN: 16113349. DOI: `10.1007/978-3-319-30303-1_11`.

[14] A N Kolmogorov. „On tables of random numbers“. In: *Theoretical Computer Science* 207 (1998), pp. 387–395.

[15] Pascal Junod et al. „Obfuscator-LLVM – Software Protection for the Masses“. In: *2015 IEEE/ACM 1st International Workshop on Software Protection*. 2015, pp. 3–9. DOI: `10.1109/SPRO.2015.10`.

[16] *Swiss startup protects SnapChat.* 2017. URL: `https://www.startupticker.ch/en/news/july-2017/swiss-startup-protects-snapchat`.

[17] Ninon Eyrolles. „Obfuscation with Mixed Boolean-Arithmetic Expressions : reconstruction, analysis and simplification tools“. PhD thesis. June 2017.

[18] Hui Xu et al. „Manufacturing Resilient Bi-Opaque Predicates Against Symbolic Execution“. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* 2018, pp. 666–677. DOI: `10.1109/DSN.2018.00073`.

Detection of Malicious Code in SSH programs

Ádám Ruman, Daniel Kouřil

Program analysis is crucial in digital forensics investigations and malware analysis processes. It is a challenging procedure, still relying on manual methods and the guidance of knowledgeable individuals. In order to streamline the analytic process, we have developed a set of techniques to identify malicious code in applications deployed on the examined system.

In this paper, we concentrate on analyzing SSH programs since they are ubiquitous tools enabling remote access to computer systems. SSH programs are also often the target of malicious actors who obtain illegitimate access to the system. Therefore, having a way to analyze SSH binaries quickly helps significantly when a potentially compromised system is being investigated.

Our solution is based on similarity comparisons of suspected malware samples to legitimate versions. It exploits the process' interaction with the operating system's API.

The API calls are structured into a graph. The structure depends on the time the call was made and the file descriptor it was associated with. Examples of such execution graphs are depicted in Fig. 1. They display a pair of a legitimate SSH client and its maliciously modified version. The visualizations are slightly skewed, but their closer examination shows an extra branch in the graph of the modified program, which refers to malicious code. The code is marked by the red line and depicts an exfiltration of information over the network.

Our work focuses on methods that enable automated analysis and evaluation of programs, describing the differences. The solution is composed of several partial areas.

To derive the information about system and library calls, we employ methods of dynamic program analysis. By running the program and monitoring its execution, we collect a list of calls and their attributes and order. Based on this information, the call graph structure is established.

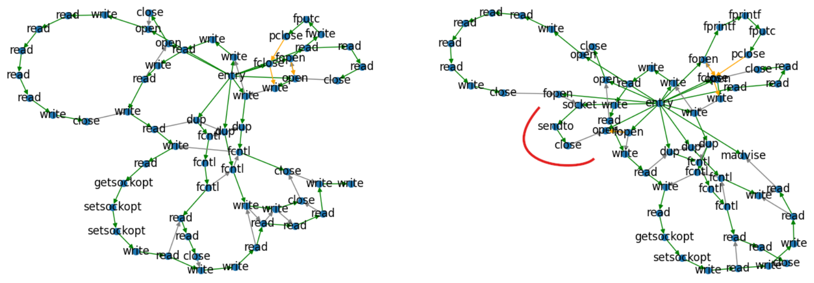


Figure 1: API call graphs for an SSH client and its modified version

We produced call graphs of a number of legitimate SSH programs from various Linux distributions. The outcome is a corpus used to analyze unknown SSH samples.

When a sample is analyzed, its call graph is established first. It is then compared to the legitimate programs in our collection. The closest match is found based on a well-established graph similarity algorithm (Graph Edit Distance) parametrized to our needs. The result is the identification of a legitimate SSH program that was most likely used as the basis when the malicious implementation was being prepared.

The second stage is to provide more detailed knowledge about the nature of the malicious code. The suspect graph and its closest match are processed using additional techniques to spot differences between the two graphs on a more fine-grained level. The differences are reported as another graph, suitable for further processing by a machine or to be presented to the analyst.

The designed methods were prototyped as a set of tools that implement all the described steps. The work has been conducted as part of the development of a diploma thesis [1] that provides detailed descriptions of all methods.

1 Evaluation of the comparison techniques

We have performed a set of experiments to validate the designed methods and to assess the efficiency of the tool on maliciously modified programs.

The results help us define an interpretation of the produced results and show us how conservative we have to be when assessing the similarity of two programs.

All the experiments were carried out in virtualized environments (Docker, VirtualBox, and VMWare), as we expect analysts not to use live systems for malware assessment. In these experiments, we used the password login setting. However, public keys and other methods like Kerberos could also be used. We also limited ourselves to the part of the SSH clients run, where user interaction happens.

1.1 Evaluation Over Legitimate Programs

For the purpose of assessing the (dis)similarity of legitimate SSH programs we collected the following set of implementations:

- Manually compiled OpenSSH binaries based on source code-base between versions 6.0 and 9.0.
- The latest OpenSSH binaries shipped by the following Linux distributions: Centos 7, Debian Stretch, Debian Buster, Debian Bullseye, Debian Bookworm, Ubuntu 16.04, Ubuntu 18.04.
- A binary of DropBear, an SSH implementation meant namely for IoT devices.

We compared all the above mentioned programs, the graphical result can be seen in Fig. 3 and 2. It is important to note that for these figures the score of the comparison is not yet interpreted in any way, and the color coding is just a simple 3-color uniform spread.

By analyzing the results of the comparisons over legitimate programs, we can make the following observations:

- The diagonal, representing comparisons of the programs with themselves, always has the score of 100 and GED 0, which is exactly the expected behavior.
- Most of the high-similarity groupings are located close to the diagonal, thus, they are made of closely related versions.
- The further away two versions are, the score tends to be smaller. The decrease, however, is not linear nor strict.

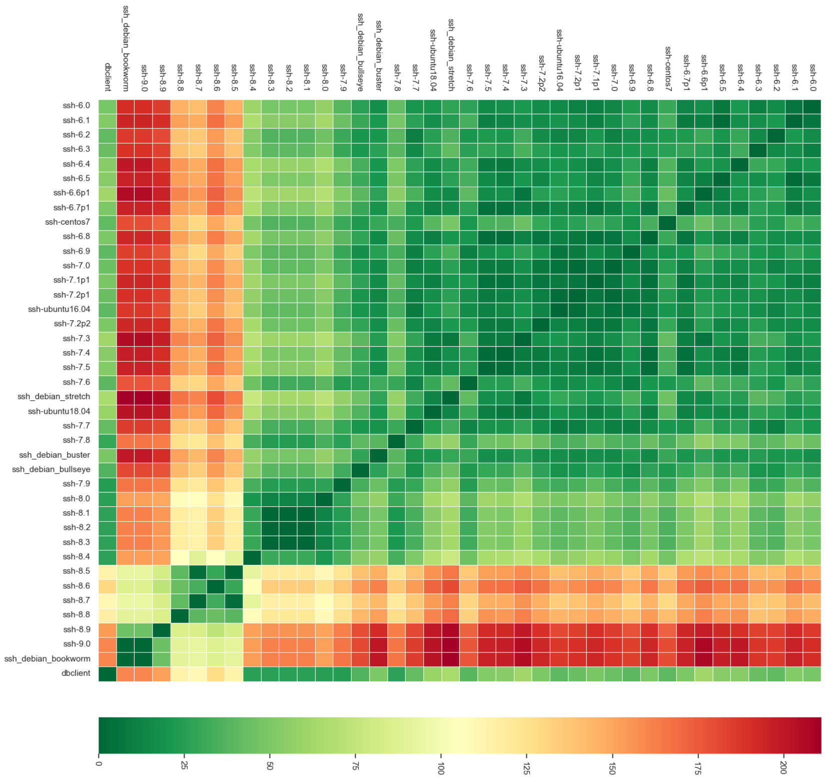


Figure 2: Comparison of legitimate SSH versions with Graph Edit Distance.

- Most changes in the OpenSSH binaries seem to happen rapidly, this is no surprise, as both our graph comparison module and GED is sensitive to new branches, and the manual analysis of the programs shows this is what usually happens at those "borders".
- We observed four groups within the collected versions with our similarity score, which seem to be closely related. These groups are defined by the aforementioned changes and are the following:
  - Versions 6.0 – 7.8 and all the distribution versions derived from them;
  - Versions 8.0 – 8.3;

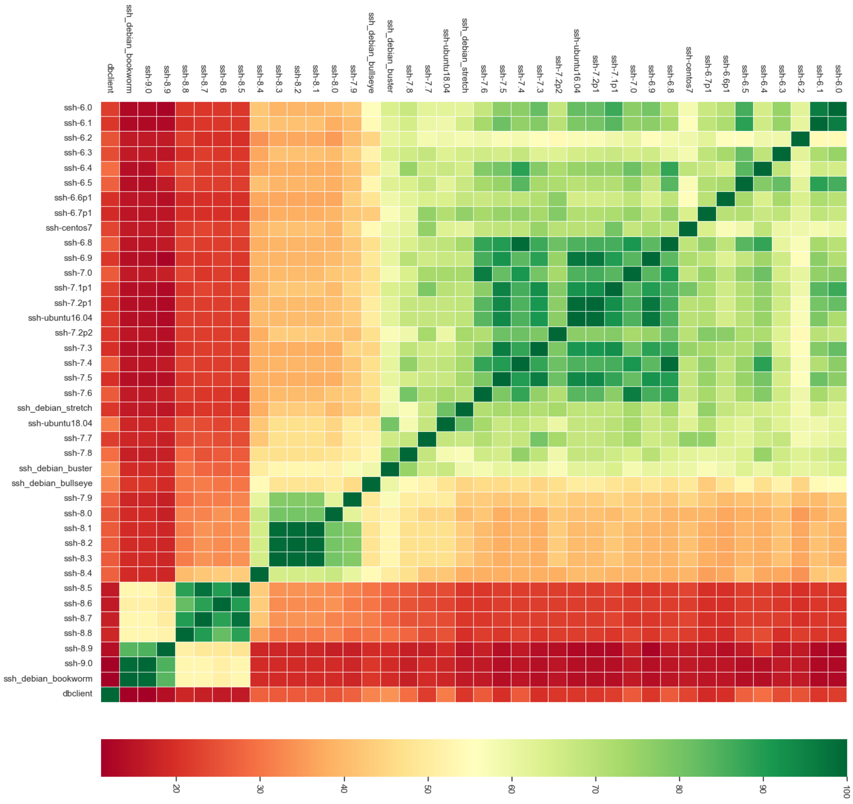


Figure 3: Comparison of legitimate SSH versions with our Similarity Score.

  - Versions 8.5 – 8.8;
  - Versions 8.9, 9.0 and the distribution version from Debian Bookworm.

- Some versions seem unique, while we can see the change in similarity with further-away versions, they are not particularly close to their neighbors. For example ssh-6.2, ssh-7.2p2 seems to be disjointed from their neighbors, this is something that needs further manual analysis. This is especially visible with our scoring system.

- A visible anomaly in Figure 3 is the green(ish) area in the top-left corner. Here, we can observe that the similarities between versions change unpredictably in a range of close versions. While this could be the results of some versions changing small functionality back-and-forth, it is still worth of further investigation.
- The DropBear client is unique, this shows us that exotic versions have to be taken care of, and as many of them as possible must be in our database for comparison for the best results.
- Comparing the GED and similarity score heatmaps, we see how the latter is more sensitive to small changes.

Further manual analysis of the mentioned outliers and anomalies shows that their trace files are very similar, except the more unique versions are missing the last couple of OS API calls. Having repeated the tracing process a few more times, we noticed that the missing parts were not consistently the same. By using other tools such as `ltrace`, we narrowed down the problem to `frida-trace`'s[1] behavior. It seems that in some cases, the end of the process makes the `frida` thread finish before it could log the OS API calls.

This inconsistency provides us a reason for other observed anomaly, where the similarity was not decaying with version distances uniformly, but rather fluctuating.

Score Interpretation Our experiments on legitimate programs give us an insight into how these programs behave and how much deviation can be considered "natural". With this know-how, we are able to define an interpretation for our scoring system.

What is left is to decide how conservative we want to be with our interpretation. While this is user-specific, we would advise being open to false positives rather than false negatives. This works well with the second result of our tool, which is a graph highlighting the differences, which can help an analyst make an educated choice more quickly. It is important to note that the score – even with an interpretation – can not tell us whether the differences are hazardous or harmless, but only how far they deviate from the expected behavior.

Our final interpretation for SSH clients is:

[1]`Frida` is the tool utilized to collect information on calls performed by the process

- Score **100 – 93** – this score range should be taken as an indicator of the programs being different only in very small implementation details, and thus considered a valid binary.
- Score **93 – 75** – this score range indicates either multiple implementation differences or small functional changes. While we should not totally rule out malicious modifications (as they might be really small changes), the probability of it is rather small. This score can also be achieved if the exact matching program is not present in the database. We recommend at least taking a look on the highlighted differences, and checking whether the found closest program is the same as we should have on the system analysed (administrators either have this information, but it can also be gathered with static analysis or from network traffic).
- Score **75 – 0** – this score range indicates that the suspect binary is with high probability modified. Further analysis should be made.

1.2 Evaluation Over Malicious Programs

Armed with the knowledge of the nature of legitimate SSH programs and versions, and an interpretation scheme for the resulting scores of our tool, we are able to test it on malicious binaries. For this we use a collection of maliciously modified binaries that were found during investigation of real incidents [2]. The collection consists of several families of trojanized SSH binaries. A family consists of either multiple base versions of SSH modified with the same malicious functionality, or the same version for different architectures.

There are some samples which are not usable anymore, as the libraries they are linked to are not available (usually pre OpenSSH v6.0 programs linked to ancient OpenSSL libraries). Some others can be run but crash upon their start, but the most interesting problem we ran into was `frida-trace` not being able to collect some API call arguments for some of the samples (for both x64 and Intel 80386 architectures).

Ultimately, 13 samples could be run and analyzed. The results of the analysis can be seen in Table 1. For this experiments, we took the two closest programs (with one exception – more about that in our discussion of results), not just the best match, to see whether there is any relation between the Graph Edit Distance and our Similarity Score. Let us discuss our findings and observations.

Malware Sample	Closest Legitimates	GED	Similarity
Akiva_Client_2	ssh_debian_stretch	82	48.634
	ssh-6.6	86	48.628
Atollon_Client_2	ssh-7.6	17	62.050
	ssh-7.9	21	44.252
Bespin_Client_2	ssh-6.3	19	67.374
	ssh_debian_bullseye	20	52.778
Crait_Client_2	ssh-6.0	13	57.835
	ssh-6.1	13	61.212
	ssh-7.1	13	53.907
Chandrila_Client_2	ssh-9.0	46	30.849
	ssh_debian_bookworm	46	30.819
Endor_Client	ssh_debian_stretch	90	35.493
	ssh-7.3	96	35.573
Endor_Client_5	ssh_debian_stretch	90	36.293
	ssh-7.3	96	34.373
Mimban_Client_2	ssh-6.4	12	53.343
	ssh-6.7	14	45.828
Mimban_Client_3	ssh-6.4	12	48.667
	ssh-6.0	20	48.977
Onderon_Client_2	ssh-7.1	10	45.331
	ssh-6.8	13	48.102
PolisMassa_Client	ssh-6.4	12	46.937
	ssh-6.7	16	37.891
Ebury_Injected_Client	ssh-6.8	20	54.214
	ssh-ubuntu16.04	21	73.584
	ssh-ubuntu18.04	–	68.582

Table 1: Experiment results on malicious SSH clients.

The first important thing to note is that GED is not reversely proportional to the similarity score in the general setting. A bigger GED does between programs A and B than between programs C and D does not rule out that A and B will be more similar. We must take into account the size of these programs (or rather, the size of the graph mined from them). However in the local setting (comparing A to B and then A to C) the reverse proportion usually holds (but it is not a rule, as GED is not as customizable as we would like it to be).

We may also observe by combining the results from legitimate comparison programs that there is no transitive relations between the closest programs based on GED. This means that for the two closest programs to our suspect A and B, it does not need to hold that the closest program to A based on GED is B. This seems to be different with our similarity score where this transitive relation is tighter.

All the malicious programs are "caught" with our method, and the defined interpretation of the score. We can observe that some of the malware comes close to the similarity score limit (75) while others are far down in score. By further manual analysis we can confirm that this deviation is the result of how much and how "blatant" the malicious activity is.

One of the most interesting malware samples is the *Ebury_ Injected_ Client*, which is unique, as the malicious code is not in the SSH binary itself, but is injected into a library which is loaded. For this sample again, we used the two best matches from the search module, but here the similarity score difference is both reversed and much higher than we would expect from the GED values. It is also important to note that the original SSH binary we used was of Ubuntu 18.04, which is not the best match neither based on GED or the similarity score.

Overall the experiments can be viewed as a success. All the malicious programs we were able to get into working shape were correctly flagged. On the other side, the cardinality of the samples is not too high, and thus further trials could potentially uncover some cases that would be missed by the tool in its actual state. The experiments also show that we could combine multiple layers of information that can be collected for more optimal results. Our choice of the tracing tool might not have been the best for some use-cases. Fortunately, alternatives are quite straightforward to integrate with our tools.

2 Conclusion

Efficient ways for analysis of binary artifacts are crucial for any analyst investigating a security incident. Using the methods described in the paper, it is possible to perform analysis by automated workflow without the involvement of an expert. The produced results can be examined by an investigator who does not need specific knowledge of binary analysis. This way, security teams can integrate quick verifications of SSH programs during the initial investigation of an incident.

While the methods were developed primarily for SSH programs, the principles employed are generic, and the mechanisms can also be used to analyze other programs.

References

[1] Ádám Ruman. *Detection of Malicious Patterns in SSH Programs.* Diploma Thesis. Faculty of Informatics, Masaryk University. 2022.

[2] Romain Dumont, Marc-Etienne M.Léveillé, and Hugo Porcher. *The Dark Side of the ForSSHe.* URL: `https://www.welivesecurity.com/wp-content/uploads/2018/12/ESET-The_Dark_Side_of_the_ForSSHe.pdf` (visited on 03/27/2022).

Creation and detection of malicious synthetic media – a preliminary survey on deepfakes

Anton Firc, Kamil Malinka, Petr Hanáček

E-mail: ifirc@fit.vut.cz, malinka@fit.vut.cz, hanacek@fit.vut.cz

Abstract

Deepfakes present an emerging threat in cyberspace. Recent developments in machine learning make deepfakes highly believable and very difficult to differentiate between what is real and what is fake. Not only humans but also machines struggle to identify deepfakes. Current biometrics systems might be easily fooled by carefully prepared malicious synthetic media – deepfakes. We provide an overview of deepfake creation and detection methods for selected visual and audio domains. We put specific emphasize on the open-source solutions. We discuss both facial and speech deepfakes, and for each domain, we define deepfake categories and differences between them and discuss available detection methods. For each deepfake category, we provide an overview of available open-source tools for their creation and datasets.

Key words: face deepfakes, voice deepfakes, biometrics systems, deepfake detection, cybersecurity

1 Introduction

Deepfake is a term that denotes a subset of synthetic media. The term itself is just a combination of words *deep learning* and *fake*. Deepfakes are created using deep neural networks, and they depict events that never happened in order to entertain, defame individuals, spread fake news, and others [1].

The constant advancements in machine learning make deepfake creation available for a broader spectrum of computer users. The most simple tools even feature a graphical user interface that lets inexperienced users create deepfakes [2, 3, 4, 5]. This fact urges the investigation of what threats and impacts deepfakes might have on cyberspace, as society is still not sure. However, it seems that deepfakes only pose a threat to individuals, not to nations or the whole world [6].

One of the sub-areas of harmful deepfake usage is spoofing biometrics systems [7, 8]. Both voice recognition and facial recognition systems are prone to be spoofed by synthetic media. Moreover, not every deepfake type might have the potential to be used against biometrics systems.

In this work, we systematically overview the deepfake area from the security point of view. The style and content of this survey aim to provide a unified overview of the deepfake problematics to security-related researchers and developers. In order to perform a security analysis focused on deepfake resilience of biometrics authentication systems, it is necessary to have orientation in all deepfake types, their properties, means for their creations, and methods for their detection.

In our survey, we focus on deepfakes in facial and speech domains. We divide facial deepfakes into categories according to the level of manipulation needed, and as an extra category, we incorporate face morphing. We divide voice deepfakes into categories according to the voice transfer technology. For each category, tools and datasets are provided. Deepfake detection methods are discussed as a whole for each domain.

Due to a high number of existing works, it is often hard to connect current research results with existing tools implementing proposed methods. We do not aim to provide extensive details on used technologies for creation and detection. Thus, we rather provide a general overview of all the concerned areas for the reader to gain an initial insight of deepfake creation and detection.

The main contributions of this article might be summarized as follows:

- We provide an overview of deepfake creation tools. We also connect them with the relevant research results, as they usually come from different authors.
- We provide a comprehensive overview of the basics of deepfake detection.
- We provide a united taxonomy for facial and speech deepfakes and define differences between each category. The facial deepfakes are categorized according to the level of manipulation needed, and the voice deepfakes according to the voice transfer technology.

The face deepfakes are discussed in Section 2. The voice deepfakes are discussed in Section 3. Section 4 summarizes all of the stated knowledge.

2 Face deepfakes

This section discusses available face deepfake creation tool, datasets and detection methods. We define an united taxonomy of facial deepfakes according to the level of manipulation needed. We provide a general overview of the available open-source tools and publicly accessible datasets for each category. Finally, we provide an overview of detection methods.

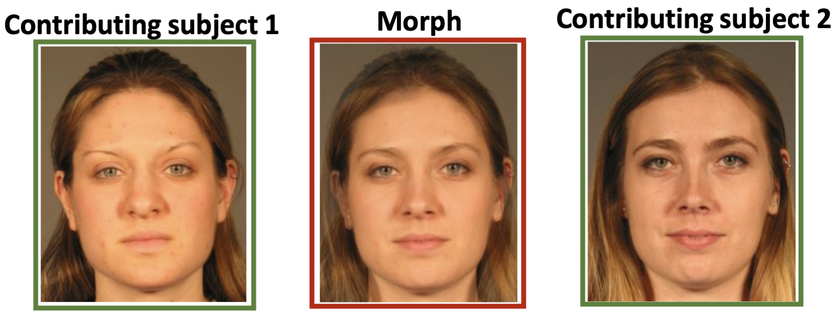
2.1 Categories

We divide face deepfakes into five categories depending on the level of manipulation needed to create such a deepfake. From the highest level of manipulation to the lowest: *face synthesis, face morphing, face swap, face reenactment* and *face manipulation.* Examples of selected categories are shown in Figure 1.

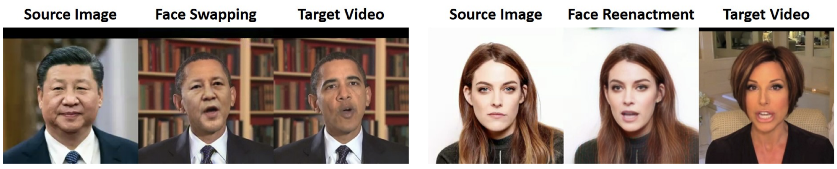
Face synthesis

We define *face synthesis* as a process of synthesizing non-existing faces based on learned high-level attributes, such as pose or identity [9]. There are different applications of face synthesis. They range from synthesizing virtual characters in the film industry to providing a human-looking representation of computer agents to interact with its users. Moreover, face synthesis might be beneficial for face recognition applications to generate needed amounts of training data [12].

Face synthesized using StyleGAN2 model [9]. Image retrieved from https://thispersondoesnotexist.com.



An example of face morphing [10].



Face swapping on the left and face reenactment on the right [11].

Figure 1: Examples of selected facial deepfake categories.

Face morphing

Morphing is a special effect in motion pictures or animations that changes one image into another using a seamless transition. Morphing is often used to depict one person turning into another [13]. For the scope of this work, we understand morphing as a method to produce a facial image that is very similar to the face of one subject but also contains facial features of the second subject.

Face swap

Face swapping refers to a technique where a face from *source* photo is transferred onto a face in a *target* photo. The result is desired to look realistic and unedited. For the scope of this work, we only refer to the one-to-one face-swapping paradigm.

Facial reenactment

Facial reenactment is a process of photo-realistic facial re-animation of a target video with expressions of a source actor. This method was formerly proposed to provide the missing visual channel used in a scenario of a digital assistant. It is essential that these methods not only generate audio and visual information, but the audio has to be synced to the motions of generated human visual [14].

Face manipulation

Face manipulation is a technique used to modify a specific part of a target's face in an image or video. The identity of the target remains unchanged. The attacker is either able to add or remove features like facial hair, glasses, and others, or to change/transfer the expressions, lighting or pose of the head [15].

2.2 Tools

An overview of tools for face deepfake creation is provided in Table 1. This list is not exhaustive and focuses solely on the tools available as open-source projects.

Name	Link	Face synthesis	Face morphing	Face swap	Facial reenactment	Face manipulation
StyleGAN3 [16]	https://github.com/NVlabs/stylegan3	✓	✗	✗	✗	✗
ProGAN [17]	https://github.com/akanimax/pro_gan_pytorch	✓	✗	✗	✗	✗
AdvFaces [18]	https://github.com/ronny3050/AdvFaces	✓	✗	✗	✗	✗
MMGeneration [19]	https://github.com/open-mmlab/mmgeneration	✓	✗	✗	✗	✗
Face-Morphing	https://github.com/Azmarie/Face-Morphing	✗	✓	✗	✗	✗
Face Morphing	https://github.com/cirbuk/face-morphing	✗	✓	✗	✗	✗
DeepFaceLab [20]	https://github.com/iperov/DeepFaceLab	✗	✗	✓	✗	✗
Realistic-Neural-Talking-Head-Models [21]	https://github.com/vincent-thevenin//Realistic-Neural-Talking-Head-Models	✗	✗	✓	✗	✗
SimSwap [22]	https://github.com/neuralchen/SimSwap	✗	✗	✓	✗	✗
FaceShifter [23]	https://github.com/mindslab-ai/faceshifter	✗	✗	✓	✗	✗
FSGAN [11]	https://github.com/YuvalNirkin/fsgan	✗	✗	✓	✗	✗
Deepfakes	https://github.com/deepfakes/faceswap	✗	✗	✓	✗	✗
NeuralVoicePuppetry [14]	https://github.com/miu200521358/NeuralVoicePuppetryMMD	✗	✗	✗	✓	✗
Face2Face [24]	https://github.com/datitran/face2face-demo	✗	✗	✗	✓	✗
You said that? [25]	https://github.com/joonson/yousaidthat	✗	✗	✗	✓	✗
First Order Model [26]	https://github.com/AliaksandrSiarohin/first-order-model	✗	✗	✗	✓	✗
Articulated Animation [27]	https://github.com/snap-research/articulated-animation	✗	✗	✗	✗	✗
InterFaceGan [28, 29]	https://github.com/genforce/interfacegan	✗	✗	✗	✗	✓
SkinDeep	https://github.com/vijishmadhavan/SkinDeep	✗	✗	✗	✗	✓
StyleMapGAN [30]	https://github.com/naver-ai/StyleMapGAN	✗	✗	✗	✗	✓
GAIA [31]	https://github.com/timsainb/GAIA	✗	✗	✗	✗	✓
ELEGANT [32]	https://github.com/Prinsphield/ELEGANT	✗	✗	✗	✗	✓

Table 1: Open-source tools for face deepfake creation. Each line represents a different tool with link to corresponding repository and an indication of what category of deepfakes does the tool create.

Table 2: Face deepfakes datasets. Each column represents dataset counting data of one category of facial deepfakes. Some of the datasets appear in multiple columns, as they contain deepfakes from more categories.

Face synthesis	Face Morphing	Face swap
non-curated images[1]	Kramer et al. [33]	WildDeepfake [34]
PGGAN [17]	Raja et al. [35]	VideoForensicsHQ [36]
iFakeFaceDB [37]		FaceForensics++[38]
TPDNE [39]		DFDC [40]
generated.photos [41]		Celeb-DF [42]
		DeeperForensics-1.0 [43]

Face reenactment	Face manipulation
FaceForensics[44]	Zhou et al. [45]
	Dang et al. [46]
	FaceForensics++[38]

[1]https://drive.google.com/drive/folders/1j6uZ_a6zci0HyKZdpDq9kSa8VihtEPCp

2.3 Datasets

The available datasets are shown in Table 2.

2.4 Detecting facial deepfakes

There are various feasible approaches for deepfake detection. Each of the previously stated categories has its own methods and approaches suited for detection. However, these methods and approaches are not exclusive, and some of them are suitable to detect more types of facial deepfakes. Rather than discussing the nuances between the detection of different categories, we provide a comprehensive overview of basic principles used for deepfake detection across all of the stated categories. We thus divided the detection methods into four categories: *artifact-based*, *deep learning-based*, *spatiotemporal-based.*

Artifact-based detection methods utilize the specific features and patterns generated by GANs in the images or videos. The media (image or video) are firstly pre-processed, to extract the desired features, then these features are most commonly fed into a neural network that makes the final decision about the realness of the provided media [47, 48, 49, 50, 51, 52, 53, 54, 45, 46, 55].

Deep learning-based detection completely relies on the abilities of deep neural networks. Often, complex convolutional neural networks are used. The network extracts self-learned features and classifies the image or video as real or fake [56, 57, 58, 59, 60, 61].

Spatiotemporal-based methods mostly exploit the physical and physiological signals. As the name suggests, temporal information is needed; thus, these methods are only suitable for videos. The physical and physiological signals are not well captured in deepfake videos and may include spontaneous and involuntary physiological activities such as breathing, pulse, eye movement, or eye blinking. As these signals are often overlooked in the process of deepfake videos creation, they are suitable to be used as indicators for detection [62, 63, 64, 65, 66, 67, 68, 69, 70].

The list of stated detection methods and approaches is not exhaustive. Additionally, the stated categories are not exclusive, as we, for example, see deep-learning-based approaches exploiting artifacts. This section thus gives the reader a brief introduction to face deepfake detection.

3 Speech deepfakes

This section discusses available speech deepfake creation tools, datasets and detection methods. Again, we define a united taxonomy according to the voice transfer technology. Firstly, categories of speech deepfakes are defined, then an overview of open-source tools and available tools is provided. Finally, we discuss the basics of deepfake speech detection.

3.1 Categories

There are two main methods for creating deepfake speech: text-to-speech synthesis (TTS) and voice conversion (VC) [71]. The main difference is in the input data. TTS, as the name suggests, consumes written text as input and produces synthesized speech that sounds like a particular individual. VC, on the other hand, consumes a source voice saying desired phrase and a target voice and outputs the source phrase spoken by the target voice [72].

Text-to-speech synthesis

Text-to-speech (TTS) synthesis is a process of generating speech from written text [73]. The goal of this process is to synthesize speech that is not only easily understandable but also indistinguishable from the speech spoken by humans [74]. This technique finds a place in providing computer-human interfaces for smart assistants or navigation systems.

Voice conversion

Voice conversion is a technique used for modifying a given speech from a source speaker to match the vocal qualities of a target speaker [75, 76]. In contrast to TTS, this process is independent of the spoken content thus does not require transcriptions. Some of the most advanced voice conversion frameworks can separately transfer components of speech such as timbre, pitch, or rhythm [77].

Speech morphing

The term *speech morphing* refers to a technique of smooth transformation from one signal to another. This combination creates a new signal with an intermediate timbre [78]. The signals should be sufficiently similar to become reasonably aligned and interpolated into the new signal [79]. Simply said, it is an combination of two voices to create an intermediate one [75].

3.2 Tools

Table 3 provides an overview of open-source speech deepfake creation tools. Currently, no tools are providing the speech morphing functionality solely, we thus provide tools that are the most relevant to the speech morphing.

3.3 Datasets

The overview of deepfake speech datasets is shown in Table 4. As stated previously in Section 3.2, the research regarding speech morphing is not active. Thus, no dataset contains morphed speech.

Table 3: Open-source tools for speech deepfake creation. Each line represents a different tool with link to corresponding repository and an indication of what category of deepfakes does the tool create.

Tool	Link	TTS	VC	Speech morphing
MozillaTTS	https://github.com/mozilla/TTS	✓	✗	✗
Real-Time-Voice-Cloning [80]	https://github.com/CorentinJ/Real-Time-Voice-Cloning	✓	✗	✗
CoquiTTS	https://github.com/coqui-ai/TTS	✓	✗	✗
TensorFlowTTS	https://github.com/TensorSpeech/TensorflowTTS	✓	✗	✗
TransformerTTS	https://github.com/as-ideas/TransformerTTS	✓	✗	✗
Flowtron [81]	https://github.com/NVIDIA/flowtron	✓	✗	✗
Emotional TTS [82]	https://github.com/Emotional-Text-to-Speech/dl-for-emo-tts	✓	✗	✗
YourTTS [83]	https://github.com/edresson/yourtts	✓	✓	✗
SpeechSplit [77]	https://github.com/auspicious3000/SpeechSplit	✗	✓	✗
FragmentVC [84]	https://github.com/yistLin/FragmentVC	✗	✓	✗
AdaptiveVC [85]	https://github.com/jjery2243542/adaptive_voice_conversion	✗	✓	✗
Sprocket [86]	https://github.com/k2kobayashi/sprocket	✗	✓	✗
StarGAN [87]	https://github.com/hujinsen/StarGAN-Voice-Conversion	✗	✓	✗
MaskCycleGAN-VC [88]	https://github.com/GANtastic3/MaskCycleGAN-VC	✗	✓	✗
Figaro	https://github.com/symphonly/figaro	✗	✗	✓
VoiceMorphing	https://github.com/nestyme/voice-morphing	✗	✗	✓
PyVoiceChanger [76]	https://github.com/juancarlospaco/pyvoicechanger	✗	✗	✓

Table 4: Speech deepfakes datasets. Each row represents a different dataset with indication of deepfakes of what category does it contain.

Name	TTS	VC
ASVspoof 2019 [89]	✓	✓
ASVspoof 2021 [90]	✓	✓
WaveFake [91]	✓	✗
FoR [92]	✓	✗
SYNSPEECHDDB [93]	✓	✓
FMFCC-A [94]	✓	✓

3.4 Detecting deepfake speech

This section provides a more general overview of deepfake speech detection to introduce the reader into the basics of this problem rather than an exhaustive list of detection methods and corresponding publications.

The former attempts in deepfake speech detection frequently relied on Hidden Markov Models (HMMs) or Gaussian Mixture Models (GMMs) [95]. Since then, deepfake creation technologies have advanced rapidly. This led to the need for more reliable detection mechanisms. As a response, many works started to utilize low-level spectro-temporal features that are being fed into various classifiers [96, 97, 98, 95].

Latest approaches in contrast rely on deep learning, to extract the best features [99, 100, 101]. Some of the methods even include data augmentation to improve the detection robustness [102, 103, 104, 105, 106, 107].

In addition to the detection methods utilizing the speech signal or some of its forms, Remaio and Tzerpos [108] or Khochare et al. [109] propose detection of deepfake speech using an image-based approach. Additionally, Wang et al. [110] propose a framework for a neural network-based speaker recognition system that monitors the activations of neurons in different network layers.

An analysis of the ASVspoof 2019 challenge revealed that the majority of the deepfake detection systems are based on deep neural networks. The best performance is obtained by combining score levels of single systems that vary by input features or training procedure. For further development of detection methods, it seems beneficial to use mixup techniques and FIR filters for coded magnitude response emulation [111].

4 Conclusions

As literature review shows, the deepfake creation tools are still on the rise, and the development of detection methods tries to hold onto this trend. The role of open-source tools in areas of deepfake creation and detection is significant. As we demonstrate in this paper, extensive amount of creation tools is provided as open-source tools. These tools often feature a community of experts and enthusiasts that continuously improve them. Moreover, the deepfake detection tools exist almost exclusively as public GitHub repositories.

Acknowledgments

This work was supported by the internal project of Brno University of Technology (FIT-S-20-6427).

References

[1] Jon Bateman. *Deepfakes and Synthetic Media in the Financial System: Assessing Threat Scenarios*. Tech. rep. Carnegie Endowment for International Peace, 2020, pp. i–ii. URL: `http://www.jstor.org/stable/resrep25783.1`.

[2] Descript. *Overdub*. online. 2021. URL: `https://www.descript.com/overdub`.

[3] Resmble AI. *Resemble AI webpage*. online. 2020. URL: `https://www.resemble.ai`.

[4] *Online Deepfake Maker*. 2022. URL: `https://deepfakesweb.com`.

[5] *Reface: be anyone and reface anything*. 2022. URL: `https://hey.reface.ai`.

[6] Valencia A. Jones. „Artificial Intelligence Enabled - Deepfake technology The Emerge of a New Threat“. Master thesis. Utica College, 2020.

[7] Anton Firc. „Applicability of Deepfakes in the Field of Cyber Security“. Supervisor Mgr. Kamil Malinka, Ph.D. Master's thesis. Brno: Brno University of Technology, Faculty of Information Technology, 2021.

[8] John Seymour and Azeem Aqil. *Your Voice is My Passport*. 2018. URL: `https://www.blackhat.com/us-18/briefings/schedule/%5C#your-voice-is-my-passport-11395`.

[9] Tero Karras et al. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: `1912.04958 [cs.CV]`.

[10] Sushma Venkatesh et al. *Face Morphing Attack Generation & Detection: A Comprehensive Survey*. 2020. arXiv: `2011.02045 [cs.CV]`.

[11] Yuval Nirkin, Yosi Keller, and Tal Hassner. „FSGAN: Subject agnostic face swapping and reenactment“. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 7184–7193.

[12] Yang Wang, Zicheng Liu, and Baining Guo. „Face Synthesis“. In: *Handbook of Face Recognition*. London: Springer London, 2011, pp. 521–547. ISBN: 978-0-85729-932-1. DOI: `10.1007/978-0-85729-932-1_20`. URL: `https://doi.org/10.1007/978-0-85729-932-1_20`.

[13] Matteo Ferrara, Annalisa Franco, and Davide Maltoni. „The magic passport“. In: *IEEE International Joint Conference on Biometrics*. 2014, pp. 1–7. DOI: `10.1109/BTAS.2014.6996240`.

[14] Justus Thies et al. *Neural Voice Puppetry: Audio-driven Facial Reenactment*. 2020. arXiv: `1912.05566 [cs.CV]`.

[15] Mariëtte van Huijstee et al. *Tackling deepfakes in European policy*. 2021. DOI: `10.2861/325063`. URL: `https://www.europarl.europa.eu/stoa/en/document/EPRS_STU(2021)690039`.

[16] Tero Karras et al. „Alias-Free Generative Adversarial Networks“. In: *Proc. NeurIPS*. 2021.

[17] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: `1710.10196 [cs.NE]`.

[18] Debayan Deb, Jianbang Zhang, and Anil K. Jain. *AdvFaces: Adversarial Face Synthesis*. 2019. arXiv: `1908.05008 [cs.CV]`.

[19] MMGeneration Contributors. *MMGeneration: OpenMMLab Generative Model Toolbox and Benchmark*. `https://github.com/open-mmlab/mmgeneration`. 2021.

[20] Ivan Perov et al. *DeepFaceLab: Integrated, flexible and extensible face-swapping framework*. 2021. arXiv: `2005.05535 [cs.CV]`.

[21] Egor Zakharov et al. *Few-Shot Adversarial Learning of Realistic Neural Talking Head Models*. 2019. arXiv: `1905.08233 [cs.CV]`.

[22] Renwang Chen et al. „SimSwap: An Efficient Framework For High Fidelity Face Swapping“. In: *MM ’20: The 28th ACM International Conference on Multimedia*. ACM, 2020, pp. 2003–2011. DOI: `10.1145/3394171.3413630`. URL: `https://doi.org/10.1145/3394171.3413630`.

[23] Lingzhi Li et al. „Faceshifter: Towards high fidelity and occlusion aware face swapping". In: *arXiv preprint arXiv:1912.13457* (2019).

[24] Justus Thies et al. *Face2Face: Real-time Face Capture and Reenactment of RGB Videos*. 2020. arXiv: 2007.14808 [cs.CV].

[25] J. S. Chung, A. Jamaludin, and A. Zisserman. „You said that?" In: *British Machine Vision Conference*. 2017.

[26] Aliaksandr Siarohin et al. „First Order Motion Model for Image Animation". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/31c0b36aef265d9221af80872ceb62f9-Paper.pdf.

[27] Aliaksandr Siarohin et al. „Motion Representations for Articulated Animation". In: *CVPR*. 2021.

[28] Yujun Shen et al. „Interpreting the Latent Space of GANs for Semantic Face Editing". In: *CVPR*. 2020.

[29] Yujun Shen et al. „InterFaceGAN: Interpreting the Disentangled Face Representation Learned by GANs". In: *TPAMI* (2020).

[30] Hyunsu Kim et al. *Exploiting Spatial Dimensions of Latent in GAN for Real-time Image Editing*. 2021. arXiv: 2104.14754 [cs.CV].

[31] Tim Sainburg et al. *Generative adversarial interpolative autoencoding: adversarial training on latent space interpolations encourage convex latent distributions*. 2019. arXiv: 1807.06650 [cs.LG].

[32] Taihong Xiao, Jiapeng Hong, and Jinwen Ma. *ELEGANT: Exchanging Latent Encodings with GAN for Transferring Multiple Face Attributes*. 2018. arXiv: 1803.10562 [cs.CV].

[33] Robin S. S. Kramer et al. „Face morphing attacks: Investigating detection with humans and computers". In: (2019). DOI: https://doi.org/10.1186/s41235-019-0181-4. URL: https://cognitiveresearchjournal.springeropen.com/articles/10.1186/s41235-019-0181-4#citeas.

[34] Bojia Zi et al. *WildDeepfake: A Challenging Real-World Dataset for Deepfake Detection*. 2021. arXiv: 2101.01456 [cs.CV].

[35] Kiran Raja et al. *Morphing Attack Detection – Database, Evaluation Platform and Benchmarking*. 2020. arXiv: 2006.06458 [cs.CV].

[36] Gereon Fox et al. „VideoForensicsHQ: Detecting High-quality Manipulated Face Videos". In: *IEEE International Conference on Multimedia and Expo (ICME 2021)*. Shenzhen, China (Virtual): IEEE, 2021. ISBN: 978-1-6654-3864-3. DOI: 10.1109/ICME51207.2021.9428101.

[37] João C. Neves et al. *GANprintR: Improved Fakes and Evaluation of the State-of-the-Art in Face Manipulation Detection*. 2019. eprint: `arXiv:1911.05351`.

[38] A. Rössler et al. „FaceForensics++: Learning to Detect Manipulated Facial Images“. In: *International Conference on Computer Vision (ICCV)*. 2019.

[39] *TPDNE dataset*. online. 2021. URL: `https://www.kaggle.com/potatohd404/tpdne-60k-128x128/version/2`.

[40] Brian Dolhansky et al. *The DeepFake Detection Challenge Dataset*. 2020. arXiv: `2006.07397 [cs.CV]`.

[41] *Faces Generated by AI dataset*. online. 2021. URL: `https://generated.photos/datasets`.

[42] Yuezun Li et al. „Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics“. In: *IEEE Conference on Computer Vision and Patten Recognition (CVPR)*. 2020.

[43] Liming Jiang et al. *DeeperForensics-1.0: A Large-Scale Dataset for Real-World Face Forgery Detection*. 2020. arXiv: `2001.03024 [cs.CV]`.

[44] Andreas Rössler et al. *FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces*. 2018. arXiv: `1803.09179 [cs.CV]`.

[45] Peng Zhou et al. *Two-Stream Neural Networks for Tampered Face Detection*. 2018. arXiv: `1803.11276 [cs.CV]`.

[46] Hao Dang et al. „On the detection of digital face manipulation“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern recognition*. 2020, pp. 5781–5790.

[47] Chih-Chung Hsu, Chia-Yen Lee, and Yi-Xiu Zhuang. *Learning to Detect Fake Face Images in the Wild*. 2018. arXiv: `1809.08754 [cs.MM]`.

[48] Francesco Marra et al. „Detection of GAN-Generated Fake Images over Social Networks“. In: *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. 2018, pp. 384–389. DOI: `10.1109/MIPR.2018.00084`.

[49] Belhassen Bayar and Matthew C. Stamm. „A Deep Learning Approach to Universal Image Manipulation Detection Using a New Convolutional Layer“. In: *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*. IH&MMSec ’16. Vigo, Galicia, Spain: Association for Computing Machinery, 2016, pp. 5–10. ISBN: 9781450342902. DOI: `10.1145/2909827.2930786`. URL: `https://doi.org/10.1145/2909827.2930786`.

[50] Yuezun Li and Siwei Lyu. *Exposing DeepFake Videos By Detecting Face Warping Artifacts*. 2019. arXiv: 1811.00656 [cs.CV].

[51] Nicolas Beuve, Wassim Hamidouche, and Olivier Deforges. „DmyT: Dummy Triplet Loss for Deepfake Detection“. In: ADGD ’21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 17–24. ISBN: 9781450386821. DOI: 10.1145/3476099.3484316. URL: https://doi.org/10.1145/3476099.3484316.

[52] Oliver Giudice, Luca Guarnera, and Sebastiano Battiato. „Fighting Deepfakes by Detecting GAN DCT Anomalies“. In: *Journal of Imaging* 7.8 (July 2021), p. 128. ISSN: 2313-433X. DOI: 10.3390/jimaging7080128. URL: http://dx.doi.org/10.3390/jimaging7080128.

[53] Jun Jiang et al. „Practical Face Swapping Detection Based on Identity Spatial Constraints“. In: *2021 IEEE International Joint Conference on Biometrics (IJCB)*. 2021, pp. 1–8. DOI: 10.1109/IJCB52358.2021.9484396.

[54] Zhiqing Guo et al. *Fake face detection via adaptive manipulation traces extraction network*. 2020. arXiv: 2005.04945 [cs.CV].

[55] Zhengzhe Liu, Xiaojuan Qi, and Philip Torr. *Global Texture Enhancement for Fake Face Detection in the Wild*. 2020. arXiv: 2002.00133 [cs.CV].

[56] Clemens Seibold et al. „Detection of Face Morphing Attacks by Deep Learning“. In: *Digital Forensics and Watermarking*. Cham: Springer International Publishing, 2017, pp. 107–120. ISBN: 978-3-319-64185-0.

[57] Ulrich Scherhag, Christian Rathgeb, and Christoph Busch. „Towards Detection of Morphed Face Images in Electronic Travel Documents“. In: *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. 2018, pp. 187–192. DOI: 10.1109/DAS.2018.11.

[58] Ulrich Scherhag, Christian Rathgeb, and Christoph Busch. „Morph Deterction from Single Face Image: A Multi-Algorithm Fusion Approach“. In: *Proceedings of the 2018 2nd International Conference on Biometric Engineering and Applications*. ICBEA ’18. Amsterdam, Netherlands: Association for Computing Machinery, 2018, pp. 6–12. ISBN: 9781450363945. DOI: 10.1145/3230820.3230822. URL: https://doi.org/10.1145/3230820.3230822.

[59] Poorya Aghdaie et al. „Detection of Morphed Face Images Using Discriminative Wavelet Sub-bands“. In: *2021 IEEE International Workshop on Biometrics and Forensics (IWBF)*. 2021, pp. 1–6. DOI: 10.1109/IWBF50991.2021.9465074.

[60] Matteo Ferrara, Annalisa Franco, and Davide Maltoni. „Face morphing detection in the presence of printing/scanning and heterogeneous image sources“. In: *IET Biometrics* 10.3 (Feb. 2021), pp. 290–303. ISSN: 2047-4946. DOI: 10.1049/bme2.12021. URL: http://dx.doi.org/10.1049/bme2.12021.

[61] Darius Afchar et al. „MesoNet: a Compact Facial Video Forgery Detection Network“. In: *2018 IEEE International Workshop on Information Forensics and Security (WIFS)* (Dec. 2018). DOI: 10.1109/wifs.2018.8630761. URL: http://dx.doi.org/10.1109/WIFS.2018.8630761.

[62] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. *In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking.* 2018. arXiv: 1806.02877 [cs.CV].

[63] David Güera and Edward J. Delp. „Deepfake Video Detection Using Recurrent Neural Networks“. In: *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).* 2018, pp. 1–6. DOI: 10.1109/AVSS.2018.8639163.

[64] Umur Aybars Ciftci, Ilke Demir, and Lijun Yin. „FakeCatcher: Detection of Synthetic Portrait Videos using Biological Signals“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). ISSN: 1939-3539. DOI: 10.1109/tpami.2020.3009287. URL: http://dx.doi.org/10.1109/TPAMI.2020.3009287.

[65] Pavel Korshunov and Sébastien Marcel. „Speaker Inconsistency Detection in Tampered Video“. In: *2018 26th European Signal Processing Conference (EUSIPCO).* 2018, pp. 2375–2379. DOI: 10.23919/EUSIPCO.2018.8553270.

[66] Rashmiranjan Das, Gaurav Negi, and Alan F. Smeaton. „Detecting Deepfake Videos Using Euler Video Magnification“. In: *Electronic Imaging* 2021.4 (Jan. 2021), pp. 272–272. ISSN: 2470-1173. DOI: 10.2352/issn.2470-1173.2021.4.mwsf-272. URL: http://dx.doi.org/10.2352/ISSN.2470-1173.2021.4.MWSF-272.

[67] Ilke Demir and Umur Aybars Ciftci. „Where Do Deep Fakes Look? Synthetic Face Detection via Gaze Tracking“. In: *ACM Symposium on Eye Tracking Research and Applications.* New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450383448. URL: https://doi.org/10.1145/3448017.3457387.

[68] Shruti Agarwal et al. „Protecting World Leaders Against Deep Fakes“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops.* June 2019.

[69] Shruti Agarwal et al. „Detecting Deep-Fake Videos from Phoneme-Viseme Mismatches“. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, pp. 2814–2822. DOI: 10.1109/CVPRW50498.2020.00338.

[70] Yipin Zhou and Ser-Nam Lim. „Joint Audio-Visual Deepfake Detection“. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 14800–14809.

[71] Run Wang et al. *DeepSonar: Towards Effective and Robust Detection of AI-Synthesized Fake Voices*. 2020. arXiv: 2005.13770 [eess.AS].

[72] Engineering National Academies of Sciences and Medicine. *Implications of Artificial Intelligence for Cybersecurity: Proceedings of a Workshop*. Washington, DC: The National Academies Press, 2019. ISBN: 978-0-309-49450-2. DOI: 10.17226/25488. URL: https://www.nap.edu/catalog/25488/implications-of-artificial-intelligence-for-cybersecurity-proceedings-of-a-workshop.

[73] Paul Taylor. *Text-to-Speech Synthesis*. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511816338.

[74] Youcef Tabet and Mohamed Boughazi. „Speech synthesis techniques. A survey“. In: *International Workshop on Systems, Signal Processing and their Applications, WOSSPA*. 2011, pp. 67–70. DOI: 10.1109/WOSSPA.2011.5931414.

[75] *Voice Conversion: A Critical Survey*. Zenodo, July 2010. DOI: 10.5281/zenodo.849853. URL: https://doi.org/10.5281/zenodo.849853.

[76] Kaizhi Qian et al. *AUTOVC: Zero-Shot Voice Style Transfer with Only Autoencoder Loss*. 2019. arXiv: 1905.05879 [eess.AS].

[77] Kaizhi Qian et al. „Unsupervised speech decomposition via triple information bottleneck“. In: *arXiv preprint arXiv:2004.11284* (2020).

[78] Pedro Cano et al. „Voice Morphing System for Impersonating in Karaoke Applications“. In: *ICMC*. 2000.

[79] Hartmut R Pfitzinger. „Unsupervised speech morphing between utterances of any speakers“. In: *Proceedings of the 10th Australian International Conference on Speech Science & Technology*. 2004, pp. 545–550.

[80] Jemine Corentin. „Real-time Voice Cloning“. Master thesis. Liège, Belgique: Université de Liège, Liège, Belgique, 2019. URL: https://matheo.uliege.be/handle/2268.2/6801?locale=en.

[81] Rafael Valle et al. *Flowtron: an Autoregressive Flow-based Generative Network for Text-to-Speech Synthesis*. 2020. arXiv: 2005.05957 [cs.SD].

[82] Brihi Joshi et al. *An exploration into Deep Learning methods for Emotional Text-to-Speech*. Version v1.0.0. June 2020. DOI: 10.5281/zenodo.3876081. URL: https://doi.org/10.5281/zenodo.3876081.

[83] Edresson Casanova et al. *YourTTS: Towards Zero-Shot Multi-Speaker TTS and Zero-Shot Voice Conversion for everyone*. 2022. arXiv: 2112.02418 [cs.SD].

[84] Yist Y. Lin et al. *FragmentVC: Any-to-Any Voice Conversion by End-to-End Extracting and Fusing Fine-Grained Voice Fragments With Attention*. 2021. arXiv: 2010.14150 [eess.AS].

[85] Ju-chieh Chou, Cheng-chieh Yeh, and Hung-yi Lee. *One-shot Voice Conversion by Separating Speaker and Content Representations with Instance Normalization*. 2019. arXiv: 1904.05742 [cs.LG].

[86] Kazuhiro Kobayashi and Tomoki Toda. *sprocket: Open-Source Voice Conversion Software*. EasyChair Preprint no. 64. EasyChair, 2018. DOI: 10.29007/s4t1.

[87] Hirokazu Kameoka et al. *StarGAN-VC: Non-parallel many-to-many voice conversion with star generative adversarial networks*. 2018. arXiv: 1806.02169 [cs.SD].

[88] Takuhiro Kaneko et al. „MaskCycleGAN-VC: Learning Non-parallel Voice Conversion with Filling in Frames“. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 2021.

[89] Junichi Yamagishi et al. *ASVspoof 2019: The 3rd Automatic Speaker Verification Spoofing and Countermeasures Challenge database*. 2019. URL: https://doi.org/10.7488/ds/2555.

[90] Junichi Yamagishi et al. *ASVspoof 2021: accelerating progress in spoofed and deepfake speech detection*. 2021. arXiv: 2109.00537 [eess.AS].

[91] Joel Frank and Lea Schönherr. *WaveFake: A Data Set to Facilitate Audio Deepfake Detection*. 2021. arXiv: 2111.02813 [cs.LG].

[92] Ricardo Reimao and Vassilios Tzerpos. „FoR: A Dataset for Synthetic Speech Detection“. In: *2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*. 2019, pp. 1–10. DOI: 10.1109/SPED.2019.8906599.

[93] Zhenyu Zhang et al. *SynSpeechDDB: a new synthetic speech detection database*. 2020. DOI: 10.21227/ta8z-mx73. URL: https://dx.doi.org/10.21227/ta8z-mx73.

[94] Zhenyu Zhang et al. *FMFCC-A: A Challenging Mandarin Dataset for Synthetic Speech Detection*. 2021. arXiv: 2110.09441 [cs.SD].

[95] Tianxiang Chen et al. „Generalization of audio deepfake detection“. In: *Proc. Odyssey 2020 The Speaker and Language Recognition Workshop.* 2020, pp. 132–137.

[96] Massimiliano Todisco, Héctor Delgado, and Nicholas Evans. „A New Feature for Automatic Speaker Verification Anti-Spoofing: Constant Q Cepstral Coefficients“. In: *Proc. The Speaker and Language Recognition Workshop (Odyssey 2016).* 2016, pp. 283–290. DOI: 10.21437/Odyssey.2016-41.

[97] Zhizheng Wu, Eng Chng, and Haizhou Li. „Detecting Converted Speech and Natural Speech for anti-Spoofing Attack in Speaker Recognition“. In: *13th Annual Conference of the International Speech Communication Association 2012, INTERSPEECH 2012* 2 (Jan. 2012).

[98] Zhizheng Wu et al. „A study on spoofing attack in state-of-the-art speaker verification: the telephone speech case“. In: Jan. 2012, pp. 1–5. ISBN: 978-1-4673-4863-8.

[99] Yanmin Qian et al. „Deep Feature Engineering for Noise Robust Spoofing Detection“. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.10 (2017), pp. 1942–1955. DOI: 10.1109/TASLP.2017.2732162.

[100] Hossein Zeinali et al. „Detecting Spoofing Attacks Using VGG and SincNet: BUT-Omilia Submission to ASVspoof 2019 Challenge“. In: *Proceedings of Interspeech.* Vol. 2019. 9. Graz, AT: International Speech Communication Association, 2019, pp. 1073–1077. DOI: 10.21437/Interspeech.2019-2892. URL: https://www.fit.vut.cz/research/publication/12086.

[101] Yexin Yang et al. „The SJTU Robust Anti-Spoofing System for the ASVspoof 2019 Challenge“. In: Sept. 2019. DOI: 10.21437/Interspeech.2019-2170.

[102] Xinhui Chen et al. *UR Channel-Robust Synthetic Speech Detection System for ASVspoof 2021.* 2021. arXiv: 2107.12018 [eess.AS].

[103] Rohan Kumar Das. „Known-unknown Data Augmentation Strategies for Detection of Logical Access, Physical Access and Speech Deepfake Attacks: ASVspoof 2021“. In: *Proc. 2021 Edition of the Automatic Speaker Verification and Spoofing Countermeasures Challenge.* 2021, pp. 29–36. DOI: 10.21437/ASVSPOOF.2021-5.

[104] Xinhui Chen et al. „UR Channel-Robust Synthetic Speech Detection System for ASVspoof 2021“. In: *Proc. 2021 Edition of the Automatic Speaker Verification and Spoofing Countermeasures Challenge.* 2021, pp. 75–82. DOI: 10.21437/ASVSPOOF.2021-12.

[105] Tianxiang Chen et al. „Pindrop Labs' Submission to the ASVspoof 2021 Challenge". In: *Proc. 2021 Edition of the Automatic Speaker Verification and Spoofing Countermeasures Challenge.* 2021, pp. 89–93. DOI: 10.21437/ASVSPOOF.2021-14.

[106] Woo Hyun Kang, Jahangir Alam, and Abderrahim Fathan. „CRIM's System Description for the ASVSpoof2021 Challenge". In: *Proc. 2021 Edition of the Automatic Speaker Verification and Spoofing Countermeasures Challenge.* 2021, pp. 100–106. DOI: 10.21437/ASVSPOOF.2021-16.

[107] Joaquín Cáceres et al. „The Biometric Vox System for the ASVspoof 2021 Challenge". In: *Proc. 2021 Edition of the Automatic Speaker Verification and Spoofing Countermeasures Challenge.* 2021, pp. 68–74. DOI: 10.21437/ASVSPOOF.2021-11.

[108] Ricardo Reimao and Vassilios Tzerpos. „Synthetic Speech Detection Using Neural Networks". In: *2021 International Conference on Speech Technology and Human-Computer Dialogue (SpeD).* 2021, pp. 97–102. DOI: 10.1109/SpeD53181.2021.9587406.

[109] Janavi Khochare et al. „A Deep Learning Framework for Audio Deepfake Detection". In: *Arabian Journal for Science and Engineering* (2021), pp. 1–12.

[110] Run Wang et al. „DeepSonar: Towards Effective and Robust Detection of AI-Synthesized Fake Voices". In: *Proceedings of the 28th ACM International Conference on Multimedia.* MM '20. Seattle, WA, USA: Association for Computing Machinery, 2020, pp. 1207–1216. ISBN: 9781450379885. DOI: 10.1145/3394171.3413716. URL: https://doi.org/10.1145/3394171.3413716.

[111] Anton Tomilov et al. „STC Antispoofing Systems for the ASVspoof2021 Challenge". In: *Proc. 2021 Edition of the Automatic Speaker Verification and Spoofing Countermeasures Challenge.* 2021, pp. 61–67. DOI: 10.21437/ASVSPOOF.2021-10.

Hardwarově akcelerovaná kryptografie s využitím FPGA

Petr Jedlička

1 Úvod

V posledním desetiletí nastal významný pokrok v oblasti vývoje kvantových počítačů. Například společnost IBM byla schopna v roce 2020 sestavit kvantový počítač s 53 qubity [1]. V následujícím roce tatáž společnost představila počítač využívající 127 qubitů a do konce roku má v plánu sestavit první kvantový počítač s více než 1000 qubity [2]. Za předpokladu, že bude výzkum kvantových počítačů pokračovat tempem, jež je nastíněn výše uvedenými čísly, je velká pravděpodobnost, že v následujícím desetiletí bude lidstvo disponovat kvantovým počítačem, jež bude schopen pomocí algoritmů určených speciálně pro kvantové počítače prolomit kryptosystémy využívající problémy založené na diskrétním logaritmu a faktorizaci velkých čísel. V současné době jsou tyto kryptosystémy využívány většinou systémů, kde je potřeba zajistit autenticitu, integritu a důvěrnost dat, což v budoucnu bude představovat bezpečnostní problém.

Odpovědí na tento problém je výzkum v oblasti takzvané postkvantové kryptografie, jež zahrnuje kryptosystémy, pro které v současné době není znám žádný typ útoku, který by ohrozil jejich bezpečnost, a to jak s použitím konvenčního počítače, tak i počítače kvantového. Momentálně probíhá třetí kolo standardizace kryptosystémů pro postkvantovou kryptografii pod záštitou NIST (National Institut of Standards and Technology) [3]. Podle dosavadních výsledků standardizace se jako nejperspektivnější ukazují kryptosystémy založené na mřížkách, jejichž problematika implementace na FPGA (Field Programmable Gate Array) bude přiblížena v následujících kapitolách tohoto článku.

2 Základní rozdíly v práci s FPGA a s mikroprocesorem

Zatímco mikroprocesor vykonává jednotlivé instrukce na základě strojového kódu uloženého v paměti programu, FPGA umožňuje návrh libovolného digitálního obvodu, kterým je nakonec i samotný mikroprocesor, který je na FPGA také možné syntetizovat. Na rozdíl od mikroprocesoru, jehož možnosti paralelizace jsou limitovány počtem jeho jader, na FPGA je možné bez problému vytvořit stovky paralelních procesů, kdy každý proces může být tvořen dalšími paralelními procesy.

Výše popsané vlastnosti umožňují na FPGA velmi rychlé vykonávání operací, jež jsou tvořeny řetězcem dílčích aritmetických a logických funkcí, které mohou běžet paralelně. Z hlediska možností paralelizace naopak nejsou příliš vhodné algoritmy, které pracují v uzavřené smyčce (iteracích). Nejméně vhodnou skupinu tvoří algoritmy, obsahující rozsáhlé rozhodovací struktury určující provedení určité operace. Implementace takových algoritmů často vede ke vzniku velkých kombinačních logik obsahujících mnoho kritických cest, které snižují maximální taktovací frekvenci dané implementace. Optimalizace takových algoritmů pro FPGA je možná, ale mnohdy komplikovaná.

3 Typické operace v postkvantové kryptografii

Nejrozšířenější typem operací, kterou vnitřně provádí všechny algoritmy založené na mřížkách, jsou aritmetické operace mezi maticemi a vektory polynomů. Jako příklad lze uvést následující operaci násobení matice A o rozměru $k \times l$ s vektorem y o délce l, kdy výsledkem je vektor w o délce k. Operaci lze popsat následující rovnicí [4]:

$$w = Ay$$

Elementy matice a vektorů jsou tvořeny polynomy 256. řádu. Aby bylo možné jednotlivé polynomy mezi sebou bodově vynásobit, čímž dojde k výraznému urychlení operace, je potřeba je převést pomocí transformace NTT (Number Theoretic Transform). [4]

V následujících kapitolách bude přiblížena problematika implementace vybraných částí postkvantových algoritmů. Konkrétní příklady pochází z implementace digitálního podpisu CRYSTALS-Dilithium a algoritmu pro šifrování a distribuci klíče CRYSTALS-Kyber.

4 Implementace transformace NTT

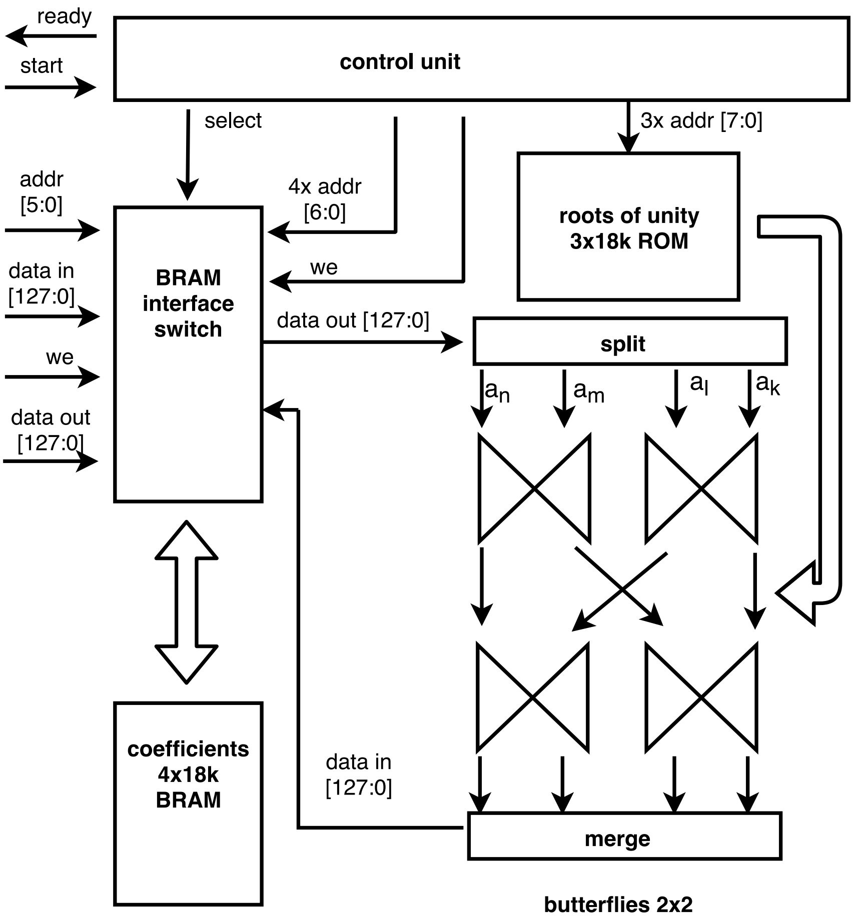
NTT transformace je algoritmus založený na FFT (Fast Fourier Transformation). Liší se však oborem čísel, nad kterým pracuje. Zatímco FFT pracuje s reálnými čísly, NTT pracuje s celými čísly v modulární aritmetice. [4]

Problémem při implementaci tohoto algoritmu je vykonávání v iteracích, kterých je pro polynom 256. řádu celkem 8 ($2^8 = 256$). Výstupy iterace slouží jako vstup pro iteraci následující, přičemž dochází navíc ke změně pořadí těchto vstupů, což na první pohled vede k nevyhnutelným úsekům čekání mezi iteracemi, avšak i takový algoritmus je možné pomocí pokročilejších metod optimalizace paralelizovat a eliminovat tak doby čekání na minimum.

Výpočet je možné urychlit paralelním nasazením několika motýlků (z anglického označení *Butterly*), což je základní výpočetní struktura algoritmu FFT i od něj odvozeného NTT. V každé z paralelizovaných iterací se musí nacházet stejný počet motýlků. Tento počet zároveň musí odpovídat násobku čísla 2^{n-1}, kde n je počet paralelizovaných iterací. Toto pravidlo vychází z matematické podstaty FFT a jeho odvození by přesahovalo rozsah článku. Pro paralelizaci dvou iterací tedy stačí 4 motýlci v rozložení 2×2 (2 motýlci v každé iteraci) nebo 8 motýlků v rozložení 4×4. Pro paralelizaci ve třech iteracích by však již bylo potřeba minimálně 12 motýlků. Je potřeba vždy zvolit správný kompromis mezi mírou paralelizace a množstvím využitých hardwarových zdrojů.

V dalším kroku je možné eliminovat čekání mezi „neparalelizovanými sadami“ iterací. Toho je možné docílit střídavým výpočtem iterací pro dva polynomy, jejichž iterace jsou navzájem nezávislé. Například pro rozložení 2×2 by bylo pořadí následovné: iterace 1-2 pro polynom 1, iterace 1-2 pro polynom 2, iterace 3-4 pro polynom 1, iterace 3-4 pro polynom 2,... Blokové schéma implementované transformace s motýlky 2×2 je na obrázku 1.

Vstupní koeficienty, mezivýsledky a výstupní hodnoty jsou ukládány do 4 blokových RAM pamětí (BRAM). Roots of unity (ekvivalent twidle faktorů u FFT) jsou uloženy v ROM paměti. Řídící jednotka (control unit)



Obrázek 1: NTT transformace

provádí čtení z obou pamětí, data přivádí na vstup motýlků a jejich výstup ukládá zpět do BRAM pamětí.

5 Implementace smyček

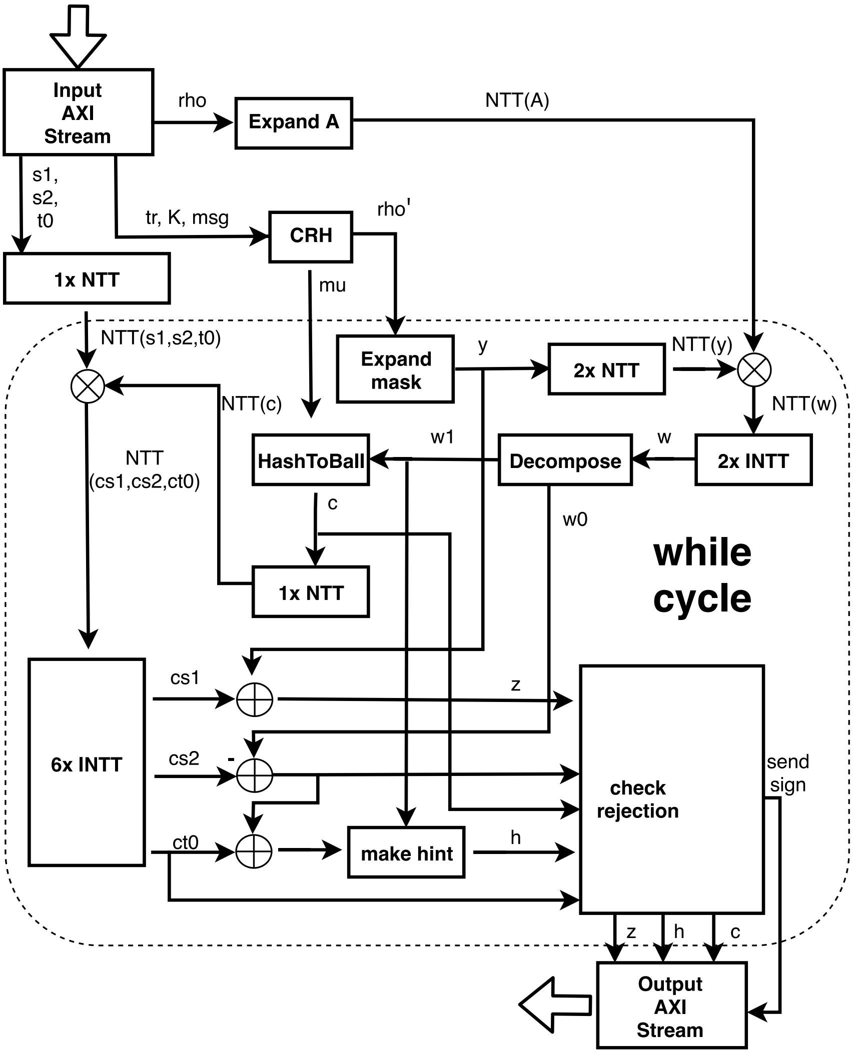
Další částí, kterou je při implementaci na FPGA potřeba optimalizovat, jsou smyčky typu *while*, které jsou přítomny v algoritmech digitálních podpisů založených na mřížkách. V případě podpisu CRYSTALS-Dilithium je ve smyčce zahrnuto přibližně 80% všech operací, z nichž je většina časově dost náročná. V případě sekvenčního vykonávání těchto operací by vždy byla aktivní pouze jedna komponenta. Paralelizaci této smyčky předcházela úvaha, že jednotlivé procesy, které v ní jsou obsaženy mohou počítat výstupy i pro budoucí iterace, aniž by bylo známo, zda tato iterace nastane. Jediným kritériem pro provedení výpočtu daným procesem se tedy stala dostupnost potřebných dat na jeho vstupu. Ve výsledku tak všechny procesy ve smyčce běží paralelně a v daném okamžiku každý z nich počítá výstupy pro jinou budoucí iteraci. Blokové schéma implementace digitálního podpisu se znázorněnou smyčkou typu *while* je na obrázku 2.

Uvnitř komponenty se nachází mnoho dílčích komponent, pro jejichž detailní popis není v tomto článku prostor. Nicméně lze dílčí komponenty zjednodušeně rozdělit do tří skupin: komponenty založené na NTT, komponenty využívající hašovací funkci Keccak a komponenty provádějící aritmetické operace nad vektory a maticemi polynomů. Komponenta podpisu používá pro komunikaci s okolím AXIStream rozhraní.

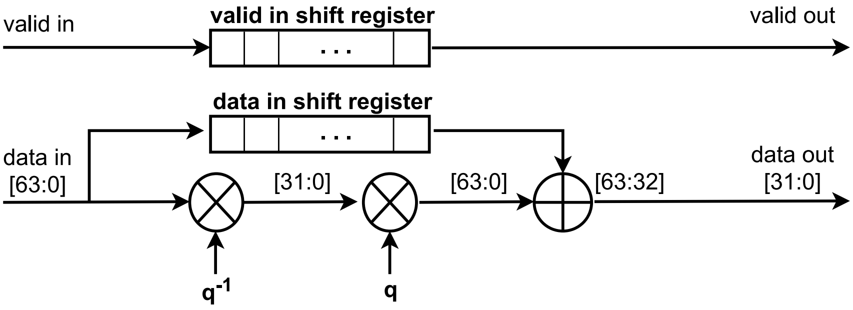
6 Implementace aritmetických operací

Aritmetické operace lze v porovnání s předchozími dvěma operacemi implementovat nejsnadněji. FPGA obvody obsahují vestavěné DSP (Digital Signal Processing) bloky, které umožňují akcelerovat aritmetické operace. Typickým příkladem je blok Montgomeryho redukce, který provádí redukci po násobení v modulární aritmetice. Komponenta se skládá z řetězce několika aritmetických operací, které mohou paralelně pracovat. Blokové schéma implementace Montgomeryho redukce je na obrázku 3.

Komponenta obsahuje 2 násobičky a jednu sčítačku. Pro všechny tyto části byly na FPGA použity DSP bloky. Délka posuvného registru pro valid signál odpovídá celkové latenci komponenty. Délka posuvného registru



Obrázek 2: Blokové schéma implementace podpisu



Obrázek 3: Blokové schéma implementace Montgomeryho redukce

pro vstupní data, která jsou používána na dvou místech v komponentě, odpovídá součtu latencí násobiček.

7 Závěr

V článku byly popsány metody optimalizace hardwarových implementací pro nejčastěji se vyskytujících operace v postkvantových kryptografických schématech CRYSTALS-Dilithium a CRYSTALS-Kyber. Konkrétně se jednalo o obecné aritmetické operace, NTT transformaci a smyčky typu *while*.

Ačkoliv se jednalo o optimalizace pro konkrétní kryptosystémy, u kterých v současné době nelze s jistotou říct, zda se stanou standardem pro postkvantovou kryptografii, lze tyto metody optimalizace uplatnit i u jiných algoritmů založených na mřížkách.

Odkazy

[1] E. Pednault, J. Gunnels a D. Maslov. *On "Quantum Supremacy"*. 2019. URL: `https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/`.

[2] J. Palyza. *IBM představila nový kvantový čip: díky němu budou výkonné počítače působit zastarale*. 2021. URL: `https://www.chip.cz/novinky/ibm-predstavila-novy-kvantovy-cip-diky-nemu-budou-vykonne-pocitace-pusobit-zastarale/`.

[3] Dr. Lily Chen, Dr. Dustin Moody a Dr. Yi-Kai Liu. *Post-Quantum Cryptography*. 2022. URL: `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`.

[4] Léo Ducas et al. „CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme“. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (ún. 2018), s. 238–268. DOI: `10.13154/tches.v2018.i1.238-268`. URL: `https://tches.iacr.org/index.php/TCHES/article/view/839`.

Phishingator aneb cvičný phishing „nejen“ na ZČU

Martin Šebela

Abstrakt

Příspěvek se zabývá vzdělávací aplikací Phishingator, která původně vznikla jako bakalářská práce autora s cílem upozornit na stále se zvyšující hrozbu phishingu. Aplikace umožňuje jednoduše vytvářet cvičné phishingové kampaně, které se skládají z vlastních podvodných webových stránek (například falešného přihlášení do univerzitních systémů) a z vlastních podvodných e-mailů, které jsou následně rozeslány konkrétním příjemcům. Phishingator poté průběžně sleduje, jakým způsobem příjemci na podvodný e-mail a stránku reagují a především, zdali se do ní nepokusili zadat platné přihlašovací údaje. Každému příjemci je zároveň poskytnuta zpětná vazba s vyznačenými indiciemi, na základě kterých bylo možné phishing rozpoznat. Na ZČU bylo díky aplikaci Phishingator provedeno již několik cvičných phishingových kampaní na různá témata. Jak na cvičné phishingové kampaně reagují uživatelé v akademickém prostředí, jak vypadá typický průběh kampaně ve Phishingatoru a je riziko phishingu na základě výsledků z Phishingatoru opravdu aktuální?

1 Phishing

S podvodnými e-maily (tzv. phishing) se pravděpodobně setkal již každý z uživatelů e-mailové komunikace. Zatímco dříve bylo možné se setkat až s téměř úsměvnými e-maily o výhrách v loterii nebo se žádostí o financování kosmického programu některé z afrických zemí, vzorky z posledních několika let ukazují, že phishing může být i velmi sofistikovaný. Útočníci rozesílaný phishing neustále vylepšují, případně i personalizují vůči konkrétní organizaci (tzv. spear phishing) tak, aby byl důvěryhodnější. Podvodné e-maily mohou rovněž zneužívat právě probíhající situaci ve světě. Z pohledu kybernetické bezpečnosti se navíc jedná o nejčastější bezpečnostní incident. Phishing se ovšem z e-mailové komunikace rozšířil i do SMS a textových zpráv (tzv. smishing) nebo telefonních hovorů (tzv. vishing). Celá řada uživatelů o phishingu bohužel neví, nebo si jeho riziko stále nepřipouští. A právě na základě těchto důvodů vznikl na ZČU nástroj Phishingator[1], který dobrovolně registrovaným uživatelům rozesílá cvičný phishing a upozorňuje na znaky, díky nimž lze rozpoznat skutečný phishing.

2 Phishingator

Celým názvem *„Systém pro rozesílání cvičných phishingových zpráv“* umožňuje administrátorům v dané instituci (například na univerzitě) jednoduše odesílat cvičný phishing konkrétním adresátům. Cílem je nenásilnou formou upozornit na nebezpečí skutečného phishingu a uživatelům, kteří cvičnému phishingu podlehnou, předat obratem zpětnou vazbu a zobrazit indicie, na základě kterých bylo možné podvod rozpoznat.

2.1 Phishingová kampaň

Administrátor Phishingatoru (typicky zaměstnanec bezpečnostního CERT/CSIRT[2] týmu) nejprve sestaví tzv. phishingovou kampaň.

Součástí phishingové kampaně je především rozesílaný podvodný e-mail, který si administrátor v systému vytvoří podobně jako v běžném e-mailovém klientovi. Jediným rozdílem je přidání tzv. indicií k phishingovému e-mailu. Každá indicie představuje jednu označenou pasáž v textu e-mailu, na základě které může uživatel phishing rozpoznat. Všechny indicie k danému podvodnému e-mailu jsou uživateli zobrazeny tehdy, pokud cvičnému phishingu podlehne nebo odešle formulář na podvodné stránce. Uživateli je díky tomu poskytnuta zpětná vazba, na co se měl v e-mailu zaměřit a možnost poučit se.

Administrátor dále volí, jaká podvodná stránka bude s cvičným phishingem svázána – tedy zdali se bude jednat např. o podvodnou stránku důvěrně napodobující přihlášení k univerzitním systémům nebo jen o generickou

[1] `https://phishingator.zcu.cz`

[2] CERT = Computer Emergency Response Team, CSIRT = Cyber Security Incident Response Team

přihlašovací stránku na nešifrovaném protokolu HTTP. Šablony podvodných stránek, stejně jako domény, na kterých jsou cvičné podvodné stránky hostovány, lze ve Phishingatoru také jednoduše nastavit.

V závěru administrátor zvolí dobu trvání kampaně a vyplní konkrétní adresáty, kteří cvičný phishing v daný den a čas obdrží. Jediným tlačítkem je následně celá phishingová kampaň spuštěna a v reálném čase je možné sledovat, jak vybraní příjemci na cvičný phishing, potažmo podvodnou stránku reagují. U každého z adresátů jsou rozlišovány následující akce:

1. bez reakce na podvodný e-mail a podvodnou stránku,
2. návštěva podvodné stránky,
3. vyplnění neplatných přihlašovacích údajů do formuláře na podvodné stránce,
4. vyplnění platných přihlašovacích údajů do formuláře na podvodné stránce.

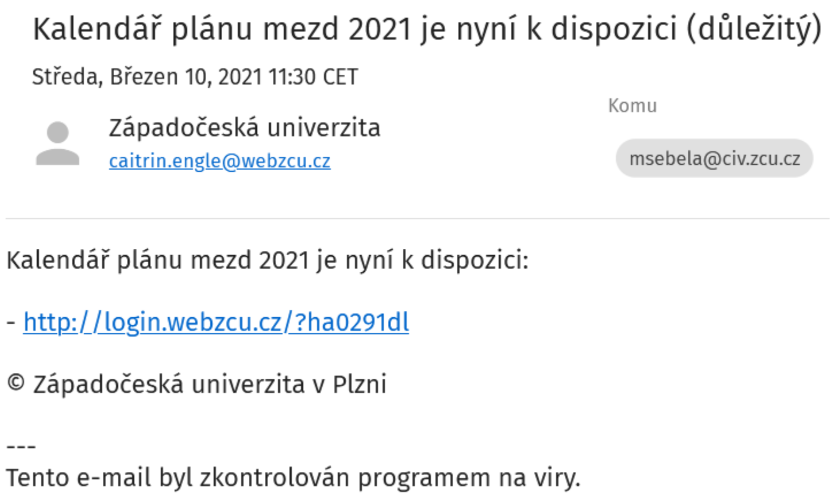
Každý uživatel, který na cvičné podvodné stránce vyplní obsah formuláře a následně jej odešle (tj. akce 3 a 4 z výše uvedeného seznamu), je obratem přesměrován na stránku s již zmíněnými indiciemi, na základě kterých bylo možné konkrétní phishing rozpoznat.

Po ukončení phishingové kampaně dochází k odeslání e-mailové notifikace všem uživatelům, kteří byli součástí dané kampaně. Notifikace zároveň obsahuje odkaz na stránku s indiciemi. I uživatel, který cvičný phishing odhalil, tak má šanci si prohlédnout, jaké všechny indicie byly v cvičném podvodném e-mailu obsaženy.

2.2 Typický průběh phishingové kampaně

V počátku roku 2021 několik českých univerzit upozornilo na podvodný e-mail, který jejich uživatele informoval o „kalendáři plánu mezd“ a obsahoval typické znaky phishingu (viz obr. 1). Phishing zneužíval hlavičku jména odesílatele, ve které byl uveden název univerzity, na níž útočník cílil. Útočník i graficky věrně napodobil přihlašovací stránku daných univerzit.

Na ZČU nebyl podobný vzorek bezpečnostním týmem registrován, došlo tak v rámci cvičné phishingové kampaně k rozeslání podobného phishingu skrze aplikaci Phishingator. S cvičným phishingem byla rovněž svázána i cvičná phishingová stránka, která vzhledově odpovídala přihlášení do


Kalendář plánu mezd 2021 je nyní k dispozici (důležitý)

Středa, Březen 10, 2021 11:30 CET

Západočeská univerzita
caitrin.engle@webzcu.cz

Komu
msebela@civ.zcu.cz

Kalendář plánu mezd 2021 je nyní k dispozici:

- http://login.webzcu.cz/?ha0291dl

© Západočeská univerzita v Plzni

Tento e-mail byl zkontrolován programem na viry.

Obrázek 1: Cvičný phishing rozeslaný z aplikace Phishingator, který se ovšem podobal skutečnému phishingu.

systémů na ZČU, hostována byla ovšem na zakoupené doméně `webzcu.cz` (oficiální doména ZČU je `zcu.cz`).

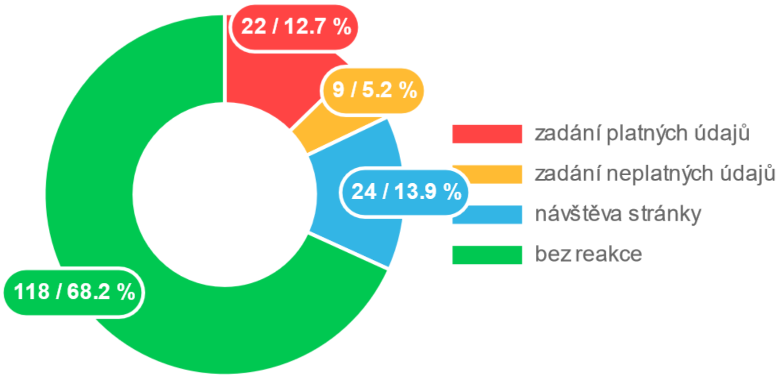
Cvičný phishing byl následně rozeslán 173 vybraným zaměstnancům na ZČU. Jak ukazuje tab. 1, první platná identita (tj. uživatel do formuláře na cvičné phishingové stránce vyplnil platné uživatelské jméno a heslo) byla získána již v první minutě kampaně.

Čas	**Milník**
11:30	Spuštění kampaně a odeslání cvičného phishingu.
11:31	První návštěva podvodné stránky.
11:31	Získána první platná identita.
11:35	Získáno již 8 platných identit.
do 12:00	Získáno již 15 platných identit.

Tabulka 1: Počátek událostí cvičné phishingové kampaně z aplikace Phishingator.

Výsledky phishingové kampaně pak ukazují (viz graf na obr. 2), že ze 173 zaměstnanců (napříč fakultami a dalšími součástmi ZČU) tomuto cvičnému phishingu, který odpovídal skutečnému phishingu, podlehlo 22 zaměstnanců (12,7 %).

K získání všech 22 platných identit navíc došlo už během prvních dvou hodin kampaně.



Obrázek 2: Graf znázorňující výsledky phishingové kampaně z aplikace Phishingator.

2.3 Reakce uživatelů

Na ZČU je aplikace Phishingator postavena především na režimu dobrovolného přihlášení – kdo má zájem o cvičný phishing, tomu stačí se do aplikace Phishingator přihlásit univerzitním kontem a nastavit si, zdali chce cvičný phishing odebírat. Zaměstnanci bezpečnostního týmu pak několikrát v roce dobrovolně registrovaným uživatelům rozešlou cvičný phishing.

Aplikace ovšem umožňuje odeslat e-mail i např. zaměstnancům konkrétní katedry či oddělení. Na základě podnětu vedoucích některých kateder, kterým byly možnosti Phishingatoru prezentovány, tak došlo i k cílenému otestování zaměstnanců v rámci periodického školení bezpečnosti. Podobně došlo k rozeslání cvičného phishingu například i studentům předmětu o bezpečnosti v IT. I díky této formě upozornění na phishing se následně někteří z vybraných uživatelů přihlásili mezi dobrovolníky, kteří cvičný phishing chtějí také odebírat, a zůstat tak ve střehu.

U uživatelů, kteří podlehli některému z prvních obdržených cvičných phishingů, lze navíc pozorovat, že další cvičné phishingové e-maily již úspěšně odhalili. Pokud někteří z těchto uživatelů navíc následně obdrželi skutečný phishing, který jim připadal nebezpečný a měli již zkušenost díky aplikaci Phishingator, upozornili na obdržený podvodný e-mail přímo helpdesk ZČU.

Celkové výsledky za všechny dosud realizované kampaně z aplikace Phishingator na ZČU ukazují, že cvičnému phishingu podlehlo 11,2 % testovaných uživatelů.

3 Závěr

Jak už bylo zmíněno v úvodu, cílem aplikace není získat co největší množství platných identit uživatelů, ale především uživatele upozornit na hrozbu a rizika skutečného phishingu. Aplikace Phishingator tak může být praktickým doplňkem k teoretickým školením o phishingu a bezpečnosti, a to jak pro zaměstnance, tak pro studenty.

Výsledky cvičné phishingové kampaně uvedené výše potvrzují, že phishing jako takový si stále dokáže najít své oběti, které mu podlehnou. Pokud uživatelům ovšem poskytneme nástroj, díky kterému budou obezřetnější a s phishingem se cvičně setkají a budou o jeho hrozbě vědět, můžeme procento úspěšných phishingových útoků snížit. Je totiž lepší, se nanečisto spálit ve Phishingatoru a poučit se, nežli předat své přihlašovací údaje útočníkovi ve skutečném phishingu.

Practical lessons of (deep)faking human speech

Anton Firc, Kamil Malinka

E-mail: ifirc@fit.vutbr.cz, malinka@fit.vutbr.cz

Abstract

Deepfakes are an emerging threat to computer security. Various usages range from spreading fake news to identity theft. A particular subset of this problem is using deepfakes to spoof biometrics systems. The attacker spoofs an individual's identity by synthesizing his voice or appearance and uses this spoofed identity to gain access into a system secured by biometrics authentication. In our former research, we demonstrated that speech deepfakes present a threat not only to voice biometrics systems but also to people. We presented that text-dependent verification is more resilient to deepfakes than text-independent verification. This paper aims to summarize our former research with its results and to highlight the powerful role of open-source tools in this area.

Key words: deepfakes, biometrics, spoofing attack, impersonation

1 Introduction

The slang term *deepfake* has no agreed-upon technical definition; it is just a combination of words 'deep learning' and 'fake' and primarily relates to content generated by an artificial neural network. Deepfakes are a subset of synthetic media created using deep neural networks that depict events that never happened to entertain, defame individuals, spread fake news, and many others [1].

The recent advancements in machine learning make the creation of deepfakes easier than ever. Even people without any technical background are able to use commercial tools with intuitive UI. These techniques and tools are being used for both illicit and legitimate purposes. One of the unexplored areas of illicit usage is using deepfakes to spoof speaker recognition systems. Currently, the approach to this topic is exclusively "technical." A significant amount of research gets published regarding deepfake detection methods, deepfake creation methods, or improvements in voice biometrics systems. However, there is still a missing link between these areas, where the proposed detection methods are tested in production environments, tested against state-of-the-art deepfakes, and finally implemented into real-world biometrics systems. This is where our research aims to fill the missing gaps and links, explore the current readiness of voice biometrics systems against deepfakes, and the steps to be taken to improve the resilience and security of deployed biometric systems.

Section 2 discusses our former research on the resilience of voice biometrics systems to deepfakes and the human ability to detect deepfakes and Section 3 concludes all of the discussed information.

2 Former research

As previously stated, one of the unexplored areas of illicit deepfake usage is using deepfakes to spoof voice biometrics systems. There are mixed opinions on the feasibility of such attacks and minimal scientific evidence [**voicebot**, 2, 3, 4, 5]. Our prior work [6, 7] evaluated the current state of readiness of voice biometrics systems and people to face deepfakes. This section describes our motivation, presents the attack schema, and discusses the experiments executed to assess the resilience of voice biometrics systems against deepfakes and the human ability to distinguish between real and deepfake speech.

2.1 Motivation

Up to date, no experiments regarding the deepfake resilience of voice biometrics systems in real-world environments exist. We decided to analyze the current situation and assess the advancements of deepfake technology and the difficulty of executing a deepfake powered attack on a scenario of customer verification in a customer care call center. In this scenario, the communication is made exclusively using the telephone, and the voice biometrics system has to be spoofed as well as the human operator. The scenario and complete attack schema is described in Section 2.2.

2.2 Attack schema

As mentioned in our motivation, our first goal is to examine how difficult it is to spoof voice biometrics systems using deepfakes and what measures might be taken to mitigate the threats posed by deepfakes to voice biometrics systems. The second goal is to evaluate the human ability to identify deepfakes.

We decided to focus on the area of customer verification in call centers (see Figure 1). A non-malicious scenario involves the customer making a telephone call to the call center. Both the voice biometrics system and the operator have to verify the customer's identity.

In a deepfake scenario of an attack on a customer care call center, the attacker synthesizes utterances with a speech of his victim in advance. The attacker then makes a phone call to the customer care service and begins to replay the prepared utterances to verify his identity and initiate an unauthorized action of his choice. The success of this scenario relies on the deepfake ability to spoof voice biometrics systems as well as humans.

Attacker model

An attacker is a person with the ability to create voice deepfakes, and his goal is to gain access into a system secured by voice authentication, such as a bank call center. The attacker is in possession of all needed personal information about his victim, all needed details about the typical scenario of the voice authentication process, and finally samples of voice belonging to the victim.

The attacker will use all of this information to synthesize utterances reproducing the victim‘s speech, and then in the most believable way possible, try to access the system secured by voice biometrics system and use the granted access to his advantage.

Victim model

A victim is any person that uses her's or his voice to authenticate into any system. We also expect the victim to be a regular computer user or

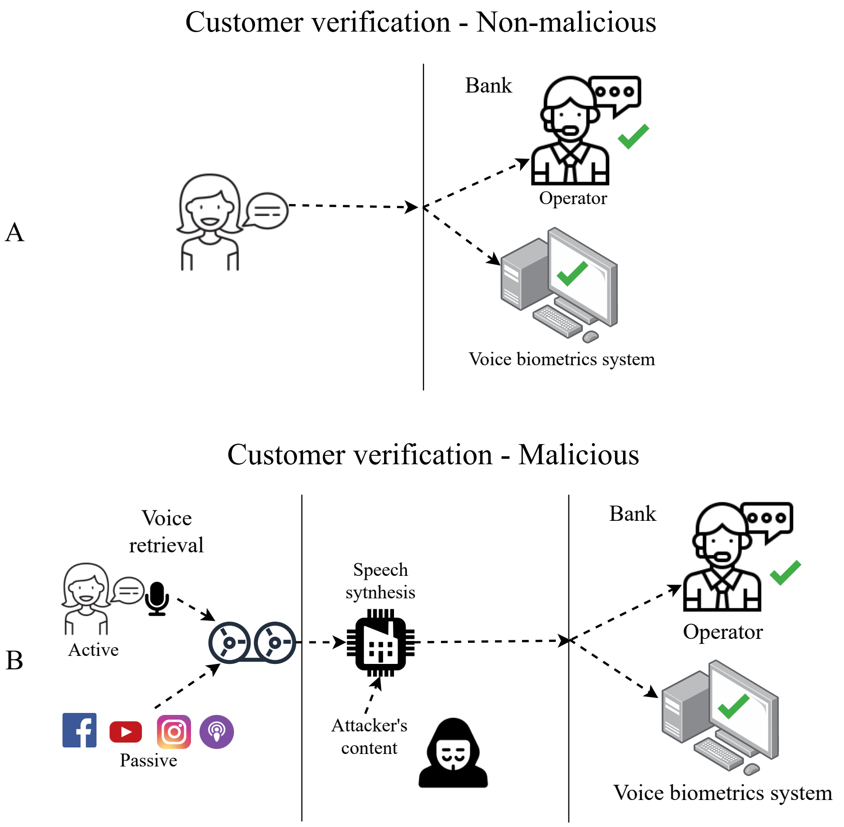


Figure 1: Attack schema. Figure A represents non-malicious (genuine) access to customer care call center, Figure B represents malicious access with target voice retrieval phase and speech synthesis.

possess a telephone. The voice samples of the victim can be retrieved from any of the content posted online or by recording a phone call.

2.3 Spoofing voice authentication

This section describes the experiments executed to assess the resilience of voice biometrics systems to deepfakes. To evaluate the threats posed by an inexperienced attacker, we decided to create our deepfake dataset using an open-source text-to-speech synthesis tool. Using the created dataset and two voice biometrics systems, we demonstrate that deepfakes do pose a serious threat to voice authentication. Finally, we present a method to mitigate these threats.

Executed experiments

The first experiment aimed to evaluate the technical feasibility of deepfake creation and acquiring knowledge about the attacker. We tested two commercial tools *Overdub* [8] and *Resemble AI* [9] and one open-source tool *Real-Time-Voice-Cloning (RTVC)* [10]. The commercial tools feature a straightforward process to synthesize speech of high quality. However, the usability of the commercial tools falls vastly behind the open-source one. The cost of increased usability lies within higher demands on effort and knowledge of attackers. Despite the needed knowledge, we assess the RTVC tool as the most powerful one, as explained later in this section.

We also examined the amount of data needed to clone an individual's voice; as little as five-second embedding is enough for the RTVC tool. To achieve higher quality results, fine-tuning is needed, which needs at least 0.2 hours of transcribed speech. More data is needed if the attacker plans to create a new model from scratch, around 20 hours of transcribed speech.

The second experiment evaluated the resilience of text-independent verification against deepfakes. We tested two voice biometrics systems: Microsoft Speaker Recognition API [**microsoftSpeech**] and Phonexia Voice Verify demo [11] against the three previously mentioned speech synthesis tools. We began by comparing genuine authentication attempts with the deepfake ones. The acquired data showed that the deepfakes created by the commercial tools get accepted as genuine speech. The deepfake speech synthesized by the open-source tool lacked a bit behind.

To further extend the scope of the second experiment, we decided to create a new deepfake dataset using the RTVC tool and to examine

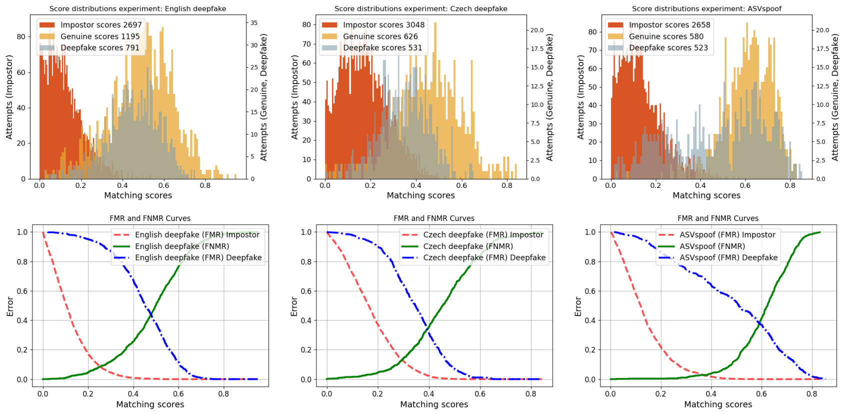


Figure 2: Matching scores distribution graphs (top) and FMR / FNMR graphs (bottom). The left plots represent created English deepfake dataset. The middle plots represent created Czech deepfake dataset. The right plots represent the ASVSpoof 2019 challenge dataset [12].

the differences between genuine and deepfake speech using MS Speaker Recognition API. We used the RTVC tool despite achieving the worst scores as we believe that the potential attacker would choose this tool over the commercial ones because of its usability and the MS Speaker Recognition API because of the definite result of the verification process that is easy to compare. Moreover, we decided to create our dataset because the current availability of English deepfake datasets is minimal, and no Czech deepfake dataset exists. Moreover, this way, we can explore all of the needed knowledge, time, and data needed to synthesize speech intended to be used maliciously and understand the attacker's profile even more deeply. The dataset consists of genuine and deepfake speech of 100 English and 60 Czech speakers selected from the Common Voice Corpus [13]. The dataset was published[1] for further use.

We collected matching scores of the genuine, impostor, and deepfake attempts using this dataset. Then we compared collected matching scores by plotting score distribution plots and FMR, FNMR curves as shown in Figure 2. The overlap of genuine and deepfake matching scores indicates

[1]`https://drive.google.com/drive/u/2/folders/1vlR-TA7gjKzjYylxzRnA_HzZEyWiLe0k`

that there was no difference in how the voice biometrics system processed genuine and deepfake speech. The increase in EER value shows that presenting deepfake speech to the voice biometrics system increases the system's number of mistakes in terms of false accepts.

The third, final experiment compared the resilience of text-dependent and text-independent verification to deepfakes. During the previous experiments, we noticed a discrepancy between scores obtained from each verification type. To further examine this behavior, we created a small dataset of 5 speakers consisting of phrases for text-dependent verification in Microsoft Speaker Recognition API [**microsoftSpeech**]. We collected matching scores for genuine and deepfake attempts using text-dependent and text-dependent verification types. As Figure 3 shows, the deepfake matching scores differ vastly from the genuine ones. This difference almost vanished when using text-independent verification. This implies that it is much easier to reproduce the matching scores of text-independent verification, which puts the text-dependent verification into a position of the more secure one when facing deepfakes. To thoroughly verify this hypothesis, more robust testing must be carried out.

Experiments conclusions

The executed experiments assessing the resilience of voice biometrics systems to deepfakes have suggested that deepfakes pose a serious threat to voice authentication. Current state-of-the-art speech synthesis models are able to synthesize speech using only a very short embedding utterance, which makes almost every person a suitable target. Additionally, considering the availability of the speech synthesis tools as an open-source projects, the level of threat to voice biometrics might be stated as significant. Fortunately, we have discovered that text-dependent verification shows to be more resilient to deepfakes than text-independent verification. As text-dependent verification is a well-known method, which is surely implemented in some systems, it might be currently used to improve the resilience of voice biometrics systems without the need for any extensive changes to the deployed systems. Even though this finding requires to be examined in deeper detail on more robust experiments, current results look promising.

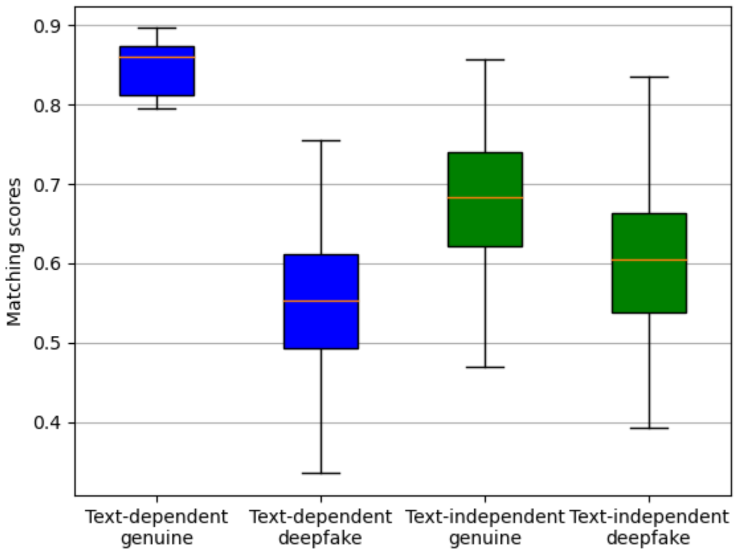


Figure 3: Comparison of scores calculated for text-dependent and text-independent verification using the same genuine and deepfake recordings for both verification types.

2.4 Spoofing people

The second important factor in the proposed call-center scenario is the human factor. In most cases, after being authenticated by the voice biometrics system, there is a human operator the customer has to talk to in order to perform the desired actions. If the operator gains any suspicion, she or he will most likely end the conversation. It means that the deepfake recording must be able to spoof the voice biometrics system and the operator simultaneously.

To examine whether the created deepfakes might be accepted by humans as genuine recordings, we created a survey where the respondents are set into a voice biometrics system. The main goal of the respondents is to decide whether each attempt is genuine or not.

The survey consists of 10 speakers from the deepfake dataset created during the first experiment based on their average deepfake matching scores to range from the lowest to the highest. The second criterion balanced sex distribution, so five female and five male speakers were selected.

For each speaker, there were two deepfake utterances and one genuine utterance. These utterances we combined in random order into the survey. Two versions were created, one with instructions in the Czech language and the second in English.

Both of the surveys were released at the beginning of March 2021, and during the course of one month, exactly 100 responses were collected. The ratio of Czech to English responses was 1 to 2, and the female to male ratio was exactly 1 to 1.

The responses were evaluated the same way as the performance of biometrics systems is evaluated. Each utterance represents a verification attempt, genuine or impostor. False accept rate and false match rate were calculated to evaluate how precise the respondents were in distinguishing between genuine and deepfake utterances. Table 1 shows results of both surveys together, and also each one separately.

The results show that approximately one of the three deepfake authentication attempts was successful. The quality of the genuine utterance had a significant impact on the false accept rate.

The achieved results show that voice deepfakes have the ability to fool humans and that there is some correlation between calculated matching scores and the accept or reject rates.

Table 1: FAR and FRR calculated from the survey results.

	both	English	Czech
FAR (%)	30.67	27.29	36.57
FRR (%)	39.06	37.04	42.57

3 Conclusions

The experiments executed on the proposed call-center model have separately examined the included factors: voice biometrics system and human operator.

Voice biometrics systems were easily fooled by synthetic speech. Using only a short embedding recording, the attacker can synthesize the speech of the selected victim and spoof the voice biometrics system. This shows how powerful open-source tools are when used properly.

In terms of human recognition of deepfakes, we showed that people could not spot a difference between genuine and deepfake recordings. Considering both of the stated factors, the presented attack schema on call-center shall be plausible. Spoofing the voice biometrics systems with deepfakes is relatively easy, and the human factor does not increase the difficulty.

The revealed information urges the need for the development and deployment of methods to increase the resilience of voice biometrics systems against deepfakes. One of the possibilities is to deploy a deepfake detection solution. However, the effort and knowledge needed to develop a reliable detector are pretty disproportional to the effort and knowledge needed to perform an attack on the described model. One of the significant problems with deepfake detection is the wide inter-class variability. Most of the developed and presented solutions currently struggle with previously unseen data codecs or modifications such as adding noise or changing the bitrate. This results in misclassifying deepfakes as genuine speech. Moreover, deploying a deepfake detection solution might also increase the genuine rejection rate. This is an unwanted side-effect that frustrates legitimate customers, and thus, the operator of such a biometrics system will most likely turn the deepfake detection off.

The other option is to use natural defenses, such as text-dependent verification. However, this approach has its own shortcomings. Implementing a text-dependent verification is not always feasible, and we are currently not aware of any other methods that will provide a sufficient level of resilience against deepfakes.

Acknowledgments

This work was supported by the internal project of Brno University of Technology (FIT-S-20-6427). Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] Jon Bateman. *Deepfakes and Synthetic Media in the Financial System: Assessing Threat Scenarios.* Tech. rep. Carnegie Endowment for International Peace, 2020, pp. i–ii. URL: `http://www.jstor.org/stable/resrep25783.1`.

[2] Sudipto Ghosh. *Are You Confident About Distinguishing Between a Computer-Generated Voice and Human Voice?* online. 2019. URL: `https://aithority.com/ait-featured-posts/are-you-confident-about-distinguishing-between-a-computer-generated-voice-and-human-voice/`.

[3] Rupert Jones. *Voice recognition: is it really as secure as it sounds?* online. 2018. URL: `https://www.theguardian.com/money/2018/sep/22/voice-recognition-is-it-really-as-secure-as-it-sounds`.

[4] Dan Simmonss. *BBC fools HSBC voice recognition security system.* online. 2017. URL: `https://www.bbc.com/news/technology-39965545`.

[5] Jon Petersen. *Combating deepfakes with voice biometric technology.* online. 2019. URL: `https://www.techradar.com/news/combating-deepfakes-with-voice-biometric-technology`.

[6] Anton Firc. „Applicability of Deepfakes in the Field of Cyber Security“. Supervisor Mgr. Kamil Malinka, Ph.D. MA thesis. Brno University of Technology, Faculty of Information Technology, 2021.

[7] Anton Firc and Kamil Malinka. „The dawn of a text-dependent society: deepfakes as a threat to speech verification systems“. In: Brno, CZ: Association for Computing Machinery, 2022. DOI: `10.1145/3477314.3507013`. URL: `https://www.fit.vut.cz/research/publication/12595`.

[8] Descript. *Overdub*. online. 2021. URL: `https://www.descript.com/overdub`.

[9] Resmble AI. *Resemble AI webpage*. online. 2020. URL: `https://www.resemble.ai`.

[10] Jemine Corentin. „Real-time Voice Cloning“. Master thesis. Liège, Belgique: Université de Liège, Liège, Belgique, 2019. URL: `https://matheo.uliege.be/handle/2268.2/6801?locale=en`.

[11] Phonexia. *Phonexia Voice Verify*. `https://www.phonexia.com/en/product/voice-verify/`. 2021.

[12] Junichi Yamagishi et al. *ASVspoof 2019: The 3rd Automatic Speaker Verification Spoofing and Countermeasures Challenge database*. 2019. URL: `https://doi.org/10.7488/ds/2555`.

[13] Rosana Ardila et al. „Common Voice: A Massively-Multilingual Speech Corpus“. English. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 4218–4222. ISBN: 979-10-95546-34-4. URL: `https://www.aclweb.org/anthology/2020.lrec-1.520`.

Simulations of DAG-based Blockchain Protocols and Attacks on the PHANTOM Protocol via Transaction Selection Strategies

Martin Perešíni, Ivan Homoliak, Kamil Malinka, Federico Matteo Benčić, Tomáš Hladký

1 Introduction

Blockchain technology is a relatively new concept that only came to the general public's attention in 2009 when Satoshi Nakamoto's original Bitcoin white paper was published [1]. As a result, the value of cryptocurrencies has soared as companies and individuals have begun to realize their benefits. Public consciousness is focused mainly on Bitcoin, closely followed by Ethereum, but other blockchain protocols are also gaining considerable attention. The use cases for blockchain technology are not only the exclusive domain of cryptocurrencies, but the application can also be found in other sectors such as healthcare, food-chain supply, e-voting systems, data storage, and others [2]. With the promise of blockchain properties and their popularity, blockchain technology's security and privacy benefits could seem that blockchains are already settled, and there are no open questions. However, security and performance issues are still a significant concern for this technology. To improve the properties of blockchain protocols, the concept of Directed Acyclic Graph (DAG)-based protocols was introduced [3, 4].

The focus of this work is to explore the issues of DAG-oriented consensus protocols. No one has yet analyzed the performance and robustness of DAG-oriented approaches in an empirical study under the assumption of real-world conditions and adversarial settings via different incentive schemes that differ from protocol design.

1.1 Motivation

We investigate existing blockchain designs with DAG structure that propose solutions to blockchain throughput problems, particularly the PHANTOM protocol and its optimization GHOSTDAG. These protocols generalize the PoW rule of longest/strongest chain and utilize stale blocks (resulting from forks) in the DAG structure with a proposed random transaction selection strategy, resulting in increased transaction throughput.

However, this assumption of random transaction selection is not adequately analyzed. As a result, we propose an attack in which actors who do not follow the protocol design and select transactions rationally (based on the highest fee) will have a significant profit advantage over honest protocol users. To properly evaluate the proposed attack, we developed a custom simulator[1] that extends the open-source simulation tool to support multiple chains and enables us to investigate the behavior of such edge cases.

2 Blockchain throughput limitations

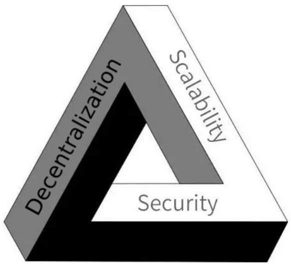


Figure 1: Blockchain trilemma concept.

Blockchains suffer from a bottleneck in processing throughput. Nakamoto's consensus (Bitcoin) proposal [1] limits the maximum block size, which implies a limited number of transactions. Bitcoin's throughput is around 3−8 transactions per second. Ethereum can process 15−30 transactions per second, compared to the centralized financial system Visa, which already

[1]`https://github.com/Tem12/DAG-simulator/`

had a maximum speed of $\sim 10000 - 20000$ transactions per second [5] is orders of magnitude different. One might think that a naive approach could solve this problem:

- **Increase block size** – and thus increase the number of transactions in one block. However, this leads to centralization because larger blocks take longer to propagate to all other miners (nodes) in the network [6]. Smaller miners will be at a disadvantage because by the time they receive the latest mined block, the larger miners (or pools) that just sent that block can immediately start mining the next block.

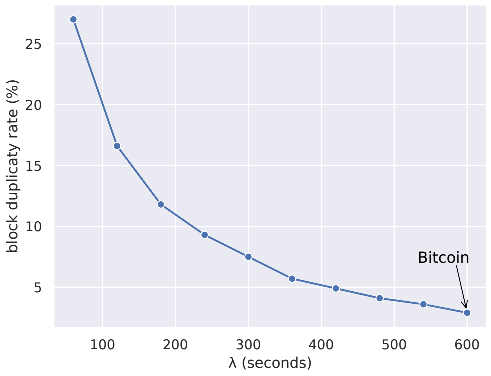


Figure 2: The relationship between the decreasing block creation time (λ) and the increased occurrence of orphan blocks in Bitcoin.

- **Decrease block creation time** – this will result in a reduction in mining difficulty. The bitcoin block creation time (parameter λ) is set to an average of 10 minutes [1]. Reducing this value will increase the orphaned block rate[2] (see Figure 2), resulting in wasted energy and resources. The occurrence and integration of orphaned blocks is resolved by consensus itself.

These naive improvements resolve the throughput problem and only worsen the balance between the so-called security-performance trade-off. If

[2]`https://www.investopedia.com/terms/o/orphan-block-cryptocurrency.asp`

we could somehow utilize stale blocks (forks), we can increase throughput $\rightarrow$ DAG-based protocols.

3 DAG-based protocols

Blockchains inherently suffer from the processing throughput bottleneck, as consensus must be reached for each block within the chain. One approach to solving this problem is to increase the rate at which blocks are created. However, this approach has its drawbacks (section 2, Figure 2).

3.1 Forks

If blocks are not propagated through the network before a new block (or blocks) is created, a *soft fork* can occur, where two blocks reference the same parent block. A soft fork resolves itself, and thus only one block is eventually accepted as valid, Figure 3. All other blocks and transactions they contain are discarded (i.e., *orphaned*). As a result, consensus nodes that created orphaned blocks waste their resources and do not get rewarded for their efforts. This is problematic because it may discourage consensus nodes from participating in the consensus protocol.

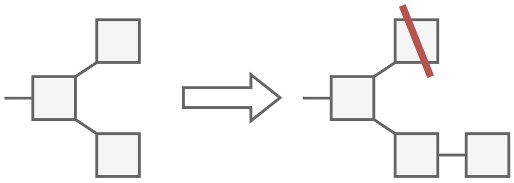


Figure 3: Soft fork.

Nakamoto (PoW) consensus uses a single chain to link the blocks. The order of blocks[3] in Bitcoin has initially been determined using the longest chain rule (see Figure 4). However, this rule was later replaced in favor of the strongest chain rule (see Figure 5), which takes into account the accumulated difficulty of the PoW puzzle.

As a response to the above issue, several proposals [3, 4] have substituted a single chaining data structure for Directed Acyclic Graph (DAG)

[3]And consequently transactions.

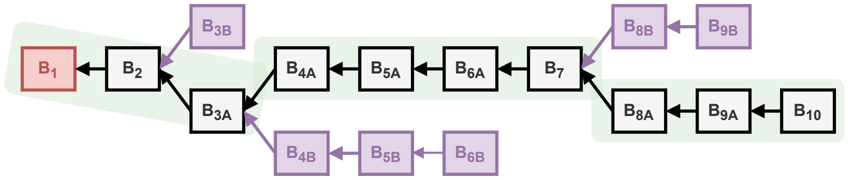


Figure 4: The longest-chain rule with orphaned blocks (**purple**) in PoW consensus.

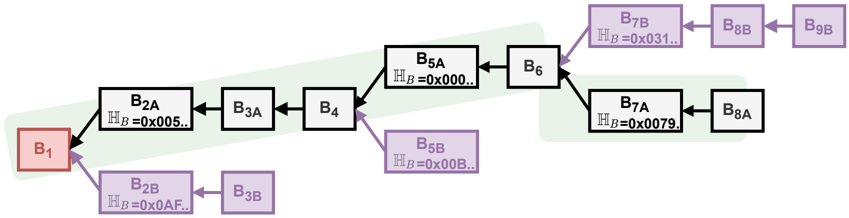


Figure 5: The strongest-chain rule with the main chain in **green** and orphaned blocks in **purple**. The hash of the block is denoted as $\mathbb{H}_B$.

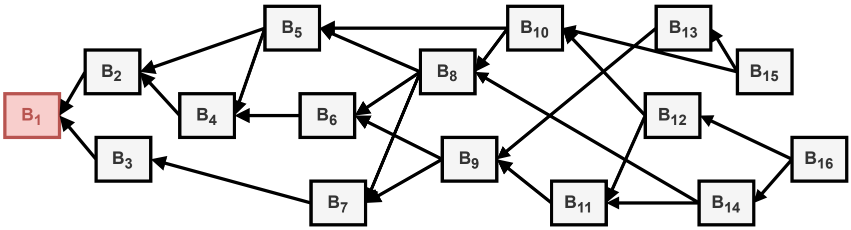


Figure 6: A DAG structure used in blockchain.

structure, as displayed in Figure 6. Such a structure can maintain multiple interconnected chains and thus increase processing throughput. The assumption of some DAG-oriented solutions is to abandon transaction selection based on the highest fee since this approach increases the probability that the same transaction is included in more than one block (hereafter *transaction collision*). Instead, these solutions use a random selection strategy to avoid transaction collisions.

3.2 PHANTOM

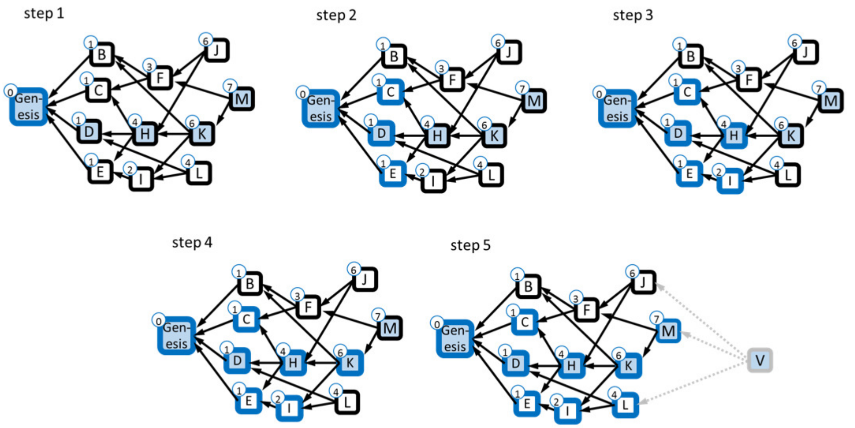


Figure 7: An example of a **blockDAG** G and the operation of **GHOSTDAG** to construct its blue set $BLUE_k(G)$ set, under the parameter $k = 3$. The circle near each block X represents its score, namely, the number of blue blocks in the DAG $past(X)$. The algorithm selects the chain **greedily**, starting from the highest-scoring tip M, then selecting its predecessor K (the highest scoring tip in $past(M)$), then H, D (breaking the C, D, E tie arbitrarily), and finally $Genesis$. A hypothetical "virtual" block V is introduced to this chain – a block whose past equals the entire current DAG. Blocks in the chain ($Genesis$, D, H, K, M, V) are marked with a light-blue shade. We construct the DAG's set of blue blocks using this chain, $BLUE_k(G)$. The set is constructed recursively and using the blue anticone principle [3].

There are several categories and designs of DAG protocols. More than thirty DAG-oriented blockchain systems have been classified based on their characteristics and principles. We skip the DAG protocols and their theoretical designs, whose complexity of recursively establishing consensus on the blocks it contains is impractical and challenging [3], we instead focus on PHANTOM and describe it. Moreover, since the use of PHANTOM is considered impractical in terms of efficiency [3], because it requires solving the NP-hard [4], the authors of PHANTOM have developed a greedy algorithm called GHOSTDAG (Figure 7, which is more convenient for implementation). We emphasize that the results of our research apply to GHOSTDAG and PHANTOM. In the context of this work, the maximum k-cluster SubDAG problem and block ordering are not crucial, so they have been abstracted in our simulations for simplicity.

4 Blockchain simulators

When proposing new changes or modifications to existing blockchains protocols, they need to be continuously tested and verified. A challenging point during this process is that some of their properties only become apparent with a large number of nodes. Since formal modeling cannot cover all properties, another option is to deploy actual nodes in an entire network, which can be a problem with a large number of nodes and require a minimal implementation of the blockchain; thus, it is an enormous computational cost. For this reason, in many cases, **simulation** remains **the only option** to verify their properties, as it simulates the behavior of an entire network. However, the design of such a simulator is complex, which is why most simulators focus on the specific aspects they implement, and only abstract the rest.

There are a lot of different blockchain simulators (Table 1): BTCsim [7], SimBlock [8], BlockSim [9], Simbit [10] and Bitcoin Mining Simulator [11] but none of them were initially intended for DAGs and did not support them.

Simulator	PoW	DES	Network Layer	Topo-logy	Block TXs	Compiled
BTCsim [7]	✓	✓	✓	✓		
Simbit [10]	✓	✓	✓		✓	
BlockSim [9]	✓	✓	✓	✓	✓	
SimBlock [8]	✓	✓	✓	✓		✓
Bitcoin MBSim [11]	✓	✓	✓	✓	✓	✓

Table 1: Comparison of different blockchain simulators.

[4]Maximum k-cluster SubDAG problem

Instead of implementing the simulation model from scratch, we decided to create an extension to an existing simulator (the Bitcoin Mining Simulator – **Bitcoin MBSim** [11]) that allows us to perform various experiments aimed at investigating the behavior of the PHANTOM protocol under attacks related to the potential identified problems. We have several requirements from the simulator that need to be fulfilled:

- Discrete Event Simulation (DES)
- Custom network topology
- Network layer with block propagation delay (custom latency between peers)
- Data layer involving transactions in blocks
- Mempool data structure
- Simulator implemented in a compiled language – optimization, for extensive experiments

Our extended simulator is a classic event-driven simulator. The created model tries to reflect as many of the necessary details as possible while abstracting away from unnecessary aspects of the protocol itself. The source code for the simulator is available at `https://github.com/Tem12/DAG-simulator/`.

The primary purpose of the simulator is to verify the profit of miners and the collision rate of transactions. Therefore, the simulator must implement a network and data layer. To this end, the simulator does not reflect actual blockchain implementations, which means that we do not simulate actual attacks on the network and assume that all simulated actors follow the consensus algorithm. Thus, we can omit the consensus layer and its features (e.g., Merkle tree hash computation, SHA256 hashing algorithm, block and transaction verification). We expect the simulation to spend most of the computation time on basic mempool (buffer of unprocessed transactions) operations such as sorting, inserting, or removing. For this reason, a mempool implementation is essential, and all mempool work must be optimized for better performance – hence the simulator is implemented in a compiled language. For a mempool, we used a specially tailored hash table. This optimization enables effective management of the mempool in the case of any transaction selection strategy[5]

Another part is the extension of the original simulator to support the simulation of (abstracted) multiple chains of DAG-based blockchains. At the same time, monitoring transaction duplicity, throughput, and the relative earned profit with respect to the used mining power (hash-rate). In addition, we have improved and added more simulation complexity to simulate each block, including the corresponding transactions (as opposed to simulating only the number of transactions in a block), added a custom process that creates transactions and broadcasts them to the network of miners. Most importantly, we implemented two different transaction selection strategies – a rational and a random strategy.

5 Evaluation

There are a lot of essential parameters and adjustments that we had to go through to create our best guideline on how to simulate blockchains properly. Every parameter in the system is sensitive and can drastically change the output. In addition to the simulator itself, we created automated scripts to create the correct simulation configuration (different network topologies, network delay distribution, etc.) and scripts to run the simulator on a multi-core system and implemented our use case for data collection and post-processing at a later stage, meaning the simulator could collect the data and the data could be processed independently at a later stage. For our simulations, we used the following parameter configuration:

5.1 Experiments

GOAL of the experiments is to examine the relative earned profits of malicious miners who follow a rational transaction selection strategy compared to honest miners who follow a random strategy.
RESULTS We implement a Bitcoin-like network topology of approximately 8000 nodes with realistic block propagation latency. We kept the frequency of block creation constant within this experiment ($\lambda = 20$ sec),

[5]Note: For example, a rational transaction strategy requires a mempool with transactions ordered by fees, while a random transaction strategy requires the hash-map data structure. Therefore, they are hard to utilize at the same time.

Parameter	Value
block creation rate (λ)	20
network propagation delay (τ)	5 seconds or realistic delay distr.
# of total blocks	20000
# of miners	7592
network topology	realistic Bitcoin-like
# of transactions in block	100
maximum size of mempool	10144 transactions
distribution of transaction fees	exponential ($\lambda_{fee} = 150$)
mining strategies	rational or random
mining power of each miner	0 to 100%

Table 2: Used parameters for the simulations.

experimented with the adversarial mining power of malicious miners in the network and monitored the relative profit for each setting. Part of the results are shown in Figure 8. They show that with a single malicious miner, the average profit of honest miners is roughly halved compared to the average profit of the malicious miners. In addition, our experiments showed that the profit advantage of malicious miners decreases as their number increases. This observation can be considered beneficial to the protocol because it discourages more miners from using a rational transaction selection strategy. On the other hand, we also observe that malicious miners are instead incentivized to **form a malicious mining pool**.

6 Conclusion

In the paper, we provide the reader with an introduction and analysis of a possible solution to blockchain throughput issues in the form of DAG-based blockchain protocols. On the other hand, we found that one such protocol, PHANTOM is susceptible to our presented attack on an incentive scheme utilizing a different transaction selection strategy. We created the open-source simulator and performed experiments that verify that malicious actors have relative profit advantages in a Bitcoin-like network and inherently decrease protocol throughput. The reader should be familiar with the problem of DAG-based protocols and simulation pitfalls to at least some extent. More details about DAG-oriented protocols, blockchain simulations and described attacks can be found in article [12].

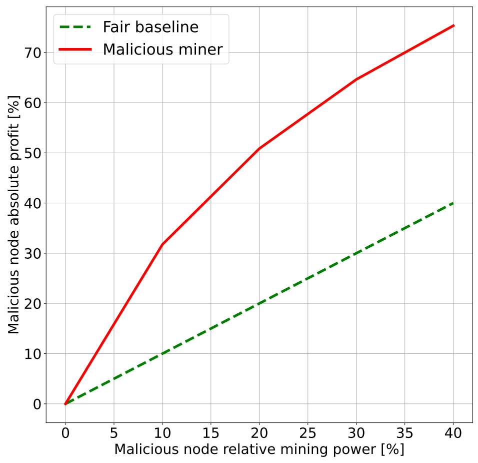


Figure 8: Profit relative to mining power of a malicious miner in Bitcoin-like network topology.

References

[1] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system.* 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[2] Thomas McGhin et al. „Blockchain in healthcare applications: Research challenges and opportunities“. In: *Journal of Network and Computer Applications* 135 (2019), pp. 62–75. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.02.027. URL: http://www.sciencedirect.com/science/article/pii/S1084804519300864.

[3] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. „PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus: September 2, 2021“. In: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies.* New York, NY, USA: Association for Computing Machinery, 2021, pp. 57–70. ISBN: 9781450390828. URL: https://doi.org/10.1145/3479722.3480990.

[4] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. „Inclusive block chain protocols“. In: *Financial Crypto.* 2015.

[5] Shihab Hazari and Qusay Mahmoud. „Improving Transaction Speed and Scalability of Blockchain Systems via Parallel Proof of Work“. In: *Future Internet* 12 (July 2020), p. 125. DOI: `10.3390/fi12080125`.

[6] J. Göbel and A.E. Krzesinski. „Increased block size and Bitcoin blockchain dynamics“. In: *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. 2017, pp. 1–6. DOI: `10.1109/ATNAC.2017.8215367`.

[7] Rafael Brune. *Bitcoin Network Simulator,* ***https://github.com/rbrune/btcsim***. online. Nov. 2013. URL: `https://github.com/rbrune/btcsim`.

[8] Shu Takayama et al. *SimBlock,* ***https://github.com/dsg-titech/simblock***. online. June 2019. URL: `https://github.com/dsg-titech/simblock`.

[9] Carlos Faria. *BlockSim: Blockchain Simulator,* ***https://github.com/carlosfaria94/blocksim***. online. Mar. 2018. URL: `https://github.com/carlosfaria94/blocksim`.

[10] S. Bowe. *Javascript P2P Network Simulator,* ***https://github.com/ebfull/simbit***. online. Mar. 2014. URL: `https://github.com/ebfull/simbit`.

[11] Gavin Andresen. *Mining / block propagation simulator,* ***https://github.com/gavinandresen/bitcoin_miningsim***. online. May 2015. URL: `https://github.com/gavinandresen/bitcoin_miningsim`.

[12] Martin Perešíni et al. *DAG-Oriented Protocols PHANTOM and GHOSTDAG under Incentive Attack via Transaction Selection Strategy.* 2021. DOI: `10.48550/ARXIV.2109.01102`. URL: `https://arxiv.org/abs/2109.01102`.

The Security Reference Architecture for Blockchains: Towards a Standardized Model for Studying Vulnerabilities, Threats, and Defenses

Ivan Homoliak

Abstract

Due to their specific features, blockchains have become popular in recent years. Blockchains are layered systems where security is a critical factor for their success. The main focus of this work is to systematize knowledge about security and privacy issues of blockchains. To this end, we propose a security reference architecture based on models that demonstrate the stacked hierarchy of various threats as well as threat-risk assessment using ISO/IEC 15408. In contrast to the previous surveys [1, 2, 3], we focus on the categorization of security vulnerabilities based on their origins and using the proposed architecture we present existing prevention and mitigation techniques. The scope of our work mainly covers aspects related to the nature of blockchains, while we mention operational security issues and countermeasures only tangentially.

1 Blockchains at a Glance

The blockchain is a data structure representing an append-only distributed ledger consisting of entries (transactions) aggregated within ordered blocks. The order of the blocks is agreed by untrusting participants running a consensus protocol. A transaction is an elementary data entry that may contain arbitrary data, e.g., an order to transfer native cryptocurrency (crypto-tokens), a piece of application code (i.e., smart contract), the execution orders of such application code, etc. Transactions sent to a blockchain are validated by all nodes that maintain a replicated state of the blockchain.

Involved Parties Blockchains usually involve the following parties. *(1) Consensus nodes* actively participate in the underlying consensus protocol. These nodes can read the blockchain and write to it by appending new transactions (which they also disseminate). Besides, they can validate the blockchain and thus check whether writes of other consensus nodes are correct and respect a specified logic. Consensus nodes can prevent malicious behavior (e.g., by not appending invalid transactions, or not following an incorrect blockchain view). In the context of Proof-of-Resource protocols (see subsection 4.2), these nodes are often referred to as *miners*. *(2) Validating nodes* read the entire blockchain, validate it, and disseminate transactions. Unlike consensus nodes, validating nodes cannot write to the blockchain. Hence, they cannot prevent malicious behavior. However, since they possess the entire blockchain, they can detect malicious behavior. *(3) Lightweight nodes* (i.e., clients) benefit from most of the blockchain functionalities, but they are equipped only with limited information about the blockchain. These nodes read only a fragment of the blockchain (usually block headers) and validate only a small number of transactions that concern them, while they rely on consensus and validating nodes for ensuring correctness of the blockchain. They can detect only a limited set of attacks.

Features of Blockchains Blockchains were initially proposed as open cryptocurrencies, but due to their features, they became appealing for other applications as well. Blockchains achieve *decentralization* via a distributed consensus protocol, which provides resilience to failures. Usually, participants are equal and no single entity pose an authority. Another important result of decentralization is censorship resistance. Blockchains are *immutable*, requiring a significant quorum of colluding nodes to change its entries retrospectively. Usually, immutability is achieved thanks to a cryptographic one-way function that creates integrity preserving links between blocks. Although blockchains are highly redundant in a storage of the data, the main advantage of such redundancy is high *availability*. This feature is of special interest to applications that cannot tolerate outages. Blockchain transactions, as well as actions of protocol participants, are usually *transparent* to other participants and in most cases even to the public. This can be a benefit for multiple applications, but it can also be seen as a disadvantage from the anonymity and privacy perspective. Beside common features, some blockchains may focus on additional features, such as energy efficiency [4, 5, 6], scalability [7], throughput [8, 9, 10], privacy [11], etc.

Types of Blockchains Based on how a new node enters a consensus protocol, we distinguish the following blockchain types. *(1) Permissionless* blockchains allow anyone to join the consensus protocol without permission. Such participation can be anonymous, and these protocols are designed to run over the Internet. To prevent Sybil attacks [12], these schemes usually require consensus nodes to establish their identities by running a Proof-of-Resource scheme, while the consensus power of a node is proportional to its resources invested into running the protocol. *(2) Permissioned* blockchains require a consensus node to obtain permission (and identity) to join the consensus protocol. The permission is given by a centralized or federated authority(ies), while nodes usually have equal consensus power (i.e., one vote per node). These schemes can be *public* if they are accessible over the Internet or *private* when they are deployed over a restricted network. *(3) Semi-Permissionless* blockchains require each new-coming consensus node to obtain a permission (i.e., cryptocurrency "stake"); however, such permission can be given by any stakeholder (i.e., consensus node). These blockchains are similar to permissionless blockchains, except a consensus is based on a stake rather than on resources spent. The node's consensus power is proportional to the stake it has. Similar to permissionless blockchains, these systems are usually intended to be run over the Internet. Novel aspects of (semi-)permissionless blockchains are incentives and network effects that are designed to increase the protocol's security and adoption.

2 Security Reference Architecture

Stacked Model To classify security aspects related to blockchains, we introduce a simplified stacked model [2] consisting of four layers (see Figure 1). In contrast to previous work [2], we keep only such granularity level that enables us to isolate various nature of security threats in blockchains.

(1) The *network layer* (see section 3) consists of the data representation and network services planes. The data representation plane deals with storing and encoding of data, while the network service plane contains discovery and communication with protocol peers, addressing, routing, and naming services. (2) The *consensus layer* (see section 4) deals with ordering

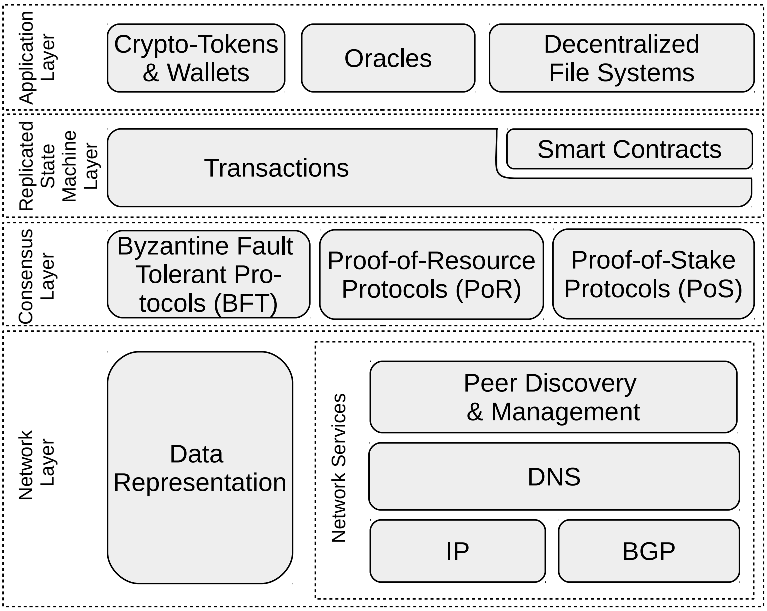

Application Layer
Crypto-Tokens & Wallets
Oracles
Decentralized File Systems
Replicated State Machine Layer
Transactions
Smart Contracts
Consensus Layer
Byzantine Fault Tolerant Protocols (BFT)
Proof-of-Resource Protocols (PoR)
Proof-of-Stake Protocols (PoS)
Network Layer
Data Representation
Network Services
Peer Discovery & Management
DNS
IP
BGP

Figure 1: Stacked model of reference architecture.

of transactions and we divide it according to a type of the protocol used to Byzantine Fault Tolerant (e.g., [13, 14, 15]), Proof-of-Resource (e.g., [7, 16, 17, 18, 19]), and Proof-of-Stake (e.g., [5, 6]) protocols. (3) The *replicated state machine (RSM) layer* (see section 5) deals with the interpretation of transactions, according to which, the state of the blockchain is updated. Smart contracts involve two special types of transactions, which represent a programming code itself and invocations of this code. (4) In the application layer (see section 6) we present the most common end-user functionalities such as crypto-tokens with wallets, oracles/data feeds, and decentralized file systems. Throughout the paper, we summarize components of particular layers with their respective security threats and protection techniques.

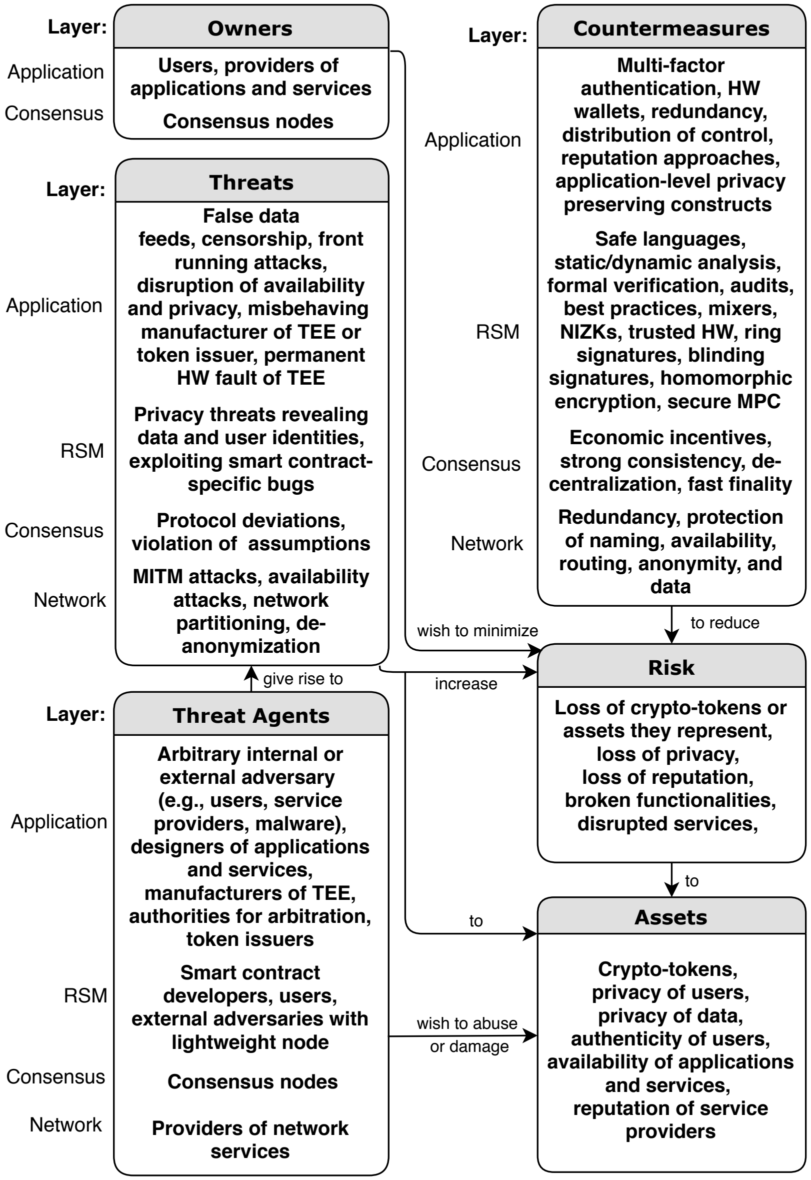
Threat-Risk Assessment Model To better capture security-related aspects of blockchain systems, we introduce a threat-risk model based on the template of ISO/IEC 15408 [20]. The model includes the following components and actors (see Figure 2). *Owners* are blockchain users who run any node type. Owners posses crypto-tokens and/or use blockchain-based applications or services. *Assets* consist of monetary value (i.e., crypto-tokens), blockchain functionalities, as well as services built on top of them (e.g., exchanges, secure logging, supply chains). *Threat agents* are malicious users whose intention is to steal assets, break functionalities, or disrupt services. *Threats* arise from vulnerabilities at the network, in smart contracts, from consensus protocol deviations, violations of protocol assumptions, or application-specific dependencies. Threats facilitate various attacks on assets and services. *Countermeasures* are provided by the security, safety, incentives, and reputation techniques that protect owners from threats. *Risks* caused by threats and their agents may lead to losses of monetary assets or service malfunctions and disruptions.

The owners wish to minimize the risk caused by threats that arise from threat agents. With the stacked model, different threat agents appear at each layer. At the network layer, there are service providers including parties managing IP addresses and DNS names. The threats at this layer come from man-in-the-middle (MITM) attacks, network partitioning, de-anonymization, and availability attacks. Countermeasures contain protection of availability, naming, routing, anonymity, and data. At the consensus layer, nodes may be malicious and wish to alter the outcome of the consensus protocol by deviating from it. The countermeasures include economic incentives, strong consistency, and decentralization. At the RSM layer, the threat agents may stand for developers who (un)intentionally introduce semantic bugs in smart contracts (intentional bugs represent backdoors). Mitigating countermeasures are safe languages, static/dynamic verification, and audits. Other threats are related to privacy of data and identity of users with mitigation techniques using mixers, cryptography constructs (e.g., non-interactive zero-knowledge proofs (NIZKs), ring signatures). At the application layer, threat agents are unspecified, since any user on the network who uses a blockchain application may pose a threat. The threats on this layer arise from false data feeds and examples of mitigation techniques are authentication or reputation systems.

3 Network Layer

3.1 Private Networks

A private network ensures low latency, a centralized administration, privacy, and meeting regulatory obligations (e.g., HIPAA). The organization owning



Layer:
Owners
Application
Users, providers of applications and services
Consensus
Consensus nodes
Layer:
Threats
Application
False data feeds, censorship, front running attacks, disruption of availability and privacy, misbehaving manufacturer of TEE or token issuer, permanent HW fault of TEE
RSM
Privacy threats revealing data and user identities, exploiting smart contract-specific bugs
Consensus
Protocol deviations, violation of assumptions
Network
MITM attacks, availability attacks, network partitioning, de-anonymization
give rise to
Layer:
Threat Agents
Application
Arbitrary internal or external adversary (e.g., users, service providers, malware), designers of applications and services, manufacturers of TEE, authorities for arbitration, token issuers
RSM
Smart contract developers, users, external adversaries with lightweight node
Consensus
Consensus nodes
Network
Providers of network services
Layer:
Countermeasures
Application
Multi-factor authentication, HW wallets, redundancy, distribution of control, reputation approaches, application-level privacy preserving constructs
RSM
Safe languages, static/dynamic analysis, formal verification, audits, best practices, mixers, NIZKs, trusted HW, ring signatures, blinding signatures, homomorphic encryption, secure MPC
Consensus
Economic incentives, strong consistency, de-centralization, fast finality
Network
Redundancy, protection of naming, availability, routing, anonymity, and data
to reduce
wish to minimize
increase
Risk
Loss of crypto-tokens or assets they represent, loss of privacy, loss of reputation, broken functionalities, disrupted services,
to
to
Assets
wish to abuse or damage
Crypto-tokens, privacy of users, privacy of data, authenticity of users, availability of applications and services, reputation of service providers

Figure 2: Threat-risk assessment model of reference architecture.

the network provides access to local participants as well as to external ones; hence systems deploying private networks belong to the group of permissioned private blockchains. The inherent feature of private networks is that authentication and access control can be provided at the network layer.

Pros

Access control is achieved by centralized authentication of users and assigning them roles. A private network has full control over routing paths and physical resources used, which enables regulation of the network topology and transmission medium best suited for requirements. *Data privacy* is ensured by permissioned settings. *User identity* is revealed only within a private group of nodes; they are immune to external attacks in contrast to public networks. *Fine-grained authorization controls* are applied by the operator of the network to implement the security principle of minimal exposure and thus mitigate insider threat attacks [21]. *Resource availability* is easier to manage, as all network participants and the deployment scenario are known ahead of time.

Cons

Virtual Private Network (VPN) connectivity is required to communicate between private networks spread over different geographical locations. While VPNs are in general secure, they inherit the disadvantages of running service over the Internet. *Applicability* of private networks is suitable only for permissioned and private blockchains.

Security Threats and Countermeasures

Insiders may pose a serious threat to security [22]. A compromised node may already have administrative privileges or obtain them by exploiting security vulnerabilities. *Countermeasures* include regular software updates, user monitoring (e.g., SIEM), prevention techniques that minimize trust and maximize trustworthiness, as well as respecting best practices [21].

3.2 Public Networks / the Internet

Public networks provide high decentralization, openness, and low entry barrier, while network latency, privacy, and network control are put aside. These networks are naturally required by all public (permissionless) blockchain systems.

Pros

High availability is attractive to multi-homed nodes since they have alternate routes to send/receive messages. Multi-homed nodes may find useful to disseminate blocks across multiple channels, thereby increasing a chance of blocks being appended to the blockchain. *Decentralization* is achieved through geographical dispersion of nodes. Public peer-to-peer (p2p) networks are harder to shut down [23]. *Openness and low entry barrier* on the Internet are achieved through wide adoption, technology interoperability (e.g., using TCP/IP), economic (e.g., low cost of broadband connection) and societal (e.g., resistance to regulations) factors.

Cons

Single-point-of-failure – DNS with its hierarchy, IP addresses, and ASes are managed by centralized parties like IANA of ICANN. *External adversaries* pose a threat to public networks. These adversaries can be classified based on their capabilities to which the blockchain network may be exposed [24]: (1) resources under attacker control (e.g., botnets, DNS and BGP servers), (2) identities are stolen or masqueraded (e.g., IP addresses participating in an eclipse attack or route manipulation), (3) MITM attacker (i.e., eavesdropping and spoofing), (4) common vulnerabilities leading to exploits, e.g., observed in DNS BIND [25], (5) revealing secrets (e.g., de-anonymizing peers). *Efficiency* – although an average Internet bandwidth has improved in recent years, a distribution of powerful infrastructure is not uniform, which results in a different latency among peers, and the overall latency of the network is increased – this, in turn, may result to loss of created blocks and thus wasting of consensus power.

Security Threats and Countermeasures

DNS attacks usually arise from cache poisoning that mainly affects nodes employing DNS bootstrapping to retrieve online peers but also users of online blockchain explorers. One *countermeasure* is a security extension of DNS, called DNSSEC, which provides authentication and data integrity. In addition, name resolution can also be made using alternate DNS servers.

Routing attacks are traffic route diversions, hijacking, or DoS attacks. Beside simple data eavesdropping or modification, these attacks may lead to network partitioning, which in turn raises the risks of 51% attacks or selfish mining attacks (see section 4). *Countermeasures* suggest nodes to be multi-homed (or using VPN) for route diversity, choosing extra peers whose connections do not pass through the same ASes, preference of peers hosted on the same AS within the same /24 prefix (to reduce risk of partitions), and fetching the same block from multiple peers [26]. Another mitigation is SABRE [27], a secure relay network that runs alongside with the Bitcoin network. The BGPsec [28] is an extension for BGP used between neighboring ASes, and it provides assurance of route origin and propagation.

Eclipse attacks hijack all connections of a node to the blockchain network. Hence, all traffic received by the node is under the full control of the attacker. Eclipse attacks arise from threats on DNS and routing in the network as well as they may be a result of vulnerabilities in p2p protocols [29, 30, 31]). Eclipse attack increase chances of selfish mining and double spending attacks (see section 4) – the eclipsed victims may vote for an attacker's chain. *Countermeasures:* Improving randomness in choosing peers was proposed in work [29] by several rules that manage the peer table. Another mitigation strategy against eclipse attacks is to use redundant network links or out-of-band connections to verify transactions (e.g., by a blockchain explorer). Also, note that countermeasures for DNS and routing attacks are applicable here as well.

DoS Attacks on connectivity may result in a loss of consensus power, thus preventing consensus nodes from being rewarded. For validating nodes, this attack leads to disruption of some blockchain dependent services. *Countermeasures:* One mitigation is to peer only with white-listed nodes. Methods to prevent volumetric DDoS include on-premise filtering (i.e., with an extra network device), cloud filtering (i.e., redirection of traffic through a cloud when DDoS is detected), or hybrid filtering [32] (i.e., combinations of the previous two).

DoS attacks on local resources, such as memory and storage, may reduce the peering and consensus capabilities of nodes [33]. An example is flooding the network with low fee transactions (a.k.a., penny-flooding), which may cause memory pool depletion, resulting in a system crash. Possible mitigation is raising the minimum transaction fee and rate-limit the number of transactions.

4 Consensus Layer

4.1 Generic Attacks

Violations of Protocol Assumptions

Adversarial Centralization of Consensus Power. In these attacks, a design assumption about the decentralized distribution of consensus power is violated. Examples of this category are *51% attacks* for PoR and PoS protocols as well as $\frac{1}{3}$ of *Byzantine nodes* for BFT protocols (and their combinations). In a 51% attack, the majority of the consensus power is held by the adversary. In *Byzantine attacks*, a quorum of $\frac{1}{3}$ adversarial consensus nodes might cause the protocol being disrupted or even halted. As a design-oriented countermeasure, it is important to promote decentralization by incentive schemes that reward honest participation and discourage [34] or punish [35, 36] protocol violations.

Time-Validation Attacks. Usually, besides system time, nodes in PoW and PoS maintain network time that is computed as the median value of the time obtained from the peers. Such a time is often put into the block header, while nodes, upon receiving a block, validate whether it fits freshness constraints. An attacker can exploit this approach by connecting a significant number of nodes and propagate inaccurate timestamps, which can slow down or speed up the victim node's network time [37]. When such a desynchronized node creates a block, this block can be discarded by a network due to freshness constraints. To avoid de-synchronization attacks, a node can build a reputation list of trusted peers or employ a timestamping authority [38].

Double Spending

This attack is possible due to the creation of two or more conflicting blocks with the same height, resulting in inconsistencies called *forks*. Therefore, some crypto-tokens might be temporarily spent in both conflicting blocks, while only a single block is later included in the honest chain. To prevent this attack in permissionless blockchains, it is recommended to wait a certain amount of time (i.e., a few next blocks) until a block "is settled."

4.2 Proof-of-Resource Protocols (PoR)

Protocols from this category require nodes to prove a spending of a scarce resource in a lottery-based fashion [39]. Scarce resources may stand for: *(1) Computation* that is represented by Proof-of-Work (PoW) protocols (e.g., Bitcoin, Ethereum). *(2) Storage* used in the setting of Proof-of-Space protocols [16] (e.g., Spacecoin [40], SpaceMint [41]). *(3) Crypto-tokens* spent for Proof-of-Burn protocols (e.g., Slimcoin [17]). *(4) Combinations and modification* of the previous types, such as storage and computation, called Proof-of-Retrievability (e.g., Permacoin [18]) and storage over time, which is represented by Proof-of-Space protocols (e.g., Filecoin [19]).

PoR protocols belong to the first generation of consensus protocols, and they are mostly based on Nakamoto Consensus [7] that utilize PoW, inheriting its pros (e.g., high scalability) and cons (e.g., low throughput). For the detailed analysis of several PoW designs, we refer the reader to [42].

Pros

In PoR protocols, malicious overriding of the history of blockchain (or its part) requires spending at least the same amount of resources as was spent for its creation. This is in contrast to principles of PoS protocols, where a big enough coalition may override the history with almost no cost.

Cons

stand mainly for a high operational cost. Moreover, these protocols provide only probabilistic finality, which enables attacks forking the last few blocks of the chain.

Security Threats and Mitigations

Selfish Mining: In selfish mining [43], an adversary attempts to privately build a secret chain and reveal it to the public only when an honest chain is "catching up" with the secret one. The longest chain rule causes honest miners to adopt the attacker's chain and invalidate the honest chain, thus wasting their consensus power. This attack is more efficient when consensus power of a selfish miner reaches some threshold (e.g., 30%). The selfish mining strategy was later generalized [44] and extended to other variants that increase the profit of the attacker [45]. *Countermeasures:* (1) For the case of the longest chain rule, the first introduced mitigation is uniform tie breaking [43], which tells consensus nodes to choose the chain to extend uniformly at random, regardless of which one they received first. However, this technique is less effective when assuming network delays [44]. (2) It is recommended to use fork choice rules that also account for the quality of solutions and make the decision deterministic, as opposed to a uniform tie breaking. An example of such a rule is to select the block based on the smallest hash value. Another example is to include partial solutions [46, 47, 48] or full (orphaned) blocks [49, 50] for computation of block's quality. (3) Another option is using a pseudo-random function [51], which moreover provides unpredictability, hence the attacker cannot determine his chances to win a tie. (4) PoW protocols can be combined with BFT protocols, where PoW is used only for joining the protocol and BFT for consensus itself (e.g., [9, 10, 51]).

Feather Forking: In this attack [52], the adversary creates incentives for rational miners to collectively censor certain transactions. Before a mining round begins, an adversary announces that he will not extend the block containing blacklisted transactions, and thus will attempt to extend a forked chain. Although this strategy is not profitable for the adversary and the success rate is dependent on his consensus power, rational honest nodes prefer to join on the censorship to avoid the potential loss. *Countermeasures:* design-oriented protection is to minimize the chance of the attacker being successful, which can be done by including (and rewarding) partial solutions [46, 47, 48] or full orphaned blocks [49, 50] into branch difficulty computation.

4.3 Byzantine Fault Tolerant (BFT) Voting Protocols

BFT protocols represent voting-based [39] consensus protocols that utilize Byzantine agreement and a state machine replication [53]. These protocols assume a fully connected topology, broadcasting messages, and a master-replicas hierarchy. *Synchronous* examples of this category are PBFT [13], RBFT, *eventually synchronous* examples are BFT-SMaRt [14], Tendermint [8], BChain, and *asynchronous* examples are SINTRA [54] and HoneyBadgerBFT [15]. For more details, we refer the reader to review of BFT protocols in [55].

Pros

BFT protocols provide high throughput and a low latency finality. To face their scalability limitation, BFT protocols are often combined with PoS or PoW. This is in line with a lottery approach [39] for selecting a portion of all nodes, referred to as committee, which further runs BFT consensus (e.g., Algorand [4], DFINITY [56]).

Cons

The main con of traditional BFT protocols [13] is a low scalability caused by a high communication complexity (i.e., $\Theta(n^2)$). Since these protocols can work efficiently only with a limited number of consensus nodes, they can be used in their pure form only in permissioned blockchains.

Security Threats and Mitigations

Many BFT protocols assume synchronous delivery of messages. However, this assumption can be violated by unpredictable network scheduler, as demonstrated in [15]. This fact motivates asynchronous BFT protocols that can be based on threshold-based cryptography, which enables reliable and consistent broadcast [54, 15]. Issues with scalability and throughput can be dealt with by applying cryptographic constructs [57, 58] and partitioning consensus nodes into shards that process transactions in parallel [9, 10]. Another option is to prune the number of nodes into committees [4]; however, this reduces security level of BFT and provides only probabilistic security guarantees depending on the committee size.

4.4 Proof-of-Stake Protocols (PoS)

Similar to the PoR category, PoS protocols are based on the lottery approach [39]. However, in contrast to PoR, no scarce resource is spent; instead, the nodes are required "to prove investment" of crypto-tokens in order to participate in a protocol, and thus potentially earn interest from the invested amount. The concept of PoS was first time proposed in Peercoin [59] as a combination with PoW – each node has its particular difficulty for PoW, which is based on the age of the coins a node owns. Although there exist a few pure PoS protocols (e.g., [5], [6]), the trend is to combine them in a hybrid setting with PoR (e.g., Proof-of-Activity [60], Peercoin [59], Snow White [61]) or BFT protocols (e.g., Algorand [4]).

In particular, a combination of PoS with BFT represents a promising approach taking advantages of both lottery and voting, while no resources are wasted.

Pros

The main feature of PoS protocols, as compared to PoR, is their energy efficiency. Although some PoS protocols are often combined with a PoR technique (e.g., [61, 59]), the overall energy spent is much less than in the case of pure PoR protocols.

Cons

Introduction of PoS protocols has brought PoS specific issues and attacks, while these protocols are still not formally proven to be secure. Next, PoS protocols are semi-permissionless – a node needs to first obtain a stake from any of existing nodes to join the protocol.

Security Threats and Mitigations

Nothing-at-Stake: Since generating a block in PoS does not cost any energy, a node can extend two or more conflicting blocks without risking its stake, and hence increase a chance to be rewarded. Such behavior increases the number of forks and thus time to finality. *Countermeasures:* Deposit-based solutions [35] require nodes to make a deposit during some fixed period/round and checkpoint-based solutions [35, 36]) employ "state freezing" at periodic snapshots, while the blockchain can be reversed maximally up to the recent checkpoint. Next option is to use cryptographic solutions [62] for revealing identity and a private key of a node that signs two conflicting blocks. Another countermeasure is to use backward penalization of nodes that produced two or more conflicting chains [36, 35]. Finally, PoS protocols can be combined with BFT approaches, and thus forks can hardly occur [4].

Grinding Attack: If the leader or committee producing a block is determined before the round starts, then the attacker can bias this process to increase his chances of being selected in future. For example, if a PoS protocol takes only a hash of the previous block for the election process, the leader of a block may bias a hash value by suitably adjusting the content of the block in a few attempts.

Countermeasures: A grinding attack can be prevented by performing a fresh leader election by an interaction of nodes (e.g., the secure multiparty coin flipping protocol [6]) or by private checking whether the output of a verifiable random function (VRF) is below a certain stake-specific threshold (e.g., [4]).

Denial of Service on a Leader/Committee: If a leader or a committee are publicly determined before the round starts [6], then the adversary may conduct a DoS attack against them and thus cause a restart of the round – this might be repeated until adversary's desired nodes are elected. *Countermeasures:* A prevention technique was proposed in Algorand [4] – a node privately determines whether it is a potential leader (or committee member), and immediately releases a block candidate (or a vote) – hence, after publishing this data, it is too late for a DoS attack. The concept of VRF approach was also utilized in other protocols [63, 56].

Long-Range Attack: In this attack [64] (a.k.a., posterior corruption [36]), an adversary can bribe previously influential consensus nodes to sell their private keys or steal the private keys by other means. Since consensus nodes may exchange their crypto-tokens for fiat money, selling their keys impose no expenses and risk. If the attacker accumulates keys with enough stake in the past, he may rerun the consensus protocol and rewrite the history of the blockchain. A variant of long-range attack that considers transaction fee-based rewarding and infrequent or no check-points is denoted as a *stake-bleeding* attack [65]. *Countermeasures:* One mitigation is to lock the deposit for a longer time than the period of participation in the consensus. The next mitigation technique is frequent periodic check-pointing, which causes the irreversibility of the blockchain with respect to the last checkpoint. Another option is to apply key-evolving cryptography and forward-secure digital signatures, which require users to evolve their private keys, while already used keys are erased (e.g., [63]). The third mitigation technique is enforcing a chain density in a time-domain [65] for the protocols where the expected number of participants in each round is known (e.g., [6]). The last option is context-sensitive transactions, which put the hash of a recent valid block into a transaction itself [65].

5 Replicated State Machine Layer

5.1 Transaction Protection

Mostly, transactions containing plain-text data are digitally signed by private keys of users, enabling anybody to verify the validity of transactions by corresponding public keys. However, such an approach provides only pseudonymous identities that can be traced to real identities, and moreover, it does not ensure confidentiality of data [66].

Security Threats and Countermeasures

Privacy Threats to User Identity. In many blockchains, user identities can be linked with their transactions by various deanonymization techniques, such as network flow analysis, address clustering, transaction fingerprinting [66, 67, 68]. Moreover, blockchains designed with anonymity and privacy features (e.g., Zcash, Monero) are also vulnerable to a few attack strategies [69]. *Countermeasures:* Various means are used for obfuscation of user identities, including centralized (e.g., CoinJoin, Mixcoin) and decentralized (e.g., CoinShuffle) mixing services, ring signatures [70], and non-interactive zero-knowledge proofs (NIZKs) [11]. Some mixers enable internal linkability by involved parties (e.g., CoinJoin) or linkability by the mixers (e.g., Mixcoin), which are also potential threats. Unlinkability for all parties can be achieved by multi-party computation [71], blinding signatures [72], or layered encryption (e.g., CoinShuffle). Ring signatures provide unlinkability to users in a signing group [70], enabling only verification of correctness of a signature, without revealing an identity of a signer.

Privacy of data. NIZKs [11] and blind signatures [73, 72] can be used for preservation of data privacy. Another method is homomorphic encryption [74], which enables to compute some operations over encrypted messages. Privacy and confidentiality for smart contract platforms can be achieved through trusted transaction managers [75], trusted hardware [76], and secure multi-party computations [77].

5.2 Smart Contracts

Smart contracts, introduced to automate legal contracts, now serve as a method for building decentralized applications on blockchains. They are usually written in a blockchain-specific programming language that may be Turing-complete and contain arbitrary programming logic or only serve for limited purposes. In the following, we describe these two contrasting types of smart contract languages.

Security Threats and Countermeasures

Turing-Complete Languages. This language category has a large attack surface due to the possibility of arbitrary programming logic. Examples are Serpent and Solidity, while Solidity is the most popular and widely-used one. *Serpent* is a high-level language that was designed to be simple and similar to the Python language. However, Serpent was designed in untyped fashion, lacking out-of-bound access checks of arrays and accepting invalid code by compilers [78], which opened the door for plenty of vulnerabilities. Hence, Serpent showed to be as an unsuccessful attempt to simplify the coding phase. *Solidity* is an object-oriented statically-typed language that is primarily used by Ethereum platform. Contracts written in Solidity can contain various types of vulnerabilities [79]. Mitigations of such vulnerabilities can be done by code analysis tools [80, 81], respecting best practices [82], utilizing known design patterns [83], audits, and testing. Various approaches are used for source code analysis, such as linters (e.g., SmartCheck [81], Solhint, Solium), fuzzers [84], semantic-based program verifiers [85], and other symbolic code analyzers [86] often using control flow-graph techniques. Note that source code of contracts is often not public in contrast to their bytecode. For this reason, bytecode decompilers (e.g., Erays [87], Porosity), analyzers [88], and automated exploit generators [89] can be utilized.

Turing-Incomplete Languages. The main pro of this category is its design-oriented goal of small attack surface and emphasis on safety but at the cost of limited expressiveness. Examples of this category are Pact, Scilla, Vyper. *Pact* is a declarative language intended for Kadena blockchain and provides type inference and module-guarded tables to prevent direct access to modules. Pact is equipped with the ability to express and check properties of its programs, also leveraging SMT solvers. *Scilla* is designed to achieve expressiveness and tractability while enabling formal reasoning about contract behavior. Every computation utilizes the automata-based model, and computations are realized as standalone atomic transitions that strictly terminate. Scilla enables external calls only as the last instruction of a contract, which simplifies proving safety and thus mitigates a few vulnerabilities. *Vyper* is an experimental language designed to ease the audit of smart contracts and increase security – it contains strong typing and bounds/overflows checks.

6 Application Layer

6.1 Crypto-Tokens and Wallets

Besides cryptocurrencies that provide native crypto-tokens, there are other blockchain applications using crypto-tokens for the purpose of providing owners with rights against the third party (i.e., counterparty tokens) or with a possibility of transferring asset ownership (i.e., ownership tokens) [90]. All types of tokens require the protection of private keys and secrets linked with user identities. For this purpose, two main categories of wallets have emerged – *self-sovereign wallets* and *hosted wallets* [91, 3, 92]. Beside technical risks, all crypto-tokens are exposed to regulatory risk, while non-native tokens are in addition exposed to legal risks [90].

Self-Sovereign Wallets Users of self-sovereign wallets locally store their private keys. These wallets differ in several aspects, e.g., isolation of the keys – there are software wallets that store the keys within the PC (e.g., Bitcoin Core, Electrum Wallet, MyEtherWallet) as well as hardware wallets that store keys in a sealed storage, while they expose only signing functionality (e.g., Trezor, Ledger, KeepKey, BitLox). Another type of wallets enables to customize functionality and security by a smart contract (e.g., Eth. MultiSigWallet, TrezorMultisig2of3).

Hosted Wallets Hosted wallets require a centralized party for interaction with the wallet and thus blockchain. If a hosted wallet has full control over private keys, it is referred to as a *server-side wallet* (e.g., Coinbase, Circle Pay Wallet, Luno Wallet), while in the case of keys stored in the users' browsers, the wallets are referred to as *client-side wallets* (e.g., Blockchain Wallet, BTC Wallet, Mycelium Wallet, CarbonWallet, Citowise Wallet). We refer the reader to works [92, 91] for a security overview of miscellaneous wallet solutions.

Security Threats and Mitigations

Since server-side wallets accounted for several compromises in the past, their popularity have attenuated in favor of client-side wallets. Although client-side wallets do not expose private keys to a 3rd party, they are dependent on the availability of the online interface provided by such a party. Contrary, self-sovereign wallets do not trust in a 3rd party nor rely on its availability. However, these wallets are susceptible to key theft (i.e., malware [93], keyloggers [3, 94]). Possible mitigation of these attacks are hardware wallets displaying transactions to the user, while the user confirms signing by a button (e.g., Trezor, Ledger, KeepKey). Another option is to protect self-sovereign wallets by multi-factor authentication using multi-signatures (e.g., TrezorMultisig2of3, Eth. MultiSigWallet), threshold-based cryptography [95], or air-gapped OTPs [92].

6.2 Oracles

Oracles are trusted entities that provide data reflecting the state of the world beyond the blockchain. *Prediction markets* (e.g., Augur, Gnosis) were created for the purpose of trading the outcome of events – individuals are incentivized to accurately wager on these outcomes, which serve as data feeds. *Dedicated data feeds* [96, 97] build on existing blockchain platforms or create dedicated oracle networks (e.g., ChainLink, Witnet) that internally run consensus protocol.

Security Threats

The provision time of prediction markets may be long for many applications and the provided set of data events may be also limited. In contrast, dedicated data feeds enrich a data domain and significantly shorten a provision time; however, they often rely on a trusted party [96, 97], which may misbehave or accidentally produce wrong data. Oracle networks eliminate trust in a single party by a consensus of the group; however, threats related to the consensus layer of this functionality also needs to be considered. Moreover, for providers that offer authenticated data feeds using trusted hardware [97, 98], a vulnerability in trusted hardware may result in a compromise of the entire data feed.

6.3 Decentralized Filesystems (DFs)

DFs serve as a data storage infrastructure running native blockchains (e.g., Storj [99], Filecoin [19], Permacoin [18]). DFs borrow ideas from peer-to-peer file storage systems, but they additionally incentivize data preservation by crypto-tokens. Alternatively to native DFs, decoupling of the stored data from the blockchain data is also possible in a few forms of integration with existing blockchains. Beside naïve storage of integrity proofs to off-chain data, cloud services (e.g., Amazon Web Services, Google Cloud, IBM), and distributed hash tables (DHT) [100] are promising approaches.

Security Threats and Mitigations

While native DFs handle availability and decentralization using consensus layer mechanisms, cloud services and DHT solutions rely on a provider's infrastructure and dedicated file sharing networks, respectively. Sybil attacks claiming redundant storage of the same piece of data can be prevented by unique encryption of each data copy [99]. Another attack might target the reputation of the network by dropping data and its redundant copies. A simple mitigation technique is to use multiple consensus nodes for a file upload, which diminishes chances of the attack being successful. Next mitigation is to hide the number of redundant copies using erasure encoding [99].

7 Conclusion

In this paper, we try to systematize the knowledge about security aspects of blockchain systems. We proposed a stack-modeled security reference architecture, which we further projected into a threat-risk assessment model that presents various threats and countermeasures. The stacked model consists of four layers. At each of the layers, we surveyed specific security issues and mitigation techniques. In future work, we plan to amend the security issues of each layer by details and evidence about real-world incidents.

References

[1] Mauro Conti et al. „A survey on security and privacy issues of bitcoin“. In: *IEEE Communications Surveys & Tutorials* 20.4 (2018).

[2] Wenbo Wang et al. „A Survey on Consensus Mechanisms and Mining Management in Blockchain Networks“. In: (2018).

[3] Joseph Bonneau et al. „Sok: Research perspectives and challenges for bitcoin and cryptocurrencies“. In: *IEEE SP*. 2015.

[4] Yossi Gilad et al. „Algorand: Scaling byzantine agreements for cryptocurrencies“. In: *SOSP*. 2017.

[5] Iddo Bentov et al. „Cryptocurrencies without proof of work“. In: *FC*. 2016.

[6] Aggelos Kiayias et al. „Ouroboros: A provably secure proof-of-stake blockchain protocol“. In: *CRYPTO'17*. 2017.

[7] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[8] Ethan Buchman et al. „The latest gossip on BFT consensus“. In: (2018).

[9] Eleftherios Kokoris-Kogias et al. „OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding“. In: *IEEE S&P*. 2018. doi: `10.1109/SP.2018.000-5`. url: `https://doi.org/10.1109/SP.2018.000-5`.

[10] Mahdi Zamani et al. „RapidChain: Scaling Blockchain via Full Sharding“. In: *ACM CCS*. 2018. doi: `10.1145/3243734.3243853`. url: `https://doi.org/10.1145/3243734.3243853`.

[11] Eli Ben Sasson et al. „Zerocash: Decentralized anonymous payments from bitcoin“. In: *IEEE SP*. 2014.

[12] John R Douceur. „The sybil attack“. In: *IPTPS*. 2002.

[13] Miguel Castro and Barbara Liskov. „Practical Byzantine Fault Tolerance“. In: *OSDI*. 1999, pp. 173–186.

[14] Alysson Bessani et al. „State Machine Replication for the Masses with BFT-SMART“. In: *IEEE/IFIP DSN*. 2014.

[15] Andrew Miller et al. „The honeybadger of BFT protocols“. In: *ACM CCS*. 2016.

[16] Stefan Dziembowski et al. „Proofs of Space“. In: *CRYPTO'15*. 2015.

[17] P4Titan. *Slimcoin: A peer-to-peer crypto-currency with proof-of-burn*. Tech. rep. 2014.

[18] Andrew Miller et al. „Permacoin: Repurposing bitcoin work for data preservation“. In: *IEEE S&P*. 2014.

[19] Protocol Labs. *Filecoin: A decentralized storage network.* Tech. rep. 2017.

[20] Common Criteria. *Common Criteria for Information Technology Security Evaluation.* Tech. rep. 2017. URL: `https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf`.

[21] Ivan Homoliak et al. „Insight Into Insiders and IT: A Survey of Insider Threat Taxonomies, Analysis, Modeling, and Countermeasures“. In: *ACM CSUR* 52.2 (2019). ISSN: 0360-0300. DOI: `10.1145/3303771`. URL: `http://doi.acm.org/10.1145/3303771`.

[22] Warwick Ashford. *Corporate networks vulnerable to insider attacks, report finds.* `http://bit.ly/2WdPGFE`. 2018.

[23] Rodrigo Rodrigues and Peter Druschel. „Peer-to-peer Systems“. In: *Commun. ACM* 53.10 (2010). ISSN: 0001-0782. DOI: `10.1145/1831407.1831427`. URL: `http://doi.acm.org/10.1145/1831407.1831427`.

[24] David S. H. Rosenthal et al. „Notes On The Design Of An Internet Adversary“. In: *CoRR* cs.DL/0411078 (2004). URL: `http://arxiv.org/abs/cs.DL/0411078`.

[25] ISC. *BIND 9 Security Vulnerability Matrix.* `http://bit.ly/2Z1PiMj`. 2019.

[26] Maria Apostolaki et al. „Hijacking bitcoin: Routing attacks on cryptocurrencies“. In: *IEEE SP*. 2017.

[27] Maria Apostolaki et al. „SABRE: Protecting Bitcoin against Routing Attacks“. In: *arXiv preprint arXiv:1808.06254* (2018).

[28] Matt Lepinski and K Sriram. *BGPSEC protocol specification.* Tech. rep. 2017.

[29] Ethan Heilman et al. „Eclipse Attacks on Bitcoin’s Peer-to-Peer Network.“ In: *USENIX Security*. 2015.

[30] Karl Wüst and Arthur Gervais. *Ethereum eclipse attacks.* Tech. rep. 2016.

[31] Yuval Marcus et al. „Low-Resource Eclipse Attacks on Ethereum’s Peer-to-Peer Network.“ In: 2018 (2018).

[32] Eyal Arazi. *Choosing the Right DDoS Solution (Part 4): Hybrid Protection.* `http://bit.ly/2XoB8V9`. 2018.

[33] Sergio Demian Lerner. *New DoS Vuln by Forcing Continuous Hard Disk Seek/Read Activity.* `http://bit.ly/2KnCA6u`. 2019.

[34] Andrew Miller et al. „Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions“. In: *ACM CCS*. 2015.

[35] Vitalik Buterin and Virgil Griffith. „Casper the friendly finality gadget“. In: (2017).

[36] Phil Daian et al. „Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake“. In: *Iacr*. 2017.

[37] Alex Boverman. *Timejacking & Bitcoin*. http://bit.ly/2WKAnbM. 2011.

[38] Pawel Szalachowski. „(Short Paper) Towards More Reliable Bitcoin Timestamps“. In: *IEEE CVCBT*. 2018.

[39] Hyperledger team. *Hyperledger Architecture, Vol. 1: Consensus*. 2017. URL: `https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf`.

[40] Sunoo Park et al. *Spacecoin: A cryptocurrency based on proofs of space.* Tech. rep. 2015.

[41] Trond Hønsi. „SpaceMint: A Cryptocurrency Based on Proofs of Space“. MA thesis. NTNU, 2017.

[42] Ren Zhang and Bart Preneel. „Lay Down the Common Metrics: Evaluating Proof-of-Work Consensus Protocols’ Security.“ In: *IEEE S&P*. 2019.

[43] Ittay Eyal and Emin Gün Sirer. „Majority is not enough: Bitcoin mining is vulnerable“. In: *Communications of the ACM* 61.7 (2018).

[44] Ayelet Sapirshtein et al. „Optimal selfish mining strategies in Bitcoin“. In: *FC*. 2016.

[45] Kartik Nayak et al. „Stubborn mining: Generalizing selfish mining and combining with an eclipse attack“. In: *IEEE EuroSP*. 2016.

[46] Alexei Zamyatin et al. *Flux: Revisiting Near Blocks for Proof-of-Work Blockchains*. `https://eprint.iacr.org/2018/415/20180529:172206`. 2018.

[47] Rafael Pass and Elaine Shi. „Fruitchains: A fair blockchain“. In: *PODC*. 2017.

[48] Pawel Szalachowski et al. „StrongChain: Transparent and Collaborative Proof-of-Work Consensus“. In: *USENIX Security*. 2019.

[49] Yonatan Sompolinsky and Aviv Zohar. „Accelerating Bitcoin’s Transaction Processing. Fast Money Grows on Trees, Not Chains.“ In: 2013.881 (2013).

[50] Ren Zhang and Bart Preneel. „Publish or perish: A backward-compatible defense against selfish mining in bitcoin“. In: *CT-RSA*. 2017.

[51] Eleftherios Kokoris Kogias et al. „Enhancing bitcoin security and performance with strong consistency via collective signing“. In: *USENIX Security*. 2016.

[52] A. Miller. *Feather-forks*. `http://bit.ly/2JVqsKG`. 2013.

[53] Fred B Schneider. „Implementing fault-tolerant services using the state machine approach: A tutorial“. In: *ACM CSUR* 22.4 (1990).

[54] Christian Cachin and Jonathan A Poritz. „Secure intrusion-tolerant replication on the Internet“. In: *DSN*. 2002.

[55] Christian Cachin and Marko Vukolić. „Blockchain consensus protocols in the wild“. In: *arXiv preprint arXiv:1707.01873* (2017).

[56] Timo Hanke et al. „Dfinity technology overview series, consensus system“. In: *arXiv preprint arXiv:1805.04548* (2018).

[57] Christian Cachin et al. „Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography“. In: *JoC* 18.3 (2005).

[58] Victor Shoup. „Practical threshold signatures“. In: *EUROCRYPT*. 2000.

[59] S. King and S. Nadal. *Ppcoin: Peer-to-peer crypto-currency with proof-of-stake*. Tech. rep. 2012. URL: `https://peercoin.net/assets/paper/peercoin-paper.pdf`.

[60] Iddo Bentov et al. „Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake.“ In: (2014).

[61] Iddo Bentov et al. „Snow White: Provably Secure Proofs of Stake.“ In: (2016).

[62] Wenting Li et al. „Securing PoS blockchain protocols“. In: *DPM*. 2017.

[63] Bernardo David et al. „Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain“. In: *EUROCRYPT*. 2018.

[64] Vitalik Buterin. *Long-range attacks: The serious problem with adaptive proof of work*. `http://bit.ly/2JUYNJE`. 2014.

[65] Peter Gaži et al. „Stake-bleeding attacks on proof-of-stake blockchains“. In: *IEEE CVCBT*. 2018.

[66] Qi Feng et al. „A survey on privacy protection in blockchain system“. In: *Journal of Network and Computer Applications* 126 (2019). ISSN: 1084-8045. DOI: `https://doi.org/10.1016/j.jnca.2018.10.020`. URL: `http://www.sciencedirect.com/science/article/pii/S1084804518303485`.

[67] Alex Biryukov et al. „Deanonymisation of clients in Bitcoin P2P network“. In: *ACM CCS*. 2014.

[68] Ivan Pustogarov. „Deanonymisation techniques for Tor and Bitcoin“. PhD thesis. University of Luxembourg, 2015.

[69] George Kappos et al. „An empirical analysis of anonymity in zcash“. In: *USENIX Security*. 2018.

[70] Shen Noether. „Ring SIgnature Confidential Transactions for Monero.“ In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 1098.

[71] Jan Henrik Ziegeldorf et al. „Secure and anonymous decentralized Bitcoin mixing“. In: *Future Generation Computer Systems* 80 (2018). ISSN: 0167-739X. DOI: `https://doi.org/10.1016/j.future.2016.05.018`. URL: `http://www.sciencedirect.com/science/article/pii/S0167739X16301297`.

[72] Luke Valenta et al. „Blindcoin: Blinded, Accountable Mixes for Bitcoin“. In: *FC*. 2015. ISBN: 978-3-662-48051-9.

[73] Ethan Heilman et al. „Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions“. In: *FC*. 2016.

[74] Pascal Paillier. „Public-Key Cryptosystems Based on Composite Degree Residuosity Classes“. In: *EUROCRYPT'99*. Ed. by Jacques Stern. 1999. ISBN: 978-3-540-48910-8.

[75] Ahmed Kosba et al. „Hawk: The blockchain model of cryptography and privacy-preserving smart contracts“. In: *IEEE S&P*. 2016.

[76] Raymond Cheng et al. „Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution“. In: *arXiv preprint arXiv:1804.05141* (2018).

[77] Guy Zyskind et al. „Enigma: Decentralized computation platform with guaranteed privacy“. In: (2015).

[78] Zeppelin Sollutions. *Serpent Compiler Audit*. `bit.ly/2MrHQbX`. 2017.

[79] Nicola Atzei et al. „A survey of attacks on ethereum smart contracts (SoK)“. In: *POST*. 2017.

[80] Reza M Parizi et al. „Empirical vulnerability analysis of automated smart contracts security testing on blockchains“. In: *CASCON*. 2018.

[81] Sergei Tikhomirov et al. „Smartcheck: Static analysis of ethereum smart contracts“. In: *WETSEB*. 2018.

[82] SmartContractSecurity. *Smart Contract Weakness Classification Registry*. `https://github.com/SmartContractSecurity/SWC-registry/`. 2019.

[83] Maximilian Wohrer and Uwe Zdun. „Smart contracts: Security patterns in the ethereum ecosystem and solidity“. In: *IWBOSE*. 2018.

[84] Bo Jiang et al. „Contractfuzzer: Fuzzing smart contracts for vulnerability detection“. In: *ASE*. 2018.

[85] Everett Hildenbrandt et al. *Kevm: A complete semantics of the ethereum virtual machine*. Tech. rep. 2017.

[86] Petar Tsankov et al. „Securify: Practical security analysis of smart contracts“. In: *ACM CCS*. 2018.

[87] Yi Zhou et al. „Erays: reverse engineering ethereum's opaque smart contracts“. In: *USENIX Security*. 2018.

[88] Ivica Nikolić et al. „Finding the greedy, prodigal, and suicidal contracts at scale“. In: *ACM ACSAC*. 2018.

[89] Johannes Krupp and Christian Rossow. „teether: Gnawing at ethereum to automatically exploit smart contracts“. In: *USENIX Security*. 2018.

[90] MME. *Conceptual Framework for Legal and Risk Assessment of Crypto Tokens*. `http://bit.ly/2WlLed2`. 2018.

[91] Shayan Eskandari et al. „A first look at the usability of bitcoin key management“. In: *arXiv preprint arXiv:1802.04351* (2018).

[92] Ivan Homoliak et al. „An Air-Gapped 2-Factor Authentication for Smart-Contract Wallets“. In: *arXiv preprint arXiv:1812.03598* (2018).

[93] Dell SecureWorks. *Cryptocurrency-Stealing Malware Landscape*. 2015. URL: `http://www.opensource.im/cryptocurrency/cryptocurrency-stealing-malware-landscape-dell-secureworks.php`.

[94] Antony Peyton. *Cyren sounds siren over Bitcoin siphon scam*. 2017. URL: `https://www.bankingtech.com/2017/01/cyren-sounds-siren-over-bitcoin-siphon-scam/`.

[95] Steven Goldfeder et al. *Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme*. 2015.

[96] Juan Guarnizo and Pawel Szalachowski. „PDFS: Practical Data Feed Service for Smart Contracts“. In: *arXiv preprint arXiv:1808.06641* (2018).

[97] Concur Technologies, Inc. *Oraclize Documentation*. `http://bit.ly/2wAR3UK`. 2008.

[98] Fan Zhang et al. „Town crier: An authenticated data feed for smart contracts“. In: *ACM CCS*. 2016.

[99] Shawn Wilkinson et al. „Storj a peer-to-peer cloud storage network“. In: (2014).

[100] Ruinian Li et al. „Blockchain for large-scale internet of things data storage and protection“. In: *IEEE Transactions on Services Computing* (2018).

E-Banking Authentication – Dynamic Password Generators and Hardware Tokens

Ondřej Hujňák, Kamil Malinka, Petr Hanáček

1 Introduction

The banking sector keeps going through continuous digital evolution as the paradigms in the finance sector are shifting. We witnessed the transfer from in-person banking to online e-banking, and this trend continues towards mobile banking. With the emergence of new European directives that affect this area and new approaches to authentication, we decided to review the current state of e-banking authentication options. This paper presents our findings and is based on our recent research of e-banking security[1].

Because the clear trend in authentication is the usage of multi-factor authentication, we present in the first section of our paper an overview of possible combinations of the multi-factor authentication schemes and evaluate their compliance with the newest standard available – the PSD2 directive of the European Union. Moreover we discuss trends in the authentication approaches and outline possible future directions.

In the second part we focus on newly emerged or recently updated authentication methods. Namely we take a closer look at dynamic password generators, which are currently very popular mean of authentication in e-banking systems, and hardware security modules, which are gaining popularity and seem like a logical next step in authentication.

We conclude the paper with a description of FIDO2 open standard for strong authentication, which is directly targeted on web environment and enables usage of advanced authentication options in any web project.

2 Authentication

The classic approach for authentication in digital systems is utilisation of basic memorized secret such as static password or PIN code. Even though we assume the usage of TLS for transport encryption, the method shows the weakest resistance against various attacks and alternative authentication options are actively researched[2]. We can divide the authentication primitives (basic principles that can be used for authentication) into three categories – knowledge, possession and inherence, and every authentication scheme can be seen as a combination or specific use of those primitives. Nowadays, direct implementations of those primitives (methods) are not considered inherently secure and to achieve satisfiable security, modern authentication schemes combine multiple methods. Such schemes are called multi-factor (MFA), where factor denotes an authentication method, and European banks are legally required to use them.

In Table 1, we describe features of common combinations of authentication methods. In the table we put checkmark if the combination of methods satisfies the requirement and cross mark if it doesn't. If the mark is in brackets, the assessment is not unambiguous, in which case we used the more common rating and added the condition in the footnote. In case the feature can be enabled possible by some additional adjustment of the basic scheme, we use 'Opt.' as in the optional feature.

The reader's main takeaway is the quick overview of the possibilities and their features with respect to the PSD2 requirements, which we consider the most advanced in the e-banking area. Those requirements are:

Cloning protection – replication of the authentication primitive is prevented,

Factor independence – breach of one factor (method) does not compromise the reliability of the other,

Dynamic linking – generated authentication code is bound to the transaction and the amount and payee are presented to the user,

Strong Customer Authentication (SCA) – the factors used must satisfy the factor independence condition and consist of at least two categories of primitives out of knowledge, possession and inherence.

For further ease of understanding, we state the category of the given scheme as defined by Frederik Mennes in his SCA requirement analysis[3]. There are four categories based on the segregation of the factors:

1aa (one-app-authentication) describes the e-banking apps with built-in authenticators,

2aa (two-app-authentication) means both authentication and e-banking apps are separate apps,

2da (two-device-authentication) extracts authentication to a separate device,

oob (out-of-band) uses a third party (such as telco service) for authentication.

2.1 Trends in authentication

What we consider interesting is the continuous evolution of a typical e-banking system, especially in the authentication and authorisation area. It moved from *password-based* authentication, over HW tokens and SMS codes, to currently used *Dynamic passwords* generated by mobile applications.

A few years ago, the *hardware tokens* (often called "calculator") were the most spread method for authentication. They represent the first true OTP systems and provided very high security but declined because of the usability constraints as they required users to carry an extra device and manually transfer the generated code into the e-banking system. They were shortly superseded by *SMS codes*, but the situation changed because, unlike SMS, hardware tokens easily satisfy the SCA requirements of PSD2, and the burden of manual code transfer is overcome by the new generation of HW Tokens with NFC/Bluetooth technologies.

The most apparent new thread is the spread of *biometrics* for authentication, which prevailed mostly in smartphones despite they need to be used only in combination with another method. The common use of biometrics is strengthening the KYC (know your customer) process for remote customer verification or device authorisation when using a hardware token, a dynamic password generator or secure enclave.

Secure enclave is a relatively recent addition to the authentication methods, which is covered in Section 4. A secure enclave is usually protected by other authentication methods (password, PIN or biometrics) and, apart from providing key storage and cryptographic operations, is used as a root of trust and checks the integrity of operating systems and applications.

Method combination	**Cloning protection**	**Factor independece**	**Dynamic linking**	**SCA**	**Comment**
SMS	(✓)[2]	✗	Opt.	✗	oob
Password + PIN	✗	✗	✗	✗	
Password + Grid card	✗	✓	✗	(✓)[1]	2da
PKI (private key)protected by password	✓	✗	✓	✗	
HW Token	✓	✓	✓	✓	2da
HW Tokenprotected by PIN	✓	✓	✓	✓	2da
HW Tokenprotected by BIO	✓	✓	✓	✓	2da
Password + SMS	(✓)[2]	✓	Opt.	(✓)[3]	oob
Integrated DPGprotected by password	(✗)[4]	(✗)[5]	✓	(✗)[5]	1aa
Separated DPGprotected by password	(✗)[4]	(✓)[4]	✓	(✓)[4]	2aa
Dynamic password + BIO	(✗)[4]	✓	✓	(✓)[4]	2aa
BIO + Password	✗	✓	✗	✓	
BIO + SMS	(✓)[2]	✓	Opt.	✓	oob
PKI (private key)protected by BIO	(✗)[4]	✗	✓	✗	
Secure Enclaveprotected by BIO	✓	✓	✓	✓	
Secure Enclaveprotected by password	✓	✓	✓	✓	

Table 1: PSD2 Features of authentication methods and their combinations.[1]

[1] Technically could be considered valid, but in reality is not used as such.
[2] Indirectly possible by attacks on telecommunication links such as SIM Swapping and SS7 attacks.
[3] To fulfil SCA requirements, the SMS receiving device have to be independent of the other one where the password is entered. Nowadays, this is difficult to achieve.
[4] Depends on OS capabilities, in case of rooted OS (called jailbreak in iOS) cannot be ensured [4]. E.g. biometrics is cloneable by design, and protection depends on second factor properties.
[5] The satisfiability of factor independence is not decided yet; if it does not satisfy factor independence, it isn't SCA.

In the future, we can expect broader usage of secure hardware and enclaves in both mobile phones and computers, which will support the multi-factor authentication in one device. We can see the trend to rise even now, when Apple utilizes their *Apple Secure Enclave* very tightly in iOS since the iPhone 5 and Windows 11 require a TPM chip to be present. Delegation of authorisation and crypto-operations into a tailored hardware will ease the access hardening and compliance with novel legislations targeted on personal security in digital world.

Another trend we might forecast is the consolidation of identity, as the national (governmental) IDs are digitalised as electronic identity (eID) and such can be used for authentication across various systems including e-banking. The situation in electronic identity varies a lot nowadays, as there are countries, where the governmental eID is used in e-banking (Belgian itsme, Estonian Mobiil-ID) while in other countries banks are the identity providers, and e-government services use e-banking identity (Norwegian Mobile BankID, Swedish BankID, Czech BankID). In this case the trend in e-banking authentication will follow the eID option and we can see two clear paths being taken in the world. Either the eID and physical ID card will be bound together and the physical ID card will be used for authentication as a hardware token. Or the identities will be decoupled, which will allow the eID to be transferred digitally and utilised in secure enclaves.

3 Dynamic password generators

Dynamic password generators (DPG) are systems, which generate one-time passwords (OTP) – a special authentication primitive, which is valid only for one authentication process and can never be reused.

The modern trend in OTPs is the usage of mobile applications as DPGs, where this application simulates HW tokens in SW and often implements a challenge-response protocol. As the DPG is an authentication method based on OTP primitive, it is a single-factor and is currently used in authentication schemes either bundled with e-banking application, or decoupled. When bundled, it is crucial that the application uses a HSM or enclave in order to satisfy the factor independence. Because the presence of HSM/enclave cannot be ensured on all user devices, banks currently prefer the decoupled variant, where they argue that the factor independence is assured by the mobile operating system and its process isolation.

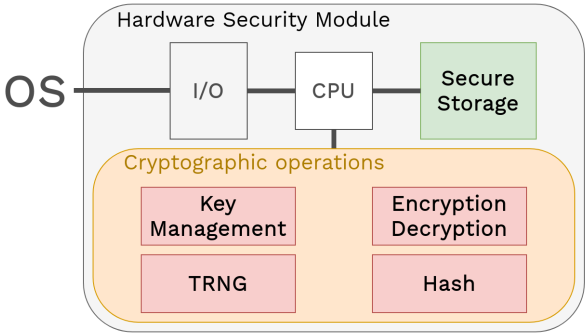


Figure 1: High level Hardware Security Module Architecture.[1]

4 Hardware security module and enclaves

Hardware security module (HSM) is a hardware module equipped with a microprocessor containing some security relevant data (keys) and algorithms for manipulating them (see Figure 1). This specialised hardware ensures both logical security by isolating such data from the system and hardware security as these modules are designed to be tamper-proof. Its features can be used for two authentication-related operations:

- cryptographic operations and private key store (HSM never reveals the key),
- operating system integrity verification (attestation).

Such a module can be either integral part of a more robust system (standalone chip on motherboard of a computer, or a smartphone), or can be equipped with some kind of interface (USB, NFC, Bluetooth) and used standalone in a form of HW token device (such as U2F/FIDO keys) or smart card.

Secure enclave implements a trusted execution environment (TEE) concept, where an application is being run in an isolated environment. The application code is isolated from the operating system and memory is encrypted. A secure enclave guarantees confidentiality, integrity, and security for the application running within it [5]. The main difference when compared with HSMs is that enclaves allow us to run arbitrary operations (opposed to a very specific set of operations in HSM) within the device

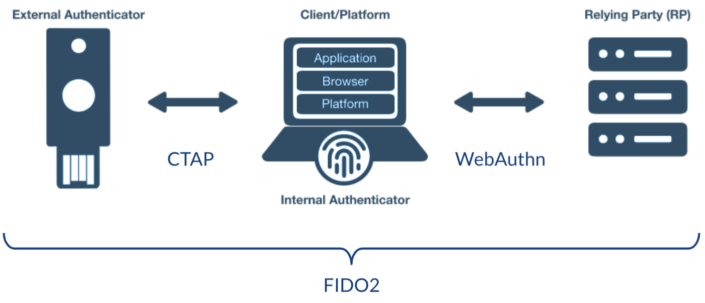


Figure 2: FIDO2 standard protocol scheme.[6]

while maintaining a high level of trust and security. Examples of secure enclave technology are Intel SGX, ARM TrustZone, and AMD Memory Encryption Technology.

5 Fast IDentity Online (FIDO)

FIDO2 ("Fast IDentity Online") is an open authentication standard, maintained by the FIDO Alliance, that consists of the W3C Web Authentication (WebAuthn) specification defining the javascript interface and the Client to Authentication Protocol (CTAP) controlling the connection between an authenticator and the platform (see Figure 2). FIDO2 is the successor of the U2F (Universal 2$^{\text{nd}}$ Factor) protocol that enables internet users to securely access any number of online services with a security key, and maintains backward compatibility.

It is important to note, that FIDO2 standard is not web specific. Bindings exist for various programming languages such as Java, Python and Go. Moreover FIDO2 can enable the usage of authenticators for various tasks on all major operating systems, such as integration with Windows Hello for log in, PAM log in or LUKS2 integration at Linux or even log in on MacOS.

5.1 WebAuthn

WebAuthn defines an API enabling the creation and use of strong, attested, scoped, public key-based credentials by web applications, for the purpose of strongly authenticating users. It is currently supported by all major browsers via javascript `navigator.credentials` object with two relevant methods corresponding to two phases:

- `create()` either registers a new account or associates a new asymmetric key pair credentials with an existing account.
- `get()` uses an existing set of credentials to authenticate to a service, either logging a user in or as a form of second-factor authentication.

First phase is the creation of the new credentials on some device (including private key) and registration of the public key for verification. The function `create` takes a `CredentialCreationOptions` dictionary as a parameter and all the options are set within. Notably *nonce* for a challenge-response part, identification of the *user* and *relying party*, *authenticator restrictions* and required *attestation type*. The registration code can be seen in Listing 1.

The `credential` object from response then contains its *type* (public-key in our case), *credentialId*, *client data* and optionally the *attestation certificate* to verify the source of the credentials.

Listing 1: WebAuthn registration

```
const credentialCreationOptions = {
    challenge: Uint8Array, // nonce
    rp: {
        name: String,
        id: String,
    },
    user: {
        id: Uint8Array, // random, stored in
                        // the authenticator
        name: String,
        displayName: String,
    },
    pubKeyCredParams: [{alg: Number,
                        type: "public-key"}],
```

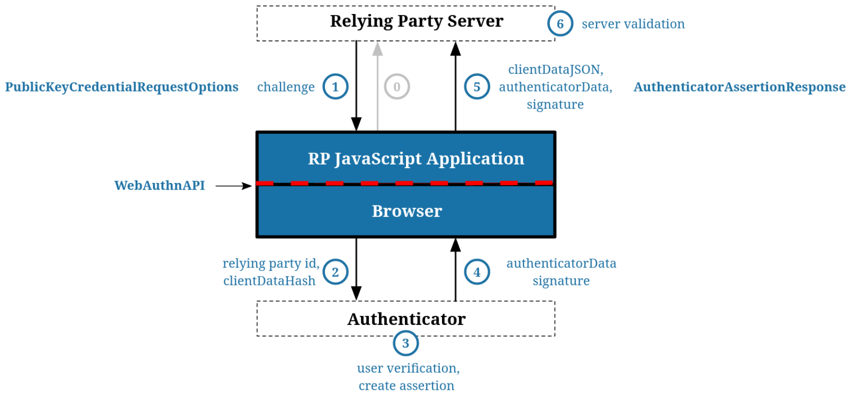

Relying Party Server
6
server validation
PublicKeyCredentialRequestOptions
challenge
1
0
5
clientDataJSON, authenticatorData, signature
AuthenticatorAssertionResponse
RP JavaScript Application
WebAuthnAPI
Browser
relying party id, clientDataHash
2
4
authenticatorData signature
Authenticator
3
user verification, create assertion

Figure 3: User authentication via WebAuthn.[7]

```
    authenticatorSelection: {
        authenticatorAttachment: "cross-platform",
        // "cross-platfor" or "platform"
    },
    timeout: Number,
    attestation: String
    // "none", "indirect" or "direct"
};

const credential=await navigator.credentials.create({
    publicKey: credentialCreationOptions
});
```

The authentication of a registered user is then handled through the `get` function, which retrieves the client identity as depicted by Figure 3. The function accepts a `CredentialRequestOptions` dictionary as a parameter and within it it sets mainly the new challenge *nonce* and the *credentialId*, which was retrieved during registration is passed in here. It can also optionally limit the transports, which can be used for authenticator connection via CTAP. The overview of the call can be seen in the Listing 2.

The response contains again the `credential` object, but this time the *client data* are signed with the user private key and *user handle* is returned as well. This allows the verification of the client data.

Listing 2: WebAuthn authentication

```
const publicKeyCredentialRequestOptions = {
    challenge: Uint8Array, // nonce
    allowCredentials: [{
        id: Uint8Array, // credentialId from
                        // credential object
        type: 'public-key',
        transports: ['usb', 'ble', 'nfc'],// optional
    }],
    timeout: Number,
}

const assertion = await navigator.credentials.get({
    publicKey: publicKeyCredentialRequestOptions
});
```

The compatible authenticators for the use with WebAuthn can be implemented in software running either on device (called platform authenticators by the standard) or off device (called roaming authenticators) accessible over some transport (such as USB, Bluetooth or NFC).

5.2 CTAP

The Client to Authenticator Protocol (CTAP) is a complementary protocol to WebAuthn API. The application developers do not interact with it directly, but via the WebAuthn interface and it enables the roaming (external), user-controlled cryptographic authenticator (such as a smartphone or a hardware security key) to interoperate with a client platform such as a laptop. The CTAP specification actually refers to two protocol versions, the CTAP1/U2F protocol and the CTAP2 protocol, where an authenticator implementing CTAP2 protocol is usually called FIDO2 authenticator.

6 Conclusion

In the paper, we provide the reader with analysis of possible multi-factor authentication method combinations with respect to PSD2 requirements. We covered the trends in e-banking authentication with clear display of current schemes and probable future directions.

The reader should be familiar with dynamic password generators, their features and shortcomings. We presented the hardware security module concept and its current implementations and usage. Lastly we provided a thorough overview of concurrent standard for strong authentication in web environment – FIDO2.

References

[1] Kamil Malinka et al. „E-Banking Security Study—10 Years Later“. In: *IEEE Access* 10 (2022), pp. 16681–16699. DOI: `10.1109/ACCESS.2022.3149475`.

[2] Joseph Bonneau et al. „The quest to replace passwords: A framework for comparative evaluation of web authentication schemes“. In: *2012 IEEE Symposium on Security and Privacy*. IEEE. 2012, pp. 553–567.

[3] Fredrik Mennes. *PSD2: Which Strong Authentication and Risk Analysis Solutions comply with the EBA's Final Draft RTS?* Accessed on Nov. 12, 2020. Apr. 2017. URL: `https://frederikmennes.wordpress.com/2017/04/19/psd2-which-strong-authentication-and-risk-analysis-solutions-comply-with-the-ebas-final-draft-rts/`.

[4] Vincent Haupert and Tilo Müller. „On App-based Matrix Code Authentication in Online Banking“. In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP*. 2018, pp. 149–160. ISBN: 978-989-758-282-0. DOI: `10.5220/0006650501490160`.

[5] Nishank Vaish. *Why are Enclaves Taking Over the Security World?* Accessed on Dec. 7, 2020. July 2019. URL: `https://fortanix.com/blog/2019/07/why-are-enclaves-taking-over-security-world/`.

[6] Anna Sinitsyna. *Beyond Passwords: FIDO2 and WebAuthn in Practice.* Accessed on Apr. 20, 2022. Dec. 2019. URL: `https://www.inovex.de/de/blog/fido2-webauthn-in-practice/`.

[7] J.C. Jones et al. *Web Authentication:An API for accessing Public Key Credentials Level 1*. W3C Recommendation. https://www.w3.org/TR/2019/REC-webauthn-1-20190304/. W3C, Mar. 2019.