FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Kritéria pro hodnocení bezpečnosti kryptografických knihoven
Student:	Matěj Douša
Vedoucí:	Ing. Josef Kokeš, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Informační bezpečnost 2021
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Práce bude sloužit jako východisko pro řešení projektu Bezpečné použití kryptografických knihoven. Cílem tedy je vypracovat metodiku, která umožní porovnat různé kryptografické knihovny mezi sebou a doporučit pro dané použití tu nejvhodnější.

1) Seznamte se s problematikou bezpečnosti softwaru - bezpečný návrh, implementace i použití.
2) Po dohodě s vedoucím vyberte 3-4 open-source knihovny realizující kryptografické protokoly. Nastudujte hlavní vlastnosti těchto knihoven.
3) Zaměřte se na technicko-organizační opatření zajišťující bezpečnost zkoumaných knihoven, například pravidla pro přispívání do projektu. Porovnejte zkoumané knihovny mezi sebou a také s OpenSSL.
4) Navrhněte metodiku, jak pro obecnou knihovnu nalézt a vyhodnotit typické chyby, ke kterým při jejím použití dochází. Takovým zdrojem může být např. seznam publikovaných zranitelností (CVE) aplikací, které knihovnu používají. Použijte tuto metodiku na knihovny z předchozích bodů.
5) Na základě předchozích zjištění určete hlavní kritéria, která ovlivňují bezpečnost jak knihovny, tak jejího použití aplikačním vývojářem. Formulujte doporučení pro vývojáře, jak zvolit a použít knihovnu tak, aby výsledná aplikace byla co nejbezpečnější.

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 26. ledna 2024 v Praze.

Bakalářská práce

KRITÉRIA PRO HODNOCENÍ BEZPEČNOSTI KRYPTOGRAFICKÝCH KNIHOVEN

Matěj Douša

Fakulta informačních technologií
Katedra informační bezpečnosti
Vedoucí: Ing. Josef Kokeš, Ph.D.
13. května 2024

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci: Douša Matěj. *Kritéria pro hodnocení bezpečnosti kryptografických knihoven.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Seznam obrázků

Seznam tabulek

Tímto bych rád poděkoval vedoucímu práce Ing. Josefu Kokešovi, Ph.D. za vstřícnost při konzultacích, cenné rady, připomínky a odborné vedení mé práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praha dne 13. května 2024

Abstrakt

Využití kryptografických knihoven je dnes velmi časté, zároveň kvalita těchto knihoven přímo ovlivňuje bezpečnost software, který je používá. Kryptografie se používá, když je potřeba ukrýt data. To bývá vyžadováno jak pro samotné uložení dat, tak pro jejich přenos. Oslabením tohoto ukrývání dat může dojít k úniku. Tato práce se zabývá kryptografickými knihovnami, a to zejména kritérii, která by se dala použít pro hodnocení jejich bezpečnosti. Jsou zde rozebrána kritéria pro hodnocení kvality open-source vývoje, hlavně jeho důrazu na bezpečnost. Dále se práce věnuje použitelnosti knihoven z pohledu vývojářů. Je zkoumáno rozhraní knihoven a jejich dokumentace. Nakonec jsou dohledávány chyby v použití vyskytující se přímo v aplikacích používajících vybrané knihovny. Výsledkem je soubor vyzkoušených kritérií, která jednak slouží k ohodnocení spolehlivosti, bezpečnosti a kvality konkrétní knihovny a také k porovnání knihoven mezi sebou.

Klíčová slova bezpečnost, kryptografie, kryptografické knihovny, použití kryptografie, vývoj open-source, návrh API, použitelnost API

Abstract

The use of cryptographic libraries is very common today, at the same time, the quality of these libraries greatly affects the security of the software that uses them. Cryptography is used when there is a need to hide data. This is required both for data storage and for data transmission. Weakening this data security can lead to leakage. This work studies the cryptographic libraries, and in particular the criteria that could be used for evaluation of their security. Criteria for evaluating the quality of open-source development, especially its emphasis on security, are discussed here. Furthermore, the usability of libraries from the point of view of developers is studied. The interface of libraries and their documentation is examined. Finally, usage mistakes found directly in applications using the selected libraries are traced. The result is a set of tested criteria, which serve both to evaluate the reliability, security and quality of a particular library, and also to compare libraries with each other.

Keywords security, cryptography, cryptographic libraries, use of cryptography, open-source developement, API design, API usability

Seznam zkratek

AES	Advanced Encryption Standard
API	Application Programming Interface
CVE	Common Vulnerabilities and Exposures
DNS	Domain Name System
DoS	Denial of Service
ECB	Electronic Code Book
FIPS	Federal Information Processing Standard
FLOSS	Free/Libre Open Source Software
FOSS	Free Open Source Software
HTTP	HyperText Transfer Protokol
HTTPS	HyperText Tranfer Protokol Secure
IV	Initialization Vector
KDF	Key Derivation Function
OS	Operating System
OSS	OpenSource Software
PR	Pull Request
SSH	Secure Shell
SSL	Secure Socket Layer
TLS	Transport Layer Security

Úvod

Bezpečnost je v dnešní době v aplikacích velmi důležitá. Naprostá většina aplikací různými způsoby pracuje s uživatelskými daty, která je potřeba chránit, a to jednak při přenosu mezi zařízeními po síti, ale také např. při jejich uložení na disk. K docílení potřebné ochrany dat je zapotřebí využití kryptografie. Ta je v dnešních aplikacích velmi často využívána, ovšem při tom se setkáváme s problémy.

V dnešní době se při vývoji software často setkáváme s využíváním knihoven, tedy implementací potřebných funkcionalit od jiného autora. Jeden z hlavních důvodů je zvýšení efektivity vývoje při snížení množství potřebných znalostí, které by bez knihoven byly nutné pro implementaci konkrétních funkcionalit.

V případě kryptografie je tento trend snad ještě častější, protože vlastní implementace kryptografie je složitá, vyžaduje silné znalosti v tomto oboru a zároveň chyby se hůře hledají a mají závažnější následky. Zvlášť v případě kryptografie se doporučuje využít již existující, ověřené implementace v podobě knihoven raději než implementace vlastní — zkušenost ukazuje, že vlastní implementace kryptografie často vede k chybám, a tedy bezpečnostním problémům [1].

Kryptografických knihoven existuje mnoho — to je způsobeno tím, že existují různé knihovny pro různé programovací jazyky, platformy či různá použití. Na bezpečnosti a návrhu kryptografických knihoven ale stojí bezpečnost software vývojářů, kteří se rozhodli těchto knihoven využít [2].

Tato práce se zabývá bezpečností kryptografických knihoven, respektive kritérii, která lze zohlednit při výběru co nejvhodnější knihovny. Problematika je zkoumána na open-source knihovnách, protože u nich je možnost nahlédnout jak do vývoje samotné knihovny, tak do vývoje aplikací, které knihovnu používají. Nejprve jsou prozkoumány již existující práce a jejich závěry v oblastech vývoje open-source a použití kryptografických knihoven, na což navazuje vlastní výzkum. Ve vlastním výzkumu jsou zkoumány vývojové procesy některých knihoven, jakým způsobem se liší a jaký to může mít dopad na bezpečnost knihoven. Dále jsou zkoumány týmy správců knihoven, jejich aktivita a vztah ke knihovně. Podobně jsou rozebrány faktory ovlivňující snadnost použití knihoven, jako jednoduchost jejich rozhraní a jeho dokumentace. Nakonec se práce věnuje porovnání knihoven podle jejich již existujících špatných použití.

Cílem této práce je shrnout kritéria, která by umožňovala open-source kryptografické knihovny mezi sebou porovnat a vybrat co nejvhodnější pro konkrétní využití, tedy pro konkrétní potřeby vývojáře. Tato kritéria by jednak měla vývojáři pomoci s výběrem nejvhodnější knihovny, ale zároveň se také přesvědčit o spolehlivosti konkrétní knihovny — ať už o bezpečnosti aktuálního stavu, či výhled na bezpečnost knihovny do budoucna.

Výsledky této práce byly zahrnuty do projektu pro NÚKIB[1] („Bezpečné použití knihoven typu OpenSource“), stejně jako výsledky prací dalších dvou studentů (Milan Špinka, Kirill Leonov) [3, 4]. Tato problematika byla v rámci projektu řešena dohromady, část práce, zejména čerpání

[1] Národní úřad pro kybernetickou a informační bezpečnost

z dřívějších akademických výzkumů souvisejících s tímto tématem, bude podobná. Téma je to ale široké, každý z členů týmu se zaměřil na trochu odlišný pohled na věc, a hlavně každý prováděl vlastní výzkum na odlišných množinách knihoven. Pro větší šíři získaných informací a závěrů je tedy vhodné prohlédnout si ostatní dvě práce.

Kapitola 1

Aplikace a kryptografie

Tato kapitola se zabývá rolí kryptografických knihoven v aplikacích. Jsou zde rozebrány obecné principy bezpečného vývoje aplikací, tedy jak by měl programátor postupovat při vývoji tak, aby výsledek byl bezpečná aplikace. Dále se zabývá důvody k použití kryptografie v aplikacích, tedy k čemu je kryptografie v aplikacích potřeba, čemu pomáhá a čemu zabraňuje. Také je zde vysvětleno proč je v programování vhodné používat knihovny, a to především ty kryptografické.

1.1 Postup bezpečného vývoje

V procesu vývoje, ať už aplikací, nebo například knihoven, se do kódu mohou dostat chyby. Chybu mohou udělat jak nezkušení, tak zkušení programátoři. To je obecný problém, který buď nelze vyřešit, nebo se to dosud nepodařilo. Z tohoto důvodu je do procesu vývoje zařazeno testování, tedy automatické kontroly kódu a lidská kontrola. To sice může pomoci odhalit některé chyby už při vývoji, ale neexistuje záruka stoprocentně bezchybného výsledku [5]. Jinými slovy, nikdy nelze s jistotou říci, že v software nejsou chyby. Tato práce se dále zabývá především chybami souvisejícími s bezpečností, ostatní chyby (např. funkční chyby typu „pouze nefunguje tlačítko“) se zde dále neřeší.

Chyby, které se týkají bezpečnosti, jsou pro účely této práce takové chyby, které narušují bezpečnost software, tedy z něj dělají software nebezpečný. Takové chyby způsobují zranitelnosti — možnosti napadnutí útočníkem. Bezpečný software chrání důvěrnost, integritu a dostupnost informací [6] (v angličtině CIA — Confidentiality, Integrity and Availability) [7].

Výskyt bezpečnostních chyb je často ovlivněn tím, že bezpečnost jako taková nedostane v procesu vývoje dostatečně vysokou prioritu. V praxi to vypadá tak, že se nejprve implementuje hlavní a vedlejší funkčnost software a bezpečnost až nakonec — a často na ni zbyde málo času. Poté, když jsou potřeba opravy, se kvůli tomu musí měnit návrh nebo větší množství kódu, což je neefektivní a drahé. Lze to omezit dodržováním bezpečnostních zásad, jako např. SD[3]:

- Secure by Design
- Secure by Default
- Secure in Deployment [6]

1.1.1 Secure by Design

Hlavní myšlenkou této zásady je zabývat se bezpečností již ve fázi návrhu software. Návrh by měl brát v potaz několik následujících doporučení:

- dodržování návrhových a programovacích konvencí
- zohlednění aktuálních doporučení v oblasti bezpečnosti od kompetentních organizací
- vyřazení nepoužívaného či zastaralého kódu
- pravidelné školení členů týmu v oblasti bezpečnosti
- alespoň jeden odborník na oblast bezpečnosti v týmu/organizaci
- pravidelné prověřování kódu
- kontrola více očí při úpravě či přidání kódu, chyby opravuje autor

Cílem této zásady je omezit bezpečnostní problémy vzniklé špatným návrhem či nedostatečným zaměřením na bezpečnost při návrhu. [6]

1.1.2 Secure by Default

Tato zásada se zaměřuje na uživatele, jejich běžné chování a správné přípravě software pro uživatele. Naprostá většina uživatelů používá software v základním nastavení — je potřeba, aby toto nastavení bylo bezpečné. Dále se věnuje zmenšení možné zranitelné plochy — je vhodné oddělit méně používané funkce tak, aby bylo možné je doinstalovat později, ale pokud je uživatel nepotřebuje, tak ho neohrožují. Software by také měl zabezpečit všechna data, která jsou zajímavá pro útočníky. Také by software měl vyžadovat pouze nutná oprávnění pro případ ovládnutí útočníkem — tím se omezí dopad takového ovládnutí. [6]

1.1.3 Secure in Deployment

Tato poslední část SD^3 se věnuje fázi, kdy software je již zveřejněný a používaný. Autor by měl poskytovat dostatečnou bezpečnostní podporu pro software:

- vydávání bezpečnostních záplat co nejdříve po objevení
- dostatek kvalitních informací o bezpečném použití
- srozumitelná dokumentace, tipy, chybová hlášení
- možnost uživatelského nastavení bezpečnosti

Tato zásada omezuje problémy vzniklé špatným použitím software, a také řeší, jak zajistit, aby byl software co nejbezpečnější i po zveřejnění uživatelům a v průběhu používání. [6]

1.2 K čemu se využívá kryptografie

Jak již bylo zmíněno výše, bezpečný software musí chránit důvěrnost, integritu a dostupnost informací [6]. K dosažení těchto cílů se využívá kryptografie. Její využití je široké. Když je potřeba ukrýt data, využívá se symetrické šifrování. Když je potřeba navázat na dálku s někým spojení, je využito také asymetrické šifrování. V dnešní době skoro jakákoliv běžně používaná aplikace využívá kryptografii — například přihlašování, otevírání většiny webových stránek, komunikace přes síť — a proto se s nutností využít kryptografii potýká stále více tvůrců software. O možnostech takových vývojářů mluví další část kapitoly.

1.3 Důvod k využití knihoven

Ve chvíli, kdy vývojář do svého software chce (či musí) zahrnout kryptografii, má dvě možnosti. První možnost je implementovat si kryptografické algoritmy sám. Tím se vývojář spolehne pouze sám na sebe a má úplnou kontrolu nad tím, co přesně jeho software dělá. Druhá možnost je využít již existující implementace v podobě knihovny. Tím se vývojář spolehne na jiné vývojáře a věří, že jím využitá knihovna dělá, co má, a že její tvůrce knihovnu důsledně kontroluje a nemá zlé úmysly. Na druhou stranu ušetří hodně času, který by jinak musel věnovat implementaci kryptografie. Která z těchto variant je ale bezpečnější?

Na první pohled může vypadat, že věřit sobě a své implementaci by mohlo vyústit v bezpečnější výsledek. Bohužel, ukazuje se, že konkrétně kryptografie je poměrně hodně složitý obor a je velmi složité si sám správně algoritmy implementovat [1]. Bezpečnější je tedy využít nějaké knihovny, nejlépe známé, používané a ověřené. V tu chvíli může buď spoléhat na jiné vývojáře / bezpečnostní experty, že knihovnu dostatečně ověřili, nebo si sám ověřit její bezpečnost. Samostatné ověření ale obsahuje procházení celého kódu knihovny (pokud je dostupný) a ujištění se u každého řádku, že nikde není chyba. Navíc takové samostatné ověření bude platit pouze pro konkrétní verzi knihovny, protože v další verzi knihovny se kód pravděpodobně změní a bude potřeba buď neměnit používanou verzi (nebezpečné), nebo opětovná kontrola všech změn. To je ale velmi náročné, většina běžných vývojářů na to nemá čas a hlavně schopnosti [8, 9].

Řešení může být jednoduše „přijmout knihovnu za bezpečnou“ — tím se ale vzdáváme jakékoli kontroly nad implementací kryptografie ve svém software. Jako přijatelná varianta tedy vychází najít obecnější, jednodušší kritéria pro výběr a ohodnocení knihovny než kontrolování celého kódu verze a všech nových změn — tím sice nelze zaručit stoprocentní bezpečnost, ale mělo by to snížit množství výskytu bezpečnostních problémů spojených s knihovnou — ať už chyb v knihovně, nebo chyb v jejím použití.

Kapitola 2

Model hrozeb kryptografických knihoven

Tato kapitola se zabývá faktory, kterými se lze zabývat při řešení bezpečnosti kryptografických knihoven. Tyto faktory představují problémy, se kterými se můžou vývojáři aplikací setkat a které mohou poškodit bezpečnost jejich software. Nejprve je potřeba zaměřit se na problémy uvnitř knihovny, tedy problémy, které jsou mimo moc uživatele knihovny opravit u sebe v kódu. Dále se podíváme na problémy s použitím knihovny způsobené nevhodnou prací s rozhraním knihovny, ani ne nutně způsobené neznalostí uživatele jako spíše nevhodným návrhem rozhraní.

2.1 Zranitelná implementace

Jednou z hrozeb pro bezpečnost knihovny jsou chyby v kódu uvnitř knihovny. Tyto chyby mohou způsobit např. špatné fungování kryptografického algoritmu, tedy zjednodušit jeho prolomení, případně zastavit šifrování dat úplně. Autor knihovny může také špatně či slabě nastavit parametry algoritmů, kdy dopadem je opět snížení bezpečnosti. Takové chyby se mohou v knihovně vyskytnout neúmyslně, nebo je někdo může do knihovny podstrčit schválně a tím cílit na oslabení bezpečnosti aplikací používajících knihovnu.

2.1.1 Zadní vrátka

Úmyslná chyba v knihovně může být využita pro napadení aplikací spoléhajících na knihovnu, tedy být úmyslně do knihovny umístěna jako zadní vrátka. Takovéto úmyslné poškození může udělat buď autor sám, nebo v případě open-source externí přispěvatel. Pokud tedy autor zlé úmysly nemá, je na něm, aby bezpečnost kódu v knihovně ohlídal. Takovéto chyby ale nemusí být snadno rozeznatelné od chyb neúmyslných, což je ostatně cílem takových útočníků.

2.1.2 Neúmyslné zranitelnosti

Chyby se do knihovny snadno dostanou neúmyslně. Ať už autor sám udělá chybu, nebo přehlédne či nedostatečně zkontroluje kód přidávaný cizím programátorem, který sám udělal chybu a neví o tom. Obranou, podobně jako výše, musí být dostatečná kontrola kódu.

Jak již bylo řečeno výše, tyto dva typy chyb často není možné rozeznat, proto nadále budou považovány za totéž.

2.2 Zranitelné použití

Další hrozba pro bezpečnost kryptografických knihoven je jejich špatné použití. Správné použití knihovny by sice mělo být starostí jejího uživatele (vývojáře, který ji používá), ale ukazuje se (více v kapitole 5), že použití kryptografie není pro běžného vývojáře jednoduché a knihovna tento problém může omezit jednoduchým a přímočarým rozhraním, rozumnými výchozími hodnotami a kvalitní dokumentací.

Kapitola 3

Open-source a closed-source

Pro účely této práce je zde vysvětleno, co je to vlastně open-source (otevřený) a closed-source (uzavřený) software. Dále jsou zde shrnuty rozdíly v bezpečnosti těchto dvou typů.

3.1 Open-source

Open-Source Software (OSS) je software, který má otevřený zdrojový kód. Otevřený kód v širším pojetí znamená veřejně dostupný — úmyslně publikovaný kód. Existuje definice open-source softwaru od *Open Source Initiative* [10], která pojem zužuje na software s otevřeným zdrojovým kódem, který navíc ale splňuje další podmínky, z nichž nejdůležitější je licence, pod kterou software musí být publikován. Tato licence musí mimo jiné umožňovat bezplatnou redistribuci a možnost modifikace. Software, který tato kritéria splňuje, se pak označuje jako FOSS (Free Open-Source Software) nebo FLOSS (Free/Libre Open-Source Software), a jeho hlavní vlastností je jeho „svoboda“, kterou ostatní OSS mohou postrádat právě kvůli přísnějším licencím.

Svoboda je jedním z hlavních důvodů rozšířenosti open-source softwaru, protože jeho využití (a modifikace) nic nestojí. Zvlášť pokud jde o široce používaný software, tak je jednoduché osvojit si jeho použití v rozumně krátké době. Jako problém se ale může projevit to, že se uživatelé open-source musí spolehnout na správce tohoto software s důvěrou, že bude nadále tento software zodpovědně udržovat. [2]

Tato údržba úzce souvisí s procesy vývoje open-source, které ovšem pod žádnou definici nespadají [11]. Často, avšak ne vždy, se zde vyskytuje vývojový model „tržiště“ [12], což znamená, že upravovat originální verzi software (přispívat svým kódem do originálního repozitáře) může kdokoliv a odkudkoliv. Naproti tomuto modelu stojí model „katedrála“ [12], kde je vývojové prostředí uzavřenější a o software se stará malá skupina vývojářů.

3.2 Closed-source

Jako *closed-source* se označuje software, který není open-source, tedy jeho zdrojový kód není veřejně dostupný. Mezi jeho další vlastnosti patří větší přísnost licencí, často zpoplatněné použití a je dostupný pouze v podobě spustitelného binárního souboru [11]. Skrytý zdrojový kód ovšem neznamená, že nikdo nemá možnost zjistit, jak program funguje. Z binárních souborů lze tyto informace vyčíst (jedná se vlastně jen o kód nižší úrovně abstrakce, pořád jde ale o instrukce programu) různými způsoby, ať už bez pomoci nástrojů (jsou potřeba hluboké znalosti), nebo s nástroji specializovanými na tuto činnost, které umí částečně z binárních souborů sestavit zpátky kód vyšší úrovně abstrakce (pořád je potřeba hodně znalostí v tomto oboru a ze strany tvůrce closed-source software to lze značně ztížit).

V closed-source vývoji neuvidíme model „tržiště“ hlavně z toho důvodu, že k tak široké spolupráci je potřeba právě veřejná dostupnost zdrojových kódů.

3.3 Bezpečnostní rozdíly

V open-source vývoji, především právě ve výše zmíněném tržištním modelu, lze vidět jak výhody, tak i nevýhody. Jednou z hlavních nevýhod může být právě spolupráce velkého množství lidí z různých míst, kteří se jednak pravděpodobně neznají a také se mnohem složitěji ověřují jejich znalosti. Znalosti kryptografie hrají v bezpečnosti jejího použití zásadní roli a ve vývoji open-source se mezi vývojáři mohou jejich úrovně znalostí značně lišit [13].

Rozdíly v bezpečnosti mezi open-source a closed-source se zkoumaly již v počátcích velké expanze open-source, a to již na počátku tohoto století [12]. Existuje více různých akademických porovnání bezpečnosti těchto dvou typů software, ať už z té doby, nebo novější [14]. Jeden ze starších materiálů s názvem *Katedrála a tržiště* (v angličtině *The cathedral and the bazaar*) popisuje jednak různé typy vývojových procesů („tržiště“ typické pro dnešní open-source a „katedrála“ typická pro closed-source), ale také vysvětluje pravidlo zvané „Linusův zákon“, které vysvětluje úspěšnost open-source s tržištním vývojem tak, že „pokud máte dostatek očí, všechny chyby jsou průhledné“ [12]. Jinými slovy, čím více lidí se do vývoje zapojí, tím větší je šance, že někdo chybu najde a že ji někdo opraví.

Proti tomu ale stojí názor, který toto pravidlo rozporuje a poukazuje na to, že samotný fakt, že kód je dostupný všem, ještě automaticky neznamená, že se najdou odborníci kteří kód zrevidují [15]. Některé studie také nalézají problémy v samotné spolupráci většího množství lidí kteří se neznají, a poukazují na chyby vzniklé právě z těchto důvodů [16]. Navíc, v procesu kontroly kódu dokonce může být i kontrola prováděná větším množstvím lidí kontraproduktivní, protože může dojít k „difúzi zodpovědnosti“, tedy k pocitu, že kontrola již není potřeba, protože revize již byla provedena někým jiným [16]. Motivace pro kontrolu kódu ze stran programátorů či analytiků s „dobrými úmysly“ tedy nemusí být nijak vysoká, naproti tomu například motivace potenciálních útočníků, kteří by software chtěli napadnout, může být zásadně vyšší, ať už pro možný zisk, nebo jiné jejich důvody. Na druhou stranu, closed-source nezvyšuje bezpečnost samotným ukrytím kódu. Dle [15] je to prakticky totéž jako problém *bezpečnost skrze neznalost* (v angličtině *Security through obscurity*) známý již přes 140 let [17, 18].

Oba protichůdné názory na open-source mají zřejmě částečně pravdu. Z různých výzkumů na toto téma nevyplývá, že by měl open-source být obecně více nebo méně bezpečný než closed-source. Dle [11] „se obě strany debaty shodují na tom, že otevřený kód usnadňuje nalézání zranitelností, pouze se neshodují v závěru, jaký dopad to má na bezpečnost“. Vývojové procesy open-source a jejich bezpečnost jsou více rozebrány v kapitole 4.

Kapitola 4
Vývoj knihoven

Tato kapitola se zabývá indikátory bezpečnosti vývoje open-source software. Shrnuje postup hodnocení kryptografických knihoven podle jejich procesů a požadavků na strukturu kódu.

4.1 Bezpečnost open-source — více do detailu

Jak již bylo řečeno v předchozích kapitolách, na bezpečnost open-source oproti closed-source není jednoznačný názor, nebylo ovšem dokázáno, že by open-source měl být lepší nebo horší než closed-source. Open-source je ovšem stále více využíván, dalo by se tedy tvrdit, že byl přes svá rizika přijat jak vývojáři, tak uživateli. Dle průzkumu [19] až 96 % aplikací obsahuje open-source jako svou součást. Vzhledem k takto širokému využití je tedy potřeba se dále jeho bezpečností zabývat, a to i nezávisle na porovnání s closed-source.

Jak již bylo řečeno, open-source velmi často využívá „tržištní“ model vývoje, díky kterému na software mohou spolupracovat vývojáři odkudkoliv, nezávisle na svém vztahu k aplikaci a ke správcům projektu [12]. Takováto rozmanitost týmu s sebou nese ale i rozmanitost vývojových procesů [20] a zároveň různé úrovně znalostí zapojených vývojářů, a to zejména v oblasti bezpečnosti [13]. Do procesu vývoje mimo jiné patří kontrola změn prováděných v repozitáři nebo požadavky na formátování kódu, které takovou kontrolu mohou zjednodušit. Tyto faktory přímo ovlivňují možnost výskytu zranitelností [21]. Je vhodné se tedy zabývat tím, jak lze nastavit vývojový proces open-source tak, aby se co nejvíce omezil výskyt zranitelností ve výsledném software.

Počet zranitelností v aplikacích by se mohl zdát jako jednoduché kritérium pro porovnání bezpečnosti aplikací a jejich procesů, tedy čím více má aplikace objevených zranitelností, tím méně je bezpečná. Tento předpoklad ale neplatí ani prvním zmíněným směrem, ani druhým — málo objevených zranitelností neznamená bezpečnou aplikaci — tedy toto kritérium použít nelze [22].

Již v nultých letech se akademici snažili o predikování výskytu zranitelností ve zdrojových souborech podle jejich vlastností. Jeden takový výzkum zjišťuje, že zranitelnosti se častěji vyskytují v takových souborech, které obsahují více změn v průběhu času. Také se více zranitelností objevuje v novějších souborech či v souborech obsahujících nedávné rozsáhlejší změny [23]. Další podobný výzkum tyto závěry potvrzuje a dodává množinu metrik pro poměrně přesné odhady výskytu zranitelností v souborech [24]. Tento výzkum využívá relativních změn v souborech, tedy např. počet změněných řádků ku počtu všech řádků v souboru.

Dále jsou přidávány další faktory, které souvisí se soubory obsahujícími zranitelnosti: složitost kódu a aktivita vývojářů [21]. Závěry tohoto materiálu popisují tři metriky, které lze poměrně hodně spolehlivě použít pro určení souborů, které pravděpodobněji obsahují zranitelnost: met-

rika změn v souboru, metrika aktivity vývojářů a kombinovaná metrika (složitost kódu, změny a aktivita vývojářů).

- **Změny v kódu:** jak již bylo řečeno, vyšší počet změn v souboru ukazuje na vyšší šanci, že soubor obsahuje zranitelnost.
- **Aktivita vývojářů:** tato metrika říká jednak to, že více zranitelností je v souborech upravovaných vývojáři, kteří nejsou zvyklí pracovat spolu na jednom zdrojovém souboru (mají málo či žádné dřívější příspěvky do stejného souboru) a také to, že se zranitelnosti objevují v souborech, kam přispívají vývojáři, kteří přispívají zároveň do mnoha jiných souborů.
- **Kombinovaná metrika:** zde se kombinují předchozí dvě metriky spolu se složitostí kódu — tedy soubor se složitějším kódem, který splňuje navíc předchozí metriky, je velmi pravděpodobným místem výskytu zranitelností.

Ze stejného výzkumu nevyplývá, jak dobrá metrika je složitost kódu samotná, pouze že je méně spolehlivá než výše zmíněné ostatní metriky.

Ve velkém měřítku byl problém prozkoumán ve výzkumu [16], který si dal za cíl ověřit platnost dříve zmíněného Linusova zákona („pokud máte dostatek očí, všechny chyby jsou průhledné“) [12]. Zde se na projektu Chromium testovaly různé hypotézy související s Linusovým zákonem a výsledky jsou překvapivé. Kromě zjištění z předchozích výzkumů, která byla opět potvrzena, zde byly nasledující zajímavé objevy:

- **Kontrola více lidí:** Kupodivu když více lidí kontroluje stejný kód (přepočteno na počet řádků kódu), tak je vyšší pravděpodobnost výskytu zranitelnosti, a to především při kontrole 3 a více lidí. Tento jev si autoři vysvětlují tím, že zranitelnosti je na rozdíl od klasických chyb snazší přehlédnout, a také při více lidech dochází k takzvané „apatii kolemjdoucího“, neboli difúzi zodpovědnosti, tedy spoléhání se na ostatní, že možné zranitelnosti najdou.
- **Rychlost kontroly:** Dalším zkoumaným jevem byl podíl „rychlých“ revizí kódu, tedy revizí, které zkontrolovaly v průměru více než 200 řádků kódu za hodinu (autory zvolená hranice). Čas revize byl počítán jako čas od otevření revize do jejího schválení. Výsledek ukázal, že u zranitelných souborů byl nižší počet těchto rychlých revizí než u ostatních — což bylo v rozporu s předpokladem, který vycházel z jiné studie [25], který tvrdil, že rychleji (tedy méně důkladně) kontrolované soubory jsou náchylnější k chybám. Tato studie se ovšem zabývala chybami obecně a ne jenom zranitelnostmi, autoři si tento jev vysvětlují právě odlišnou povahou zranitelností od klasických defektů — je snazší je přehlédnout bez dostatečných znalostí a zkušeností v této oblasti.
- **Schvalovatelé změn:** Soubory se zranitelnostmi sice celkově kontrolovalo více lidí, celkově méně z nich bylo zkušených v oblasti bezpečnosti než u neutrálních souborů. Tato část tedy říká, že na odhalení zranitelností jsou potřeba vývojáři, kteří s tím již mají zkušenosti. Vývojáři ovšem často dostatečně zkušení nejsou, dle [13] a také [15].
- **Vazby mezi vývojáři:** Zranitelné soubory častěji revidovali lidé, kteří v minulosti autorův kód kontrolovali méně — tedy lepší „vzájemná znalost“ vývojářů snižuje riziko neodhalených zranitelností. Dle autorů tyto vzájemné zkušenosti s revizemi zlepší komunikaci mezi vývojáři, a tedy i efektivitu a kvalitu jejich spolupráce.

Další zajímavé závěry z jiných studií [26] jsou např., že zranitelným kódem častěji přispějí „zkušenější“ programátoři — tedy takoví, kteří již vícekrát přispívali do daného projektu. Bohužel, samotný počet různých příspěvků zřejmě nekoreluje se schopností odhalit/nenapsat nebezpečný kód. Dále se ale ukazuje, že kód méně zkušených vývojářů bývá častěji zranitelný než kód ostatních a také že zranitelnost objeví většinou programátor zkušenější než autor kódu [26]. Je

zde tedy opět vidět rozdíl mezi zkušeností se samotným psaním kódu a zkušeností se zranitelným kódem.

Předešlé závěry sice zpochybňují platnost Linusova zákona v určitých oblastech (například samotný vyšší počet lidí při kontrole kódu neznamená vyšší šanci na nalezení zranitelnosti), podporují jej ale v jiných (například ve větším počtu lidí se spíše najde někdo zkušený, kdo je schopný zranitelnosti nalézat a opravovat). To nám vlastně potvrzuje „neporovnatelnost“ open-source s closed-source, každý typ software je jednoduše jiný a nese s sebou jiná rizika. Je velmi těžké, možná nemožné, říci, že jsou některá větší než jiná.

Velké pozornosti se procesům vývoje open-source software dostalo v roce 2014 a po něm, kdy byla objevena zranitelnost v OpenSSL zvaná „Heartbleed“, která byla způsobena chybou v implementaci protokolu SSL/TLS, konkrétně v rozšíření „Heartbeat“. Tato zranitelnost zasáhla velmi široké spektrum aplikací a služeb — už jenom open-source webové servery Apache a Nginx, které využívaly v té době zranitelné verze OpenSSL, byly používány 66 % webových stránek [27]. Dle dalších výzkumů bylo zasaženo až 55 % nejpopulárnějších webových stránek využívajících HTTPS [28].

V návaznosti na objevení tohoto problému byly v organizaci a vývojových procesech OpenSSL udělány zásadní změny [22]. Problémy v knihovně před objevením zranitelnosti „Heatbleed“ byly následující:

- nízká aktivita, pouze dva hlavní vývojáři
- objemný, složitý a špatně spravovatelný kód
- nekonzistentní styl psaní kódu
- nedefinované procesy jako plán vydávání verzí nebo reakce na zranitelnosti
- neměřilo se pokrytí kódu testy

V reakci na Heartbleed byly ovšem tyto problémy nalezeny a odstraněny. Z dvoučlenného týmu hlavních vývojářů se tým rozrostl na 15 lidí, z čehož 4 byli zaměstnanci na plný úvazek. Tím byla docílena dostatečná motivace pro aktivitu ve vývoji, kontrole kódu a rychlé reakci na zranitelnosti. Nějakou dobu po Heartbleedu se vývojový tým zaměřil na problémy s kódem — kód byl zredukován, byla velmi snížena jeho složitost, byl zpřehledněn a byly vymýceny nekonzistence ve stylu psaní. Postupně byly popsány procesy — jak reakce na zranitelnosti nebo plán vydávání nových verzí, tak například požadavky na styl psaní kódu, aby se nadále udržoval přehledný. Také se začaly používat nástroje na kontrolu pokrytí kódu testy. [22]

Jednou z dalších reakcí na Heartbleed bylo vytvoření *Best-Practices Badge* organizací CII[1] (později OpenSSF[2]), což je certifikace (doslovný překlad je „odznak“), která udává sadu kritérií, která hodnotí, co jsou dobré praktiky vývoje open-source [29]. OpenSSL tento odznak získala začátkem roku 2016. Před změnami v reakci na Heartbleed knihovna OpenSSL splňovala pouze necelé dvě třetiny kritérií [22]. Kritéria *Best-Practices Badge* se dělí do následujících 6 kategorií:

- **Základní požadavky:** Tato skupina vyžaduje obecné postupy pro samotnou použitelnost vyvíjeného software. Patří sem obsah hlavní stránky projektu, kde je vyžadován rozumný popis určení software, proces poskytování zpětné vazby a nebo například postup přispění vlastním kódem. Dále je vyžadována FLOSS licence na projekt, dokumentace nebo např. podpora HTTPS na hlavních stránkách.
- **Správa změn:** Tato skupina určuje, kde má probíhat vývoj. Je vyžadován veřejně dostupný repozitář s kontrolou verzí a změn — musí být dohledatelné kdo, kdy a jakou změnu provedl. Dále je nutné unikátně označovat verze a důkladně sepisovat seznam změn v dané verzi.

[1] Core Infrastructure Iniciative
[2] Open Source Security Foundation

- **Nahlašování chyb:** Pro nahlašování klasických chyb musí být popsaný postup. Zároveň se tým knihovny musí aktivně vyjadřovat jak k nahlášeným chybám, tak k návrhům ke zlepšení. Stejně tak hlášení zranitelností musí být jasně popsané, pokud možno soukromé a úvodní reakce by neměla trvat déle než dva týdny.
- **Kvalita:** Tato část se věnuje zejména udržení kvality kódu. Vyžaduje možnost automatického sestavení software ze zdrojového kódu, automatické testování, dostupný návod, jak si sám spustit testovací sadu, a požadavky na testování nově přidaného kódu.
- **Bezpečnost:** Projekt musí mít alespoň jednoho člena zkušeného v oblasti psaní bezpečného kódu a také člena, který zná běžné chyby způsobující zranitelnosti v druhu software, který projekt vytváří. Dále jsou zde kritéria vyžadující použití bezpečných kryptografických algoritmů prověřených odborníky a kritéria zakazují vlastní implementaci kryptografie pro software, kde to není primární účel. Zároveň se v software nesmí vyskytovat neopravené zveřejněné zranitelnosti starší 60 dnů.
- **Analýza kódu:** Zde jsou vyžadovány nástroje pro analýzu kódu — statická analýza je přímo nutná, dynamická analýza je pouze silně doporučená. [30]

I s těmito doporučeními se ovšem nedá zamezit výskytu dalších zranitelností v open-source, dá se pouze omezit. Například v knihovně GnuTLS byla v roce 2020 nahlášena zranitelnost CVE-2020-13777 [31], která umožňovala pasivní dešifrování komunikace protokolem SSL/TLS. Způsobeno to bylo použitím slabého kryptografického algoritmu.

Dalším zajímavým materiálem od organizace OpenSSF je příručka pro hodnocení open-source software, která vyšla ke konci roku 2023 [32]. Tato příručka je určena pro programátory, kterým má pomoci vyhodnotit, zda konkrétní závislost do svého projektu chtějí přidat, a také zjistit, jak moc rizikové by takové přidání bylo.

Příručka mimo jiné obsahuje následující důležité body:

- **Je možné se závislosti vyhnout?** První bod rovnou vybízí programátora k zamyšlení, zda závislost doopravdy potřebuje. Čím méně je software závislý na cizích produktech, tím méně se musí programátor spoléhat na ostatní, že budou k vývoji přistupovat správně a poctivě [2].
- **Je software aktivně vyvíjený?** Naprostá většina software potřebuje aktivně spravovat a vyvíjet, reagovat na zranitelnosti a vylepšovat se — nevyvíjený software je tedy riziko.
- **Je vidět snaha o vytvoření bezpečného software?** Tento bod radí zaměřit se na to, jakým způsobem — pokud vůbec — se vývojáři snaží o zaručení bezpečnosti svého projektu. Doporučuje se zjistit, zda projekt má dříve zmíněný odznak dobrých praktik, jakým způsobem reagoval na bezpečnostní audity (pokud nějaké byly; údajně je to poměrně vzácný jev), v jakém čase opravuje chyby a především zranitelnosti či jakým způsobem se testuje.
- **Je software jednoduché použít bezpečně?** Je důležité, aby výchozí nastavení bylo bezpečné (více v části 1.1.2 — Secure by Default) a jednoduché ukázky použití vedly k bezpečnému programu. API by mělo být navrženo pro jednoduché bezpečné použití (dále v kapitole 5).
- **Existují instrukce pro hlášení zranitelností?** Nahlásit zranitelnost by mělo být snadné a rychlé, nesmí tomu bránit nedostatečná informovanost o tomto procesu.
- **Je software široce používán?** U více používaného software se pravděpodobně bude více hledět na bezpečnost, mohlo by tedy být lepší využít právě takový produkt.
- **Jakou má software licenci?** Licence sice zdánlivě nemusí souviset s bezpečností, ale projekty bez jasných informací o licenci mohou méně dodržovat dobré praktiky vývoje. Je tedy dobré licenci zkontrolovat u každé součásti software.

Kromě zmíněných doporučení, na které se lze zaměřit při zkoumání open-source vývojových procesů, stojí za zmínku, že analýza pokrytí kódu testy hraje poměrně důležitou úlohu pro testování. Vývojáři chápou procenta pokrytí jako hru a snaží se mít co nejlepší skóre. Je tedy vhodné se v repozitářích podívat, zda je dostupná právě tato analýza, a pokud ano, v jakém stavu pokrytí kódu je [33].

4.2 Předpokládaná kritéria

Předchozí sekce shrnuje, jaké vlastnosti mívají soubory, ve kterých se vyskytují zranitelnosti, co může být příčinou toho, že se zranitelnosti dostanou do vydané verze, a také doporučení pro bezpečný vývoj a návod, jak takový vývoj poznat. Zde následuje souhrn vlastností vývoje a organizace open-source kryptografických knihoven, na které je vhodné se zaměřit, ať už podle předchozí sekce, nebo podle našich vlastních předpokladů. Vlastnosti jsou rozděleny do skupin podle toho jak spolu souvisí z teoretického hlediska a nebo praktického hlediska — některé informace jsou dohledatelné na podobných místech.

- Základní informace o knihovně

  Tato kategorie se zabývá důležitými informacemi, které by o sobě knihovna měla dávat vědět zřetelně a jasně, buď na hlavních stránkách knihovny, nebo v README souboru v kořenu repozitáře. Tyto informace jsou:

  - zamýšlené využití knihovny, k čemu je určena
  - popis procesů pro hlášení chyb, podávání návrhů na zlepšení a přispívání do kódu knihovny
  - popis hlášení zranitelností a způsob, jakým tým knihovny reaguje
  - licence, pod kterou je knihovna publikována

  Dále je důležitý repozitář samotný — knihovna musí pro vývoj používat verzovací systém, musí být jasné, co se kdy stalo v kódu za změny a kdo je udělal. Také by knihovna měla obsahovat „release notes“ pro každou verzi (seznam změn, které verze obsahuje). Ještě je vhodné, pokud knihovna obdržela certifikaci dobrých praktik open-source vývoje.

- Tým knihovny (správce)

  Je potřeba se zabývat také tím, kdo se o knihovnu stará a odkud se vzal / jakou má motivaci. Je vhodné, aby se o knihovnu nestaral jeden samotný člověk — i schopný vývojář si může do kódu sám zanést chybu a přehlédnout ji, je tedy důležité, aby se o knihovnu starali alespoň dva lidé. Dále pro zajištění motivace vývojářů, a tedy vyšší aktivitu a rychlejší reakce, je důležité, aby v týmu knihovny byl alespoň jeden vývojář zaměstnaný na plný úvazek přímo na vývoj knihovny. K tomu jsou potřeba finance — je dobré zjistit, odkud se takové finance berou. Také je vhodné, aby se v týmu vyskytoval alespoň jeden odborník na bezpečný kód a aktivně se zapojoval do revizí kódu. Celkově ale nejdůležitější vlastností týmu musí být aktivita — musí být za poslední dobu vidět aktivní práce na projektu v podobě nových úprav kódu či reakce na založená issues (nahlášené chyby či návrhy ke zlepšení).

- Proces přispívání

  Kromě požadavku na popsání procesu přispívání do knihovny, který byl zmíněn v první kategorii, je nutné se na tento proces detailněji zaměřit. Důležité jsou dvě části, a to kdo schvaluje nové změny a jaké jsou na tyto nové změny požadavky.

  - Změnu musí vždy zkontrolovat a schválit alespoň jeden jiný člověk než její autor.
  - Je důležité, aby změnu vždy prohlédl a schválil alespoň jeden člen týmu knihovny.
  - Není vhodné, aby změny schvalovalo více lidí než dva.

Tyto požadavky by měly zajistit, že kód vidí vždy alespoň čtyři oči a zároveň správci projektu mají kontrolu nad tím, co se v kódu děje. Zároveň by se mělo zamezit možné difúzi zodpovědnosti [16]. Dále mají být definovaná pravidla pro psaní kódu, tedy kódovací konvence, které by měly zaručit dlouhodobou spravovatelnost a srozumitelnost kódu. Také při přidání nového kódu má vývojář doplnit potřebné testy — tyto dva požadavky by měly být vyžadovány při kontrole změn v kódu.

- Bezpečnostní informace

  Tato kategorie řeší kritéria přímo související s aktuální bezpečností knihovny. Nejdůležitější takové kritérium vyžaduje, aby všechny závažné zveřejněné zranitelnosti, které jsou starší než 60 dní, byly v aktuální verzi opraveny. Dále je důležité, aby knihovna sama o sobě říkala, že je bezpečná — toto tvrzení se svým odůvodněním by mělo být dostupné v dokumentaci, případně v repozitáři. Dále je výhodou, pokud knihovna podstoupila v nedávné době bezpečnostní audit. Pokud ano, je důležité, aby všechny nálezy z takového auditu byly opraveny, případně (pokud nemají vysokou závažnost) k nim existovalo vyjádření. Takové audity ale jsou málo časté [32]. Také existují certifikace implementací kryptografie, např. FIPS 140-2 či FIPS 140-3 — je výhodou, pokud knihovna takovou certifikaci obdržela.

- Kontrola kvality kódu

  Knihovna by měla testovat správnost a kvalitu svého kódu. Co lze v repozitáři či dokumentaci hledat, je obecně sada testů, které by měly pokrývat všechnu funkcionalitu knihovny a které by se měly spouštět při každé změně kódu. Dále by knihovna měla používat nástroje na analýzu kódu, a to jak statickou, tak dynamickou. V neposlední řadě lze využívat fuzz testing[3]. Kromě různých testů a nástrojů na analýzu kódu je vhodné, když knihovna má soubory s kódem v přehledné struktuře a jejich obsah není moc složitý a zanořený.

4.3 Aplikace na vybrané kryptografické knihovny

Dále jsou kritéria zkoumána přímo na vybraných open-source kryptografických knihovnách.

4.3.1 OpenSSL

Knihovna OpenSSL je jednou z nejznámějších a historicky nejpoužívanějších kryptografických knihoven. Je vyvíjena již od roku 1998 v jazyce C a její použití je velmi široké — knihovna je robustní a nabízí hodně možností využití, jako je obsáhlé kryptografické API, ověřování a tvorba certifikátů a protokoly SSL/TLS, DTLS a QUIC. [34]

Základní informace

- Knihovna má v repozitáři[4] jasně popsáno své zamýšlené využití.
- Procesy pro hlášení chyb a podávání návrhů jsou popsány.
- Proces přispívání je popsán.
- Hlášení zranitelností je jasně vysvětleno a reakce na ně je popsána.
- Licence je jasně veřejně vystavena (Apache Licence Version 2.0).
- Knihovna využívá verzovací systém (git).

[3] technika testování využívající „předgenerování“ všech možných vstupů
[4] `https://github.com/openssl/openssl`

- Seznamy změn pro konkrétní verze jsou dostupné a existují i pro nejnovější verze.
- Knihovna obdržela odznak dobrých praktik v roce 2016[5].

Tým knihovny

- Dle dokumentace je v týmu, který se stará o správu knihovny, celkem 18 lidí.
- Organizace OpenSSL Software Services platí vývojáře přímo pro práci na OpenSSL [35].
- Finance se získávají jednak od sponzorů (je publikovaný seznam) a dále z poskytování placené podpory knihovny.
- V týmu je více lidí se zkušenostmi s kryptografií a bezpečným kódem, není ale oddělený konkrétní člověk.
- Aktivita:
  - Nejnovější verze knihovny je z dubna 2024 (psáno 25. dubna 2024).
  - Vývoj je velmi aktivní, za poslední čtyři měsíce vzniklo více než 500 pull requestů[6], je ale zvláštní, že existuje necelých 350 neuzavřených PR (pull requestů), z nichž některé jsou staré několik let. Kvůli tomu repozitář vypadá trochu neorganizovaně.
  - Issues[7] jsou často řešena, zodpovídána a uzavírána, existuje ovšem necelých 2000 otevřených issue, což opět značí neorganizovanost repozitáře.

  Aktivita je tedy vysoká, ale organizovanost částí repozitáře je slabší.

Proces přispívání

- Obecně možnost přispět svým kódem má kdokoliv. Pokud se ale jedná o netriviální změny[8], je nutná smlouva (CLA — Contributor Licence Agreement).
- Změny vždy musí být schváleny dvěma členy týmu *commiters* (celý tým, který spravuje projekt — člen se nazývá *commiter*), jeden z nich musí navíc patřit do technické komise (OTC — OpenSSL Technical Committee). Roli *commiter* lze získat od manažerské komise (OMC — OpenSSL Managemet Committee) na doporučení OTC. OMC může kdykoliv roli *commiter* vývojáři odebrat. Mezi schvalovateli nesmí být autor kódu.
- Pravidla pro psaní kódu jsou na stránkách knihovny popsána a jsou vyžadována při kontrole kódu.
- Jsou vyžadovány testy pokrývající nový kód. Při jakýchkoli změnách musí kód projít testy průběžné integrace.

Bezpečnostní informace

- Všechna nedávná zveřejněná CVE byla opravena.
- Není nikde souhrn důvodů, proč by měl uživatel věřit, že je knihovna bezpečná. Jsou popsány jen konkrétní problémy, které jsou údajně aktuálně mimo zaměření tvůrců knihovny.
- Knihovna nezveřejňuje informace o provedených bezpečnostních auditech.
- Kryptografický modul knihovny obdržel certifikaci FIPS 140-2.

[5] `https://www.bestpractices.dev/en/projects?q=openssl`
[6] požadavek na přidání změn do hlavního kódu
[7] dotazy od komunity — mohou to být návrhy na změnu, nahlášené chyby i jen problémy s použitím
[8] triviální jsou například opravy komentářů a překlepů

Kontrola kvality

- Testy průběžné integrace se spouští automaticky, jejich pokrytí se pohybuje kolem 67 %[9].
- Knihovna využívá fuzz testing.
- Chybí celkový souhrnný popis testovací metodologie, není zmíněno použití nástrojů na analýzu kódu.

Souhrn

Dle získaných informací je dobře nastavena organizace i vývojové procesy knihovny. Zejména u procesu kontroly změn je vidět, že autoři bezpečnost procesu berou vážně. Bohužel chybí některé informace, jako např. jakými všemi způsoby se testuje, důvod, proč by se knihovna měla považovat za bezpečnou, a v repozitáři jsou velmi dlouho otevřené některé pull requesty a issues.

V následující tabulce jsou shrnuta důležitější kritéria:

[9] `https://coveralls.io/github/openssl/openssl?branch=openssl-3.3`

Tabulka 4.1 Hodnocení vývoje OpenSSL

Bod	Kritéria	
Základní informace	popsáno zamýšlené využití	✔
	popsány procesy hlášení chyb a návrhy na zlepšení	✔
	popsán proces přispívání do kódu	✔
	popsáno, jak hlásit zranitelnosti, a popsána reakce	✔
Tým knihovny	knihovnu spravují alespoň dva lidé	✔
	alespoň 1 člověk je zaměstnán na práci na knihovně	✔
	vývoj je aktivní	✔
Proces přispívání	autor kódu nemůže sám kód schválit	✔
	změny musí projít kontrolou člena týmu knihovny	✔
	existují pravidla pro styl psaní kódu	✔
	vyžadují se testy	✔
Bezpečnostní informace	nedávné zveřejněné zranitelnosti jsou opraveny	✔
	knihovna jasně říká, proč je bezpečná	✗
	knihovna zveřejňuje záznamy o auditech a certifikacích	✓
Kontrola kvality	spouští se automatické testy při nových změnách	✔
	měří se pokrytí kódu	✔
	využívají se další techniky testování a kontroly kódu jako fuzz testing a statická a dynamická analýza	✓

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

4.3.2 Mbed TLS

Mbed TLS je kryptografická knihovna napsaná v jazyce C vyvíjená od roku 2006. Nabízí kryptografická primitiva, manipulaci s certifikáty a protokoly SSL/TLS a DTLS. Snaží se cílit na malé množství kódu, tedy je vhodná pro vestavné systémy. [36]

Základní informace

- Knihovna má v repozitáři[10] jasně popsáno své zamýšlené využití.
- Procesy pro hlášení chyb a podávání návrhů jsou popsány.
- Proces přispívání je popsán.
- Hlášení zranitelností je jasně vysvětleno.
- Reakce na zranitelnosti je údajně popsána, odkaz z repozitáře ale vede na neexistující stránku. Zřejmě bylo zapomenuto na aktualizaci odkazu, protože popis reakce pro celou firmu Trusted Firmware je dostupný.
- Licence je jasně veřejně vystavena (duální licence: Apache 2.0 a GPL 2.0-or-later).
- Knihovna využívá verzovací systém (git).
- Seznamy změn pro konkrétní verze jsou dostupné a existují i pro nejnovější verze.
- Nebyl nalezen záznam o certifikaci dobrých vývojových praktik.

Tým knihovny

- Tým se skládá hlavně z lidí, kteří jsou zaměstnáni v Trusted Firmware / ARMu. Snaží se do svých řad nabrat i vývojáře z komunity, aktuálně je v týmu jeden takový člověk. Dle pull requestů a jejich schvalování (PR za posledních cca půl roku) bylo napočítáno 12 lidí, z nichž cca 5 je velmi aktivních. Oficiální seznam správců knihovny není vystaven.
- Vývojáři jsou sice zaměstnáni také pro práci na knihovně, není ale dostupná informace o nikom, kdo by byl zaměstnán výlučně pro vývoj knihovny.
- Finance na práci na knihovně se získávají z placené komerční podpory knihovny, případně z dalších výdělečných činností firem Trusted Firmware a ARM.
- Celá firma Trusted Firmware je zaměřena na bezpečný software[11], lze tedy předpokládat že vývojáři mají s bezpečným kódem zkušenosti. Veřejně ale není dedikovaný člověk pro knihovnu, který by tuto oblast zastřešoval.
- Aktivita:
  - Nejnovější verze knihovny je z března 2024 (psáno 25. dubna 2024).
  - Vývoj je velmi aktivní, za posledního půl roku vzniklo více než 500 pull requestů. Podobně jako u OpenSSL i zde je poměrně velký počet neuzavřených PR (kolem 250) a issues (kolem tisíce), z nichž některé jsou otevřené roky.

  Aktivita je tedy velká, jen organizace repozitáře (PR a issues) je opět trochu slabší.

[10] `https://github.com/Mbed-TLS/mbedtls`
[11] `https://www.trustedfirmware.org/`

Proces přispívání

- Kdokoliv může založit pull request a tím zažádat o přidání svého kódu do knihovny.
- Nový kód musí projít testy průběžné integrace.
- Kromě lidí z komunity (můžou přidat svoje review) musí kód schválit dva ověření schvalovatelé (trusted reviewers). Autor si opět nemůže kód schválit sám.
  - Kontrola těchto schvalovatelů má dle popisu[12] vysokou laťku — schvalovatel musí kódu perfektně porozumět a nemít žádné pochyby.
  - Kontroluje se:
    * podepsané commity (potvrzení o tom, že se kód stane součástí projektu)
    * pokrytí nového kódu testy
    * čistota a čitelnost kódu, dodržování standardů Mbed TLS
    * bezpečnost kódu
- Po schválení ještě musí někdo z užší skupiny oprávněných správců (trusted gatekeepers) přidat kód do knihovny (kontrola, zda vše proběhlo, jak mělo).

Bezpečnostní informace

- Všechna nedávná zveřejněná CVE byla opravena.
- Knihovna poskytuje poměrně dobrý popis toho, jak která bezpečnostní rizika řeší, uživatel může tedy dle těchto informací věřit, že autoři berou bezpečnost vážně.
- Knihovna nezveřejňuje informace o provedených bezpečnostních auditech.
- Knihovna usiluje o certifikaci FIPS 140-2.

Kontrola kvality

- Testy průběžné integrace se spouští automaticky, jejich pokrytí se sice údajně měří, není ale veřejně vystavené.
- Existuje popis celé testovací metodologie. Testuje se následující:
  - testy kompilace na různých systémech a platformách
  - analýza pokrytí kódu testy
  - testy různých konfigurací
  - paměťové testy (integrita, úniky)
  - statická analýza — hledání možných problémů a nesrovnalostí v kódu
  - fuzz testing

[12] `https://mbed-tls.readthedocs.io/en/latest/reviews/review-for-contributors/`

Souhrn

Dle dostupných informací o procesech a organizaci knihovny to vypadá, že je vývoj nastaven bezpečně. Hlavním nedostatkem u Mbed TLS může být to, že žádný vývojář není veřejně určen výlučně pro práci na knihovně — alokace vývojářů v případě potřeby musí správně nastavit zaměstnávající firma.

V následující tabulce jsou shrnuta důležitější kritéria:

Tabulka 4.2 Hodnocení vývoje Mbed TLS

Bod	Kritéria	
Základní informace	popsáno zamýšlené využití	✔
	popsány procesy hlášení chyb a návrhy na zlepšení	✔
	popsán proces přispívání do kódu	✔
	popsáno, jak hlásit zranitelnosti, a popsána reakce	✔
Tým knihovny	knihovnu spravují alespoň dva lidé	✔
	alespoň 1 člověk je zaměstnán na práci na knihovně	✓
	vývoj je aktivní	✔
Proces přispívání	autor kódu nemůže sám kód schválit	✔
	změny musí projít kontrolou člena týmu knihovny	✔
	existují pravidla pro styl psaní kódu	✔
	vyžadují se testy	✔
Bezpečnostní informace	nedávné zveřejněné zranitelnosti jsou opraveny	✔
	knihovna jasně říká, proč je bezpečná	✔
	knihovna zveřejňuje záznamy o auditech a certifikacích	✓
Kontrola kvality	spouští se automatické testy při nových změnách	✔
	měří se pokrytí kódu	✔
	využívají se další techniky testování a kontroly kódu jako fuzz testing a statická a dynamická analýza	✔

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

4.3.3 GnuTLS

GnuTLS je kryptografická knihovna napsaná v jazyce C a částečně v assembleru. Je vyvíjená od roku 2000. Implementuje protokoly SSL/TLS, DTLS a technologie okolo nich. Vznikla jako

součást projektu GNU, dnes je ale vývoj nezávislý — i tak se s knihovnou můžeme nejčastěji setkat právě na GNU systémech. [37]

Základní informace

- Knihovna má jasně popsáno své zamýšlené využití.
- Proces nahlašování chyb je popsán.
- Proces přispívání je popsán.
- Hlášení zranitelností je popsáno.
- Reakce na zranitelnosti je popsána.
- Licence je jasně veřejně vystavena (GNU Lesser General Public License version 2.1).
- Knihovna využívá verzovací systém (git).
- Seznamy změn pro konkrétní verze jsou dostupné a existují i pro nejnovější verze.
- Dle repozitáře[13] knihovna splňuje kritéria OpenSSF dobrých vývojových praktik.

Tým knihovny

- Tým se skládá z několika dobrovolníků, nevypadá to, že by byli něčím vázaní, zřejmě jsou to nadšenci do kryptografie. Dle dokumentace jsou v aktuálním týmu knihovny čtyři lidé. Dle analýzy pull requestů jsou dva schvalovatelé velmi aktivní, jsou ale často vidět všechna čtyři jména z dokumentace.
- Nikdo tedy není zaměstnán na práci na knihovně.
- Knihovna nemá odděleného člověka se zkušenostmi s bezpečným kódem.
- Aktivita:
  - Nejnovější verze knihovny je z dubna 2024 (psáno 25. dubna 2024).
  - Vývoj je poměrně aktivní, za posledního půl roku je vidět kolem 50 pull requestů. Není mnoho otevřených pull requestů (kolem 20), ale i tak se mezi nimi najdou přes tři roky staré.
  - Na issues se průběžně reaguje, poměrně velká část je ale otevřená (cca 20 %) a některá jsou stará i kolem 10 let.

  Aktivita tedy nějaká je, úměrná velikosti týmu a jeho dobrovolnickému vztahu ke knihovně.

[13] `https://gitlab.com/gnutls/gnutls`

Proces přispívání

- Kdokoliv může založit pull request a tím zažádat o přidání svého kódu do knihovny.
- Nový kód musí projít testy průběžné integrace.
- Kromě lidí z komunity (můžou přidat svoje review) musí kód schválit jeden člověk z týmu knihovny.
  - Kontrola je zamýšlena tak, aby probíhala co nejrychleji. Hlavní část kontroly je ponechána na automatických testech.
  - Kontroluje se:
    * podepsané commity
    * zda byly přidány testy
    * dodržování standardů kódu GnuTLS
    * žádné očividné chyby
- Jsou velmi podrobně zdokumentovány požadavky na kódovací standardy GnuTLS, dokumentace jak lidských, tak strojových kontrol nového kódu je ale dost slabá.

Bezpečnostní informace

- Všechna nedávná zveřejněná CVE byla opravena.
- Není nikde souhrn důvodů, proč by měl uživatel věřit, že je knihovna bezpečná.
- Knihovna zveřejňuje informace u některých zranitelností, že byly nálezem auditu.
- Knihovna nezveřejňuje informaci o žádné certifikaci kryptografické implementace.

Kontrola kvality

- Testy průběžné integrace se spouští automaticky, jejich pokrytí se měří, je vystavené v repozitáři a testy pokrývají kolem 75 % kódu.
- Testy nejsou moc popsané, jsou k nalezení pouze zmínky o několika typech testování, jako práce s pamětí a fuzz testing (pokrytí 35 %)

Souhrn

Dle dostupných informací o procesech a organizaci knihovny vypadá vývoj nastaven poměrně bezpečně, je zde ale více slabších bodů. První bod se týká aktivity a motivace správců, kteří pravděpodobně nemusí mít dostatek času a motivace opravovat možné zranitelnosti včas. Dalším možným problémem je samotná kompetence správců knihovny, kterou jednak nelze z dostupných informací zaručit, ale také ji zpochybňují další lidé [38, 39].

V následující tabulce jsou shrnuta důležitější kritéria:

Tabulka 4.3 Hodnocení vývoje GnuTLS

Bod	Kritéria	
Základní informace	popsáno zamýšlené využití	✔
	popsány procesy hlášení chyb a návrhy na zlepšení	✔
	popsán proces přispívání do kódu	✔
	popsáno, jak hlásit zranitelnosti, a popsána reakce	✔
Tým knihovny	knihovnu spravují alespoň dva lidé	✔
	alespoň 1 člověk je zaměstnán na práci na knihovně	✗
	vývoj je aktivní	✓
Proces přispívání	autor kódu nemůže sám kód schválit	✔
	změny musí projít kontrolou člena týmu knihovny	✔
	existují pravidla pro styl psaní kódu	✔
	vyžadují se testy	✔
Bezpečnostní informace	nedávné zveřejněné zranitelnosti jsou opraveny	✔
	knihovna jasně říká, proč je bezpečná	✗
	knihovna zveřejňuje záznamy o auditech a certifikacích	✓
Kontrola kvality	spouští se automatické testy při nových změnách	✔
	měří se pokrytí kódu	✔
	využívají se další techniky testování a kontroly kódu jako fuzz testing a statická a dynamická analýza	✓

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

4.3.4 WolfSSL

WolfSSL je kryptografická knihovna napsaná v jazyce C. Nabízí hlavně SSL/TLS protokol, je postavená na kryptografických funkcích knihovny wolfCrypt od stejné firmy (nadále považováno za jednu knihovnu). Klade si za cíl malé množství kódu a rychlost, je tedy vhodná pro vestavné systémy a obecně pro běh v prostředí s omezenými zdroji. [40]

Základní informace

- Knihovna má v repozitáři[14] jasně popsáno své zamýšlené využití.
- Proces hlášení chyb není vyloženě popsán jako „hlášení chyb“, ale zřejmě jde o společný postup jako pro dotazy na technickou podporu.
- Postup pro navrhování nových funkcionalit je popsán.
- Proces přispívání není nikde rozumně popsán — existuje pouze informace o nutnosti mít smlouvu s WolfSSL Inc. pro možnost přidání změn.
- Hlášení zranitelností je popsáno.
- Reakce na zranitelnosti je popsána, cíl knihovny je opravit zranitelnost do 36 hodin po objevení.
- Licence je jasně veřejně vystavena (duální licence: GPLv2 a komerční licence).
- Knihovna využívá verzovací systém (git).
- Seznamy změn pro konkrétní verze jsou dostupné a existují i pro nejnovější verze.
- Nebyl nalezen záznam o certifikaci dobrých vývojových praktik.

Tým knihovny

- O knihovnu se stará tým společnosti WolfSSL Inc. Dle pull requestů a jejich schvalování (PR za posledních cca půl roku) bylo napočítáno cca 15 lidí, z nichž 5 je velmi aktivních. Oficiální seznam správců knihovny není vystaven.
- Vývojáři jsou sice zaměstnáni také pro práci na knihovně, není ale dostupná informace o nikom, kdo by byl zaměstnán výhradně pro vývoj knihovny.
- Finance na práci na knihovně se získávají z komerčních licencí knihovny a dalších produktů WolfSSL Inc.
- Celá firma WolfSSL Inc. je zaměřena na bezpečný software, lze tedy předpokládat, že vývojáři mají s bezpečným kódem zkušenosti. Veřejně ale není dedikovaný člověk pro knihovnu, který by tuto oblast zastřešoval.
- Aktivita:
  - Nejnovější verze knihovny je z března 2024 (psáno 25. dubna 2024).
  - Vývoj je velmi aktivní, za posledního půl roku vzniklo více než 500 pull requestů. Na rozdíl od ostatních zkoumaných knihoven má WolfSSL poměrně málo otevřených issues a PR (obojí kolem 80), z nichž minimum je staré několik let. Aktivita je tedy velká a organizace repozitáře svědčí o kvalitní péči o knihovnu.

[14] `https://github.com/wolfSSL/wolfssl`

Proces přispívání

- Pro založení requestu a přidání kódu do knihovny je potřeba smlouva s WolfSSL Inc.
- Nový kód musí projít velkým množstvím testů.
- Kromě lidí z komunity (můžou přidat svoji kontrolu) musí kód schválit jeden či více členů týmu (zřejmě záleží na povaze změn, tato informace není veřejně zdokumentována). Schvaluje někdo jiný než autor.
- Při kontrole změn v kódu se řeší hlavně neměnnost API, správnost algoritmů a konzistence kódu (tedy jeho soulad se standardy knihovny).
- Po schválení poslední schvalovatel provede merge.
- Požadavky na styl kódu nejsou veřejně vystaveny. Zřejmě podobně jako samotný proces kontroly kódu jsou určeny pro zaměstnance / členy týmu a jsou dostupné vývojářům s uzavřenou smlouvou s WolfSSL Inc.

Bezpečnostní informace

- Všechna nedávná zveřejněná CVE byla opravena.
- Knihovna podrobně popisuje, jak přistupuje k bezpečnosti.
- Knihovna nezveřejňuje informace o provedených bezpečnostních auditech.
- Knihovna obdržela certifikaci FIPS 140-2 (konkrétně modul WolfCrypt), certifikace FIPS 140-3 je v procesu.

Kontrola kvality

- Knihovna má velmi podrobně popsané testovací techniky.
- Testuje se:
  - testy kompilace na různých systémech a platformách
  - unit testy a pravidelná analýza pokrytí
  - testy různých konfigurací
  - paměťové testy (integrita, úniky)
  - statická analýza kódu
  - fuzz testing
  - ověření konzistence API
  - ověření algoritmů a modulů

  Správa knihovny klade na testy velký důraz; zároveň je běh testů velmi přehledně vidět v Github repozitáři u každého pull requestu.

Souhrn

Tato knihovna působí velmi organizovaně, má bezpečně nastavené procesy a kvalitní testování. Bohužel, vzhledem k povaze správců, některé informace nejsou veřejné a jsou určeny pro interní potřeby vývoje knihovny.

V následující tabulce jsou shrnuta důležitější kritéria:

Tabulka 4.4 Hodnocení vývoje WolfSSL

Bod	Kritéria	
Základní informace	popsáno zamýšlené využití	✔
	popsány procesy hlášení chyb a návrhy na zlepšení	✓
	popsán proces přispívání do kódu	✗
	popsáno, jak hlásit zranitelnosti, a popsána reakce	✔
Tým knihovny	knihovnu spravují alespoň dva lidé	✔
	alespoň 1 člověk je zaměstnán na práci na knihovně	✓
	vývoj je aktivní	✔
Proces přispívání	autor kódu nemůže sám kód schválit	✔
	změny musí projít kontrolou člena týmu knihovny	✔
	existují pravidla pro styl psaní kódu	✓
	vyžadují se testy	✔
Bezpečnostní informace	nedávné zveřejněné zranitelnosti jsou opraveny	✔
	knihovna jasně říká, proč je bezpečná	✔
	knihovna zveřejňuje záznamy o auditech a certifikacích	✔
Kontrola kvality	spouští se automatické testy při nových změnách	✔
	měří se pokrytí kódu	✔
	využívají se další techniky testování a kontroly kódu jako fuzz testing a statická a dynamická analýza	✔

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

4.4 Shrnutí

Z aplikace kritérií na vybrané kryptografické knihovny lze vidět, že co se týká open-source vývoje, tak knihovny splňují většinu požadavků. Dalo by se to vysvětlit jednak tím, že tyto konkrétní knihovny jsou poměrně známé a používané, tedy vývojové procesy už musí být nastavené rozumně. Zároveň, s dnešním častým využitím open-source, už není nutné správné praktiky „vymýšlet“, ale stačí následovat již vymyšlené pokyny pro bezpečný vývoj.

Co se týče zkoumaných knihoven, podle kritérií vývoje vypadá nejlépe knihovna Mbed TLS. Dále jsou knihovny WolfSSL a OpenSSL. Všechny tyto 3 knihovny splňují naprostou většinu kritérií a vydají, že z jejich využití neplyne velké riziko. Z využití GnuTLS by také nemělo dle těchto kritérií plynout velká hrozba, je zde ale riziko spojené s motivací a časem aktuálních správců knihovny a jejich schopnosti reakce na případné problémy.

Tato část kritérií obecně hodnotí, jak velkou důvěru může vývojář v knihovnu vložit. Porovnává, jakou hrozbu přináší do software využití závislosti v podobě kryptografické knihovny — protože se jedná o část kritérií, kde znalost a zkušenost vývojáře aplikace nemůže snížit výskyt možných problémů. Další kapitola se podívá na část kritérií, která se týkají právě použití knihovny aplikačním vývojářem. To je část, kterou takový vývojář ovlivnit může.

Kapitola 5
Použití knihoven

Tato kapitola se věnuje pohledu aplikačního vývojáře na kryptografii a kryptografické knihovny. Je zde popsáno, jakou roli hrají chyby v použití kryptografie v bezpečnosti, proč k takovým chybám dochází a jak jim lze předcházet. Dále tuto problematiku části kapitoly zkoumají přímo na vybraných knihovnách.

5.1 Role a důvody chyb v použití

Uvažujme na chvíli knihovnu, která neobsahuje žádné implementační chyby. Ať už díky plnění požadavků probraných v předchozí kapitole, nebo „jednoduše“ bezchybností vývojářů. Je taková knihovna bezpečná? Jak říká kapitola 2, bezpečnost knihovny závisí také na bezpečnosti jejího použití aplikačním vývojářem. I v případě bezchybné implementace knihovny může vývojář narazit na úskalí spočívající v použití knihovny. Z různých důvodů lze rozhraní knihoven, a to zejména kryptografických, použít špatně.

Zranitelnosti vnesené do software samotnými kryptografickými knihovnami sice snižují jeho bezpečnost, ukazuje se ale, že špatné použití kryptografie je častějším a závažnějším problémem [41]. Důvodů je více, hlavním ale je, že průměrný vývojář není kryptografickým expertem [41, 9]. Rozhraní kryptografických knihoven často vyžaduje poměrně velké znalosti kryptografických konceptů, což vede k nepochopení rozhraní a následně jeho špatnému použití [8].

Existují studie, které zkoumají, jak časté takové špatné použití kryptografie bývá. Např. studie open-source projektů v jazyce Java ukázala, že více než 72 % takových projektů špatně pracuje s kryptografií alespoň na jednom místě [42]. Další studie ukazují podobné výsledky, například v aplikacích pro systém Android [43]. Tyto závěry nám jednak ukazují, jak častý tento problém je, také ale vybízí k otázce, zda s tím lze něco udělat.

Dle článku [9] je povaha chyb v použití kryptografie jiná než povaha jiných chyb. Jsou méně viditelné (nemusí narušovat samotnou funkcionalitu software, ovšem oslabují bezpečnost), a tedy se hůře hledají.

Jak již bylo zmíněno, využití kryptografie je mnoho. Kryptografické knihovny se často využívají pro úkoly jako symetrické šifrování, pak také podepisování a ověřování, generování klíčů a v neposlední řadě navazování zabezpečeného spojení [44]. Vývojáři pro takové úkoly preferují vysokoúrovňové API, které odstíní složitost kryptografie, a tím je pro vývojáře srozumitelnější [44]. Na toto zjištění navazuje článek [9], který se více zabývá důvody špatného použití knihoven a vyjmenovává nejčastější chyby.

Tyto chyby jsou:

- špatná validace certifikátů

  Tato chyba může vést k tomu, že aplikace se sice protokolem TLS bezpečně připojí, ovšem k jinému protějšku než zamýšlela. Důvodem vzniku bývá složitost rozhraní pro kontrolu certifikátu a příliš velká volnost nastavení této kontroly.

- nebezpečné šifrovací módy

  Knihovny často nabízejí rozhraní, které vyžaduje kryptografické znalosti na straně vývojáře a hlavně je jednoduché jej použít špatně. Například nutnost volby šifrovacích módů s sebou nese riziko, že si vývojář zvolí slabý mód, a tím oslabí bezpečnost šifrování.

- nebezpečné generátory náhodných čísel

  Bezpečnostní protokoly se často spoléhají na dostupnost nepředvídatelných náhodných čísel, která jsou poskytována kryptograficky bezpečnými pseudonáhodnými generátory. Mnoho aplikací ovšem používá nebezpečné generátory, čímž snižují svou bezpečnost.

Na tyto body je mimo jiné vhodné se zaměřit při hodnocení použitelnosti kryptografického rozhraní.

Dále stejný článek definuje 10 principů pro vytvoření použitelného a bezpečného kryptografického rozhraní. Tyto principy mimo jiné obsahují:

- Kryptografie má být zahrnuta do standardních API.

  Ve chvíli, kdy je kryptografie zahrnuta do nekryptografického API, se vývojář může přímému využití kryptografie úplně vyhnout — nemůže ji tedy využít špatně.

- API má být dostatečně mocné.

  Pokud API nenabízí vše, co od něj vývojář požaduje, může se vývojář snažit různými způsoby funkcionalitu ohnout či obejít bezpečnostní část. To oslabí bezpečnost výsledného software.

- API má být snadno naučitelné a pochopitelné.

  Kryptografické pozadí API je pro většinu vývojářů příliš složité na pochopení, je tedy vhodné jej před vývojáři ukrýt.

- API nemá rozbíjet vývojářská paradigmata.

  Není vhodné vytvořit rozhraní zásadně jiným způsobem, než jsou vývojáři zvyklí.

- API má jít jednoduše použít i bez dokumentace.

  Vývojáři málokdy čtou dokumentaci, je tedy vhodné, aby se nutnost do ní podívat vyskytovala co nejméně. Jako příklad jsou uvedené návratové kódy funkcí OpenSSL, které se svým významem mohou v různých funkcích lišit — vývojář může mít bez dokumentace mylný předpoklad, že jsou návratové kódy vždy stejné, což zaručeně povede k chybě.

- API má být složité použít špatně.

  V případě špatného použití by se knihovna měla co nejvíce bránit. Například při nebezpečném nastavení by měla vývojáře alespoň varovat nebo i třeba házet výjimku.

- Výchozí hodnoty mají být jasné a bezpečné.

  Vývojáři, zvlášť když nerozumí tomu, co dělají, často zvolí právě výchozí nastavení, aby se nemuseli nastavováním zabývat. Proto by toto základní nastavení mělo být co nejbezpečnější. Například by výchozí šifrovací mód neměl být mód ECB, který některé knihovny dle článku jako výchozí měly.

- Knihovna má nabízet testovací mód.

  Bezpečnostní mechanizmy přidávají do kódu složitost a vývojáři je často potřebují pro účely testování vypnout. Tím se sice oslabí bezpečnost pouze na testovacím prostředí, ovšem jen pokud se nezapomene při nasazení/vydání software všechno zase nastavit zpátky na bezpečnou variantu. Testovacím módem by knihovna poskytla vývojářům jednodušší způsob, jak docílit chtěného chování na testovacích prostředích a zároveň zajistit, že jinde nebude bezpečnost ohrožena. [9]

Doporučením pro kryptografické knihovny se věnují i další články. Článek [41] kromě již výše zmíněných doporučení přidává doporučení pro dokumentaci, příklady použití knihovny nebo odstraňování zastaralých funkcí. Podobně studie [45] dává sadu doporučení, opět mluví např. o výchozích hodnotách i dokumentaci a zmiňuje, v čem je užitečné striktní typování. Článek [46] porovnává použitelnost kryptografických rozhraní v jazyku Rust. Také nabízí sadu doporučení, které zahrnují zejména rady, jak psát dokumentaci. Důležité body jsou:

- popsat scénář využití u každého algoritmu
- vyznačit zranitelné či slabší algoritmy
- důkladně popsat všechny parametry
- v případě více možností přidat doporučení
- poskytovat ukázky použití
- udržovat ukázkový kód stejně jako kód související s bezpečností

Jedním z hlavních jmenovaných doporučení je jednoduchost rozhraní. Studie zabývající se vlivem zjednodušení kryptografického rozhraní na chyby v použití ale říkají, že jednoduchost API není vše. Například nekvalitní dokumentace či chybějící požadovaná funkcionalita v rozhraní mohou správné a bezpečné použití ztížit i přes zmíněnou jednoduchost [47, 45]. Je také zmíněna relevance diskuzních fór, kam vývojáři často chodí pro rady a mohou si z takových nedůvěryhodných zdrojů zanést do kódu chyby [48, 49].

5.2 Soubor kritérií pro porovnání návrhu knihoven

V předchozí části bylo shrnuto, kde a proč vývojáři běžně dělají chyby. Dále byla vyjmenována některá doporučení pro lepší a bezpečnější návrh knihovny. Zde následuje souhrn vlastností kryptografických knihoven, které je vhodné prozkoumat při ověřování použitelnosti knihovny. Vlastnosti jsou rozděleny na dvě kategorie — API a dokumentace. Každá z těchto kategorií je důležitá jiným způsobem. Požadavky na API jsou důležité, protože to je to, s čím vývojáři přímo pracují a s čím pravděpodobně jako s prvním přijdou do styku. Dokumentace je oficiální zdroj informací o API, je tedy vhodné poskytnout kvalitní popis rozhraní a návod, jak ho použít.

- Důležité vlastnosti API:
  - **API dostatečně pokrývá potřebnou funkcionalitu:** Je důležité, aby uživatel pro provedení nějaké operace nemusel žádným způsobem rozhraní ohýbat, nastavovat nebezpečně či přejít k využívání jiné knihovny. Knihovna má nabízet vše potřebné ke kryptografickým úkonům. Například pro zašifrování souboru heslem je potřeba kromě symetrické šifry také z hesla vytvořit klíč, je tedy potřeba vhodná funkce pro tento úkol (KDF — key derivation function). Dále také musí API nabízet bezpečné způsoby generování náhodných hodnot, generování klíčů či inicializačních vektorů, aby uživatel nemusel využívat jinou, pravděpodobně méně bezpečnou variantu.

- **Vyšší abstrakce pro běžné scénáře:** Tento požadavek vyplývá z běžných chyb vývojářů, jako je špatné ověřování certifikátů či používání slabých parametrů. Pokud to lze udělat, je vhodné nenechat volbu vyžadující znalosti kryptografie na vývojáři, například volbu operačního módu šifer. V takovém případě je ale vhodné, aby byly k dispozici informace o použitých volbách, případně způsob umožňující nastavit je podle sebe. Tento způsob by ale neměl být jednoduchý a měl by vývojáře varovat před možným rizikem nebezpečnějšího nastavení.
- **Popsané a bezpečné výchozí hodnoty:** Výchozí hodnoty u všech funkcí musí být nejen bezpečné (protože vývojáři často zvolí právě ty), ale je potřeba je i jasně popsat, aby bylo výchozí chování jasné.
- **Chyby musí být srozumitelné:** Z chyby musí být jasné, čím byla způsobena. Nejasné chyby se mnohem hůře opravují. Dle [45] je vhodné, aby chyby byly srozumitelné i pro vývojáře nepříliš znalého v oblasti kryptografie. Výsledky běhu funkcí by měly být srozumitelné a konzistentní napříč celým rozhraním. Pokud lze použít výjimky, je vhodné je použít. Pokud nelze, je vhodné mít konzistentní návratové kódy, případně pro výsledek běhu funkce využít výčtový typ (enumeraci).
- **Striktní typování argumentů:** Argumenty by měly být složité zaměnit nebo umístit špatně. Např. klíč a IV mohou typově být pole bytů, což znamená, že v argumentech je možné je prohodit. Použitím např. různých struktur pro klíč a IV tato možnost nebude jednoduchá provést, program nepůjde při jednoduchém prohození zkompilovat.
- **Nebezpečné volby — nebezpečná jména:** Tento požadavek spočívá v tom, že pokud pojmenujeme nebezpečné volby tak, aby bylo ze jména jasné nebezpečí jejich použití, upozorní to vývojáře, který díky tomu bude vědět o riziku. Například do jména lze přidat slova jako „DANGER“, „DANGEROUS“ nebo „UNSAFE“ [3].

Kromě výše zmíněných je vhodné, aby knihovna nabízela testovací mód, který by vývojářům umožnil vypnutí bezpečnostních funkcionalit pro účely testování, a to bez zásahů do kódu.

- Důležité vlastnosti dokumentace
  - **Základní požadavky:** Dokumentace musí existovat, být dostupná a musí být jasné, že patří ke knihovně. Dále je nutné, aby dokumentace obsahovala vysvětlení instalace knihovny a následně vysvětlení použití všech jejích součástí.
  - **Ukázkový kód s použitím:** Pro často používané funkce by měly být poskytovány základní ukázky použití. Tyto ukázky by neměly vynechávat žádnou důležitou součást procesu jako například způsob generování klíče.
  - **Varování u zastaralých a nebezpečných algoritmů:** Dokumentace by měla jasně vyznačit nebezpečí použití u takových algoritmů, které se považují za slabé nebo prolomené. Samozřejmě existuje také vhodná možnost takové algoritmy vůbec neposkytovat.
  - **Přehlednost dokumentace:** Pro dobrou použitelnost by dokumentace měla být přehledná. Mělo by být jasné, co kde hledat, navigace by měla být srozumitelná a snadno použitelná a dokumentace by měla nabízet vyhledávání.
  - **Využití algoritmů:** U konkrétních algoritmů by se kromě samotného jejich popisu měly vyskytovat i běžné scénáře použití. Například symetrická šifra může mít v takových příkladech šifrování souboru.
  - **Popis parametrů:** Každý parametr by měl být důkladně popsán — jeho typ a zejména požadavky na jeho podobu, například jestli musí projít inicializací a jakým způsobem.
  - **Doporučení při možnosti volby:** V případě, kdy si lze zvolit z více možností, by dokumentace měla poskytovat doporučení pro konkrétní scénáře.

5.3 Aplikace kritérií na knihovny

V této části jsou kritéria aplikována a zkoumána (podobně jako v kapitole 4) na vybraných kryptografických knihovnách.

5.3.1 OpenSSL

Zde jsou kritéria návrhu knihoven zkoumána na OpenSSL. Stručný popis této knihovny lze nalézt v části 4.3.1.

API

- API knihovny OpenSSL je velmi mocné, umožňuje uživateli udělat prakticky cokoliv, co by v oblasti kryptografie mohlo být požadováno.
- Pro použití OpenSSL jsou nutné poměrně rozsáhlé znalosti kryptografických konceptů, knihovna nenabízí žádné abstrakce vyšší úrovně. Je na vývojáři, aby sám vybral bezpečné nastavení, na což jsou právě potřeba znalosti, které často vývojářům chybí.
- Knihovna nenabízí výchozí hodnoty. Vývojář si tedy vše musí zvolit ručně sám. Nezkušení vývojáři (s knihovnou) pravděpodobně budou hledat řešení na diskuzních fórech jako Stack Overflow [48], kde mohou dostat odpovědi vedoucí k nebezpečnému kódu, případně rovnou zvolí špatně. Stále je to ale lepší, než kdyby knihovna výchozí hodnoty nabízela a byly nebezpečné.
- Knihovna používá návratové hodnoty v podobě čísla, což není moc přehledná a srozumitelná volba. Také se v návratových kódech vyskytují nekonzistence — kontrolu návratového kódu je snazší udělat špatně.
- Dále také knihovna nemá striktní typování argumentů. Například klíč a IV u blokové šifry je jednoduché prohodit.

Dokumentace

- Dokumentace je dostupná na webových stránkách knihovny. Je kompletní, popisuje vše důležité.
- Přehlednost dokumentace je velmi špatná, špatně se i vyhledává a naviguje.
- Jsou poskytnuty ukázky použití, ale bývají jednak složité, ale také jejich využití nemusí vézt na bezpečný výsledek — využívají se např. konstantní klíče nebo zastaralé algoritmy.
- Nejsou obsažena varování u nebezpečných/zastaralých algoritmů.
- Požadavky na argumenty nejsou dostatečně popsané, uživatel tedy nemusí vědět, jakým způsobem argument připravit pro použití ve funkci.

Souhrn

Použitelnost OpenSSL má velmi špatné výsledky, zejména kvůli nízké kvalitě dokumentace. V API je hodně prostoru pro zlepšení, tomu ovšem může překážet nutnost zpětné kompatibility, která může u takto staré a používané knihovny hrát důležitou roli. Vylepšení dokumentace ovšem takovéto překážky nemá, to je místo, kde by tým OpenSSL měl hodně zapracovat.

V následující tabulce jsou shrnuta důležitější kritéria:

Tabulka 5.1 Hodnocení použitelnosti OpenSSL

Bod	Kritéria	
API	pokrývá potřebnou funkcionalitu	✔
	abstrakce pro běžné scénáře	✗
	bezpečné výchozí hodnoty	✗
	konzistentní a srozumitelné návratové hodnoty	✗
Dokumentace	přehledná, dobře se naviguje a vyhledává	✗
	obsahuje ukázky použití vedoucí k bezpečnému kódu	✗
	obsahuje varování u zastaralých algoritmů a nebezpečných módů	✗
	dostatečně popisuje argumenty a jejich předpoklady	✗

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

5.3.2 Mbed TLS

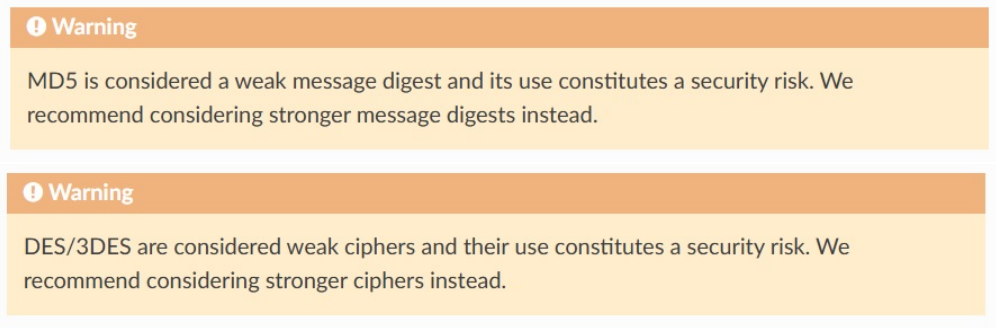
Zde jsou kritéria návrhu knihoven zkoumána na Mbed TLS. Stručný popis této knihovny lze nalézt v části 4.3.2.

API

- API poskytuje velmi mnoho možností, pokrývá všechno potřebné pro bezpečné využití kryptografie.
- API neobsahuje vysokoúrovňové abstrakce, k použití jsou tedy nutné znalosti kryptografie. Vývojář může z neznalosti snadno udělat chybu např. ve volbě argumentů.
- Knihovna nenabízí výchozí hodnoty.
- Knihovna používá návratové hodnoty v podobě enumu (výčtový typ) — je poměrně srozumitelné, co jaká návratová hodnota znamená. Návratové hodnoty jsou konzistentní.
- Knihovna nemá striktní typování argumentů.

Dokumentace

- Dokumentace je dostupná na webových stránkách knihovny. Je kompletní, popisuje vše důležité.
- Přehlednost dokumentace je poměrně dobrá. Lze v ní vyhledávat, i když navigace zrovna moc přehledná není — je nutné chvíli dokumentaci procházet, např. při hledání dokumentace šifry AES.
- Jsou poskytnuty ukázky použití, jsou poměrně jednoduché a vždy je důkladně popsáno, jaké jsou požadavky na různé hodnoty, například jak vygenerovat klíč či IV, není použita fixní hodnota. Nejsou ale pokryty všechny běžné scénáře, například šifrování souboru heslem.



Obrázek 5.1 Varování před použitím nebezpečných algoritmů u Mbed TLS

- U zastaralých algoritmů je výrazné varování o nebezpečí použití.

  Operační mód ECB sice je nabízen a nemá takové varování, knihovna ale brání uživatele před špatným použitím tím, že nedovolí tímto módem zašifrovat více než jeden blok dat.
- Požadavky na argumenty jsou rozumně popsané — je jasné, jakými funkcemi musel být připraven např. kontext šifry.

Souhrn

Knihovna Mbed TLS sice nabízí poměrně nízkoúrovňové API, dohání to ale docela dobrou dokumentací. Bohužel, v dokumentaci chybí ukázkový kód pro některé běžnější scénáře.

V následující tabulce jsou shrnuta důležitější kritéria:

Tabulka 5.2 Hodnocení použitelnosti Mbed TLS

Bod	Kritéria	
API	pokrývá potřebnou funkcionalitu	✔
	abstrakce pro běžné scénáře	✗
	bezpečné výchozí hodnoty	✗
	konzistentní a srozumitelné návratové hodnoty	✔
Dokumentace	přehledná, dobře se naviguje a vyhledává	✓
	obsahuje ukázky použití vedoucí k bezpečnému kódu	✓
	obsahuje varování u zastaralých algoritmů a nebezpečných módů	✔
	dostatečně popisuje argumenty a jejich předpoklady	✔

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

5.3.3 GnuTLS

Zde jsou kritéria návrhu knihoven zkoumána na GnuTLS. Stručný popis této knihovny lze nalézt v části 4.3.3.

API

- API poskytuje velmi mnoho možností, pokrývá naprostou většinu potřebné funkcionality pro bezpečné využití kryptografie. Oproti např. Mbed TLS má některé části jednodušší, tím jsou ale některé možnosti více omezené, jako třeba generování náhodných dat. GnuTLS nabízí jednu jednoduchou funkci — tím ale upírá možnost uživateli vylepšit generování náhodných dat pro lepší entropii.
- API neobsahuje vysokoúrovňové abstrakce, k použití jsou tedy nutné znalosti kryptografie. Vývojář může z neznalosti snadno udělat chybu např. ve volbě argumentů.
- Knihovna nenabízí výchozí hodnoty.
- Knihovna používá návratové kódy, které nejsou příliš konzistentní. Číslo 0 většinou znamená úspěch, někdy také ne.
- Knihovna nemá striktní typování argumentů.

Dokumentace

- Dokumentace je dostupná na webových stránkách knihovny. Je kompletní, popisuje vše důležité.
- Přehlednost dokumentace není moc dobrá. Vyhledávání a navigace jsou poměrně složité. Není jasné, co je kde umístěno a jak se dostat např. ke konkrétnímu algoritmu.
- Ukázkový kód je dostupný pouze pro spojení protokolem SSL/TLS, je zde příklad klienta i serveru. Ukázky pro kryptografické API chybí.
- U zastaralých algoritmů chybí varování o nebezpečí použití i přes to, že jsou nabízeny. Operační mód ECB není možné zvolit.
- Požadavky na argumenty nejsou moc dobře popsané. Není jasné, jaké podmínky mají argumenty funkcí splňovat.

Souhrn

Knihovna GnuTLS nabízí sice v něčem jednoduší, celkově ale podobně složité API jako předchozí dvě knihovny. Nejslabší v této oblasti jsou návratové kódy, ve kterých není jednoduché se vyznat. Dokumentace je sice zásadně lepší než u OpenSSL, i tak ale hodně zásadních bodů chybí.

V následující tabulce jsou shrnuta důležitější kritéria:

Tabulka 5.3 Hodnocení použitelnosti GnuTLS

Bod	Kritéria	
API	pokrývá potřebnou funkcionalitu	✔
	abstrakce pro běžné scénáře	✗
	bezpečné výchozí hodnoty	✗
	konzistentní a srozumitelné návratové hodnoty	✗
Dokumentace	přehledná, dobře se naviguje a vyhledává	✗
	obsahuje ukázky použití vedoucí k bezpečnému kódu	✗
	obsahuje varování u zastaralých algoritmů a nebezpečných módů	✗
	dostatečně popisuje argumenty a jejich předpoklady	✗

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

5.3.4 WolfSSL

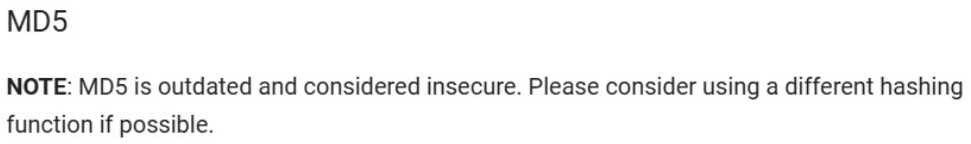
Zde jsou kritéria návrhu knihoven zkoumána na WolfSSL. Stručný popis této knihovny lze nalézt v části 4.3.4.

API

- API poskytuje širokou nabídku algoritmů a dalších bezpečnostních funkcionalit — pokrývá tedy vše důležité.
- API neobsahuje vysokoúrovňové abstrakce, k použití jsou tedy nutné znalosti kryptografie. Vývojář může z neznalosti snadno udělat chybu např. ve volbě argumentů.
- Knihovna nenabízí výchozí hodnoty.
- Knihovna používá návratové hodnoty v podobě enumu (výčtový typ) — je poměrně srozumitelné, co jaká návratová hodnota znamená. Návratové hodnoty jsou konzistentní.
- Knihovna nemá striktní typování argumentů.

Dokumentace

- Dokumentace je dostupná na webových stránkách knihovny. Je kompletní, popisuje vše důležité.
- Přehlednost dokumentace je velmi dobrá. Má dobře udělané vyhledávací pole a navigace je udělána velmi přehledně.
- Jsou poskytnuty ukázky použití u všech funkcí. Jsou poměrně jednoduché a vždy je důkladně popsáno, jaké jsou požadavky na různé hodnoty, například jak vygenerovat klíč či IV. Není použita fixní hodnota. Nejsou ale pokryty všechny složitější běžné scénáře, například šifrování souboru heslem.



Obrázek 5.2 Varování před použitím nebezpečných algoritmů u WolfSSL

- U zastaralých algoritmů sice bývá varování, ale není moc výrazné. U některých algoritmů (např. DES/3DES) dokonce chybí. Operační mód ECB sice je nabízen a nemá takové varování, knihovna ale brání uživatele před špatným použitím tím, že nedovolí tímto módem zašifrovat více než jeden blok dat.
- Požadavky na argumenty jsou rozumně popsané — je jasné, jakými funkcemi musel být připraven např. kontext šifry.

Souhrn

Knihovna WolfSSL má poměrně dobré výsledky, zejména proto, že má velmi přehlednou dokumentaci. Chybí ale ukázky kódu pro některé složitější, ale přesto časté scénáře jako zašifrování souboru heslem. Také varování u zastaralých algoritmů by mělo být všude a výrazné, což zde neplatí. API opět postrádá vyšší úroveň abstrakce, která by vývojáře oprostila od přímého styku s kryptografickými koncepty.

V následující tabulce jsou shrnuta důležitější kritéria:

Tabulka 5.4 Hodnocení použitelnosti WolfSSL

Bod	Kritéria	
API	pokrývá potřebnou funkcionalitu	✔
	abstrakce pro běžné scénáře	✗
	bezpečné výchozí hodnoty	✗
	konzistentní a srozumitelné návratové hodnoty	✔
Dokumentace	přehledná, dobře se naviguje a vyhledává	✔
	obsahuje ukázky použití vedoucí k bezpečnému kódu	✓
	obsahuje varování u zastaralých algoritmů a nebezpečných módů	✓
	dostatečně popisuje argumenty a jejich předpoklady	✔

(✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

5.4 Chyby v aplikacích

Tato sekce se pokouší pohlížet na problém špatného návrhu knihoven z druhé strany. Tedy, místo zkoumání a hodnocení samotného návrhu, se snaží nalézt chyby v použití knihoven způsobené právě špatným návrhem knihovny. Z těchto chyb by poté bylo možné zjistit, jaké jsou problémy s použitím konkrétní knihovny v praxi.

5.4.1 Postup

Před hledáním chyb v použití knihoven je ještě potřeba shrnout, jakým způsobem lze tyto chyby hledat. Hledání vhodných aplikací není úplně jednoduché. Některé knihovny mají na svých stránkách použitelné seznamy aplikací, které velmi zjednoduší proces hledání, ale některé tyto informace nemají či neposkytují, je tedy potřeba hledat jinak.

Způsob hledání aplikací používající knihovny

Jak bylo řečeno výše, jedna (nejjednodušší) možnost je využít seznam aplikací poskytnutý knihovnou. Pokud ale tento seznam nestačí nebo vůbec poskytován není, lze využít známých vývojových prostředí, jako Github či Gitlab, kde lze vyhledávat konkrétní řetězce ve veřejných repozitářích (open-source) aplikací.

Tímto způsobem lze hledat přímo jméno knihovny — zde je ale velká šance, že výsledkem bude velké množství falešně pozitivních výsledků — jméno knihovny může být jednoduše zmíněno někde v komentářích nebo issues přes to, že aplikace knihovnu nepoužívá. Nabízí se tedy hledat podle typických řetězců pro knihovny, které se mnohem pravděpodobněji budou vyskytovat tam, kde je knihovna použita — inicializační funkce knihovny či typické konstanty knihovny. Takto lze přes uživatelské rozhraní např. Githubu nebo přes API získat seznam aplikací, ze kterého si lze nadále vybírat.

Na otázku, které aplikace je vhodné vybrat, není jasná odpověď. Lze ale předpokládat, že více chyb bude objevitelných v aktivněji vyvíjených aplikacích dostatečně známých na to, aby v nich někdo hledal a nahlásil chyby/zranitelnosti. Dále lze zohlednit typ aplikace — vyzkoušet aplikace, kde je kryptografie důležitá součást, ale i aplikace, kde není zaměření na bezpečnost vysoké — vývojář bude pravděpodobně méně znalý kryptografie a může snadněji udělat chybu v použití knihovny, když mu to knihovna dovolí.

Způsob hledání CVE pro dané aplikace

Když již byla zvolena aplikace na prozkoumání, lze začít s hledáním chyb v použití. Jako jedna schůdná cesta se jeví prozkoumat CVE, která by mohla souviset s kryptografií, a tedy s knihovnou a jejím použitím.

Pro CVE existuje více různých databází, jako MITRE[1] a americký NIST[2]. Je vhodné vyzkoušet různé databáze pro případ, že by někde byla CVE, která jinde nejsou. Zde dobře funguje hledání CVE dle jména aplikace — u zranitelnosti je v popisu vždy napsané, které aplikace se problém týká.

Lze předpokládat, že počet CVE pro aplikaci může být velký a velká část nebude souviset s knihovnou — je možné CVE vyfiltrovat. Opět se nabízí otázka: podle čeho filtrovat? Je možné použít jméno knihovny a tím nalézt CVE, která mají v popisu jak jméno konkrétní aplikace, tak knihovny — těch je ale nízký počet a většina bývá zranitelnost aplikace způsobená zranitelností knihovny samotné. Další varianta je použít klíčová slova související s kryptografií. Tímto způsobem získáme více výsledků, z nichž více může souviset s použitím knihovny.

[1] `https://cve.mitre.org/`
[2] `https://nvd.nist.gov/vuln/search`

Použitý seznam klíčových slov:

- encrypt
- decrypt
- signature
- certificate
- hash
- crypto
- cipher
- sign
- verify

Souvislost CVE s použitím knihovny

Pro CVE vybraná způsobem zmíněným výše je nadále potřeba hlouběji prozkoumat, jak a jestli vůbec souvisí se špatným použitím knihovny. Většinou, pokud to není jasné z popisu problému, je potřeba dohledat commit/změnu v repozitáři aplikace, která chybu opravila. Ve chvíli kdy změna zasahuje do kódu přímo pracujícím s knihovnou, tak je zde velká šance na úspěch. Je možné ještě prozkoumat, co přesně problém způsobilo, jestli to byla jen obecná programátorská chyba (jako špatné pořadí podmínek nesouvisejících s knihovnou, nevhodně umístěná funkce do podmínky...), nebo opravdu šlo o chybu způsobenou nepochopením kryptografie nebo rozhraní knihovny. Takové CVE jsou zde označena jako související se špatným použitím knihovny a naznačují špatně navržené API, případně nedostatečnou dokumentaci a příklady použití.

5.4.2 OpenSSL

Následuje analýza vybraných aplikací a jejich CVE pro knihovnu OpenSSL.

Analyzované aplikace

Zde je seznam zkoumaných aplikací, které používají knihovnu OpenSSL.

- Ceph[3] — systém úložiště
  - Celkem 59 CVE.
  - Přes klíčová slova bylo vybráno 10 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- Wget[4] — nástroj pro stahování
  - Celkem 53 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- Pacemaker[5] — správa zdrojů v clusteru
  - Celkem 19 CVE.
  - Přes klíčová slova bylo vybráno 1 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- NgIRCd[6] — chatovací server

[3] `https://github.com/ceph/ceph`
[4] `https://github.com/mirror/wget`
[5] `https://github.com/clusterlabs/pacemaker`
[6] `https://github.com/ngircd/ngircd`

- Celkem 8 CVE.
- Přes klíčová slova bylo vybráno 1 CVE pro hlubší zkoumání.
- Kódu pro použití OpenSSL se netýkalo žádné CVE.

- GnuPG[7] — Gnu Privacy guard (bezpečnostní nástroje)
  - Celkem 94 CVE.
  - Přes klíčová slova bylo vybráno 22 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- Exim[8] — mail agent
  - Celkem 142 CVE.
  - Přes klíčová slova bylo vybráno 5 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- Mongoose[9] — Embedded Web Server
  - Celkem 42 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- OpenVPN[10] — Open-source VPN
  - Celkem 131 CVE.
  - Přes klíčová slova bylo vybráno 18 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se týkala 2 CVE: CVE-2005-2532, CVE-2005-2531.
- OpenMPT[11] — nástroj pro vykreslování zvuku
  - Celkem 10 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- TizenRT[12] — operační systém
  - Celkem 4 CVE
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- MySQL Server[13] — databázový server
  - Celkem cca 1100 CVE.
  - Přes klíčová slova bylo vybráno 36 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- OpenWRT[14] — Linuxový OS pro vestavné systémy
  - Celkem 68 CVE.
  - Přes klíčová slova byla vybrána 3 CVE pro hlubší zkoumání.

[7] https://github.com/gpg/gnupg
[8] https://github.com/exim/exim
[9] https://github.com/cesanta/mongoose
[10] https://github.com/openvpn/openvpn
[11] https://github.com/openmpt/openmpt
[12] https://github.com/samsung/tizenrt
[13] https://github.com/mysql/mysql-server
[14] https://github.com/openwrt/openwrt

  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- Lighttpd[15] — web server
  - Celkem 49 CVE.
  - Přes klíčová slova byla vybrána 4 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se týkalo 1 CVE: CVE-2013-4508.
- LuCI[16] — OpenWRT konfigurační rozhraní
  - Celkem 15 CVE.
  - Přes klíčová slova bylo vybráno 1 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- Curl[17] — nástroj pro přenos dat s URL syntaxí
  - Celkem 385 CVE.
  - Přes klíčová slova bylo vybráno 46 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se týkala 3 CVE: CVE-2020-8286, CVE-2024-0853, CVE-2009-2417.
- CJose[18] — jose pro c/c++
  - Celkem 2 CVE.
  - Přes klíčová slova byla vybrána 2 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádné CVE.
- Nginx[19] — webový server
  - Celkem 225 CVE.
  - Přes klíčová slova bylo vybráno 16 CVE pro hlubší zkoumání.
  - Kódu pro použití OpenSSL se netýkalo žádne CVE.
- OpenSSH[20] — implementace SSH protokolu
  - Celkem 156 CVE.
  - Přes klíčová slova bylo vybráno 18 CVE pro hlubší zkoumání.
  - kódu pro použití OpenSSL se netýkalo žádné CVE.

Analýza nalezených problémů a CVE

Nalezené problémy:

- OpenVPN — CVE-2005-2532[21]
  - **Problém:** Aplikace nečistila správně frontu chyb OpenSSL, čímž bylo možné způsobit DoS.

[15] https://github.com/lighttpd/lighttpd1.4
[16] https://github.com/openwrt/luci
[17] https://github.com/curl/curl
[18] https://github.com/cisco/cjose
[19] https://github.com/nginx/nginx
[20] https://github.com/openssh/openssh-portable
[21] https://nvd.nist.gov/vuln/detail/CVE-2005-2532

- **Dopad:** Možnost odepření služby, když útočník poslal hodně chybných paketů.
- **Závažnost:** 5.0 Medium (CVSS verze 2.0)
- **Souvisí to s knihovnou?** Ano.

- OpenVPN — CVE-2005-2531[22]
  - **Problém:** Podobný případ jako předchozí — aplikace nečistila správně frontu chyb knihovny OpenSSL, čímž bylo možné způsobit DoS.
  - **Dopad:** Možnost odepření služby, když selhala autentizace klientského certifikátu.
  - **Závažnost:** 5.0 Medium (CVSS verze 2.0)
  - **Souvisí to s knihovnou?** Ano.
- lighttpd — CVE-2013-4508[23]
  - **Problém:** Aplikace nastavovala za určitých podmínek slabé šifry pro SSL/TLS.
  - **Dopad:** Snížení bezpečnosti — slabší šifry, snadnější prolomit pro útočníky.
  - **Závažnost:** 7.5 High (CVSS verze 3.x)
  - **Souvisí to s knihovnou?** Ano.
- curl — CVE-2020-8286[24]
  - **Problém:** Špatná validace revokovaných certifikátů.
  - **Dopad:** Možnost použít neplatné certifikáty útočníky.
  - **Závažnost:** 7.5 High (CVSS verze 3.x)
  - **Souvisí to s knihovnou?** Ano.
- curl — CVE-2024-0853[25]
  - **Problém:** SSL/TLS session se udržovala moc dlouho — bylo možné ji znovu použít a tím přeskočit důležité ověřovací kroky.
  - **Dopad:** Možnost přeskočit důležitá ověření, a tím navázat nebezpečné připojení.
  - **Závažnost:** 5.3 Medium (CVSS verze 3.x)
  - **Souvisí to s knihovnou?** Ano.
- curl — CVE-2009-2417[26]
  - **Problém:** Aplikace mylně předpokládala, že řetězce v certifikátu jsou zakončené nulovým znakem.
  - **Dopad:** Možnost přečtení kratšího řetězce, než se v certifikátu doopravdy vyskytoval.
  - **Závažnost:** 7.5 High (CVSS verze 2.0)
  - **Souvisí to s knihovnou?** Ano.

Celkový součet CVE:

- 18 zkoumaných aplikací

[22] https://nvd.nist.gov/vuln/detail/CVE-2005-2531
[23] https://nvd.nist.gov/vuln/detail/CVE-2013-4508
[24] https://nvd.nist.gov/vuln/detail/CVE-2020-8286
[25] https://nvd.nist.gov/vuln/detail/CVE-2024-0853
[26] https://nvd.nist.gov/vuln/detail/CVE-2009-2417

- cca 2560 CVE celkem
- 183 CVE vybráno přes klíčová slova v popisu a prozkoumáno detailněji
- 6 CVE zasahovalo (respektive jejich oprava) do kódu pracujícím s knihovnou
- 6 CVE souviselo se špatným použitím knihovny — souvisí jak se špatným návrhem funkcí, tak s dokumentací.

5.4.3 Mbed TLS

Následuje analýza vybraných aplikací a jejich CVE pro knihovnu Mbed TLS.

Analyzované aplikace

Zde je seznam zkoumaných aplikací, které používají knihovnu Mbed TLS.

- Mongoose[27] — Embedded Web Server
  - Celkem 42 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- OpenVPN[28] — Open-source VPN
  - Celkem 131 CVE.
  - Přes klíčová slova bylo vybráno 18 CVE pro hlubší zkoumání.
  - Kódu pro použití mbedTLS se týkalo 1 CVE: CVE-2017-7522.
- Zephyr[29] — Bezpečný operační systém pro zařízení s omezenými zdroji
  - Celkem 111 CVE.
  - Přes klíčová slova bylo vybráno 6 CVE pro hlubší zkoumání.
  - Kódu pro použití mbedTLS se netýkalo žádné CVE.
- FreeRTOS[30] — Operační systém
  - Celkem 21 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- RiotOS[31] - Operační systém pro IoT[32]
  - Celkem 37 CVE.
  - Přes klíčová slova bylo vybráno 1 CVE pro hlubší zkoumání.
  - CVE zřejmě nemá velký dopad ani prioritu a dodnes není opraveno. Dle dohledatelných informací nesouvisí se špatným použitím Mbed TLS.
- Moddable[33] — nástroj na tvorbu nástrojů pro mikrokontroléry
  - Celkem 25 CVE.

[27] `https://github.com/cesanta/mongoose`
[28] `https://github.com/openvpn/openvpn`
[29] `https://github.com/zephyrproject-rtos/zephyr`
[30] `https://github.com/FreeRTOS/FreeRTOS`
[31] `https://github.com/riot-os/riot`
[32] Internet of Things — Internet věcí
[33] `https://github.com/moddable-opensource/moddable`

  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- OpenMPT[34] — nástroj pro vykreslování zvuku
  - Celkem 10 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- TizenRT[35] — operační systém
  - Celkem 4 CVE
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- LiteOS[36] — operační systém
  - Celkem 5 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- Cuberite[37] — minecraft server
  - Celkem 1 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- MySQL Server[38] — databázový server
  - Celkem cca 1100 CVE.
  - Přes klíčová slova bylo vybráno 36 CVE pro hlubší zkoumání.
  - Kódu pro použití mbedTLS se netýkalo žádné CVE.
- OpenWRT[39] — Linuxový OS pro vestavné systémy
  - Celkem 68 CVE.
  - Přes klíčová slova byla vybrána 3 CVE pro hlubší zkoumání.
  - Kódu pro použití mbedTLS se netýkalo žádné CVE.
- Lighttpd[40] — web server
  - Celkem 49 CVE.
  - Přes klíčová slova byla vybrána 4 CVE pro hlubší zkoumání.
  - Kódu pro použití mbedTLS se netýkalo žádné CVE.
- Curl[41] — nástroj pro přenos dat s URL syntaxí
  - Celkem 385 CVE.
  - Přes klíčová slova bylo vybráno 46 CVE pro hlubší zkoumání.
  - Kódu pro použití mbedTLS se týkalo 1 CVE: CVE-2016-3739.

[34] `https://github.com/openmpt/openmpt`
[35] `https://github.com/samsung/tizenrt`
[36] `https://github.com/LiteOS/LiteOS`
[37] `https://github.com/cuberite/cuberite`
[38] `https://github.com/mysql/mysql-server`
[39] `https://github.com/openwrt/openwrt`
[40] `https://github.com/lighttpd/lighttpd1.4`
[41] `https://github.com/curl/curl`

Analýza nalezených problémů a CVE

Nalezené problémy:

- OpenVPN — CVE-2017-7522[42]
  - **Problém:** Funkce používaná při kontrole platnosti certifikátu vrací jiný typ stringu, než volající očekává. Na základě toho pak volající funkce spadne.
  - **Dopad:** DoS[43] v případě, že vstupní string obsahuje NUL hodnotu.
  - **Závažnost:** 6.5 Medium (CVSS verze 3.x)
  - **Souvisí to s knihovnou?** Ne. Typ stringu vraceného konkrétní funkcí aplikace nesouvisí s knihovnou, je to problém aplikace samotné (tato chyba se ovšem objevila pouze v kódu pracujícím s Mbed TLS — ne v kódu pracujícím s ostatními podporovanými knihovnami).
- curl — CVE-2016-3739[44]
  - **Problém:** Při kontrole certifikátu serveru se za určitých podmínek mohla přeskočit funkce nastavující jméno serveru — jméno se tedy nezkontrolovalo. Tento problém vznikl nevhodným umístěním funkce do podmínky.
  - **Dopad:** Snížení bezpečnosti — klientovi nevadí se připojit k serveru se špatným jménem.
  - **Závažnost:** 5.3 Medium (CVSS verze 3.x)
  - **Souvisí to s knihovnou?** Ne. Jedná se o programátorskou chybu nesouvisející s knihovnou — vývojář věděl, že má jméno zkontrolovat, jen tuto kontrolu špatně umístil.

Celkový součet CVE:

- 14 zkoumaných aplikací
- cca 1990 CVE celkem
- 114 CVE vybráno přes klíčová slova v popisu a prozkoumáno detailněji
- 2 CVE zasahovala (respektive jejich oprava) do kódu pracujícím s knihovnou
- Žádné ve výsledku nesouviselo se špatným použitím knihovny — tedy na těchto vzorcích nenalezen problém se špatným návrhem knihovny.

5.4.4 GnuTLS

Následuje analýza vybraných aplikací a jejich CVE pro knihovnu GnuTLS.

Analyzované aplikace

Zde je seznam zkoumaných aplikací, které používají knihovnu GnuTLS.

- TigerVNC[45] — nástroj na vzdálený displej
  - Celkem 20 CVE.
  - Přes klíčová slova byla vybrána 2 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se netýkalo žádné CVE.

[42] `https://nvd.nist.gov/vuln/detail/CVE-2017-7522`
[43] Denial of Service — odepření služby
[44] `https://nvd.nist.gov/vuln/detail/CVE-2016-3739`
[45] `https://github.com/tigervnc/tigervnc`

- Wget[46] — nástroj pro stahování
  - Celkem 53 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- Pacemaker[47] — správa zdrojů v clusteru
  - Celkem 19 CVE.
  - Přes klíčová slova bylo vybráno 1 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se netýkalo žádné CVE.
- WeeChat[48] — chatovací klient
  - Celkem 13 CVE.
  - Přes klíčová slova byla vybrána 2 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se týkala obě CVE: CVE-2022-28352 a CVE-2011-1428.
- NgIRCd[49] — chatovací server
  - Celkem 8 CVE.
  - Přes klíčová slova bylo vybráno 1 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se netýkalo žádné CVE.
- Knot[50] — DNS server
  - Celkem 14 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- GnuPG[51] — Gnu Privacy guard (bezpečnostní nástroje)
  - Celkem 94 CVE.
  - Přes klíčová slova bylo vybráno 22 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se týkalo 1 CVE: CVE-2003-0971.
- Exim[52] — mail agent
  - Celkem 142 CVE.
  - Přes klíčová slova bylo vybráno 5 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se netýkalo žádné CVE.
- OpenMPT[53] — nástroj pro vykreslování zvuku
  - Celkem 10 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- TizenRT[54] — operační systém

[46] https://github.com/mirror/wget
[47] https://github.com/clusterlabs/pacemaker
[48] https://github.com/weechat/weechat
[49] https://github.com/ngircd/ngircd
[50] https://github.com/cz-nic/knot
[51] https://github.com/gpg/gnupg
[52] https://github.com/exim/exim
[53] https://github.com/openmpt/openmpt
[54] https://github.com/samsung/tizenrt

  - Celkem 4 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- MySQL Server[55] — databázový server
  - Celkem cca 1100 CVE.
  - Přes klíčová slova bylo vybráno 36 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se netýkalo žádné CVE.
- OpenWRT[56] — Linuxový OS pro vestavné systémy
  - Celkem 68 CVE.
  - Přes klíčová slova byla vybrána 3 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se netýkalo žádné CVE.
- Lighttpd[57] — web server
  - Celkem 49 CVE.
  - Přes klíčová slova byla vybrána 4 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se netýkalo žádné CVE.
- Curl[58] — nástroj pro přenos dat s URL syntaxí
  - Celkem 385 CVE.
  - Přes klíčová slova bylo vybráno 46 CVE pro hlubší zkoumání.
  - Kódu pro použití GnuTLS se týkalo 1 CVE: CVE-2007-3564.

Analýza nalezených problémů a CVE

Nalezené problémy:

- WeeChat — CVE-2022-28352[59]
  - **Problém:** Aplikace po změně nastavení GnuTLS neprovede znovu kontrolu certifikátu serveru, jak by měla.
  - **Dopad:** Nezkontroluje se certifikát serveru a aplikace se tedy připojí k čemukoliv.
  - **Závažnost:** 4.8 Medium (CVSS verze 3.x)
  - **Souvisí to s knihovnou?** Ne. Vývojář se snažil implementovat novou funkčnost a v rámci toho dal kontrolu certifikátu na špatné místo.
- WeeChat — CVE-2011-1428[60]
  - **Problém:** Špatné použití API pro kontrolu certifikátu.
  - **Dopad:** Nezkontroluje se správně certifikát serveru a útočník může podvrhnout certifikát a server.
  - **Závažnost:** 5.8 Medium (CVSS verze 2.0)
  - **Souvisí to s knihovnou?** Ano.

[55] https://github.com/mysql/mysql-server
[56] https://github.com/openwrt/openwrt
[57] https://github.com/lighttpd/lighttpd1.4
[58] https://github.com/curl/curl
[59] https://nvd.nist.gov/vuln/detail/CVE-2022-28352
[60] https://nvd.nist.gov/vuln/detail/CVE-2011-1428

- gnupg — CVE-2003-0971[61]
  - **Problém:** Špatné použití ElGamal komponenty knihovny, stejným způsobem se šifruje jako podepisuje.
  - **Dopad:** Možnost zjištění privátního klíče z podpisu.
  - **Závažnost:** 5.0 Medium (CVSS verze 2.0)
  - **Souvisí to s knihovnou?** Ano.
- curl — CVE-2007-3564[62]
  - **Problém:** Při kontrole certifikátu serveru se zapomnělo na kontrolu času platnosti certifikátu.
  - **Dopad:** Konkrétní servery mohly mít expirovaný certifikát, aplikace by se přes to připojila.
  - **Závažnost:** 7.5 High (CVSS verze 2.0)
  - **Souvisí to s knihovnou?** Ano.

Celkový součet CVE:

- 14 zkoumaných aplikací
- cca 1980 CVE celkem
- 122 CVE vybráno přes klíčová slova v popisu a prozkoumáno detailněji
- 4 CVE zasahovala (respektive jejich oprava) do kódu pracujícím s knihovnou
- 3 CVE souvisela se špatným použitím knihovny — netýkala se sice přímo špatného návrhu funkcí, ale spíš chabé dokumentace, případně chybějících správných příkladů použití.

5.4.5 WolfSSL

Následuje analýza vybraných aplikací a jejich CVE pro knihovnu WolfSSL.

Analyzované aplikace

Zde je seznam zkoumaných aplikací, které používají knihovnu wolfSSL

- MySQL Server[63] — databázový server
  - Celkem cca 1100 CVE.
  - Přes klíčová slova bylo vybráno 36 CVE pro hlubší zkoumání.
  - Kódu pro použití wolfSSL se netýkalo žádné CVE.
- OpenWRT[64] — Linuxový OS pro vestavné systémy
  - Celkem 68 CVE.
  - Přes klíčová slova byla vybrána 3 CVE pro hlubší zkoumání.
  - Kódu pro použití wolfSSL se netýkalo žádné CVE.
- Lighttpd[65] — web server

[61] https://nvd.nist.gov/vuln/detail/CVE-2003-0971
[62] https://nvd.nist.gov/vuln/detail/CVE-2007-3564
[63] https://github.com/mysql/mysql-server
[64] https://github.com/openwrt/openwrt
[65] https://github.com/lighttpd/lighttpd1.4

- Celkem 49 CVE.
- Přes klíčová slova byla vybrána 4 CVE pro hlubší zkoumání.
- Kódu pro použití wolfSSL se netýkalo žádné CVE.

- Curl[66] — nástroj pro přenos dat s URL syntaxí
  - Celkem 385 CVE.
  - Přes klíčová slova bylo vybráno 46 CVE pro hlubší zkoumání.
  - Kódu pro použití wolfSSL se netýkalo žádné CVE.
- Mongoose[67] — Embedded Web Server
  - Celkem 42 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).
- Gearman[68] — aplikační framework
  - Nebylo nalezeno žádné CVE.
- Fb4nds[69] — Facebookový klient do chatu obsahující Nintendo hry
  - Nebylo nalezeno žádné CVE.
- ChibiOS[70] — operační systém
  - Nebylo nalezeno žádné CVE.
- Open Vehicle Monitoring System 3[71]
  - Nebylo nalezeno žádné CVE.
- RiotOS[72] — Operační systém pro IoT
  - Celkem 37 CVE.
  - Přes klíčová slova bylo vybráno 1 CVE pro hlubší zkoumání.
  - CVE zřejmě nemá velký dopad ani prioritu a dodnes není opraveno. Dle dohledatelných informací nesouvisí se špatným použitím WolfSSL.
- OpenVPN[73] — Open-source VPN
  - Celkem 131 CVE.
  - Přes klíčová slova bylo vybráno 18 CVE pro hlubší zkoumání.
  - kódu pro použití wolfSSL se netýkalo žádné CVE.
- OpenResty[74] — výkonná webová platforma
  - Celkem 4 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).

[66] https://github.com/curl/curl
[67] https://github.com/cesanta/mongoose
[68] https://github.com/gearman/gearmand
[69] https://github.com/linoma/fb4nds
[70] https://github.com/chibios/chibios
[71] https://github.com/openvehicles/Open-Vehicle-Monitoring-System-3
[72] https://github.com/riot-os/riot
[73] https://github.com/openvpn/openvpn
[74] https://github.com/openresty/openresty

- OpenSSH[75] — implementace SSH protokolu
  - Celkem 156 CVE.
  - Přes klíčová slova bylo vybráno 18 CVE pro hlubší zkoumání.
  - kódu pro použití wolfSSL se netýkalo žádné CVE.
- FreeRTOS[76] — Operační systém
  - Celkem 21 CVE.
  - Žádné nebylo vybráno pro hlubší zkoumání (filtr přes klíčová slova).

Celkový součet CVE:

- 14 zkoumaných aplikací
- cca 2000 CVE celkem
- 126 CVE vybráno přes klíčová slova v popisu a prozkoumáno detailněji
- Žádná CVE nezasahovala do kódu pracujícím s knihovnou

5.5 Shrnutí

Z literární rešerše jednoznačně vyplývá, že návrh kryptografických knihoven hraje velkou roli v bezpečnosti software, který tyto knihovny používá. Existuje více různých seznamů doporučení pro správný návrh a dobrou dokumentaci, přesto z vlastního výzkumu lze zjistit, že důležité kryptografické knihovny stále postrádají některé potřebné kvality. U návrhu API tomu tak může být z více důvodů. Knihovny zkoumané v části 5.3 jsou všechny vyvíjené již poměrně dlouhou dobu, kvůli zpětné kompatibilitě tedy může být problém API předělat. Také fakt, že jsou tyto knihovny napsané v C, může být ideálnímu návrhu překážkou. Všechny by ale mohly přidat vrstvu abstrakce pro běžnější scénáře, aby i kryptografičtí ne-experti mohli knihovny bezpečně použít. Kvalita dokumentace ovšem zpětnou kompatibilitou zatížena není — konkrétně knihovna OpenSSL má v této oblasti silné nedostatky, ale i např. knihovna GnuTLS má co zlepšovat. Například přidat ukázkový kód pro kryptografické API.

Ze zkoumaných knihoven si nejlépe stojí WolfSSL a Mbed TLS, které obě, kromě postrádané vyšší úrovně abstrakce pro běžné scénáře, zvládly splnit jak většinu požadavků na návrh, tak o hodně předčit ostatní dvě knihovny v kvalitě dokumentace.

Co se týče dohledávání chyb použití v aplikacích, je sice možné tímto způsobem nějaké chyby najít, ale i pro takový úkol je tato metoda velmi neefektivní. Pro porovnání knihoven či jejich ohodnocení by bylo potřeba prozkoumat mnohem více aplikací a zřejmě i jiným způsobem. Způsob, který byl zde použit (dohledávání chyb přes CVE aplikací), pouze ukázal, že chyby existují, a více podložil některé poznatky z části 5.1, a to jaké jsou obecně běžné chyby v použití kryptografie.

[75] `https://github.com/openssh/openssh-portable`
[76] `https://github.com/FreeRTOS/FreeRTOS`

Kapitola 6
Závěr

Cílem práce bylo provést analýzu možných kritérií, která ovlivňují jak bezpečnost kryptografických knihoven samotných, tak bezpečnost jejich použití vývojářem. Dále bylo cílem tato kritéria shrnout v obecnou příručku pro vývojáře, díky které by si mohli sami ověřit, která knihovna je pro ně ta pravá, nebo také pouze ohodnotit bezpečnost použití konkrétní knihovny.

Kryptografické knihovny jsou velmi důležitým nástrojem pro vývojáře, kteří chtějí ve svém software použít kryptografii, ale chtějí si ušetřit práci s její implementací. Ta by jinak byla velmi náročná a náchylná k těžko odhalitelným chybám. Takovéto porovnání a ohodnocení knihoven je vhodné, protože použití knihovny (tedy cizího kódu) s sebou nese potenciálně nebezpečnou závislost. Kritérii, která byla cílem sepsat, se vývojář může přesvědčit o míře takového rizika.

Tato kritéria jsou dle své povahy rozdělena do kapitol 4 a 5. V příslušných kapitolách byla nejprve kritéria navržena s přihlédnutím k již existujícím výzkumům. Poté byla kritéria aplikována na vybrané open-source kryptografické knihovny a bylo poté vyhodnoceno, jak jsou která kritéria efektivní a užitečná. Dohledávání špatných použití kryptografie v aplikacích se ukázalo jako nejpracnější část a zároveň jako část dávající nejméně použitelných výsledků. Zbylá kritéria se jeví jako použitelný návod pro ohodnocení bezpečnosti kryptografických knihoven.

Kritéria byla zkoumána na čtyřech kryptografických knihovnách: OpenSSL, Mbed TLS, GnuTLS a WolfSSL. Z těchto knihoven se jako dobré volby jeví Mbed TLS a WolfSSL. Knihovna WolfSSL jednoznačně předčila ostatní v přehlednosti dokumentace a i ostatní požadavky splnila vcelku obstojně. Knihovna Mbed TLS zase kromě méně přehledné, ale i tak kvalitní dokumentace nejlépe obstála v kategorii vývojových procesů.

Knihovny GnuTLS a OpenSSL se naopak snaží předčit v tom, která splní požadavků méně. V této soutěži je těžké rozhodnout, obě knihovny totiž mají různé základní nedostatky. Dokumentace a návrh knihovny OpenSSL velmi snižují šanci na bezpečné použití. GnuTLS má zase nejmenší a nejméně aktivní tým správců, hrozí tedy pomalejší reakce na objevené zranitelnosti.

I přes splnění všech zde zmíněných požadavků může jakákoli knihovna obsahovat zranitelnost či se do ní může zranitelnost dostat. Také obrana před špatným použitím není všemocná, vždy se může najít někdo, kdo ji využije špatně. Záruka bezpečnosti v software nikdy nebude stoprocentní, ale i tak je vhodné se při výběru kryptografické knihovny přesvědčit o snaze vývojářů zajistit bezpečnost co nejvíce.

Bibliografie

1. BLESSING, Jenny; SPECTER, Michael A; WEITZNER, Daniel J. You Really Shouldn't Roll Your Own Crypto: An Empirical Study of Vulnerabilities in Cryptographic Libraries. *arXiv preprint arXiv:2107.04940*. 2021. Dostupné také z: `https://arxiv.org/abs/2107.04940`.
2. BOUGHTON, Lina; MILLER, Courtney; ACAR, Yasemin; WERMKE, Dominik; KÄSTNER, Christian. Decomposing and Measuring Trust in Open-Source Software Supply Chains. 2024. Dostupné také z: `https://www.cs.cmu.edu/~ckaestne/pdf/icsenier24.pdf`.
3. ŠPINKA, Milan. *Kritéria pro hodnocení bezpečnosti kryptografických knihoven*. 2024. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
4. LEONOV, Kirill. *Kritéria pro hodnocení bezpečnosti kryptografických knihoven*. 2024. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
5. ISTQB. *Certified Tester Foundation Level (CTFL) — istqb.org* [online]. 2024. [cit. 2024-04-17]. Dostupné z: `https://www.istqb.org/certifications/certified-tester-foundation-level`.
6. LEBLANC, D.; HOWARD, M. *Writing Secure Code*. Pearson Education, 2002. Developer Best Practices. ISBN 9780735637405. Dostupné také z: `https://books.google.cz/books?id=nZVCAwAAQBAJ`.
7. SAMONAS, Spyridon; COSS, David. The CIA strikes back: Redefining confidentiality, integrity and availability in security. *Journal of Information System Security*. 2014, roč. 10, č. 3. Dostupné také z: `https://www.proso.com/dl/Samonas.pdf`.
8. HAZHIRPASAND, Mohammadreza; NIERSTRASZ, Oscar; SHABANI, Mohammadhossein; GHAFARI, Mohammad. Hurdles for developers in cryptography. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, s. 659–663. Dostupné také z: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9609232`.
9. GREEN, Matthew; SMITH, Matthew. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*. 2016, roč. 14, č. 5, s. 40–46. Dostupné také z: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7676144`.
10. OPEN SOURCE INITIATIVE. *The Open Source Definition* [online]. 2007. [cit. 2024-04-16]. Dostupné z: `https://opensource.org/osd`.
11. SCHRYEN, Guido. Security of open source and closed source software: An empirical comparison of published vulnerabilities. *AMCIS 2009 Proceedings*. 2009, s. 387. Dostupné také z: `https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1390&context=amcis2009`.

12. RAYMOND, Eric. The cathedral and the bazaar. *Knowledge, Technology & Policy*. 1999, roč. 12, č. 3, s. 23–49. Dostupné také z: https://www.root.cz/knihy/katedrala-a-trziste/. Do češtiny přeložil Miroslav Nič.

13. WEN, Shao-Fang; KIANPOUR, Mazaher; KATT, Basel. Security knowledge management in open source software communities. In: *Innovative Security Solutions for Information Technology and Communications: 11th International Conference, SecITC 2018, Bucharest, Romania, November 8–9, 2018, Revised Selected Papers 11*. Springer, 2019, s. 53–70. Dostupné také z: https://link.springer.com/chapter/10.1007/978-3-030-12942-2_6.

14. WEN, Shao-Fang. Software security in open source development: A systematic literature review. In: *2017 21st Conference of Open Innovations Association (FRUCT)*. IEEE, 2017, s. 364–373. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8250205.

15. SCHRYEN, Guido; KADURA, Rouven. Open source vs. closed source software: towards measuring security. In: *Proceedings of the 2009 ACM symposium on Applied Computing*. 2009, s. 2016–2023. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/1529282.1529731.

16. MENEELY, Andrew; TEJEDA, Alberto C Rodriguez; SPATES, Brian; TRUDEAU, Shannon; NEUBERGER, Danielle; WHITLOCK, Katherine; KETANT, Christopher; DAVIS, Kayla. An empirical investigation of socio-technical code review metrics and security vulnerabilities. In: *Proceedings of the 6th International Workshop on Social Software Engineering*. 2014, s. 37–44. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/2661685.2661687.

17. PETITCOLAS, Fabien AP. Kerckhoffs' principle. In: *Encyclopedia of Cryptography, Security and Privacy*. Springer, 2023, s. 1–2. Dostupné také z: https://link.springer.com/referenceworkentry/10.1007/978-3-642-27739-9_487-2.

18. KERCKHOFFS, Auguste. La cryptographie militaire. *Journal des Sciences Militaires*. 1883, s. 161–191.

19. SYNOPSIS. *[Analyst Report] 2024 Open Source Security and Analysis Report (OSSRA) — Synopsys — synopsys.com* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html.

20. SCACCHI, Walt; FELLER, Joseph; FITZGERALD, Brian; HISSAM, Scott; LAKHANI, Karim. *Understanding free/open source software development processes*. Sv. 11. Wiley Online Library, 2006. Č. 2.

21. SHIN, Yonghee; MENEELY, Andrew; WILLIAMS, Laurie; OSBORNE, Jason A. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE transactions on software engineering*. 2010, roč. 37, č. 6, s. 772–787. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5560680.

22. WALDEN, James. The impact of a major security event on an open source project: The case of OpenSSL. In: *Proceedings of the 17th international conference on mining software repositories*. 2020, s. 409–419. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/3379597.3387465.

23. GRAVES, Todd L; KARR, Alan F; MARRON, James S; SIY, Harvey. Predicting fault incidence using software change history. *IEEE Transactions on software engineering*. 2000, roč. 26, č. 7, s. 653–661. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=859533.

24. NAGAPPAN, Nachiappan; BALL, Thomas. Use of relative code churn measures to predict system defect density. In: *Proceedings of the 27th international conference on Software engineering*. 2005, s. 284–292. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/1062455.1062514.

25. MCINTOSH, Shane; KAMEI, Yasutaka; ADAMS, Bram; HASSAN, Ahmed E. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: *Proceedings of the 11th working conference on mining software repositories*. 2014, s. 192–201. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/2597073.2597076.

26. BOSU, Amiangshu. Characteristics of the vulnerable code changes identified through peer code review. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. 2014, s. 736–738. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/2591062.2591200.

27. SYNOPSYS, Inc. http://www.synopsys.com/. *Heartbleed Bug — heartbleed.com* [online]. 2020. [cit. 2024-04-17]. Dostupné z: https://heartbleed.com/.

28. DURUMERIC, Zakir; LI, Frank; KASTEN, James; AMANN, Johanna; BEEKMAN, Jethro; PAYER, Mathias; WEAVER, Nicolas; ADRIAN, David; PAXSON, Vern; BAILEY, Michael et al. The matter of heartbleed. In: *Proceedings of the 2014 conference on internet measurement conference*. 2014, s. 475–488. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/2663716.2663755.

29. WHEELER, David A. *Core Infrastructure Initiative (CII) Best-Practices Badge Criteria*. JSTOR, 2022. Dostupné také z: https://www.ida.org/-/media/feature/publications/c/co/core-infrastructure-initiative-cii-bestpractices-badge-criteria/d-8054.ashx.

30. OPENSSF. *BadgeApp — bestpractices.dev* [online]. 2021. [cit. 2024-04-23]. Dostupné z: https://www.bestpractices.dev/en/criteria/0.

31. *NVD - CVE-2020-13777 — nvd.nist.gov* [online]. 2020. [cit. 2024-04-17]. Dostupné z: https://nvd.nist.gov/vuln/detail/CVE-2020-13777.

32. OPENSSF. *Concise Guide for Evaluating Open Source Software — best.openssf.org* [online]. 2023. [cit. 2024-04-17]. Dostupné z: https://best.openssf.org/Concise-Guide-for-Evaluating-Open-Source-Software.

33. TROCKMAN, Asher; ZHOU, Shurui; KÄSTNER, Christian; VASILESCU, Bogdan. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In: *Proceedings of the 40th international conference on software engineering*. 2018, s. 511–522. Dostupné také z: https://dl.acm.org/doi/abs/10.1145/3180155.3180209.

34. OPENSSL FOUNDATION, Inc. *openssl.org* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://openssl.org/.

35. OPENSSL FOUNDATION, Inc. *Who Writes OpenSSL? - OpenSSL Blog — openssl.org* [online]. 2023. [cit. 2024-04-25]. Dostupné z: https://www.openssl.org/blog/blog/2023/07/17/who-writes-openssl/index.html.

36. *Mbed TLS documentation — mbed-tls.readthedocs.io* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://mbed-tls.readthedocs.io/en/latest/.

37. *GnuTLS — gnutls.org* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://gnutls.org/.

38. CHU, Howard. *GnuTLS considered harmful — openldap.org* [online]. 2008. [cit. 2024-04-25]. Dostupné z: https://www.openldap.org/lists/openldap-devel/200802/msg00072.html.

39. VAUGHAN-NICHOLS, Steven. *GnuTLS: Big internal bugs, few real-world problems — zdnet.com* [online]. 2014. [cit. 2024-04-25]. Dostupné z: https://www.zdnet.com/article/gnutls-big-internal-bugs-few-real-world-problems/.

40. WOLFSSL. *Embedded SSL/TLS Library — wolfssl.com* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://www.wolfssl.com/.

41. DAS, Somak; GOPAL, Vineet; KING, Kevin; VENKATRAMAN, Amruth. IV= 0 security: Cryptographic misuse of libraries. *Massachusetts Institute of Technology*. 2014. Dostupné také z: https://courses.csail.mit.edu/6.857/2014/files/18-das-gopal-king-venkatraman-IV-equals-zero-security.pdf.

42. HAZHIRPASAND, Mohammadreza; GHAFARI, Mohammad; KRÜGER, Stefan; BODDEN, Eric; NIERSTRASZ, Oscar. The impact of developer experience in using Java cryptography. In: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, s. 1–6. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8870184.

43. EGELE, Manuel; BRUMLEY, David; FRATANTONIO, Yanick; KRUEGEL, Christopher. An empirical study of cryptographic misuse in android applications. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, s. 73–84. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/2508859.2516693.

44. NADI, Sarah; KRÜGER, Stefan; MEZINI, Mira; BODDEN, Eric. Jumping through hoops: Why do Java developers struggle with cryptography APIs? In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, s. 935–946. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/2884781.2884790.

45. LUO, Junwei; YANG, Xuechao; YI, Xun; HAN, Fengling; GONDAL, Iqbal; HUANG, Guang-Bin. A Comparative Study on Design and Usability of Cryptographic Libraries. In: *Proceedings of the 2023 Australasian Computer Science Week*. 2023, s. 102–111. Dostupné také z: https://dl.acm.org/doi/pdf/10.1145/3579375.3579388.

46. MINDERMANN, Kai; KECK, Philipp; WAGNER, Stefan. How usable are rust cryptography APIs? In: *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, s. 143–154. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8424966.

47. ACAR, Yasemin; BACKES, Michael; FAHL, Sascha; GARFINKEL, Simson; KIM, Doowon; MAZUREK, Michelle L; STRANSKY, Christian. Comparing the usability of cryptographic apis. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, s. 154–171. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7958576.

48. HAZHIRPASAND, Mohammadreza; GHAFARI, Mohammad. Worrisome patterns in developers: A survey in cryptography. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2021, s. 185–190. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9680282.

49. FISCHER, Felix; BÖTTINGER, Konstantin; XIAO, Huang; STRANSKY, Christian; ACAR, Yasemin; BACKES, Michael; FAHL, Sascha. Stack overflow considered harmful? the impact of copy&paste on android application security. In: *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, s. 121–136. Dostupné také z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7958574.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF