FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Kritéria pro hodnocení bezpečnosti kryptografických knihoven
Student:	Kirill Leonov
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

Kirill Leonov

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

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

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především mému vedoucímu Ing. Josefu Kokešovi, Ph.D. za jeho vedení, cenné rády při zpracování práce a vynikající výuku oborových předmětů.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 16. května 2024

Abstrakt

Tato bakalářská práce prezentuje zopakovatelnou metodiku pro vyhodnocení libovolné open-source kryptografické knihovny z pohledu její důvěryhodnosti a programátorské přívětivosti a na příkladu několika knihoven (libsodium, Botan, OpenSSL) znázorňuje uplatnění dané metodiky.

Klíčová slova open-source, kryptografická knihovna, bezpečný software, bezpečné použití softwaru.

Abstract

This bachelor thesis presents a repeatable methodology for evaluating an arbitrary open-source cryptographic library in terms of its trustworthiness and programmer-friendliness and illustrates the application of the methodology on the example of several libraries (libsodium, Botan, OpenSSL).

Keywords open-source, cryptographic library, secure software, safe use of software.

Seznam zkratek

ABI	Application Binary Interface
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
API	Application Programming Interface
BSI	Bundesamt für Sicherheit in der Informationstechnik
CBC	Cipher Block Chaining
CLA	Contributor Licence Agreement
CVE	Common Vulnerabilities and Exposures
DFD	Data Flow Diagram
DHCP	Dynamic Host Configuration Protocol
ECB	Electronic Code Book
FFI	Foreign Function Interface
OMC	OpenSSL Technical Committee
OTC	OpenSSL Technical Committee
PKCS	Public-Key Cryptography Standards
SDL	Security Development Lifecycle
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Module

Úvod

Použití kryptografie ve dnešní době lze v určité míře potkat skoro všude. Nutnost manipulovat se soukromými daty uživatelů tak, aby nedošlo k jejich úniku, je běžným požadavkem na moderní aplikace. Je také dobře známo, že aplikační vývojáři by neměli implementovat kryptografii vlastními sílami, jelikož se v tom dá velmi snadno udělat chyba, která zkompromituje veškerou bezpečnost. Mnohem jednodušším a bezpečnějším řešením zavedení kryptografie do aplikace je využití již existujících implementací požadovaných algoritmů — což právě nabízí open-source knihovny.

Ale použití jakékoliv knihovny také nemusí hned zamezit výskytům kryptografických chyb, je nutné zvolit vhodnou a důvěryhodnou knihovnu a navíc ji správně použít. Tomu, co obnáší důvěryhodnost a pravděpodobnost správného použití, se tato práce věnuje.

Cílem tedy je zmapování informací, které by mohly sloužit jako vodítka při hodnocení bezpečnosti open-source kryptografických knihoven, specifikace hlavních zdrojů rizik jejich bezpečnostních slabin a vypracování metodiky, která umožní porovnat různé kryptografické knihovny mezi sebou a odhadnout pro dané použití tu nejvhodnější.

První kapitola je věnována problematice bezpečnosti softwaru — popisuje totiž obecný cyklus bezpečného vývoje libovolného softwaru. Druhá kapitola představuje hrozby specifické pro open-source kryptografické knihovny, vlastnosti, které tyto hrozby zmirňuji a způsoby tyto vlastnosti odhalit. Třetí kapitola je věnována analýze vybraných knihoven. Poslední, čtvrtá kapitola shrnuje výsledky provedené analýzy a mapuje je na cíle dané práce.

Kapitola 1

Problematika bezpečnosti softwaru

Při vývoji softwaru bezpečnost je často odsunuta na vedlejší kolej, tj. je považována za aktivitu, která následuje po vývoji. To je od základu chybná myšlenka, která následně způsobuje bezpečnostní chyby [1]. Tato kapitola poskytuje odpovědi na to, jak lze bezpečný software navrhnout i implementovat a co přispívá k tomu, aby jeho použití bylo také bezpečným.

1.1 Základní pojmy

Na začátku uvedeme, co vůbec můžeme rozumět pod pojmem bezpečný software. Bezpečný software — software, který se dokáže bránit proti libovolným útokům tak, aby byly zachovány důvěrnost, integrita a dostupnost kritických aktiv [1]. Jde totiž o požadavek na dodržení CIA triády (důvěrnost, integrita, dostupnost), která je základní v celé oblasti kybernetické bezpečnosti [2].

- Důvěrnost: Zajištění důvěrnosti znamená, že data budou dostupná pouze oprávněné osobě, vylučuje se zneužití informace.
- Integrita: Zajištění integrity znamená, že data nejsou pozměněna, tj. jsou doručena uživateli bez úprav, resp. s úpravami ověřenými a nepopiratelnými.
- Dostupnost: Zajištění dostupnosti znamená, že data jsou dostupná v kvalitě a čase, kdy jsou potřebná oprávněnému uživateli.

Před tím, než říci, jak probíhá vývoj bezpečného softwaru, uvedeme jaký software existuje a co obnáší. Veškerý software lze rozdělit podle přístupnosti zdrojového kódu na:

- Open-source — je software, jehož zdrojový kód je veřejně přístupný. V užším smyslu dle definice Open Source Initiative má být zveřejněn pod licencí, která dovoluje bezplatné použití, volnou distribuci a modifikaci kýmkoliv [3].
- Closed-source — je takový software, který nespadá pod definici open-source, tj. jeho zdrojový kód není přístupný. Většinou se jedná o software vyvíjený jednou společností, která ho distribuuje za poplatek v zkompilovaném a spustitelném formátu.

1.2 Open-source vs closed-source z pohledu bezpečnosti

Z povahy open-source plyne, že na jeho vývoji se může podílet kdokoliv z celosvětové komunity, protože kód projektu je veřejně přístupný a volně modifikovatelný. Oproti tomu vývoj closed-source softwaru obvykle probíhá v užší skupině vývojářů, kde každý již má svou přiřaženou specifickou roli. Autor eseje *The Cathedral and the Bazaar* pro takové modely vývoje zavedl pojmy *bazaar*[1] a *cathedral*[2] a zformuloval tzn. Linusův zákon: *"given enough eyeballs, all bugs are shallow"*[3], který z pohledu bezpečnosti vyzdvihuje první model [4]. Tento zákon je odrazem častého názoru, že open-source software je bezpečný jenom proto, že zveřejněný kód již vidělo mnoho lidí. Ale empirické studie tomu nenacházejí žádné důkazy [5], a k tomu existuje několik důvodů [6]:

1. Vývojáři neradi zkoumají cizí kód.

   Pravdou je, že vývojáři mají mnohem větší motivaci tvořit něco nového a vlastního, než zkoumat cizí kód. Proces code review je pomalý a nudný pro většinu z nich.

2. Nutnost porozumění bezpečnostním chybám.

   Obeznámenost s tím, co vůbec obnáší zranitelnost a jaké existují, je klíčovým předpokladem pro to, aby někdo nějakou chybu v tom kódu vůbec nalezl. Není to ale základní znalostí každého programátora, naopak pro většinu vývojářů to neplatí, protože to vyžaduje odborné znalosti v oblasti bezpečnosti.

Navíc některé zranitelnosti v open-source projektech byly objeveny až po uplynutí mnoha let [6]:

- 15 let Sendmail e-mail server (CVE-2003-0161)[4]
- 10 let MIT's Kerberos authentication protocol (CVE-2003-0060)[5]
- 7 let SAMBA file and print (CVE-2003-0085)[6]
- 5 let MIT's Kerberos authentication protocols (CVE-2005-1689)[7]
- 5.5 let Eric Raymond's Fetchmail e-mail server (CVE-2002-0146)[8]

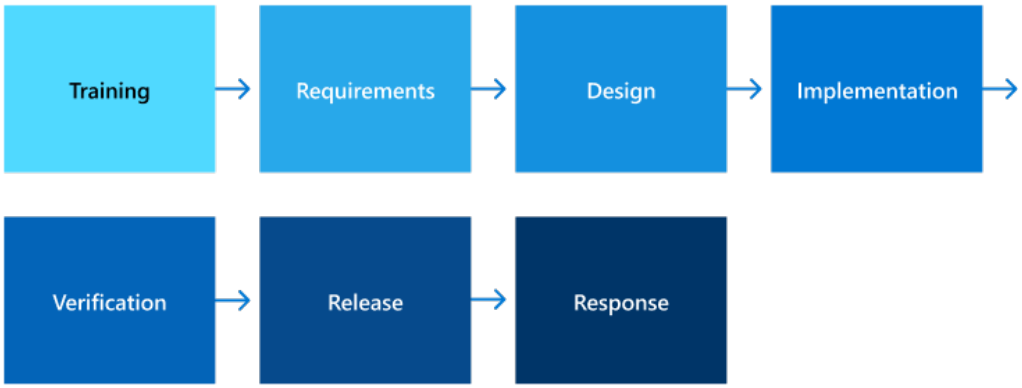
Podle názoru autorů knihy *Security Development Lifecycle* [6] — bezpečným software může udělat pouze správný cyklus vývoje, kde důraz na bezpečnost je kladen v každé z vývojových fází.

1.3 Bezpečnostní životní cyklus vývoje software

Hlavní cíle bezpečnostního cyklu vývoje softwaru (SDL): snížení počtu zranitelností ještě v průběhu jeho vývoje a snížení závažnosti možných, ale ještě neodhalených bezpečnostních chyb. Proč je důležité dbát o bezpečnost již během vývoje?

1. Záplatovaní chyb po vydaní softwaru je často náročné a může vyžadovat kompletní změny již implementované funkcionality.
2. Každá nová zranitelnost je reputační škoda a obnáší finanční ztráty [1].

[1] tržiště
[2] katedrála
[3] pokud máte dostatek očí, všechny chyby jsou průhledné
[4] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2003-0161`
[5] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0060`
[6] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0085`
[7] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1689`
[8] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0146`



Obrázek 1.1 Fáze SDL [7]

V Microsoft se počet bezpečnostních chyb po použití SDL snížil přibližně o 50 až 60 procent [8]. Je důležité zmínit, že základní oporou celého tohoto procesu je vzdělání a povědomí vývojářů. Pokud vývojáři nejsou obeznámeni se základními bezpečnostními koncepty, tak není možné očekávat, že by vyprodukovali bezpečný software. Proto je v Microsoftu veškerý inženýrský personál povinen navštěvovat bezpečnostní školení alespoň jednou za rok [6].

1.3.1 Definování bezpečnostních cílů

Před tím, než investujeme čas a prostředky do zabezpečení, je potřeba zajistit správnou organizaci celého procesu a zjistit, kolik úsilí a prostředků to bude vyžadovat. Proto bychom na začátku projektu měli:

1. Stanovit tým/osobu zodpovědnou za vedení bezpečnostního procesu.

   Hlavním cílem je koordinace a sdělování stavu všech bezpečnostních problémů mezi ostatními týmy, stanovení bezpečnostních politik.

2. Zajistit to, že proces sledování chyb zahrnuje pole pro bezpečnostní chyby.

   Měla by být zajištěna možnost zachytit a roztřídit bezpečnostní chyby dle závažnosti.

3. Určit, které části projektu budou vyžadovat modelování hrozeb.

   K tomu je potřeba rozpoznat nejrizikovější komponenty softwaru a zjistit stav z pohledu známých zranitelností.

4. Určit rozsah požadavků na testování.

   Např. určit ve kterých částech projektu je zapotřebí penetrační testování.

1.3.2 Bezpečný návrh

Při návrhu je důležité pečlivě zvažovat bezpečnost, aby nedošlo k připojení bezpečnostních prvků k produktu těsně před koncem anebo po konci návrhového procesu. Základní principy bezpečnostního návrhu [9]:

- Úspornost mechanismů.

  Kód by měl být kompaktní a srozumitelný, to ulehčí budoucí údržbu.

- Kompletní zprostředkování.

  Každý přístup do každého chráněného objektu by měl být kontrolován.

- Bezpečné výchozí nastavení.

  Výchozí instalace musí být maximálně bezpečná, protože se používá nejčastěji.

- Otevřený návrh.

  Bezpečnost systému by neměla záviset na utajení jeho struktury.

- Princip nejmenších oprávnění.

  Ve výchozím nastavení by měly být povolené pouze nezbytné funkce a přidělena pouze nezbytná oprávnění za účelem minimalizace útočného povrchu.

- Princip oddělených oprávnění.

  Lze chápat jako vícevrstevnou ochranu. Každá citlivá akce by neměla záviset pouze na jedné podmínce. Příkladem může být použití dvoufaktorové autentizace místo jednoduché pomocí hesla.

- Co nejméně společných mechanismů.

  Minimalizování sdílených prostředků, např. souborů mezi uživateli.

- Psychologická přijatelnost.

  Bezpečnost by neměla být překážkou a ztěžovat použití.

1.3.2.1 Analýza rizik

Hlavní náplní této fáze je modelování hrozeb. To je proces, který slouží k porozumění potenciálním bezpečnostním hrozbám systému, určení rizik a stanovení vhodných opatření k jejich zmírnění. Skládá se z několika kroků:

1. Studium aplikace

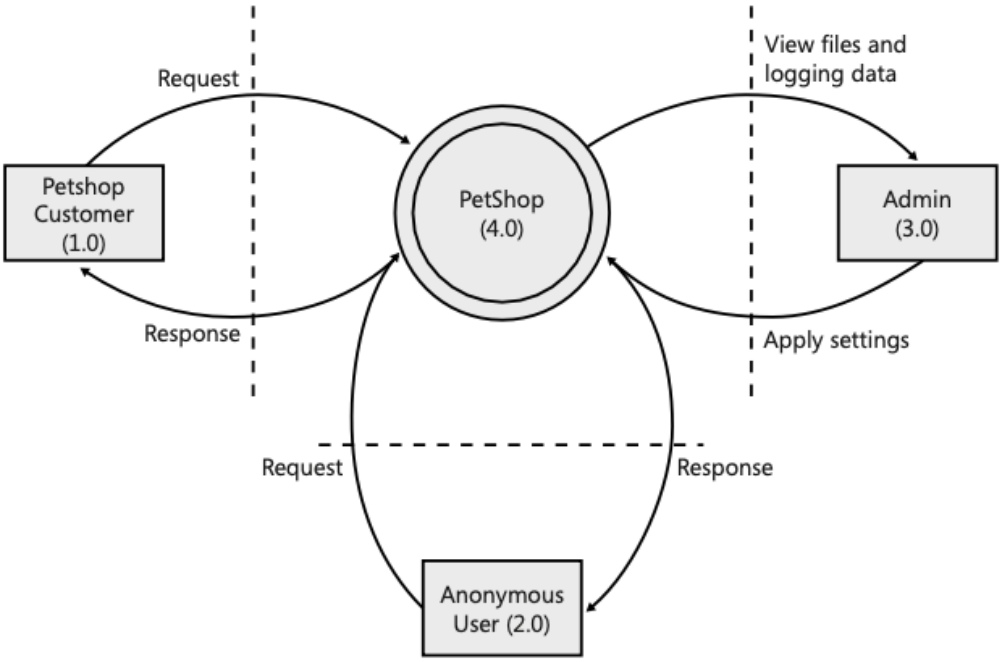
   K tomu, abychom mohli identifikovat potenciální hrozby, je potřeba namodelovat chování systému. Jedna z možností je použití Data Flow Diagramů (DFD). V následující tabulce je uveden seznam komponent, ze kterých se skládá a které bude potřeba určit pro aplikaci.

Tabulka 1.1 Komponenty Data Flow Diagramu [6]

Útvar	Typ komponenty	Popis
dvojitý kruh	multiproces	logická reprezentace procesu, který provádí mnoho různých operací
kruh	proces	logická reprezentace procesu, který provádí jednu diskrétní operaci
obdélník	interactor	reprezentace vstupu do systému
rovnoběžky	datové úložiště	reprezentace místa, na kterém se ukládají dočasná nebo trvalá data
šipka	tok dat	reprezentace prostředků, kterými putují data
přerušovaná čára	hranice důvěry	reprezentace hranic, které vymezují tok dat mezi různě důvěryhodnými zónami

2. Dekompozice

   Ke tvorbě diagramů je potřeba určit scénáře použití, které chceme takovým způsobem popsat. Pak můžeme vymezit důvěryhodné a nedůvěryhodné komponenty systému, rozdělit je pomocí hranic důvěry a patřičně rozmístit. Na následujícím obrázku je uveden příklad modelu, který můžeme takto získat. Komponenty získaného modelu lze pak nadále dekomponovat.



Obrázek 1.2 Příklad Data Flow Diagramu [6]

3. Identifikace hrozeb a zranitelností

   K identifikaci hrozeb budeme uvažovat komponenty získané dekompozicí jakožto cíle útoků. Někdy (např. to platí pro Microsoft [6]) se ke klasifikaci hrozeb se používá model **STRIDE**:

   - *Spoofing of Identity* — podvržení identity.
   - *Tampering* — neautorizované pozměnění dat.
   - *Repudiation* — popření uskutečněné akce. Stává se, když takové akce nejsou v systému zaznamenány.
   - *Information Disclosure* — únik informací.
   - *Denial of Service* — odepření služby.
   - *Elevation of Privilege* — neautorizované zvýšení oprávnění.

   Je potřeba aplikovat STRIDE na každou komponentu dle následujícího vztahu:

Tabulka 1.2 Vztah mezi komponenty DFD a STRIDE [6]

Komponenta	S	T	R	I	D	E
Interactor	✔		✔			
Tok dat		✔		✔	✔	
Úložiště		✔	✔	✔	✔	
Proces	✔	✔	✔	✔	✔	✔

Až zjistíme, které hrozby z těchto kategorií jsou možné a co obnáší, je nutno vyhodnotit jejich závažnost. Na to existuje např. model **DREAD**:

- *Damage potential* — potenciál škody.
- *Reproducibility* — zopakovatelnost hrozby.
- *Exploitability* — jak je těžké útok provést.
- *Affected Users* — rozsah postižených uživatelů.
- *Discoverability* — jak je těžké zranitelnost odhalit.

Každé z kategorií lze přiřadit hodnotu od 0 do 10 a následně vypočítat průměr jako hodnotu výsledné závažnosti. Hrozby lze seřadit dle takto spočítané závažnosti a odhadnout, které se nejpravděpodobněji mohou stát zranitelnostmi.

4. Zmírnění hrozeb

   Lze vybrat jednu z následujících strategií:

   - Nedělat nic.
     Lze to někdy uplatnit pro nejméně závažné hrozby.
   - Odstranit problematickou funkci.
     Je jediný způsob, jak se úplně zbavit rizika.
   - Upozornit uživatele o přítomné hrozbě.
     Je možné říci uživateli, na co by měl dávat pozor.
   - Zavést protiopatření.

Tabulka 1.3 Protiopatření platné pro STRIDE [6]

Hrozba	Protiopatření
Spoofing	Autentizace
Tampering	Chráněni integrity: hašovaní, MAC, digitální podpisy
Repudiation	Logování, digitální podpisy, časové otisky
Information dislocure	Autorizace, šifrování
DoS	Zajištění dostupnosti: filtrování, autorizace
EoP	Autorizace

Nelze vytvořit bezpečný produkt, pokud vývojáři nerozumí hrozbám a zranitelnostem, které takový produkt přináší, a tomu, jak tyto hrozby lze zmenšit.

1.3.3 Implementace

Při vývoji softwaru je rovněž velmi důležité nezapomenout na to, že takový software bude chtít někdo použít. Proto během této fáze se především myslí na koncového uživatele: probíhá samotný vývoj v souladu s politikami bezpečného kódu a vytváří se pomůcky pro bezpečné nasazení a použití.

1.3.3.1 Vytvoření dokumentace a ”best practices” pro uživatele

Protože bezpečnost a použitelnost mohou být v konfliktu, je důležité informovat uživatele jak o existujících hrozbách, tak i o kompromisech mezi funkcionalitou a bezpečnostními riziky produktu, a to skrz dokumentaci.

Instalační dokumentace by měla navádět k tomu, jak při prvním spouštění dostat aplikaci do co nejbezpečnějšího stavu. Hlavní dokumentace by měla vyhnout se všem radám, které propagují zjevně nebezpečné praktiky a zmínit všechny bezpečnostní nedostatky aplikace, tj. možné případy, kde může k narušení bezpečnosti dojít. Pokud se nabízí vývojové prostředí skrz API je třeba opatřit každou funkci bezpečnostní poznámkou a příkladem použití.

Obrázek 1.3 Příklad popisu funkce `strncpy` s bezpečnostní poznámkou [6]

1.3.3.2 Politiky bezpečného kódu

Již během vývoje lze předejít množství chyb díky dodržování standardů bezpečného kódu. Seznam hlavních doporučení:

- Použití nejnovější verze kompilátoru a nabízených ochranných nástrojů.

  Například, pokud se jedná o jazyk C/C++, kompilátor Misrosoft Visual Studio nabízí sadu ochranných flagů:

  1. /W4 pro maximální úroveň varování při kompilaci.
  2. /GS pro dodání security "cookie"[9] před návratovou adresou na zásobníku za účelem ochrany proti přetečení bufferu.

[9] jedná se o náhodné hodnoty

3. /NXCOMPAT pro zajištěni *Data Execution Prevention*, což je technika, která zaručuje to, že paměťové stránky, které přináleží zásobníku, se stávají nespustitelnými.

- Použití nástrojů pro analýzu zdrojového kódu.

  Existuje hodně nástrojů jak na statickou (např. PREfast nebo FxCop), tak i na dynamickou analýzu (např. valgrind), které pomáhají odhalovat bezpečnostní chyby v kódu.

- Dodržování bezpečnostního standardu jazyka, pokud takový je.

  Například pro jazyk C existuje standard ISO/IEC TS 17961:2013, který popisuje hlavní bezpečnostní pravidla, obsahuje seznam zakázaných a doporučených funkcí [10].

1.3.4 Verifikace

Tato fáze slouží k odhalení změn, ke kterým došlo během vývoje, případnou aktualizaci modelů hrozeb, provedení revizí starého a nového kódu, testování. Testováním ale nelze produkt zabezpečit, lze pouze odhalit mnoho chyb ještě během vývojové fáze.
Základní typy testování jsou:

- Fuzz testování

  Fuzzing je proces testování, kdy se aplikace nechá konzumovat neočekávaná a většinou náhodně generovaná data. To ověřuje zejména to, jak aplikace nakládá s daty, které pochází zvenčí mimo hranice důvěryhodnosti a jak kontroluje nesprávné vstupy. V Microsoftu je např. 20 až 25 % bezpečnostních chyb odhaleno během fuzzingu ještě před vydáním produktu [6].

- Penetrační testování

  Tento proces testování je simulovaným útokem, účelem kterého je identifikovat všechna slabá místa zkoumaného systému a odhalit případné bezpečnostní chyby. Většinou je prováděn třetí stranou jakožto součást bezpečnostního auditu.

- Unit testování

  To je proces testování, kdy se testuje každá nově přidaná jednotka kódu. V ideálním případě by měl být každý testovaný případ nezávislý na ostatních. Při testování se snaží testovaná část izolovat od ostatních částí programu.

1.3.5 Vydání produktu

Během této fáze dochází již ke zveřejnění produktu. Před tím ale je potřeba provést několik důležitých kroků:

1. Ujistit se, že všechny požadavky na SDL jsou dodrženy, nezbývají žádné neopravené chyby, dokumentace je připravená a modely hrozeb jsou aktuální, tj. provést finální bezpečnostní kontrolu.

2. Naplánovat reakce na bezpečnostní incidenty, ke kterým může docházet již pro vydání produktu.

   V reálném světě není možné vytvořit produkt, který bude 100-procentně bezpečný, i když byly dodrženy všechny zmíněné bezpečnostní fáze. K tomu existuje několik důvodů:

   a. Vývojáři jsou také lidé, kteří se mohou dopouštět chyb.
   b. Objevují se nové druhy zranitelností. To co bylo bezpečné včera, nemusí být bezpečným dnes. Hezkým příkladem toho je kryptografie. S pokrokem v oblasti kryptoanalýzy se staré algoritmy stávají prolomitelnými.

A proto je nutno být připraveným k výskytu nových chyb, tj.:

- Mít zveřejněný kontaktní bod, který lze použít ke sdělení bezpečnostních incidentů a chyb.
- Patřičně zpracovávat všechny nahlášené bezpečnostní chyby. Tj. musí se vyhodnotit jejich závažnost a naplánovat se vhodná reakce, kdy a jak tyto chyby by měly být opraveny.
- Zajistit podporu používaných verzí produktu.

  Je důležité produkovat potřebné aktualizace pro všechny podporované verze (nejen tu nejnovější). Často zákazníci nechtějí být nuceni k přechodu na novou verzi, jenom aby se chránili před zneužitím nějaké zranitelnosti.
- Zajistit jednoduchý proces aktualizace, který vyžaduje co nejméně manuálních zásahů.

Po ukončení všech bezpečnostních kontrol a naplánovaní další údržby lze produkt vydat.

Kapitola 2

Způsoby hodnocení bezpečnosti kryptografických knihoven

Bezpečnost lze hodnotit z mnoha stránek, a proto je potřeba říci, co jí rozumíme v této práci. Tato kapitola uvádí uvažované hrozby a seznam vlastností, které budou zmíněné hrozby zmenšovat.

2.1 Model hrozeb

Jelikož cílem je vypracovaní obecné metodiky hodnocení aplikovatelné na libovolnou open-source kryptografickou knihovnu (a zčásti i na libovolný open-source projekt), která by mohla pomoct vývojářům vybrat tu nejbezpečnější, určitě není v záměru zkoumat zdrojový kód jednotlivých knihoven na přítomnost bezpečnostních chyb, protože by se jednalo o velmi náročný proces, který by poskytoval informace pouze o bezpečnosti jedné verze a byl by těžce zopakovatelný běžným vývojářem. Proto budeme uvažovat následující hrozby:

- Selhání vývojových procesů

  Jak již bylo zmíněno, specifika modelu vývoje open-source projektů spočívají v tom, že se do toho může přispívat kdokoliv. To ale přináší rizika toho, že *kdokoliv* nemusí být důvěryhodná osoba, ale možný útočník, který chce do kódu vložit zranitelnost. Názorným příkladem může být v nedávné době objevená zranitelnost CVE-2024-3094[1] v knihovně XZ Utils. Útočníkovi se podařilo dostat se na seznam správců knihovny a vložit tam zadní vrátka pomocí několika commitů.[2] To se stalo nejpravděpodobněji v důsledku toho, že hlavní správce knihovny neměl ani dostatečnou motivaci ani čas věnovat se projektu s plným úsilím.[3]

  Z toho plyne i jiný problém, který spočívá v tom, že open-source projekty nejsou vždy schopny zajistit tak kvalitní proces vývoje, jaký byl probrán v předchozí kapitole. A to protože prostě na to nemají ani finanční prostředky, ani dostatečný počet lidí, který by se stále věnoval projektu.

- Chybné použití knihovny vývojářem

  Bezpečný software by neměl vytvářet prostor pro vznik bezpečnostních chyb v jiných aplikacích. To se zvláště tyče kryptografických knihoven. Od začátku své existence kryptografické knihovny trpí problémy spojenými s použitelností. Vývojáři knihoven navrhují rozhraní tak,

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-3094
[2] https://boehs.org/node/everything-i-know-about-the-xz-backdoor
[3] https://www.mail-archive.com/xz-devel@tukaani.org/msg00567.html

že je použitelné pouze stejnými odborníky, což vede k výskytu mnoha neúmyslných zranitelností v jiných aplikacích. Např. článek *Why does cryptographic software fail? A case study and open problems* uvádí, že studie 269 kryptografických zranitelností od ledna 2011 do května 2014 odhalila, že 83 procent všech nalezených zranitelností se týkalo nesprávného použití knihoven a jenom 17 procent jsou chyby v knihovnách [11].

2.2 Žádoucí vlastnosti open-source kryptografické knihovny

Tato sekce uvádí seznam vlastností, které by měly pozitivně ovlivňovat výslednou bezpečnost zkoumané knihovny, tj. zmenšovat zmíněné hrozby a svědčit o korelaci vývojových procesů knihovny s SDL.

2.2.1 Důvěryhodnost a kvalita vývojových procesů

Open Source Security Foundation (OpenSSF) zveřejnila průvodce hodnocením open-source softwaru [12], od kterého lze odrazit při formulování následujících kriterií:

- Projekt nastavuje a dodržuje pravidla pro přispívání do kódu.

  Projekt by měl jasným způsobem definovat, kdo do jeho kódu může přispívat a jak se tento příspěvek následně kontroluje. Jestliže tomu nebude věnovaná dostatečná pozornost, pak může dojít ke vložení zranitelného kódu útočníkem.

- Projekt je spravován lidmi, kteří mají prokazatelnou motivaci se na tom podílet.

  Existence dostatečné motivace je základním požadavkem na údržbu vývojových procesů v kvalitním stavu. Pakliže je projekt spravován jenom nadšenci jako jejich hobby (většinou jde o to, že za to nedostávají zaplaceno), snadno dojde k něčemu, co bylo zmíněno u CVE-2024-3094 v předchozí sekci.

- Projekt je široce používán.

  Za prvé se dají případné nedostatky softwaru odhalit jenom během použití a za druhé, pokud již hodně vývojářů tomuto softwaru důvěřuje, pak lze předpokládat, že ho někdo již zkoumal.

- Projekt popisuje postup hlášení a reakce na zranitelnosti.

  Zveřejnění takových postupů svědčí o tom, že správci projektu se snaží udržovat projekt v bezpečném stavu a jsou odhodláni řešit bezpečnostní problémy.

- Projekt je aktivně vyvíjen.

  Můžeme říci, že projekt je aktivně vyvíjen, jestliže po vydaní poslední verze ještě neuplynul rok (nebo alespoň je vidět aktivitu v repozitáři za poslední rok) a nahlášené chyby se nenechávají bez reakcí a opravy.

- Projekt se podrobil bezpečnostnímu auditu.

  Za prvé to svědčí o tom, že je opravdu kladen důraz na bezpečnost. A za druhé výsledky auditu dávají přehled o bezpečnosti tohoto projektu. Jenom je potřeba pamatovat, že pokud byl proveden již dávno, pak může mít jenom omezenou vypovídající hodnotu.

- Projekt používá průběžnou integraci (CI), ve které se spouští sada unit testů, fuzz testy, měří se pokrytí kódu testy.

  Testování kódu je vždy velmi dobrou vlastností, je součástí SDL a napomáhá zbavit se chyb ještě během vývoje. Čím větší testovací sada a pokrytí, tím menší pravděpodobnost výskytu chyby.

2.2.2 Bezpečnost použití a programátorská přívětivost

Jak již bylo zmíněno, problém použitelnosti kryptografických knihoven je dávný. Byly provedeny různé studie, které se tímto zabývaly a formulovaly různá doporučení pro vývoj a návrh knihoven [13], [14], [15]. Lze ale vybrat dva základní faktory, které ovlivňují použitelnost: návrh API a zdokumentovanost.

2.2.2.1 Dokumentace

Důležitost dokumentace pro libovolný software již byla zmíněna v sekci 1.3.3.1. Tady doplníme vlastnosti, které jsou obzvlášť kýžené u dokumentace kryptografické knihovny.

- Dokumentace knihovny je přehledná, navigovatelná, lze v ní snadno vyhledávat a je rozdělená na sekce podle nabízených funkcí a primitiv.

  Je žádoucí, aby dokumentace sloužila jako primární zdroj informace a neodrazovala, jinak je velmi pravděpodobné, že vývojář sáhne po neoficiálních zdrojích, které mohou obsahovat chybné a zastaralé informace.

- Dokumentace pokrývá veškerou nabízenou funkcionalitu.

  Pokud nepokrývá, jak lze očekávat správné použití u takových funkcí?

- Dokumentace popisuje, k čemu slouží nabízená primitiva/funkce a jak lze s jejich pomocí pokrýt běžné kryptografické požadavky.

  Například užitečná je přítomnost odpovědi na otázku: Jakým způsobem lze zašifrovat soubor pomocí hesla?

- Dokumentace obsahuje varování týkající se nebezpečných primitiv/funkcí a nabádá k použití bezpečnějších.

  Je pravděpodobné, že běžný vývojář použije to, na co narazí nejdřív a nebude zkoumat, co je bezpečnější.

2.2.2.2 API

API je způsob zpřístupnění interního kódu knihovny vývojáři pomocí sady funkcí. I když samotný knihovní kód nebude neobsahovat žádnou chybu, vývojář ji může jednoduše vložit do své aplikací tím, že nesprávně použije API funkci. Pravděpodobnost špatného použití u kryptografické knihovny mají zmenšovat následující vlastnosti:

- Použití různých datových typů pro vstupní argumenty.

  Jde zejména parametry jako např. inicializační vektor (`iv`) a klíč (`key`). Ukážeme to na příkladu OpenSSL:

  ```
  1 int EVP_EncryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type,
        ENGINE *impl, const unsigned char *key, const unsigned char *iv);
  ```

  Pokud by byly různých datových typů (ne `unsigned char *`), kompilátor by mohl detekovat jejich případné proházení a upozornit na to.

- Přítomnost jednoduchého a srozumitelného mechanismu informování o nastalých chybách.

  Je velmi důležité mít možnost sdělit vývojáři, jestli se volání funkce nepodařilo a co konkrétně to způsobilo. Pokud jazyk a operační systém podporují výjimky — tak to je nejlepší volba. Výjimka přeruší tok kódu a donutí vývojáře ji ošetřit. Navíc může obsahovat podrobný popis problému uvnitř.

- Zpřístupnění kvalitního generátoru náhodných čísel.

  Bezpečnost kryptografických primitivů většinou záleží na náhodnosti vstupních parametrů jako jsou klíč a inicializační vektor. Pokud vývojář nepoužije dostatečně kvalitní zdroj náhodnosti pro generování těchto parametrů, pak zkompromituje jejich použití. Proto by knihovna měla využívat kvalitních zdrojů náhodnosti, jaké poskytuje např. operační systém, zpřístupňovat je jednoduchým způsobem a navíc nabádat k jejich použití.

- Použití vysokoúrovňových abstrakcí, které jsou pro běžného vývojáře srozumitelné.

  Pod tímto kriteriem lze rozumět schopnost knihovny nabídnout takové abstrakce, které vývojář bez znalostí v oblasti kryptografii může použít správně. Např. nevyžadování od uživatele volit operační mód nebo algoritmus pro padding, sloučení několika šifrovacích fází do jedné. V nejlepším případě by takové abstrakce měly být nástavbou nad něčím nízkoúrovňovým a flexibilnějším, co vývojář může, ale nemusí použít.

- Použití bezpečných výchozích hodnot.

  Pokud knihovna nevyžaduje od uživatele specifikaci některých parametrů jako např. operační mód, inicializační vektor nebo algoritmus pro padding, měly by být použity co nejbezpečnější. Určitě by neměl být ve výchozím stavu použít operační mód ECB nebo nulový inicializační vektor.

- Přítomnost pomocných funkcí.

  Jedná se o funkce, které neslouží přímo k vykonání kryptografických operací, ale mají za účel je usnadnit nebo dodat zabezpečení. Mezi takové lze např. zařadit:

  - Bezpečné alokace.

    Nejčastěji takové funkce umisťují kanárky před a po alokovaném bloku paměti, což má zabránit přetečení bufferu, a zamezují swapování alokovaných stránek paměti na disk a tím prozrazení citlivých údajů, např. klíčů.
  - Hexadecimální a Base64 kódovaní/dekódovaní.
  - Aritmetické operace s velkými čísly.
  - Bezpečné operace s řetězci.

    Např. porovnání stejně velkých úseků paměti za stejný čas za účelem prevence útoků postranními kanály.

  Jejich výhoda je navíc v tom, že zbavují programátora nutnosti psát je ručně nebo využívat dalších závislostí.

2.3 Jak zmapovat teoretické vlastnosti a procesy s vybranou knihovnou?

K tomu, abychom mohli knihovnu vyhodnotit, je potřeba získat potřebné podklady. Naštěstí díky tomu, že analyzujeme open-source software, většinu informací můžeme čerpat z veřejných zdrojů. Hlavními zdroji určitě budou webové stránky a repozitář knihovny. Tam můžeme nahlédnout do toho, jak se knihovna vyvíjí — kdo přispívá a kdo tyto příspěvky reviduje, kdo spravuje repozitář atd. Pokud chceme se dozvědět víc informací o vývojářích (motivace, vztah ke knihovně, profesionální zkušenosti), tak užitečnou může být i sociální síť LinkedIn[4].

Jiným užitečným zdrojem mohou být aplikace, které vybranou knihovnu používají. Lze se pokusit s jejich pomocí dohledat, jak se knihovna osvědčila v praxi a k jakým případným problémům při jejím použití docházelo.

[4] `https://www.linkedin.com`

2.3.1 Jak hledat aplikace používající knihovnu?

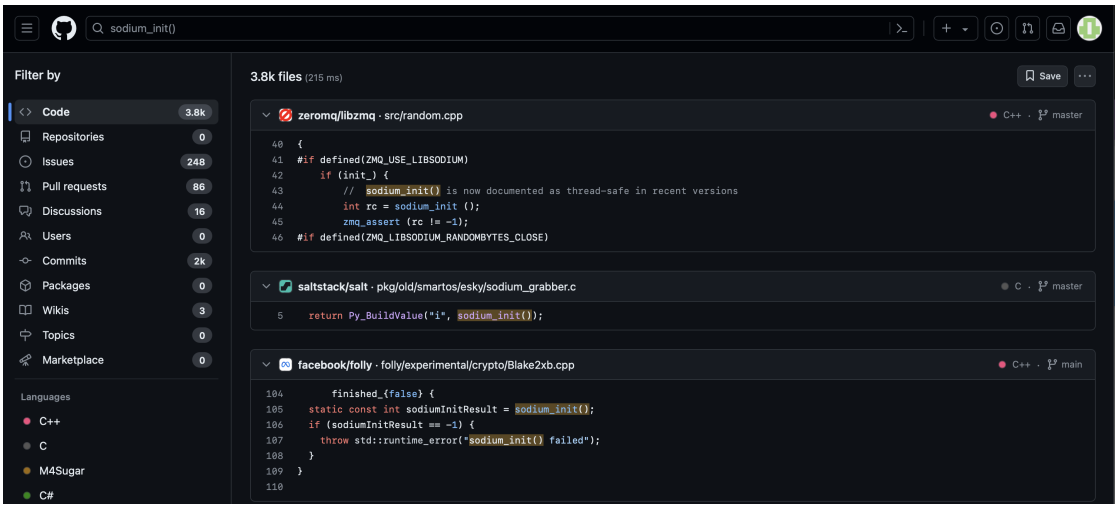
Je potřeba říci, že se rovněž zajímáme pouze o open-source aplikace. Protože až začneme hledat chyby v těchto aplikacích, tak budeme chtít následně prozkoumat jejich souvislost s knihovnou, což v praxi skoro není realizovatelné v případě closed-source softwaru. Existuje několik způsobů, jak takové aplikace najít:

1. Stránky knihovny někdy přímo obsahují seznam aplikací.

   Často ale vůbec není poskytován nebo není zdaleka úplný.

2. Vyhledávání ve veřejných webhostinzích pro open-source projekty jako jsou GitHub[5] a GitLab[6].

   Většinu aplikací lze dohledat na zmíněných stránkách pomocí pro knihovnu specifických řetězců: hlavičkové soubory, inicializační funkce nebo konstanty.



Obrázek 2.1 Vyhledávání aplikací pro knihovnu libsodium na GitHub

3. Vyhledávání pomocí nástrojů.

   Užitečným může být např. příkaz `apt-cache rdepends libname`, který poskytuje seznam na knihovně závisících balíčků v distribuci Linuxu, na které je spouštěn.

Vhodnými aplikacemi pro další průzkum budou ty, které jsou nejrozšířenější a pokrývají co nejvíc knihovní funkcionality.

2.3.2 Jak hledat chyby pro vybranou aplikaci?

Chyba se v závislosti na závažnosti může, ale nemusí objevit v databázi zranitelností (CVE)[7,8]. Proto první cestou se nabízí prozkoumat všechny pro danou aplikaci zveřejněné zranitelnosti, které se nějakým způsobem týkají kryptografie. Lze je vyfiltrovat pomocí klíčových slov jako např. *crypto, encrypt, cipher* atd.

Jak již bylo zmíněno, tak se snadno může stát, že nějaká chyba není tak závažná, aby se objevila na seznamu zranitelností, ale přece byla někde zmíněna a opravena. Proto vhodnými

[5] `https://github.com`
[6] `https://gitlab.com`
[7] `https://nvd.nist.gov/vuln`
[8] `https://cve.mitre.org`

zdroji jsou také issues, pull-requesty a commity v repozitáři aplikace. Např. rozhraní GitHubu umožňuje hledání ve zmíněných sekcích, proto na to také můžeme aplikovat klíčová slova.

2.3.3 Souvislost nalezené chyby s použitím knihovny

Pro účely této práce chceme u nalezené chyby prozkoumat, jestli souvisí s knihovnou, a pokud ano, chceme se dozvědět, co ji zapříčinilo.

Pokud máme nalezenou chybu přímo v repozitáři, např. v issues, nejspíš tam nalezneme i odkaz na commit, který už problém řeší.

Pokud máme nalezenou zranitelnost z nějaké CVE databáze, tak buď máme štěstí a popis přímo obsahuje odkaz na opravný commit, nebo alespoň zmiňuje release, ve kterém byl problém opraven. Tedy můžeme zkusit dohledat v repozitáři tento release a analyzovat commity, které zahrnuje.

U opravného commitu je nejdřív potřeba se ujistit, že se změny vůbec týkají kódu, který pracuje s knihovnou. Pokud ano, lze již zkoumat případnou souvislost s knihovnou. Za takovou souvislost považujeme např. nedostatek v návrhu API nebo nedostatečnou zdokumentovanost problematické funkce.

Kapitola 3
Analýza vybraných knihoven

Tato kapitola je věnována zmapování popsáných v sekci 2.2 vlastností s několika vybranými knihovnami. Výsledek hodnocení je vždy shrnut na konci ve formě tabulky.

3.1 Libsodium

Libsodium je fork NaCl s kompatibilním, ale rozšířeným API pro ještě další zlepšení použitelnosti. Hlavním cílem při vytváření NaCl bylo "vyhnout se různým typům kryptografických katastrof, kterými trpěly předchozí kryptografické knihovny" [16]. Cílem libsodium je "poskytnout všechny základní operace potřebné k vytvoření kryptografických nástrojů vyšší úrovně" [17].

Z popisu tedy můžeme odvodit, že knihovna má být programátorsky přívětivá, což znamená, že by měla zmenšovat minimálně jednu z hrozeb, které analyzujeme v této práci.

3.1.1 Open-source — vývoj

- Projekt se aktivně vyvíjí, poslední verze vyšla v září 2023[1], commity přibývají skoro každý týden.
- Licence *ISC*.
- Knihovna je poměrně rozšířena, například na standardní distribuci Linuxu Ubuntu 22.04 na ní závisí až 72 balíčků[2].

3.1.1.1 Tým a správa projektu

Správce a vývojář je jednotlivec Frank Denis. Na LinkedIn je dohledatelný jeho profil[3]:

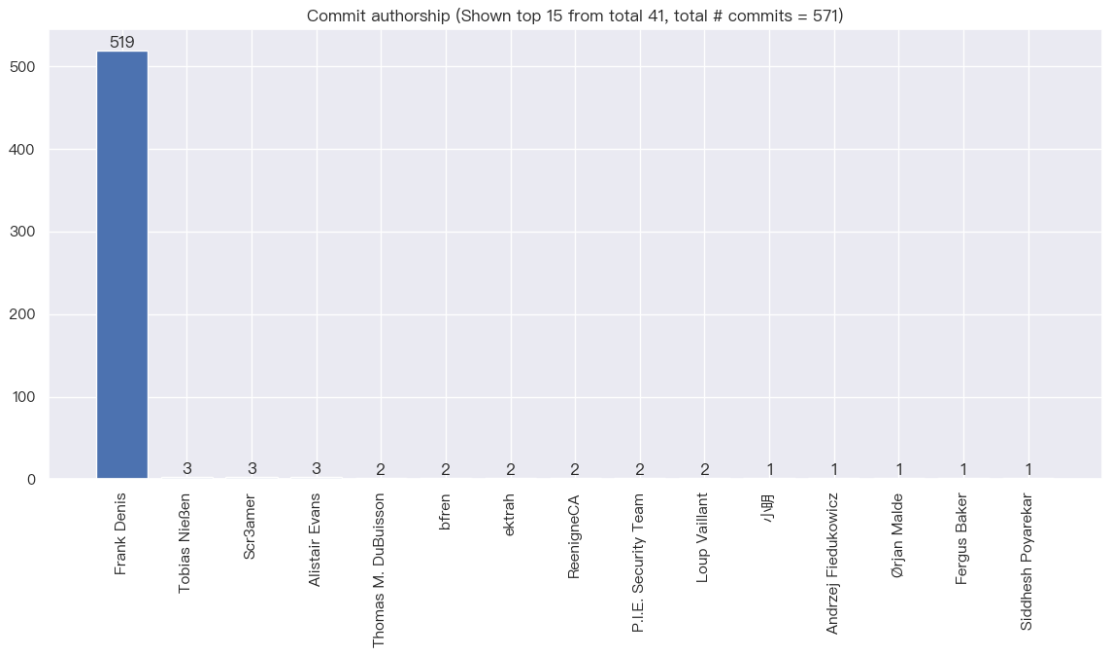
- Je softwarový inženýr a profesionální fotograf.
- Od roku 2017 funguje jako Independant Contractor: nabízí služby spojené s fotografováním a softwarovým inženýrstvím se záměrem na zpracování obrazů.

Působí dojmem, že vývoj knihovny pro něj nemá prokazatelnou motivaci, protože se nejedná o primární zaměstnání, lze to brát spíš jako hobby.

[1] zkontrolováno 8. května 2024
[2] Zjištěno příkazem `apt-cache rdepends libsodium23`
[3] `https://www.linkedin.com/in/frankdenis/`

3.1.1.2 Podmínky přispívání

- Je dostupný mailing-list `sodium-subscribe@pureftpd.org`, který dle popisu slouží k diskuzím ohledně knihovny.
- Jinak knihovna nemá nastavené žádné procesy přispívání do kódu. Správce knihovny na e-mail s otázkou ohledně těchto procesů neodpovídá. Z dostupných informací vyplývá, že kdokoliv může na GitHubu otevřít pull-request a počkat na schválení správcem knihovny.
- Střední doba mezi vytvořením pull-requestu a jeho schválením (mergem) tvoří **71** hodin (cca. **3** dny).[4]
- Znázornit procesy spojené s přispíváním nám pomohou: GitHub REST API a Python skripty[5,6]. Podíváme se, kdo do knihovny v posledních letech přispíval:



Obrázek 3.1 Libsodium: autoři commitů od roku 2020

Na obrázku je dobře vidět, že knihovnu spravuje jednotlivec a přispívání do kódu není tak častým jevem.

3.1.1.3 Financování

V repozitáři je odkaz na stránku, kde se dá objednat předplatné nebo zaslat jednorázový příspěvek.[7] Na této stránce se dá exportovat statistiku všech transakcí. Celkový obnos od roku 2019 do současné doby[8] tvoří **5520 USD**.

[4] vypočítáno na celé historii pull-requestů
[5] `https://github.com/mspi21/commitfetch`
[6] `https://github.com/mspi21/commitstats`
[7] `https://opencollective.com/libsodium/contribute`
[8] zkontrolováno 17. března 2024

3.1.2 Open-source — bezpečnost

- Knihovna nemá hlášená žádná CVE.[9]

3.1.2.1 Hlášení zranitelností a reakce

- Je zveřejněn veřejný klíč, který má být použít ke šifrování zprávy poslané na e-mail `j@pureftpd.org`.
- Proces reakce není popsán.

3.1.2.2 Bezpečnostní audity

V roce 2017 se libsodium verze 1.0.12 a 1.0.13 podrobil bezpečnostnímu auditu sponzorovanému společností Private Internet Access, která využívá knihovnu ke svým interním účelům. Výsledek auditu říká, že libsodium je bezpečná, vysoce kvalitní knihovna, která splňuje stanovené cíle použitelnosti a efektivity a během zkoumání nebyly odhalené žádné zranitelnosti, jenom několik málo závažných problémů [18].

3.1.3 Kód

- Knihovna je napsána v jazyce C se vsuvkami assembleru.
- Chybí návod na to, jak by měl vypadat styl nového kódu.
- Testovací procesy jsou popsány v dokumentaci.[10] Repozitář obsahuje unit testy, je napojen na nástroj OSS Fuzz. Bylo však odhaleno, že jeden z nástrojů na statickou analýzu kódu byl naposledy spouštěn v roce 2019[11] a neměří se vůbec pokrytí kódu testy.

3.1.4 API

API knihovny libsodium je převzat z knihovny NaCl a o něco rozšířen. Jeho hlavními vlastnostmi jsou výchozí bezpečnost a jednoduchost použití. Z pohledu designu má knihovna dva druhy API funkcí:

- 1. Obecné: skrývají před uživatelem použitá kryptografická primitiva.

```
#define MESSAGE ((const unsigned char *) "test")
#define MESSAGE_LEN 4
#define CIPHERTEXT_LEN (crypto_secretbox_MACBYTES
+ MESSAGE_LEN)

unsigned char key[crypto_secretbox_KEYBYTES];
unsigned char nonce[crypto_secretbox_NONCEBYTES];
unsigned char ciphertext[CIPHERTEXT_LEN];

crypto_secretbox_keygen(key);
randombytes_buf(nonce, sizeof nonce);
crypto_secretbox_easy(ciphertext, MESSAGE,
MESSAGE_LEN, nonce, key);

unsigned char decrypted[MESSAGE_LEN];
```

[9] https://nvd.nist.gov/
[10] https://doc.libsodium.org/internals
[11] https://scan.coverity.com/projects/2397

```
if (crypto_secretbox_open_easy(decrypted, ciphertext,
CIPHERTEXT_LEN, nonce, key) != 0) {
    /* message forged! */}
```

Výpis kódu 3.1 Obecné API libsodium: autentizované šifrování [17]

2. Specifické: přímo ve jménu obsahují název použitého primitiva.

```
#define MESSAGE (const unsigned char *) "test"
#define MESSAGE_LEN 4
#define ADDITIONAL_DATA (const unsigned char *) "123456"
#define ADDITIONAL_DATA_LEN 6

unsigned char nonce[crypto_aead_chacha20poly1305_NPUBBYTES];
unsigned char key[crypto_aead_chacha20poly1305_KEYBYTES];
unsigned char ciphertext[MESSAGE_LEN +
crypto_aead_chacha20poly1305_ABYTES];
unsigned long long ciphertext_len;

crypto_aead_chacha20poly1305_keygen(key);
randombytes_buf(nonce, sizeof nonce);

crypto_aead_chacha20poly1305_encrypt(ciphertext, &ciphertext_len,
                                     MESSAGE, MESSAGE_LEN,
                                     ADDITIONAL_DATA, A
                                     DDITIONAL_DATA_LEN,
                                     NULL, nonce, key);

unsigned char decrypted[MESSAGE_LEN];
unsigned long long decrypted_len;
if (crypto_aead_chacha20poly1305_decrypt(decrypted, &decrypted_len,
                                         NULL,
                                         ciphertext, ciphertext_len,
                                         ADDITIONAL_DATA,
                                         ADDITIONAL_DATA_LEN,
                                         nonce, key) != 0) {
    /* message forged! */ }
```

Výpis kódu 3.2 Specifické API libsodium: autentizováné šifrování [17]

Obecné API funkce mají sloužit především uživatelům, kteří nevědí, který algoritmus je třeba zvolit a jenom chtějí mít bezpečnou kryptografii pro nějaký dílčí účel. Specifické API už nabízí větší flexibilitu, protože umožňuje vývojáři vybrat algoritmus (v nabídce ale zůstávají pouze ověřené a bezpečné algoritmy).

API lze také rozdělit podle funkcionality:

- Nízkoúrovňové API

  Dle dokumentace má sloužit pouze jakožto stavební bloky pro customní konstrukce a interoperabilitu s jinými knihovnami a aplikacemi. Proto mohou být nabízená primitiva nebezpečná, pokud s nimi nebude zacházeno správně.[12] Například knihovna nabízí použití šifry ChaCha20 bez autentizace, což není ve výchozím nastavení bezpečné.

- Vysokoúrovňové API

  Zde knihovna nabízí pouze primitiva bezpečná i ve výchozím nastavení. Jedná se o rychlé a ověřené kryptografické algoritmy.

[12] `https://doc.libsodium.org/advanced`

3.1.4.1 Další vlastnosti API

- Přítomnost pomocných funkcí.

  API je opatřeno pomocnými funkcemi: zabezpečené alokace, aritmetické operace s velkými čísly, vyčištění zásobníku za účelem smazání citlivých údajů, porovnání stejně velkých úseků paměti za stejný čas, hexadecimální a Base64 kódování/dekódování.[13]

- Informování o chybách je řešeno jednoduše pomocí návratového kódu. 0 je vracená v případě úspěchu, -1 v případě jakéhokoliv neúspěchu. Není žádná možnost zjistit, co se konkretně nepodařilo. Lze ale zmínit, že většinou je několik prováděných operací spojeno do jednoho API volání, které není moc flexibilní ve svých parametrech, proto taková potřeba může nastat zřídka kdy.

- Knihovna zpřístupňuje generátor náhodných čísel, který je poskytován operačním systémem.

3.1.5 Dokumentace

- Je hezky zpracována a dá se v ní dobře navigovat.

- Na začátku obsahuje nápovědu, které z nabízených kryptografických primitiv má vývojář použít při příslušných cílech.

- Skoro každá z funkcí, která dle dokumentace spadá do vysokoúrovňového API, je opatřena příkladem použití, účelem a popisem případných problémů.

- Dokumentace nízkoúrovňového API již nemá tak podrobný popis a často chybí příklady použití. To je možné vysvětlit tím, že funkce tam zmíněné autor knihovny nedoporučuje k běžnému využití.

3.1.6 Analyzované aplikace

Jak již bylo zmíněno, jedním ze směrů analýzy je snaha zkontrolovat aplikace, které tuto knihovnu používaly, a zkusit odhalit případné problémy, na které narazili vývojáři.

Dále je uveden seznam nalezených aplikací, které k nějakému účelu používaly libsodium (přesný účel je uveden pouze tehdy, když bylo možné jej jednoduše zjistit).

- Vim — konzolový textový editor.[14]
  - Používá libsodium ke čtení a zápisu zašifrovaných souborů.
  - Jsou issues, která stojí za prozkoumání.
- Drawpile — kolaborativní kreslicí program.[15]
  - Používá libsodium k hašování hesel.
  - Žádná zajímavá issues ani CVE nejsou.
- SoftEther VPN — VPN klient a server.[16]
  - Používá libsodium k implementaci protokolu WireGuard.
  - Žádná zajímavá issues ani CVE nejsou.

[13] `https://doc.libsodium.org/helpers`
[14] `https://github.com/vim/vim`
[15] `https://github.com/drawpile/Drawpile`
[16] `https://github.com/SoftEtherVPN/SoftEtherVPN`

- Warzone 2100 — strategická hra v reálném čase.[17]
  - Používá libsodium k generování soukromých klíčů hráčů a šifrování relací a zpráv.
  - Žádná zajímavá issues ani CVE nejsou.
- Valve's Game Networking Sockets — základní transportní vrstva pro hry.[18]
  - Používá libsodium jako jednu z realizací rozhraní šifrování (společně s OpenSSL).
  - Existuje CVE, které stojí za prozkoumání.
- ZeroMQ — knihovna, která rozšiřuje standardní soketová rozhraní.[19]
  - Používá libsodium jako šifrovací rozhraní pro protokol CURVE[20].
  - Existuje CVE, které stojí za prozkoumání.

3.1.7 Analýza nalezených problémů

V této sekci je znázorněna analýza souvislosti nalezených problémů s knihovnou. Všechna nalezená CVE mají odkaz na opravný commit, což analýzu výrazně usnadňuje.

3.1.7.1 Vim

Analýza issue: sodium encryption misuses crypto_pwhash API[21]

- **Problém:** Vim nikam neukládá parametry použité pro funkce pro odvozování klíče z hesla. Používá výchozí parametry, které se dle dokumentace knihovny mohou v budoucnu měnit.
- **Dopad:** Potenciální neschopnost dešifrování dříve zašifrovaných souborů v budoucnu.
- **Souvisí to s knihovnou?** Ne. Dokumentace k pwhash API říká: *Keep in mind that to produce the same key from the same password, the same algorithm, the same salt, and the same values for opslimit and memlimit must be used. Therefore, these parameters must be stored for each user [17].*

Analýza pull-requestu: sodium: use memory locking for the keys[22]

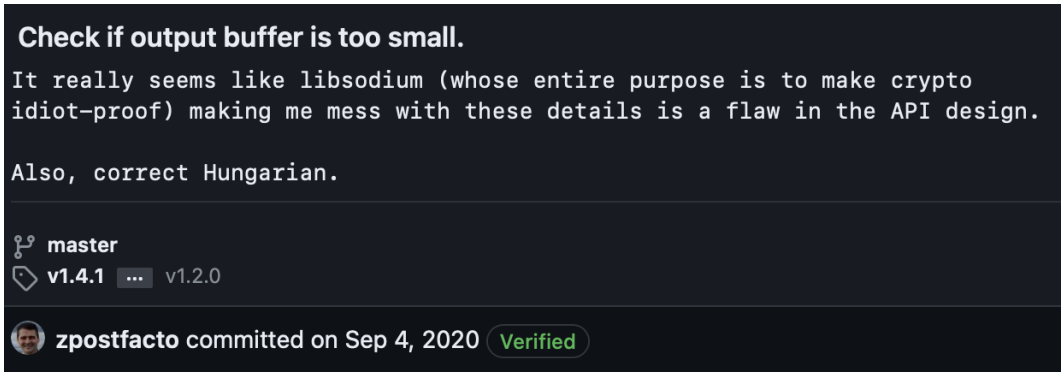
- **Problém:** Stránky paměti, kde leží heslo a od něj odvozený klíč, se mohou objevit ve swap partition.
- **Dopad:** Potenciální přečtení klíče a hesla jiným procesem ze swap partition.
- **Souvisí to s knihovnou?** Ne. Knihovna má zdokumentované API pro ochranu paměti. Je na programátorovi, aby ho použil [17].

[17] `https://github.com/Warzone2100/warzone2100`
[18] `https://github.com/ValveSoftware/GameNetworkingSockets`
[19] `https://github.com/zeromq/libzmq`
[20] protokol definující mechanismus pro bezpečnou autentizaci a důvěrnost komunikace mezi klientem a serverem
[21] `https://github.com/vim/vim/issues/12204`
[22] `https://github.com/vim/vim/pull/8657`

3.1.7.2 Valve's Game Networking Sockets

Analýza CVE-2020-6018[23]

- **Problém:** *Valve's Game Networking Sockets prior to version v1.2.0 improperly handles long encrypted messages in function `AES_GCM_DecryptContext::Decrypt()` when compiled using libsodium.*
- **Dopad:** Porucha paměti a případné vzdálené spuštění kódu.
- **Souvisí to s knihovnou?** Ano. Tady je vhodné se podívat do popisu a kódu commitu, který tuto zranitelnost řeší.[24]



Obrázek 3.2 CVE-2020-6018: popis opravného commitu

```
bool AES_GCM_DecryptContext::Decrypt(
        const void *pEncryptedDataAndTag, size_t cbEncryptedDataAndTag,
        const void *pIV,
        void *pPlaintextData, uint32 *pcbPlaintextData,
        const void *pAdditionalAuthenticationData, size_t cbAuthenticationData
) {
-   unsigned long long pcbPlaintextData_longlong;
-
+       // Make sure caller's buffer is big enough to hold the result
+       if ( cbEncryptedDataAndTag > *pcbPlaintextData + crypto_aead_aes256gcm_ABYTES )
+       {
+               *pcbPlaintextData = 0;
+               return false;
+       }
+
+   unsigned long long cbPlaintextData_longlong;
    const int nDecryptResult = crypto_aead_aes256gcm_decrypt_afternm(
-               static_cast<unsigned char*>( pPlaintextData ), &pcbPlaintextData_longlong,
+               static_cast<unsigned char*>( pPlaintextData ), &cbPlaintextData_longlong,
                nullptr,
                static_cast<const unsigned char*>( pEncryptedDataAndTag ), cbEncryptedDataAndTag,
                static_cast<const unsigned char*>( pAdditionalAuthenticationData ), cbAuthenticationData,
                static_cast<const unsigned char*>( pIV ), static_cast<const crypto_aead_aes256gcm_state*>( m_ctx )
        );

-   *pcbPlaintextData = pcbPlaintextData_longlong;
+   *pcbPlaintextData = cbPlaintextData_longlong;

    return nDecryptResult == 0;
}
```

Obrázek 3.3 CVE-2020-6018: kód opravného commitu

[23] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-6018

[24] https://github.com/ValveSoftware/GameNetworkingSockets/commit/bea84e2844b647532a9b7fbc3a6a8989d66e49e3

Na řádcích 82-87 je dodána kontrola, že výstupní buffer je dostatečně velký na to, aby se do něj vešel výsledný dešifrovaný text. Autor commitu zmiňuje, že to je chyba v API designu knihovny. To je pravda, zde je vidět, že API obsahuje potenciálně zranitelnou konstrukci, protože nepočítá s velikostí výstupního bufferu. Jedná se o starý a známý problém jazyka C, který např. se snaží řešit v standardní knihovně zavedením nových bezpečnějších funkcí jako např.

```
errno_t memcpy_s (void *restrict dest, rsize_t destsz,
const void *restrict src, rsize_t count);
```

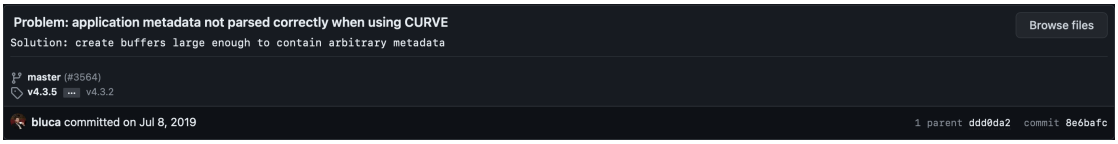
kde novým vstupním parametrem je velikost výstupního bufferu `destsz` [10].

3.1.7.3 ZeroMQ

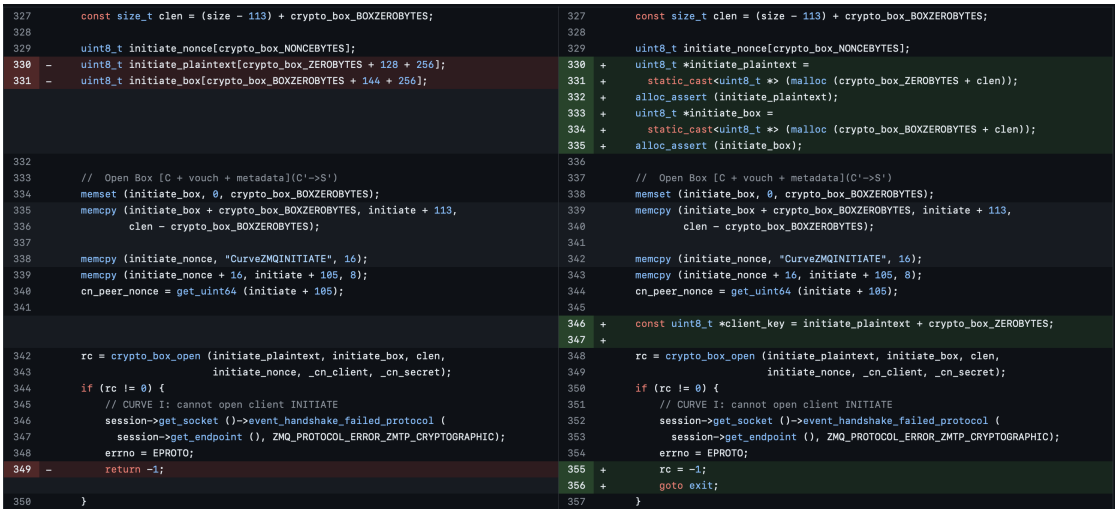
O tom, že API opravdu má zmíněný nedostatek, svědčí i další nalezená zranitelnost.

Analýza CVE-2019-13132[25]

- **Problém:** *A remote, unauthenticated client connecting to a libzmq application, running with a socket listening with CURVE encryption/authentication enabled, may cause a stack overflow and overwrite the stack with arbitrary data, due to a buffer overflow in the library.*
- **Dopad:** Odepření služby.
- **Souvisí to s knihovnou?** Ano. Tady je vhodné se podívat do commitu[26], který tuto zranitelnost řeší.



Obrázek 3.4 CVE-2019-13132: popis opravného commitu



Obrázek 3.5 CVE-2019-13132: kód opravného commitu

[25] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13132
[26] https://github.com/zeromq/libzmq/commit/8e6bafcbe08ece7ae06128d6d01093f07d202ca5

Pro naše účely není potřeba zacházet do detailů této zranitelnosti, jenom ukážeme souvislost s knihovnou. Problém nastává ve funkci, která zpracovává příchozí zprávu ze socketu. Velikost proměnné `clen` (řádek 327) je dynamická a závisí na proměnné `size`, která je deklarována na řádku 274 jako velikost příchozí zprávy. Na zásobníku alokujeme dva buffery fixní délky: `initiate_plaintext` pro dešifrovaná data, `initiate_box` pro zašifrovaná (řádky 329-331). První přetečení nastává kvůli tomu, že kopírujeme zašifrovaná data ze zprávy do bufferu fixní délky bez dodatečné kontroly velikostí (řádky 335-336). Druhé přetečení nastává, když se pokoušíme dešifrovat data do bufferu fixní délky. Takže jde znovu o ten samý nedostatek v API knihovny.

3.1.8 Shrnutí

Knihovna má zprostředkovat základní kryptografické operace i neznalým v oblasti kryptografie programátorům. Sice neposkytuje tak širokou nabídku primitiv a protokolů (chybí podpora TLS a certifikatů), ale může předejít množství špatných použití v běžných aplikacích pomocí jednoduchosti API a přehledné dokumentace.

Lze říci, že se jejím hlavním nedostatkem jeví kvalita vývojových procesů: správa pouze jedním člověkem a absence prokazatelné motivace zvětšují pravděpodobnost hrozby jejich selhání.

Shrnutí bezpečnostních vlastností knihovny je v následující tabulce.

Tabulka 3.1 Hodnocení libsodium (✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

Bod	Shrnutí	
Open-source vývoj	▬ správce a vývojář – jeden člověk	✗
	▬ vývoj je aktivní	✔
	▬ financování: 5520 USD od roku 2019	✗
	▬ přispívání: kdokoliv	✗
Open-source bezpečnost	▬ bezpečnostní audit kódu naposledy v roce 2017	✓
	▬ proces hlášení zranitelností je popsán	✔
	▬ proces reakce na zranitelností není popsán	✗
Kód	▬ provádí se statická (jeden z nástrojů se nespouští), dynamická analýza, fuzzing, unit testy, neměří se pokrytí kódu testy	✓
	▬ nejsou pokyny na styl nového kódu	✗
API	▬ jednoduché a stručné, srozumitelné pro běžného vývojáře	✔
	▬ high-level API nabízí pouze bezpečná primitiva ”by default”	✔
	▬ nabízí se nerozsáhlé low-level API s potenciálně nebezpečnými primitivy	✓
	▬ zpřístupňuje kvalitní generátor náhodných čísel	✔
	▬ návratový kód značící chybu je vždy stejný	✗
	▬ funkce nekontrolují velikost výstupního bufferu	✗
	▬ existují pomocné funkce	✔
Dokumentace	▬ hezky zpracovaná a dobře navigovatelná	✔
	▬ obsahuje nápovědu ohledně účelů funkcí	✔
	▬ pro vysokoúrovňové API skoro vždy obsahuje příklad použití	✔
	▬ pro nízkoúrovňové API je chudá	✗

3.2 Botan

Cílem Botanu je stát se nejlepší volbou pro kryptografii v jazyce C++ poskytováním nástrojů nezbytných pro implementaci široké škály praktických systémů, jako je protokol TLS, certifikáty X.509, moderní AEAD šifry, podpora PKCS#11 a hardwarového modulu TPM, hašování hesel a post-kvantová kryptografická schémata [19]. Z cílů knihovny plyne, že disponuje velmi širokou nabídkou kryptografických primitiv, a proto se často používá jako náhrada nebo alternativa OpenSSL.

Správci Botanu byl adresován e-mail s obecnými otázkami ohledně knihovny, odkud budou čerpány citace v příslušných sekcích (obsah celé zprávy je v příloze A).

3.2.1 Open-source — vývoj

- Projekt se aktivně vyvíjí, poslední verze vyšla v únoru 2024.[27]
- Licence *Simplified BSD*.
- Knihovna není široce rozšířena, příkaz `apt-cache rdepends` nedává žádné výsledky.

3.2.1.1 Tým a správa projektu

Hlavní správce a vývojář je Jack Lloyd. Znovu je dohledatelný profil na LinkedIn[28]:

- Je crypto inženýr.
- Od roku 2021 pracuje v společnosti DFINITY[29] v týmu zaměřeném na kryptografii v blockchainu.

Od roku 2015 knihovnu podporuje Bundesamt für Sicherheit in der Informationstechnik[30], dále jenom BSI. Byly založeny 2 projekty[31]:

1. Zaměřený na zlepšení knihovny a její dovedení do standardů BSI a následující správa a podpora.
2. Zaměřený na implementaci post-kvantové kryptografie.

Vykonavatel projektů: Rohde & Schwarz Cybersecurity GmbH.[32]

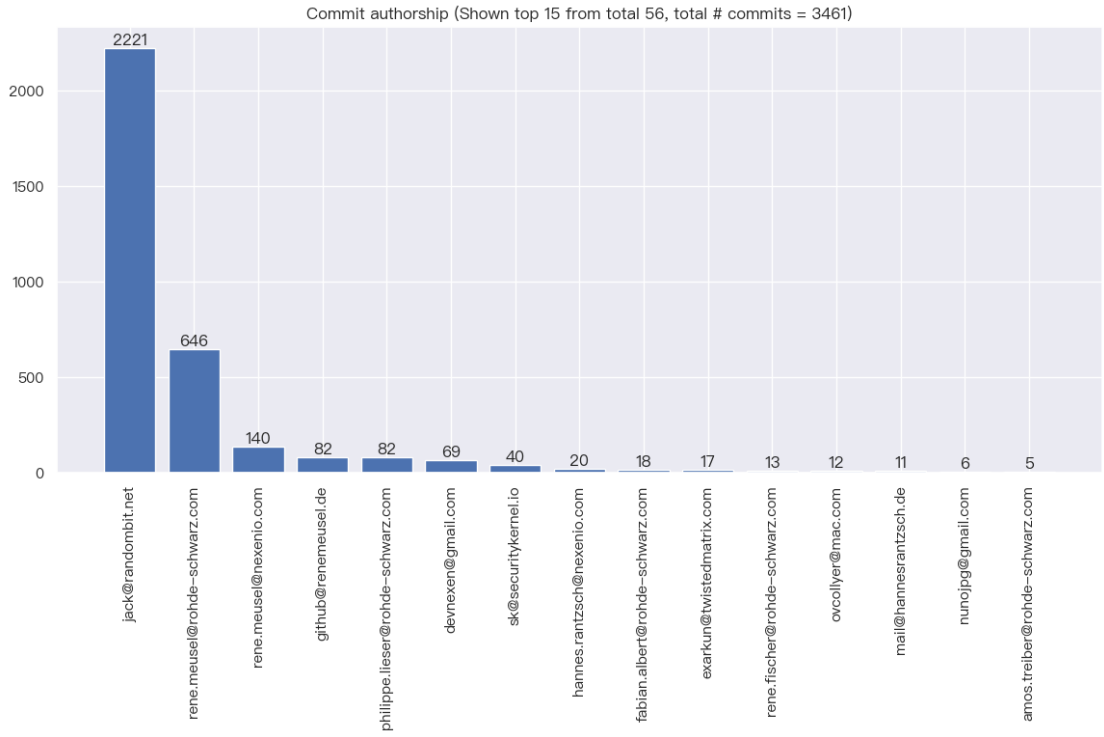
3.2.1.2 Podmínky přispívání

Knihovna nemá přesně definované procesy přispívání do kódu. V dokumentaci je popis toho, jak by kód měl vypadat u případného příspěvku. Z emailu:
— *"Do you have any special requirements for contributing in Botan repository? As I see anyone can open the pull request and if it will be reviewed, for example, by you, it could be merged."*
— *"Right, there is no CLA or anything of that sort and we'll accept contributions from anyone as long as the change is in concord with the project goals."*

- Střední doba mezi vytvořením a schválením (mergem) pull-requestu je **151** hodin (cca. **6** dní).[33]

[27] zkontrolováno 8. května 2024
[28] `https://www.linkedin.com/in/jacklloyd/`
[29] `https://dfinity.org`
[30] z němčiny Federální úřad pro informační bezpečnost
[31] `https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Kryptografie/Kryptobibliothek-Botan/kryptobibliothek-botan_node.html`
[32] `https://www.rohde-schwarz.com/cz/solutions/cybersecurity/about-us/research/kbls_254244.html`
[33] vypočítáno na celé historii pull-requestů

- Graf přispívatelů:



Obrázek 3.6 Botan: autoři commitů od roku 2020

Z mailových adres na obrazku už je celkem vidět, že většina autorů patří do společnosti Rohde & Schwarz Cybersecurity GmbH., která je vykonavatelem projektů BSI.

3.2.1.3 Financování

Od roku 2015 dostává finanční podporu od BSI na vykonávání zmíněných projektů.

3.2.2 Open-source — bezpečnost

Na stránce knihovny je od počátku roku 2014 evidována 31 chyba v kódu, CVE databáze obsahuje 26 z nich.[34] Zde je důležité rovněž podotknout, ze se skoro všechna CVE objevila ve chvíli, kdy se na vývoji začal podílet tým odborníků. Odsud plyne to, ze počet nahlášených zranitelností velmi slabě koreluje s celkovou bezpečností knihovny. Menší počet zranitelností svědčí pouze o tom, že projekt věnuje málo úsilí k jejich hledání [20].

3.2.2.1 Hlášení zranitelností a reakce

- Hlášení probíhá na e-mail `jack@randombit.net` a je zveřejněn veřejný klíč, který by měl být použít k šifrování zprávy.
- Reakce není popsána v dokumentaci, ale ve výsledné projektové zprávě je zmíněno, že další správa knihovny zahrnuje případnou opravu chyb nalezených v kódu maximálně 4 týdny od objevení chyby [21].

[34] `https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Botan`

3.2.2.2 Bezpečnostní audity

Audit proběhl jakožto součást prvního projektu BSI, byla hodnocená kvalita a složitost kódu, návrh, pokrytí kódu testy, odolnost proti útokům postranními kanály [21]. Botan dosáhl nejlepších výsledků v porovnání s ostatními hodnocenými knihovnami a proto byl zvolen pro další rozvoj a podporu, nalezené chyby byly následně opraveny.

3.2.3 Kód

- Knihovna je napsána v C++, ale nabízí se Foreign Function Interface (FFI) pro jazyk C.
- Existují pokyny na styl nového kódu.[35]
- Lze dělit na moduly.

  Jedním z důvodů, proč BSI pro svoje účely zvolil Botan, je jeho modularizace, tj. možnost vybrat, jaké moduly budou součásti výsledné binárky [21]. To např. umožňuje odstranit nebezpečné algoritmy ještě na úrovni kompilace.
- Repozitář obsahuje unit testy (navíc knihovna disponuje samostatným frameworkem na jejich vytvoření), je napojen na nástroj OSS Fuzz, měří se pokrytí kódu testy (dosahuje 92 %)[36]. Všechno se spouští během průběžné integrace (CI).

3.2.4 API

Tady je nutno zmínit jeden hlavní problém, o kterém píše autor knihovny v mailu:
— *"Did you noticed any problems, that the developers have faced to, when using Botan? If so, can you name them."*
— *"One big issue is that Botan (like almost all general purpose crypto libraries) is not particularly safe to use by non-experts. Being safe in this way (as say Google's Tink attempts to be) limits the API to where it is not practical to build general purpose systems, which was considered a more important goal by me since I started writing the library specifically to build the distributed systems I was interested in writing."*
Takže už ze záměru Botanu jeho API nemusí být bezpečně použitelné pro neodborníky, s tím je potřeba počítat.

Celé API knihovny lze nazvat nízkoúrovňovým, protože dává flexibilitu ve volbě parametrů a primitiv.

```
Botan::AutoSeeded_RNG rng;

const std::string plaintext("text");

const auto enc = Botan::Cipher_Mode::create_or_throw(
   "AES-128/CBC/PKCS7", Botan::Cipher_Dir::Encryption);

// generate key
Botan::secure_vector<uint8_t> key =
rng.random_vec(enc->minimum_keylength());
enc->set_key(key);

// generate fresh nonce (IV)
Botan::secure_vector<uint8_t> iv =
rng.random_vec(enc->default_nonce_length());
```

[35] https://botan.randombit.net/handbook/dev_ref/contributing.html
[36] https://coveralls.io/github/randombit/botan

```
// Copy input data to a buffer that will be encrypted
Botan::secure_vector<uint8_t> pt(plaintext.data(),
   plaintext.data() + plaintext.length());

enc->start(iv);
enc->finish(pt);
```

Výpis kódu 3.3 Botan: šifrování AES-128 v módu CBC [19]

Z této ukázky kódu je vidět, že uživatel už musí vybrat šifrovací algoritmus, operační mód a algoritmus pro padding.

3.2.4.1 Další vlastnosti API

- Přítomnost pomocných funkcí.

  API je opatřeno pomocnými funkcemi pro bezpečné alokace, práci s velkými čísly a hexadecimální/Base64 kódovaní a dekódovaní.

- Srozumitelné informování o chybách.

  Zde knihovna obsahuje hezkou implementaci hlášení chyb pomocí C++ výjimek, které obsahují popis chyby. Takže pokud výjimka není ošetřena, program spadne. Příklad: v předchozí ukázce kódu jsme zvolili algoritmus pro padding PKCS7, ale když tam zadáme CTS (Ciphertext stealing), který potřebuje alespoň jeden celý blok plus jeden bajt a necháme krátkou zprávu, bude vyhozena výjimka `libc++abi: terminating due to uncaught exception of type Botan::Encoding_Error: Encoding_error: AES-128/CBC/CTS: insufficient data to encrypt`.

- FFI C API

  Za zmínku stojí také C API této knihovny. I když není primární, některé aplikace jej využívají. Zde například jsou vyřešené problémy, kterými trpí API knihovny libsodium:

  1. Hlášení chyb probíhá pomocí různých návratových kódů, takže vývojář má možnost zjistit, co se konkrétně nepodařilo.
  2. Všechny funkce, které pracují s buffery, přijímají délku jak vstupního, tak i výstupního bufferu, a pokud není délka výstupního dostačující, bude vrácen příslušný chybový kód.

  ```
  int botan_cipher_update(botan_cipher_t cipher, uint32_t flags,
  uint8_t output[], size_t output_size, size_t *output_written,
  const uint8_t input_bytes[], size_t input_size, size_t *
      input_consumed)
  ```

3.2.5 Dokumentace

Dokumentaci lze rozdělit na 2 části:

1. C++ API
   - Je rozdělená na sekce, pomocí kterých se lze snadno dostat ke hledanému primitivu nebo funkci.
   - Obsahuje doporučení, která primitiva jsou bezpečná a kterým je lépe se vyhnout.
   - Obsahuje příklady použití, ale ne vždy. Podrobný popis nabízených funkcí to ale někdy nahrazuje.

> **Warning**
>
> In almost all cases, a bare block cipher is not what you should be using. You probably want an authenticated cipher mode instead (see Cipher Modes) This interface is used to build higher level operations (such as cipher modes or MACs), or in the very rare situation where ECB is required, eg for compatibility with an existing system.

Obrázek 3.7 Botan: varování o použití nebezpečného operačního módu (ECB) [19]

2. C API
   - Neobsahuje žádné příklady použití.
   - Je zastaralá, byla objevena inkonzistence s reálným použitím.[37]

Celkem lze dokumentaci vyhodnotit jako dostačující, ale lze ji nadále vylepšovat, což je také součástí zmíněných projektů. Nedostatky v dokumentaci FFI C API lze vysvětlit tím, že nejde o primární API, ale jenom o rozšiřující.

3.2.6 Analyzované aplikace

- KeePass — správce hesel.[38]
  - Dřív používal sadu různých knihoven, před 3 lety nahradil všechno Botanem.
  - Žádná zajímavá CVE ani issues nejsou.
- Kea — customní DHCP server.[39]
  - Používá Botan na TLS a jiné operace.
  - Žádná zajímavá CVE ani issues nejsou.
- Trantor — neblokující I/O TCP síťová knihovna.[40]
  - Používá Botan a OpenSSL jako interfacy pro TLS.
  - Žádná CVE, issues jsou jenom týkající se OpenSSL.
- FiveM — modifikační framework pro PC verzi Grand Theft Auto V.[41]
  - Používá Botan na TLS a jiné operace.
  - Žádná zajímavá CVE ani issues nejsou.
- SoftHSM — část projektu OpenDNSSEC.[42]
  - Používá Botan a OpenSSL jako 2 krypto-backendy.
  - Žádná zajímavá CVE nejsou, issues jsou jenom tykající se OpenSSL.
- RNP — vysoce výkonná C++ OpenPGP[43] knihovna používaná v Mozilla Thunderbird.[44]
  - Používá Botan a OpenSSL jako 2 krypto-backendy.
  - Jsou nalezeny problémy, které stojí za prozkoumání.

[37] chybný návratový kód se neshodoval s dokumentovaným
[38] `https://github.com/keepassxreboot/keepassxc`
[39] `https://gitlab.isc.org/isc-projects/kea`
[40] `https://github.com/an-tao/trantor`
[41] `https://github.com/citizenfx/fivem`
[42] `https://github.com/opendnssec/SoftHSMv2`
[43] protokol pro šifrování a podepisování dat
[44] `https://github.com/rnpgp/rnp`

3.2.7 Analýza nalezených problémů

Předchozí CVE obsahovala přímo v popisu odkaz na commit na GitHubu, kde se problém řeší. Zde uvidíme jenom referenci na verzi, ve které byla provedena oprava. Takže commit musíme zkusit dohledat vlastními silami.

Byly nalezeny dvě zranitelnosti, které by dle popisu s knihovnou souviset určitě mohly:

- **CVE-2021-33589**[45]
  Ribose RNP before 0.15.1 does not implement a required step in a cryptographic algorithm, resulting in weaker encryption than on the tin of the algorithm.
- **CVE-2023-29480**[46]
  Ribose RNP before 0.16.3 sometimes lets secret keys remain unlocked after use.

Dále provedeme ukázkovou analýzu CVE-2021-33589:

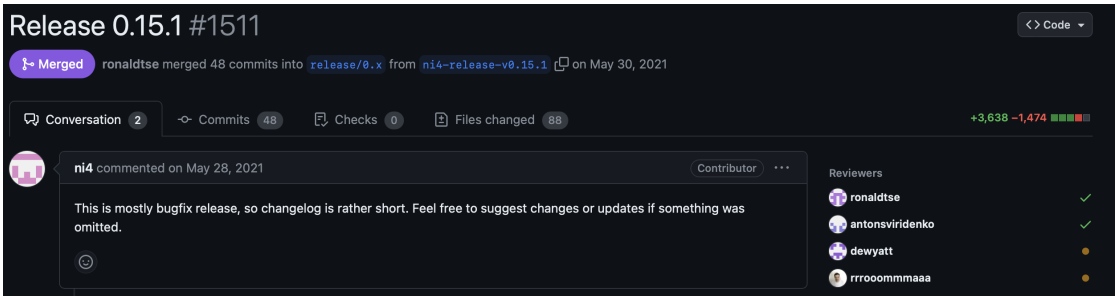
1. V changelogu[47] k verzi 0.15.1 je vidět:

```
#### Security

* Fixed issue with cleartext key data after the `rnp_key_unprotect()`/`rnp_key_protect()` calls (CVE-2021-33589).
```

Obrázek 3.8 RNP verze 0.15.1: changelog

2. Nalezený pull-request[48] k zkoumanému releasu:



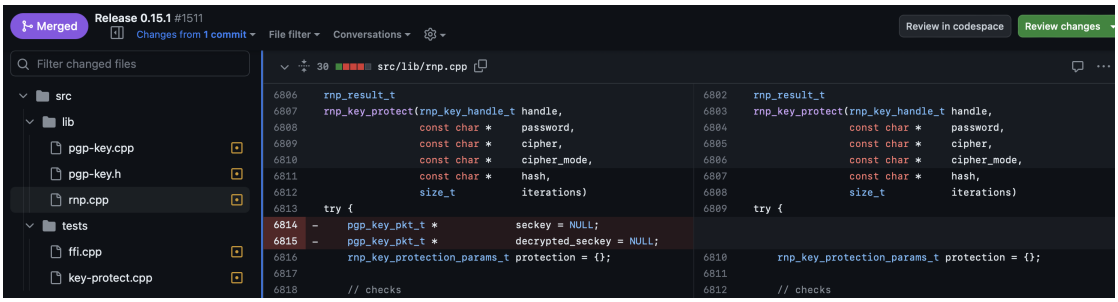
Obrázek 3.9 RNP verze 0.15.1: pull-request

3. Nalezený commit[49], který nějak řeší key protection:

Obrázek 3.10 RNP verze 0.15.1: zajímavý commit

[45] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-33589
[46] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-29480
[47] https://github.com/rnpgp/rnp/releases/tag/v0.15.1
[48] https://github.com/rnpgp/rnp/pull/1511
[49] https://github.com/rnpgp/rnp/pull/1511/commits/936764fb34cfdd37cc1daff78efb4a87c6f1df51

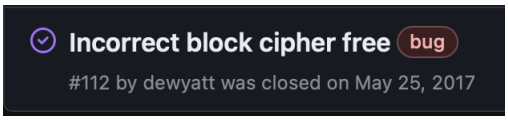
4. A tady nejsou žadné změny v souborech spojených s knihovnou, protože použití knihoven se nachází v souborech umístěných v `src/lib/crypto`.



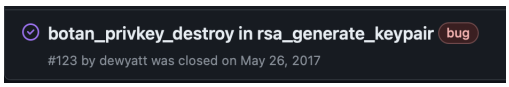
Obrázek 3.11 RNP verze 0.15.1: kód zajímavého commitu

Výsledkem zkoumání je, že CVE nejspíš s knihovnou nesouvisí. Podrobnější zkoumání aplikace není naším cílem. Po aplikaci stejného postupu na CVE-2023-29480 také nebyla nalezena souvislost s knihovnou.

Ještě byly nalezeny dva drobné problémy spojené s použitím C API:



(a) Nesprávné free[a]



(b) Memory leak[a]

[a] `https://github.com/rnpgp/rnp/issues/112`

[a] `https://github.com/rnpgp/rnp/issues/123`

Obrázek 3.12 RNP: nalezené drobné problémy

První spočíval v tom, že ke zrušení objektu `botan_block_cipher_t` bylo použito volání `free` místo speciálně určeného `botan_block_cipher_destroy`. Druhý spočíval v tom, že na vytvořený objekt typu `botan_privkey_t` chybělo následné volání `botan_privkey_destroy`, což vedlo k memory leaku. Obě tyto chyby lze považovat za související s knihovnou, protože dokumentace C API je velmi chudá, což může vývojáře zmást.

Velmi zajímavým faktem je ale to, že obě tyto chyby byly opraveny přímo vývojářem knihovny Botan — uživatelem `@randombit`. Toto navíc nejsou výjimečné případy, před tím například byla ve stejné aplikaci realizována náhrada OpenSSL Botanem stejným uživatelem.[50] V aplikaci se `@randombit` rovněž zúčastnil náhrady gcrypt, libsodium a libargon2 Botanem.[51] V aplikaci Kea v commitu, který přidává použití Botanu, se taky zmiňuje, že to bylo uděláno za pomocí týmu knihovny Botan.[52]

V e-mailu adresovanému správci Botanu proto byla položena otázka:
— *"When I was researching the applications, that use Botan, I have noticed the tendence, that you are helping developers to use Botan correctly. So any developer can turn to you, when he wants to add Botan's support, am I right?"*, na kterou odpověděl:
— *"Right, at least to the extent my time permits. Since one goal is that the library being functional and easy to use, something being unclear is kind of intrinsically a bug: an API is non-obvious, something is poorly documented, etc. By helping the user the nature of the underlying issue can become more clear."*

[50] `https://github.com/rnpgp/rnp/issues/8`
[51] `https://github.com/keepassxreboot/keepassxc/pull/6209`
[52] `https://gitlab.isc.org/isc-projects/kea/-/commit/baa19fbf7a61b0bcefd7552cc0497c6c04c3b25a`

3.2.8 Shrnutí

Ve výsledku se Botan zdá být nadějnou knihovnou. Sice není v současné době příliš rozšířená a populární, ale má k tomu předpoklady a je velmi pozitivně hodnocená vývojáři, kteří ji již používají.[53] Není náhodou, že byla zvolena německým úřadem pro další rozvoj a využití.

Zajímavá je ještě stránka v repozitáři knihovny[54], která obsahuje popis chyb, kterých se tvůrci dopustili během vývoje. Tam se např. zmiňuje, že by bylo lepší, kdyby knihovna byla napsána v jazyce C, protože by to poskytlo stabilní ABI, umožnilo aplikacím v C používat knihovnu a usnadnilo přepisování knihovny do Rustu.

Shrnutí bezpečnostních vlastností knihovny je v následující tabulce.

[53] `https://github.com/an-tao/trantor/pull/238`
[54] `https://github.com/randombit/botan/blob/master/doc/dev_ref/mistakes.rst`

Tabulka 3.2 Hodnocení Botan (✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

Bod	Shrnutí	
Open-source vývoj	na vývoji se podílí velký tým odborníků	✔
	vývoj je aktivní	✔
	financování ze strany německého úřadu	✔
	zpětná vazba, poskytování pomocí vývojářům	✔
	přispívání: kdokoliv	✗
	poměrně malá rozšířenost	✗
Open-source bezpečnost	bezpečnostní audit jakožto součást projektu BSI	✔
	proces hlášení zranitelností je popsán	✔
	proces reakce na zranitelnosti je popsán, ale ne na stránkách knihovny	✓
Kód	provádí se statická, dynamická analýza, unit testy, fuzzing, pokrytí kódu testy dosahuje 92 %	✔
	existují pokyny na styl nového kódu	✔
API	nepoužívá abstrakce, které jsou pro běžného uživatele srozumitelné	✗
	nenabízí bezpečná primitiva ”by default”	✗
	je mocné a flexibilní	✔
	hlášení chyb pomocí výjimek	✔
	zpřístupňuje kvalitní generátor náhodných čísel	✔
	existují pomocné funkce	✔
Dokumentace	hezky zpracovaná a dobře navigovatelná	✔
	neobsahuje nápovědu ohledně účelů funkcí	✗
	obsahuje varování ohledně nebezpečných primitiv	✔
	pro C API je chudá	✗

3.3 OpenSSL

OpenSSL je robustní, plně vybavená sada kryptografických nástrojů pro skoro cokoliv, co si uživatel může přát. Je nejrozšířenější a nejznámější open-source kryptografickou knihovnou.

3.3.1 Open-source — vývoj

- Projekt se aktivně vyvíjí, poslední verze 3.3.0 vyšla 9. dubna 2024.[55]
- Licence *Apache License Version 2.0.*

3.3.1.1 Tým a správa projektu

Projekt knihovny má velmi podrobné informace o tom, kým a jak je spravován. Všechno se řídí interní směrnicí projektu.[56] Technická rozhodnutí jsou řízena OpenSSL Technical Committee (OTC) a vedení projektu je řízeno OpenSSL Management Committee (OMC). Zdrojový kód OpenSSL je spravován týmem commiterů.[57] To jsou lidé, kteří jsou zodpovědni za správu obsahu repozitáře, mají nárok tam přidávat nové commity a schvalovat příspěvky. Stát se commiterem je možné pouze pozvánkou z OTC a odsouhlasením členů OMC. Taková pozice je vazána smlouvou Contributor Licence Agreement a může být kdykoliv odvolána.

3.3.1.2 Podmínky přispívání

Každý, kdo chce přispět netriviální změnou do kódu, musí podepsat zmíněnou (CLA) smlouvu. Za triviální změnu je považováno např. oprava gramatických nebo typografických chyb, manipulace s bílými znaky a někdy i jednořádkový bugfix. Ale na tom, že změna je triviální, se musí shoudnout autor a všichni, kdo změnu revidují.[58] Všechny příspěvky musí být revidovány alespoň dvěma commitery, z nichž jeden musí být rovnou členem OTC. Ani jeden z nich ale nesmí být autorem revidované změny. Všechny schválené změny se musejí aplikovat nejdřív po 24 hodinách od momentu schválení. Výjimkou jsou pouze opravy s označenou kritičností: *urgent*.[59] Rovněž je na stránkách projektu popis toho, jak má vypadat styl kódu.[60]

3.3.1.3 Financování

Projekt OpenSSL závisí na finanční podpoře od uživatelské komunity. Tato finanční podpora je možná buď formou sponzorských darů nebo smluv o technické podpoře. Nabízí se 3 druhy smluv: Basic (**15000 USD** ročně), Vendor (**25000 USD** ročně) a Premium (**50000 USD** ročně).[61]

3.3.2 Open-source — bezpečnost

CVE databáze obsahuje přes 240 zranitelností za celou dobu existence OpenSSL. Znovu nelze říci, že to je znakem nebezpečného software, naopak svědčí o tom, že je kladen důraz na jejich hledání.

[55] zkontrolováno 8. května 2024
[56] `https://www.openssl.org/policies/omc-bylaws.html`
[57] `https://www.openssl.org/community/committers.html`
[58] `https://www.openssl.org/policies/cla.html`
[59] `https://www.openssl.org/policies/general/committer-policy.html`
[60] `https://www.openssl.org/policies/technical/coding-style.html`
[61] `https://www.openssl.org/support/contracts.html`

3.3.2.1 Hlášení zranitelností a reakce

Hlášení chyb probíhá přes e-mail `openssl-security@openssl.org`. Následně se chyby třídí dle závažnosti: CRITICAL, HIGH, MODERATE, LOW. Závažnost pak ovlivňuje postup další reakce. Kritické chyby by měly být opraveny co nejdřív a způsobí release pro každou podporovanou verzi.[62]

3.3.2.2 Bezpečnostní audity

Projekt se podrobil minimálně třem bezpečnostním auditům:

1. Po objevení zranitelnosti Heartbleed společnost Core Infrastructure Initiative sponzorovala bezpečnostní audit v roce 2015, který zahrnoval revizi kódu a fuzz testing. Bylo odhaleno několik závažných chyb [20].
2. BSI zveřejnil výsledky projektu *Quellcode-basierte Untersuchung von kryptographisch relevanten Aspekten der OpenSSL-Bibliothek*[63] v roce 2015, který zahrnoval analýzu dokumentace a implementace generátoru náhodných čísel, hodnocení odolnosti proti známým zranitelnostem. Byl sepsán seznam potenciálně slabých míst a doporučení týkající se jejich opravy.
3. The Open Source Technology Improvement Fund (OSTIF) zveřejnil v roce 2019 audit verze OpenSSL 1.1.1 s důrazem na protokol TLS 1.3 a generátor náhodných čísel.[64] Znovu bylo odhaleno několik závažných chyb, které se podařilo opravit před releasem [20].

3.3.3 Kód

- Knihovna je napsána v jazyce C se vsuvkami assembleru.
- Existuje návod na to, jak má vypadat styl nového kódu.[65]
- Repozitář obsahuje unit testy, fuzz testy pro nástroj OSS Fuzz, měří se pokrytí kódu testy nástrojem Coveralls (pokrytí kolem 65 %)[66]. Všechno se spouští během průběžné integrace (CI).

3.3.4 Dokumentace

- Dokumentace OpenSSL je známa svou nepřehledností. Je velmi špatně navigovatelná, např. neobsahuje ani interní způsob vyhledávání, používá se Google s argumentem `site:www.openssl.org`.
- Chybí rozdělení na sekce. Dokumentace obsahuje např. návod na použití TLS[67], který se ale jmenuje *ossl-guide-tls-introduction* (lze dohledat jenom přes tento název) a není na něj ani odkaz z hlavní stránky.
- Někdy dokonce postrádá popis nabízených funkcí. Např. zmiňuje se funkce `EVP_CIPHER_CTX_set_app_data`, ale nejde dohledat ani použití, ani co je její záměr.
- Popis návratových hodnot je někde úplně jinde, než popis samotné funkce.

[62] `https://www.openssl.org/policies/general/security-policy.html`
[63] `https://www.bsi.bund.de/DE/Service-Navi/Publikationen/Studien/OpenSSL-Bibliothek/opensslbibliothek.html`
[64] `https://ostif.org/wp-content/uploads/2019/01/18-04-720-REP_v1.2.pdf`
[65] `https://www.openssl.org/policies/technical/coding-style.html`
[66] `https://coveralls.io/github/openssl/openssl`
[67] `https://www.openssl.org/docs/man3.2/man7/ossl-guide-tls-introduction.html`

- Obsahuje nějaké příklady použití ve formě zdlouhavých ukázek kódu, které nepokrývají celou funkcionální sadu a někdy navíc obsahují použití fixních textových klíčů.
- Dokumentace neobsahuje žádné bezpečnostní poznámky ohledně např. zastaralosti algoritmů.

Lze říci, že taková dokumentace spíš odradí od svého použití a vývojář bude hledat potřebnou informaci v neoficiálních zdrojích. Zde ale lze zmínit, že kvůli rozšířenosti knihovny jsou k dispozici i dobré neoficiální zdroje, jako např. knihy, které vysvětlují kryptografické koncepty na příkladu OpenSSL. Např. *Demystifying Cryptography with OpenSSL 3.0: Discover the best techniques to enhance your network security with OpenSSL 3.0* nebo *Network Security with OpenSSL: Cryptography for Secure Communications*. Otázkou je, jestli si běžný vývojář bude chtít pořídit knihu a věnovat tomu čas, nebo převezme jenom kód ze Stack Overflow nebo z umělé inteligence. Ve většině případů spíš zvítězí ta druhá varianta.

3.3.5 API

- API je nízkoúrovňové, flexibilní a nabízí velmi širokou funkcionalitu, ale za cenu toho, že se v něm běžný vývojář těžce vyzná.
- Obsahuje pomocné funkce jako jsou např. bezpečné alokace, rozhraní pro práci s velkými čísly nebo Base64 kódovaní/dekódovaní. Nejsou ale nějak zmíněny nebo vyjmenovány.
- Funkce, které pracují s buffery, nekontrolují velikost výstupního bufferu, čímž se vystavují možnému přetečení.
- Informování o chybách je víceúrovňové. Když selže nějaká volaná funkce, obvykle to je signalizováno návratovou hodnotou a chybový kód je uložen ve frontě chyb aktuálního vlákna. Knihovna `err` poskytuje rozhraní k získání těchto chybových kódů a textových chybových zpráv.[68] Proto existuje možnost zjistit, co se konkrétně nepodařilo, jenom je to komplikované a často se zanedbává.

3.3.6 Shrnutí

OpenSSL působí velmi důvěryhodně z pohledu organizačních procesů vývoje. Kvůli své rozšířenosti a popularitě libovolná chyba v OpenSSL může způsobit velké škody napříč celým Internetem, jak to bylo po objevení zranitelnosti Heartbleed. Zasaženy byly webové servery Apache a nginx, na kterých zavisely podle průzkumu Netcraft z dubna 2014 více než 66 % aktivních webových stránek.[69]

Shrnutí bezpečnostních vlastností knihovny je v následující tabulce.

[68] `https://www.openssl.org/docs/man1.0.2/man3/err.html`
[69] `https://heartbleed.com`

Tabulka 3.3 Hodnocení OpenSSL (✔— Bod splněn, ✓— částečné splněn, ✗— nesplněn)

Bod	Shrnutí	
Open-source vývoj	▬ vývoj je spravován několika týmy (OTC, OMC a commitery)	✔
	▬ interní směrnice definuje požadavky na přispívání	✔
	▬ financování ve formě poskytování podpory zákazníkům	✔
	▬ nejrozšířenější knihovna	✔
Open-source bezpečnost	▬ několik bezpečnostních auditů	✔
	▬ proces hlášení zranitelností je popsán	✔
	▬ proces reakce na zranitelnosti je popsán	✔
Kód	▬ provádí se statická a dynamická analýza, unit testy, fuzzing, pokrytí kódu testy jenom kolem 65 %	✓
	▬ existují pokyny na styl nového kódu	✔
API	▬ nepoužívá abstrakce, které jsou pro běžného vývojáře srozumitelné	✗
	▬ nenabízí bezpečná primitiva ”by default”	✗
	▬ je mocné a flexibilní	✔
	▬ informování o chybách je komplikované	✓
	▬ zpřístupňuje kvalitní generátor náhodných čísel	✔
	▬ existují pomocné funkce, ale jsou těžce dohledatelné	✓
Dokumentace	▬ nepřehledná, nerozdělená na sekce a špatně navigovatelná	✗
	▬ neobsahuje nápovědu ohledně účelů funkcí	✗
	▬ neobsahuje varování ohledně nebezpečných primitiv	✗
	▬ deklarace funkcí, jejich popis, návratové hodnoty a příklady použití jsou rozházené	✗

Kapitola 4
Výsledky analýzy

V sekci 1.3 byl popsán obecný proces vývoje bezpečného software, v sekci 2.1 uvažovaný model hrozeb s ohledem na specifika open-source kryptografických knihoven a k tomu v sekci 2.2 seznam faktorů, které zmíněné hrozby mají zmirňovat. Takže ve výsledku byla hodnocena kvalita a důvěryhodnost vývojových procesů a programátorská přivětivost API a dokumentace.

4.1 Efektivita provedené analýzy

Kromě oficiálních stránek a repozitáře knihovny byly za vhodný zdroj pro získání podkladů hodnocení také považovány aplikace, které zkoumanou knihovnu používají. A přesněji chyby v těchto aplikacích, které nějakým způsobem souvisejí s použitím knihovny. V průběhu empirického výzkumu se ale ukázalo, že takové chyby sice mohou poskytnout nějakou užitečnou informaci, ale taková informace nemá dostatečnou vypovídající hodnotu v poměru s vynaloženými sily, protože vyhledávání takových chyb je velmi časově náročný proces, který vyžaduje zkontrolování velkého množství zbytečných informací. Navíc se často jedná pouze o potvrzení nebo důsledek toho, co lze odvodit jenom ze zkoumání stránek knihovny. Proto např. provedení takové analýzy pro knihovnu OpenSSL kvůli rozsahu aplikací a případných chyb nebylo vůbec považováno za přínosné pro účely této práce.

4.2 Porovnání knihoven

OpenSSL je referencí velmi dobře nastavených vývojových procesů a z toho pohledu maximálně zmírňuje hrozbu jejich selhání. Co se týče hrozby špatného použití, tak tady už prohrává. Pokud budeme uvažovat zejména oficiální zdroje informace, tak je velmi pravděpodobné nepochopení ze strany vývojáře neodborného v oblasti kryptografie a následné špatné použití v aplikaci.

Knihovna libsodium se maximálně snaží zmírnit hrozbu špatného použití díky přehlednosti dokumentace a jednoduchosti API. Byly sice odhaleny nedostatky, které i přes jednoduchost vedly k několika zranitelnostem (sekce 3.1.7.2, 3.1.7.3). Rovněž je pravděpodobné selhání ve vývojových procesech, protože knihovna se spoléhá pouze na malé finanční příspěvky a nachází se pod správou jednoho člověka, u kterého daná činnost není ani primárním zaměstnáním.

U knihovny Botan došlo k výraznému zlepšení vývojových procesů po zařazení do projektů německého úřadu (BSI). Knihovna, která předtím byla spravována jednotlivcem, dostala financovaní a tým odborníků, došlo k výraznému nárůstu její funkcionality a použitelnosti. Během analýzy byl zmíněn hlavní problém použitelnosti — knihovna není bezpečně použitelná pro neodborníky, protože chce nabídnout dostatek funkcí pro vývoj *general-purpose* systémů. Ale na rozdíl od OpenSSL se Botan snaží následky takového řešení maximálně zmírnit — a to díky přehlednosti

a podrobnosti dokumentace a odhodlání opravovat dříve dopuštěné chyby. Vývojový proces není zaměřen na zpětnou kompatibilitu (např. ve verzi 3 došlo k přepracování mnoha věcí v porovnání s verzí 2[1]). Navíc velkou výhodou projektu je možnost zpětné vazby s vývojáři. To lze vysvětlit poměrně malou rozšířeností v současné době (ale např. u knihovny libsodium na adresovaný mail nebylo zodpovězeno ani po dobu 2 měsíců).

Lze říci, že Botan se zdá být zlatou střední cestou mezi vybranými knihovnami, finální výběr by se ale měl odrážet od dalších faktorů, jako odbornost vývojáře a účel použití.

4.3 Závěr

Cílem práce bylo vypracovat metodiku, která pomůže aplikačnímu vývojáři zvolit a použít knihovnu tak, aby výsledná aplikace byla co nejbezpečnější. žádoucí vlastnosti knihoven představené v sekci 2.2 a následný postup jejich zmapování v sekci 2.3 lze považovat za takovou metodiku. Je zopakovatelný a poskytuje alespoň odhad toho, jak bezpečné bude použití. Ale jde jenom o odhad, protože to pak může ovlivňovat ještě několik faktorů:

- Odbornost vývojáře v oblasti kryptografie

  Pokud vývojář disponuje znalostmi v dané oblasti, menším zlem by bylo spíš se smířit s případnými nedostatky v uživatelské přívětivosti a rozhodnout se ve prospěch projektu s kvalitnějšími organizačními procesy jako je např. Botan nebo OpenSSL. V opačném případě by bylo lepším řešením přijmout případné nedostatky vývojových procesů a rozhodnout se ve prospěch projektu, který minimalizuje riziko chybného použití, jako např. libsodium.

- Účel a rozsah použití

  Vždy je potřeba rovněž uvažovat, k čemu knihovna bude sloužit v aplikaci, protože uživatelská přívětivost často koliduje s funkcionalitou. Např. pokud se plánuje použití protokolu TLS, jednoduchost API knihovny libsodium se nevyplatí, protože bude potřeba budovat svou implementaci protokolu, což velmi pravděpodobně skončí s chybami.

Jinak ve smyslu zopakovatelnosti a rozsahu poskytovaných informací předpokládáme, že tohoto cíle bylo dosaženo.

[1] `https://botan.randombit.net/handbook/migration_guide.html`

Příloha A

Zpráva správci knihovny Botan

Adresovaná zpráva:

Hello!

I am a student of Information security in CTU university in Prague. Currently I am working on my bachelor thesis: Criteria for the security evaluation of open-source cryptographic libraries. And I have chosen Botan library as one of my research directions.

My approach to this research is not to look into the code of library, but look up for open-source applications, which are using the library and look into the issues or CVEs, that are somehow related to the usage of library.

Since I found you as the main maintainer of Botan repository in GitHub, I would really appreciate, if you will answer some of my questions: What was the main goal, when creating the Botan library?

Do you have any special requirements for contributing in Botan repository? As I see anyone can open the pull request and if it will be reviewed, for example, by you, it could be merged.

When I was researching the applications, that use Botan, I have noticed the tendence, that you are helping developers to use Botan correctly. So any developer can turn to you, when he wants to add Botan's support, am I right?

Did you noticed any problems, that the developers have faced to, when using Botan? If so, can you name them.

Your answer will be very helpful for me, I am looking forward to it.

Regards,

Leonov Kirill

Odpověď:

Hi Leonov,

> What was the main goal, when creating the Botan library?

I think https://github.com/randombit/botan/blob/master/doc/goals.rst captures it pretty well, but as a one liner, I wanted a reliable library for building distributed systems. [*]

[*] You must keep in mind the first code written in Botan was circa 1998 with first release in 2001. At the time the only options available were cryptlib, Crypto++, and SSLeay (now known as OpenSSL). All of which did (and still do, IMO) have very significant usability issues.

> Do you have any special requirements for contributing in Botan repository? As I see anyone can open the pull request and if it will be reviewed, for example, by you, it could be merged.

Right, there is no CLA or anything of that sort and we'll accept contributions from anyone as long as the change is in concord with the project goals.

> When I was researching the applications, that use Botan, I have noticed the tendence, that you are helping developers to use Botan correctly. So any developer can turn to you, when he wants to add Botan's support, am I right?

Right, at least to the extent my time permits. Since one goal is that the library being functional and easy to use, something being unclear is kind of intrinsically a bug: an API is non-obvious, something is poorly documented, etc. By helping the user the nature of the underlying issue can become more clear.

> Did you noticed any problems, that the developers have faced to, when using Botan? If so, can you name them. One big issue is that Botan (like almost all general purpose crypto libraries) is not particularly safe to use by non-experts. Being safe in this way (as say Google's Tink attempts to be) limits the API to where it is not practical to build general purpose systems, which was considered a more important goal by me since I started writing the library specifically to build the distributed systems I was interested in writing.

Another one is that some parts of the API are quite confusing due to historical design decisions.

You may also find `https://github.com/randombit/botan/blob/master/doc/dev_ref/mistakes.rst` interesting for your research.

Best,

Jack Lloyd

Bibliografie

1. KHAN, Rafiq Ahmad; KHAN, Siffat Ullah; ILYAS, Muhammad. Exploring Security Procedures in Secure Software Engineering: A Systematic Mapping Study. In: *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*. Gothenburg, Sweden: Association for Computing Machinery, 2022, s. 433–439. EASE '22. ISBN 9781450396134. Dostupné z DOI: 10.1145/3530019.3531336.
2. HAM, Jeroen Van Der. Toward a Better Understanding of "Cybersecurity". *Digital Threats*. 2021, roč. 2, č. 3. Dostupné z DOI: 10.1145/3442445.
3. OPEN SOURCE INITIATIVE. *The Open Source Definition* [online]. 2006. [cit. 2024-04-26]. Dostupné z: https://opensource.org/osd.
4. RAYMOND, Eric S.; O'REILLY, Tim. *The Cathedral and the Bazaar*. 1st. USA: O'Reilly & Associates, Inc., 1999. ISBN 1565927249.
5. FAVATO, Danilo; ISHITANI, Daniel; OLIVEIRA, Johnatan; FIGUEIREDO, Eduardo. Linus's Law: More Eyes Fewer Flaws in Open Source Projects. In: *Proceedings of the XVIII Brazilian Symposium on Software Quality*. Fortaleza, Brazil: Association for Computing Machinery, 2019, s. 69–78. SBQS '19. ISBN 9781450372824. Dostupné z DOI: 10.1145/3364641.3364650.
6. HOWARD, Michael; LIPNER, Steve. *The Security Development Lifecycle*. USA: Microsoft Press, 2006. ISBN 0735622140.
7. MICROSOFT. *Microsoft Security Development Lifecycle (SDL)* [online]. 2023. [cit. 2024-04-26]. Dostupné z: https://learn.microsoft.com/en-us/compliance/assurance/assurance-microsoft-security-development-lifecycle.
8. HOWARD, Michael. *A Look Inside the Security Development Lifecycle at Microsoft* [online]. 2019. [cit. 2024-04-26]. Dostupné z: https://learn.microsoft.com/en-us/archive/msdn-magazine/2005/november/a-look-inside-the-security-development-lifecycle-at-microsoft.
9. SALTZER, J.H.; SCHROEDER, M.D. The protection of information in computer systems. *Proceedings of the IEEE*. 1975, roč. 63, č. 9, s. 1278–1308. Dostupné z DOI: 10.1109/PROC.1975.9939.
10. CERT SECURE CODING TEAM. *SEI CERT C++ Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems*. 2016. Dostupné také z: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=494932.
11. LAZAR, David; CHEN, Haogang; WANG, Xi; ZELDOVICH, Nickolai. Why does cryptographic software fail? a case study and open problems. In: *Proceedings of 5th Asia-Pacific Workshop on Systems*. Beijing, China: Association for Computing Machinery, 2014. APSys '14. ISBN 9781450330244. Dostupné z DOI: 10.1145/2637166.2637237.

12. OPEN SOURCE SECURITY FOUNDATION BEST PRACTICES WORKING GROUP. *Concise Guide for Evaluating Open Source Software* [online]. 2023. [cit. 2024-04-26]. Dostupné z: https://best.openssf.org/Concise-Guide-for-Evaluating-Open-Source-Software.

13. 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*. New York, NY, USA: Association for Computing Machinery, 2023, s. 102–111. ACSW '23. ISBN 9798400700057. Dostupné z DOI: 10.1145/3579375.3579388.

14. 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 z DOI: 10.1109/MSP.2016.111.

15. GRABOVSKÝ, Matěj. Measuring the Usability of Cryptographic Libraries. 2018. Dostupné také z: https://is.muni.cz/th/m5uy0/thesis_Archive.pdf.

16. BERNSTEIN, Daniel J.; TANJA, Lange; SCHWABE, Peter. The security impact of a new cryptographic library. 2012. Dostupné také z: https://cr.yp.to/highspeed/coolnacl-20120725.pdf.

17. LIBSODIUM TEAM. *Libsodium documentation* [online]. 2023. [cit. 2024-05-08]. Dostupné z: https://doc.libsodium.org.

18. PIA TEAM. *Libsodium v1.0.12 and v1.0.13 Security Assessment* [online]. Private Internet Access, 2017 [cit. 2024-04-26]. Dostupné z: https://www.privateinternetaccess.com/blog/libsodium-v1-0-12-and-v1-0-13-security-assessment/.

19. BOTAN TEAM. *Botan documentation* [online]. 2024. [cit. 2024-05-08]. Dostupné z: https://botan.randombit.net/handbook/.

20. 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*. Seoul, Republic of Korea: Association for Computing Machinery, 2020, s. 409–419. MSR '20. ISBN 9781450375177. Dostupné z DOI: 10.1145/3379597.3387465.

21. BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK. Sichere Implementierung einer allgemeinen Kryptobibliothek [online]. 2017 [cit. 2024-05-08]. Dostupné z: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Projektzusammenfassung_Botan.pdf?__blob=publicationFile&v=1.