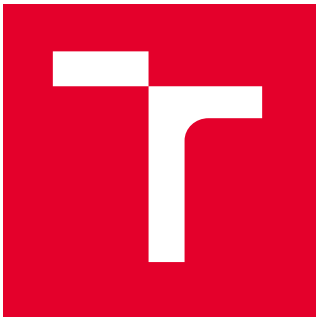


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A IMPLEMENTACE DIGITÁLNĚ PODEPSANÉHO REPORTU V PROGRAMU APACHE JMETER

DESIGN AND IMPLEMENTATION OF A DIGITALLY SIGNED REPORT IN THE APACHE JMETER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Martin Procházka**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. Ing. Pavel Šeda, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Martin Procházka ***ID:*** 240970

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Návrh a implementace digitálně podepsaného reportu v programu Apache JMeter

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem bakalářské práce je návrh a implementace modulu pro generování reportu z testování v programu Apache JMeter. Student se seznámí se softwarem pro zátěžové testování Apache JMeter a již existujícími moduly rozšiřující možnosti testování. V teoretické části práce analyzujte možnosti programu Apache JMeter, generování reportů ve formátu .html a .pdf a dále alternativy k digitálnímu podpisu. Vytvořené řešení bude rozšiřovat existující moduly o možnost výběru, z kterých testů se má vygenerovat report a generování reportu, který bude digitálně podepsán. Funkčnost aplikace důkladně otestujte na vhodných scénářích.

DOPORUČENÁ LITERATURA:

[1] RODRIGUES, Antonio Gomes, Bruno DEMION a Philippe MOUAWAD. Master Apache JMeter - From Load Testing to DevOps: Master performance testing with JMeter. Packt Publishing Ltd, 2019.
[2] ATAR, Afsana. Mastering JMeter 5.0. Packt Publishing, 2020.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 28.5.2024

Vedoucí práce: RNDr. Ing. Pavel Šeda, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zabývá vývojem zásuvného modulu do nástroje Apache JMeter pro automatické generování záznamů z výsledků zátěžového testování poskytnuté ostatními moduly. V teoretické části se věnuje možnostem nástroje Apache JMeter, zátěžovému testování obecně, struktuře PDF dokumentu, teorii digitálního podpisu a k tomu relevantním kryptografických náležitostem. Praktická část se zabývá implementací generátoru reportů. Nejprve přibližuje obsah reportu a následně se věnuje principům generování HTML a PDF reportu. Nakonec se práce zabývá vložením digitálního podpis do PDF dokumentu.

KLÍČOVÁ SLOVA

Apache JMeter, zátěžové testování, DDoS, digitální podpis, digitální certifikát

ABSTRACT

This thesis focuses on the development of a plugin for Apache JMeter to automatically generate report from the results of performance testing provided by other modules. In the theoretical part, it discusses the capabilities of Apache JMeter, stress testing in general, the structure of PDF document, the theory of digital signature and relevant cryptographic requirements. The practical part deals with the implementation of the report generator. First, it introduces the content of the report, then it discusses the principles of HTML and PDF report generation. Finally, the paper discusses the insertion of a digital signature into a PDF document.

KEYWORDS

Apache JMeter, performance testing, DDoS, digital signature, digital certificate

PROCHÁZKA, Martin. *Návrh a implementace digitálně podepsaného reportu v programu Apache JMeter*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: RNDr. Ing. Pavel Šeda, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Martin Procházka
VUT ID autora:	240970
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Návrh a implementace digitálně podepsaného reportu v programu Apache JMeter

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu RNDr. Ing. Pavlu Šedovi, Ph.D za odborné vedení, trpělivost, podnětné rady a časté konzultace. Dále bych rád poděkoval Ing. Jakubu Jedličkovi za odbornou pomoc v praktické části práce. V neposlední řadě patří poděkování rodině za podporu při studiu a tvorbě této práce.

Obsah

Úvod **11**

1 Testování pomocí nástroje Apache JMeter **13**
1.1 Nástroj Apache JMeter . . . 13
1.2 Možnosti testování nástrojem Apache JMeter . . . 13
1.3 Výkonnostní testování . . . 14
1.4 Struktura testu v Apache JMeter . . . 15
1.5 Existující moduly . . . 17

2 Teoretický rámec pro implementaci **21**
2.1 Značkovací jazyk HTML . . . 21
2.2 PDF dokument . . . 21
2.3 Programovací jazyk Java . . . 23
2.4 Tvorba přídavných modulů do nástroje Apache JMeter . . . 24
2.5 Elektronický podpis . . . 25
2.6 Jednosměrné (hešovací) funkce . . . 26
2.7 Asymetrická kryptografie . . . 26
2.8 Digitální certifikát . . . 26
2.9 Digitální certifikát standardu X.509 . . . 27
2.10 Digitální podpis . . . 28
2.11 Soubor PKCS#12 . . . 29
2.12 Knihovna OpenSSL . . . 30
2.13 Generování certifikátu a privátního klíče pomocí OpenSSL . . . 31
2.14 Java KeyStore . . . 32
2.15 Digitální podpis PDF souboru . . . 33

3 Implementace generátoru reportů **35**
3.1 Grafické uživatelské rozhraní modulu . . . 36
3.2 Veličiny obsažené v reportu . . . 36
3.3 Výběr testů pro generování reportu . . . 37
3.4 Generování HTML reportu . . . 38
3.4.1 HTML šablony . . . 39
3.5 Generování PDF reportu . . . 41

4 Implementace digitálního podpisu **42**
4.1 Výběr knihovny pro digitální podpis PDF . . . 42
4.2 Uživatelské rozhraní pro digitální podpis . . . 42
4.3 Inicializace privátního klíče a certifikátu do Java KeyStore . . . 44

4.4 Knihovna Bouncy Castle 44
4.5 Vložení digitálního podpisu do PDF dokumentu 46

5 Testování a oprava chyb v ICT testeru **50**
5.1 Zamrznutí GUI při generování Stairs thread group grafu 50
5.2 Chybějící čas začátku a konce testu v reportu 50
5.3 Implementace reportu z útoků z více zdrojových adres 51
5.3.1 Úprava Slow HTTP modulů 51
5.3.2 Úprava zpracování reportu 52
5.4 Rozšíření Reachability testeru o ICMP protokol 53
5.5 Změna zdroje reachability hodnot u reportu ze Slow HTTP testů . . 53

Závěr **56**

Literatura **57**

Seznam symbolů a zkratek **62**

Seznam příloh **64**

A Úryvky kódu implementace digitálního podpisu **65**
A.1 Metoda SignOperation.sign() 65

B Ukázkový report ze zátěžového testování **67**

C Obsah elektronické přílohy **73**
C.1 Sestavení modulu 73
C.2 Instalace modulu do Apache JMeter 73

Seznam obrázků

3.1 Diagram průběhu testování a generování reportu 35
3.2 Tabulka pro výběr testů 37
4.1 Grafické uživatelské rozhraní pro nastavení digitálního podpisu 43
5.1 Graf ve Stairs Thread Group modulu 51

Seznam výpisů

2.1 Generování soukromého klíče a certifikátu 31
2.2 Export certifikátu a privátního klíče do PKCS#12 souboru 32
3.1 Mapování XML souboru na Java třídu TestResults 38
3.2 Příklad proměnné obsahující substituční řetězec 40
3.3 Metoda pro nahrazování substitučních proměnných daty 41
4.1 Implementace metody loadKeystore ve třídě SignOperation 45
4.2 Implementace metody getSignature ve třídě SignOperation 47
5.1 Implementace ICMP protokolu v Reachability testeru 54

Úvod

V dnešní době se stále zvyšuje počet služeb připojených k Internetu, bez kterých se v běžném životě neobejdeme a je neustále potřebné je mít dostupné. Současně s tímto trendem ale narůstá počet útoků, které se snaží tyto služby znepřístupnit, jedná se například o tzv. *Denial of Service* (DoS) útoky. Při vývoji softwaru, webové aplikace, je nutností na tuto hrozbu myslet. To by ovšem bylo obtížné bez dostupného nástroje, který by tento typ útoků dokázal simulovat a odolnost dané služby otestovat. K tomuto účelu slouží nástroj Apache JMeter [1] nabízející spoustu možností zátěžového testování podporující rozšiřitelnost pomocí zásuvných modulů (pluginů). Na Fakultě elektrotechniky a komunikačních technologií Vysokého učení technického v Brně probíhá vývoj rozšíření nástroje Apache JMeter o další možnosti testování, například modul pro zátěžové testy typu *Distributed Denial-of-Service* (DDoS).

Tato práce se zabývá vytvořením modulu, který umožňuje vytvořit přehledný report o průběhu jednotlivých testů. Výsledný modul zpracovává výstupní soubory z ostatních modulů, sloužících pro samotné vykonávání testů, které následně přetransformuje do *Hypertext Markup Language* (HTML) [2] stránky a v posledním kroku do *Portable Document Format* (PDF) souboru. Modul umožňuje uživateli vložit vlastní digitální certifikát a soukromý klíč pro digitální podepsání výsledného PDF reportu.

První část práce se zabývá představením nástroje Apache JMeter, jednotlivých typů zátěžového testování, způsoby a možnostmi sestavení testu v nástroji Apache JMeter, a také již existujícími moduly vyvíjenými v rámci projektu „Technologie pro testování kyberbezpečnosti ICT“ [3] na FEKT VUT v Brně (dále jen „ICT tester“).

Druhá část práce poskytuje teoretický rámec pro následný popis implementace. Zabývá se například strukturou PDF dokumentu, jakým způsobem se tvoří moduly do nástroje Apache JMeter, dále teorií elektronického, resp. digitálního podpisu a kryptografickými metodami a prvky, které podpis využívá. Poslední částí této kapitoly je teorie o vkládání digitálního podpisu přímo do PDF dokumentu.

Třetí část se věnuje samotné implementaci modulu generátoru. V první řadě se zabývá uživatelským rozhraním, dále veličinami, které musí report obsahovat, a nakonec popisem implementace generování HTML a PDF reportu.

Předposlední část práce má za úkol čtenáři objasnit implementaci digitálního podpisu PDF reportu. Zabývá se opět uživatelským rozhraním a co je v něm možné nastavit, dále náležitostmi jazyka Java pro nahrání digitálního certifikátu a nakonec vytvořením a vložením digitálního podpisu do PDF dokumentu.

Závěrečná kapitola se věnuje testování, opravě chyb a implementaci chybějících funkcí v rámci celého projektu ICT tester [3], který byl zpracováván na FEKT VUT

v Brně. Popisuje například rozšíření testeru dostupnosti oběti o protokol *Internet Control Message Protocol* (ICMP) nebo opravu zasekávání uživatelského rozhraní při generování grafu se schodovitým průběhem. Kapitola popisuje i rozšíření modulu zajišťujícího Slow *Hypertext Transfer Protocol* (HTTP) [4] testy o možnost spuštění testu z více zdrojových adres, avšak z pohledu úpravy modulu pro generování reportů.

1 Testování pomocí nástroje Apache JMeter

Tato kapitola slouží čtenáři k přiblížení nástroje Apache JMeter [1], jeho možností a způsobu testování. Dále obsahuje základní informace o zátěžovém testování a jeho dělení. Nakonec zběžně přiblíží již existující moduly rozšiřující JMeter o další možnosti zátěžového testování.

1.1 Nástroj Apache JMeter

Apache JMeter [1] je software sloužící pro výkonnostní testování. Aplikace je vyvíjená společností Apache Software Foundation a je poskytována jako software s otevřeným zdrojovým kódem (open-source). Je vyvíjená v programovacím jazyce Java. Nejdříve byl Apache JMeter koncipován pouze jako tester webových aplikací, později byl postupně rozšiřován o široké spektrum možností testování. V nejnovějších verzích je možné testovat webové stránky (aplikace) pomocí protokolů HTTP / *Hypertext Transfer Protocol Secured* (HTTPS) spolu s *Representational State Transfer* (REST) a *Simple Object Access Protocol* (SOAP) *Application Programming Interface* (API), dále také *File Transfer Protocol* (FTP) servery, výkonnost databází pomocí *Java Database Connectivity* (JDBC), přístup k adresářovému serveru pomocí *Lightweight Directory Access Protocol* (LDAP), poštovní služby fungující na protokolech *Post Office Protocol* (POP3), *Internet Message Access Protocol* (IMAP) a *Simple Mail Transfer Protocol* (SMTP) (a jejich zabezpečených variantách) nebo také v neposlední řadě výkonnost Java objektů a mnoho dalších [1].

1.2 Možnosti testování nástrojem Apache JMeter

Ačkoliv se tak může zdát, Apache JMeter není webový prohlížeč. Pracuje pouze na protokolových vrstvách a nepodporuje veškeré operace jako standardní internetový prohlížeč – v základu neumí vyobrazovat HTML stránky, ani načítat a spouštět JavaScriptový kód [1]. Tento nástroj proto není například vhodný pro testování grafických uživatelských rozhraní, jako například frameworky Selenium [5] nebo Cypress [6].

Z těchto důvodů se tento nástroj hodí spíše pro testování serverových (backend) částí aplikací. Základními typy testů, které je možné pomocí JMeteru provádět, se řadí [7]:

- testování výkonu – více v kapitole 1.3,

- testování API – typ testování, při kterém se volá API dané aplikace a ověřují se návratové hodnoty v odpovědi. Tento typ testování je možné spojit s výkonnostním testováním, při kterém se server zahltí mnoha požadavky a testuje se, zda-li odpovědi obsahují správné hodnoty i pod zátěží,
- testování bezpečnosti – automatické testování pro odhalování bezpečnostních chyb aplikace, např. nezabezpečených zdrojů, bezpečnostních zranitelností nebo tzv. „Side Spidering“[1].

Apache JMeter [1] také podporuje vytváření zásuvných modulů (pluginů), které jej mohou rozšiřovat o spoustu dalších funkcí. Příkladem může být rozšíření programu o další možnosti zátěžového testování nebo například vytvoření přizpůsobeného generátoru zpráv o průběhu testování. Tomuto tématu se více věnuje kapitola 2.4.

1.3 Výkonnostní testování

Výkonnostní testování softwaru je typ testování, při kterém se webová aplikace (nebo jakákoliv jiná služba) vystavuje určitému počtu požadavků a měří se její schopnost tyto dotazy zpracovávat. Sledovanými parametry mohou být čas odezvy, stabilita, škálovatelnost, spolehlivost a rychlost, případně míra spotřeby zdrojů, které aplikace pod zátěží využívá. Hlavní dělení výkonnostního testování je na **load testing**, **stress testing**, **capacity testing** a **soak testing** [9, 10, 11].

Load testing

Load testing je typ výkonnostního testování, při kterém je daná aplikace zatížená jako během normálních provozních podmínek, případně při špičkovém zatížení. Během tohoto typu testování se měří doba odezvy, propustnost a míra spotřeby zdrojů [10].

Stress testing

Stress testing je druh výkonnostního testování, při kterém je testovaná aplikace zatížená nad svoje běžné provozní limity. Při tomto testování se hledají bezpečnostní problémy, které se mohou objevit pouze při enormní zátěži. Může jít například o úniky dat z paměti, problémy se synchronizací a další slabé stránky. Dále se

[1] Typ testování, kdy se vytváří jakási mapa odhalováním viditelných zdrojů webové aplikace, které by mohly obsahovat citlivé nezabezpečené informace nebo by mohly být jinak využity ke kompromitování aplikace [8].

také může měřit a sledovat vliv aplikace na zatížení hardwaru a na jeho případné poškození [10].

Capacity testing

Capacity testing sleduje při jak velkém zatížení (počtu uživatelů) bude aplikace schopna zajišťovat službu a přitom stále splňovat definované výkonnostní podmínky (například odezva). Používá se především pro analýzu prostředků potřebných pro běh aplikace a plánování jejich rozšiřování (například potřeba rozšířit diskové úložiště, přidání operační paměti, atp.) [10].

Soak testing

Soak testing je podobný jako Capacity testing, avšak s tím rozdílem, že je tento test prováděn v delším časovém úseku. Zátěž aplikace se v čase konstantně zvyšuje, přičemž tento proces trvá od několika hodin až po několik dní. Během běhu tohoto testu se sleduje spotřeba operační paměti, využití a spotřeba diskového prostoru, a případně zda-li nedochází k neočekávanému chování aplikace [11].

1.4 Struktura testu v Apache JMeter

Strukturu testu v JMeteru tvoří jednotlivé komponenty, které ve výsledku tvoří stromovou strukturu. Každá komponenta má svou funkci a ovlivňuje výsledné chování vytvářeného testu [12]. Tato kapitola může být čtenáři užitečná pro pochopení struktury modulů vyvíjených v rámci projektu ICT tester [3] na FEKT VUT v Brně, viz kapitola 1.5. Nyní následuje seznam komponent se stručným popisem jejich účelu:

Test Plan

Test Plan je základním stavebním prvkem celého testu. Je kořenem v pomyslném stromu komponent a dělá z testu celek. Mimo jiné ve svém *Graphical User Interface* (GUI) umožňuje nastavit globální proměnné pro celý test (například adresa web serveru, apod...), které mohou následně být využity ke konfiguraci ostatních částí testu [12].

Thread Group

Komponenta, která představuje virtuální uživatele (vlákna), kteří budou vykonávat daný test a určovat zátěž posílanou proti cílovému serveru. Jedno vlákno se typicky rovná jednomu virtuálnímu uživateli. Základní Thread Group komponenta umožňuje

nastavit počet uživatelů (vláken), čas náběhu (tzn. v jakém časovém rozestupu bude přidáno další vlákno) a také počet opakování daného testu [12].

Sampler

Jedna z nejdůležitějších komponent pro samotný běh testu. Tato komponenta generuje provoz (requesty) pro veškeré testy a zpracovává případné odpovědi (responses). Výstupem jsou data ve formě SampleResult objektu, který obsahuje celou řadu atributů a informací o generovaném provozu. Tyto data mohou být dále zpracována Listener komponentami [12].

Logic Controller

Je prvek, který seskupuje jednotlivé Samplery a určuje jejich pořadí, v jakém je bude JMeter spouštět. V základu jsou dostupné různé typy kontrolerů, např: Simple Controller, If Controller (na základě určené podmínky), ForEach Controller (cyklus), Random Controller a další. Tato komponenta nemusí být povinně použita [12].

Listener

Je komponenta, která slouží k vizualizaci, případně ukládání (exportu) průběhu testu. Ve výchozích pluginech jsou výsledky nejčastěji ukládány ve formátu *Extensible Markup Language* (XML) do souboru s příponou „.jtl“. Je ovšem možnost výsledky ukládat také do formátu *Comma-separated values* (CSV), což je vzhledem k výkonosti lepší možnost. CSV soubor ale většinou nemůže obsahovat tolik detailních informací, jako XML [12].

Pokud Listenery slouží jako vizualizéry – tzn. zobrazují data z testu přímo v uživatelském rozhraní – mohou být velmi paměťově i výpočetně náročné, obzvlášť když je dat z testu mnoho. Proto může být výhodnější používat pouze Listenery, které ukládají data pouze do souborů a jsou vizualizována později [12].

Configuration element

Slouží jako konfigurační prvky pro Samplery. Jsou spouštěny na začátku úrovně testu, ve které se nacházejí. Tím pádem mohou před každou úrovní testu definovat určité proměnné a hodnoty, podle kterých se Sampler při běhu testu bude řídit. Příkladem je nastavení FTP připojení, nebo například manažer HTTP přihlášení [12].

Assertion

Je komponenta, která umožňuje provádět kontrolu jednotlivých vzorků ze Samplerů. V základu JMeter obsahuje několik druhů Assertation komponent, například Response Asseration, která kontroluje podle určitého klíče (regulérního výrazu), zda-li odpověď na požadavek obsahuje správné atributy (payload, kód odpovědi). V případě, že kontrola neprojde, je tento vzorek označen jako neúspěšný / nesprávný. Assertion komponenta je spuštěna po každém Sampleru ve stejné úrovni testu nebo po nadřazeném Sampleru [12].

Timer

Umožňuje pozastavit aktuální vlákno na určitý časový úsek. Může pomoci nasimulovat reálné chování uživatele, který nevykonává všechny akce v okamžité časové posloupnosti. Vlákno může být pozastaveno na konstantní čas (Constant Timer), náhodný čas (Uniform Random Timer) nebo může například sesynchronizovat více vláken – počká, než určitý počet vláken bude Timerem pozastaven, a poté je vypustí všechny najednou (Synchronizing Timer) [12].

Pre-Processor

Komponenta, která je spuštěná před spuštěním Sampleru, ke kterému náleží. Může například dynamicky získávat data potřebná k testu z databáze, nebo měnit parametry HTTP dotazu na základě hodnoty získané z předchozího dotazu [12].

Post-Processor

Je obdobná komponenta jako předcházející, ale s tím rozdílem, že je spouštěna po příslušném Sampleru a před Assertion komponentami. To znamená, že může získávat data například z HTTP odpovědí, která se následně uloží do proměnné a mohou být použita v dalších vláknech nebo Samplerech. Může také například obnovit stav databáze po každém vzorku do určitého výchozího stavu [12].

1.5 Existující moduly

Aktuální verze ICT testeru [3] pro Apache JMeter [1] obsahuje 7 knihoven, jejichž popis je níže. Každá z těchto knihoven má formu zásuvného modulu do JMeteru, přičemž každý obsahuje několik druhů komponent (viz předchozí podkapitola).

AdvancedThreadGroups

Tento modul obohacuje JMeter o 2 druhy Thread Group komponenty. Prvním je **Stairs Thread Group**, která přidává možnost postupně schodovitě zvyšovat zátěž podle předem navoleného nastavení – lze například navolit „zvyšuj zátěž každých 10 sekund o 5 paketů za vteřinu (pps), maximální zátěž bude 100 pps držená po dobu 60 sekund, a poté postupně zátěž snižuj tempem 5 pps každou vteřinu“.

Druhou Thread Group komponentou obsaženou v tomto modulu je **Complex Thread Group**, která umožňuje daleko komplexnější nastavení zvyšování a snižování zátěže. Umožňuje definovat tabulku, ve které každý nový řádek přidává určitou zátěž započatou s konkrétním zpožděním, daným náběhem a dobou trvání.

DDoS

Je nejobsáhlejší modul z celého ICT testeru. Obsahuje opět spoustu druhů Samplerů pro jednotlivé druhy DDoS útoků – například ICMP Flood, DNS Flood, Slowloris, SYN Flood, UDP Flood a další. Obsahuje také další komponenty Thread Group vytvořené přímo pro přidání výše zmíněných Samplerů. Generování síťového provozu (paketů) zajišťuje nástroj Trafgen [13].

NetAnalyzer

Je modul zajišťující měření provozu na síťových rozhraních a vytížení systémových prostředků během běhu zátěžových testů. Jeho uživatelské rozhraní umožňuje zobrazení aktuální zátěže zvoleného síťového rozhraní (přijaté a odeslané bity) a zobrazení grafu velikosti jednotlivých hodnot (zátěž procesoru, RAM a množství provozu na rozhraní) v závislosti na čase od začátku testu. V neposlední řadě generuje soubor ve formátu XML, který je možné následně vložit do modulu ReportGenerator pro vygenerování přehledného reportu. Umožňuje také jednoduchý export hodnot do formátu PDF.

NetEmulator

Modul sloužící k emulaci přenosových parametrů mezi programem JMeter a síťovým rozhraním, přes které je veden provoz. Funguje na principu vložení virtuálního mezilehlého prvku, který upravuje parametry sítě. Modul umožňuje upravovat následující parametry:

- propustnost,
- zpoždění,
- ztrátovost paketů,

- záměna pořadí paketů,
- duplikace paketů,
- poškození paketů.

V GUI se objevuje ihned po instalaci jako nesmazatelný prvek ve stromu testových komponent. Lze poté načíst aktuální aktivní síťová rozhraní, která dané zařízení obsahuje – následně je možné ke každému rozhraní přidat prvky „Line“ obsahující konfigurace výše zmíněných parametrů.

ServerEmulator

Je zásuvný modul, který funguje jako emulátor HTTP nebo FTP serveru. Slouží jako testovací cílový server, proti kterému je možné pouštět zátěžové testy. Po nainstalování je plugin dostupný jako Config komponenta (viz kapitola 1.4) a je možné ji přidat do Test Planu pouze jednou. V základní konfiguraci umožňuje ručně spustit nebo zastavit oba druhy serverů, případně zapnout jejich automatické spouštění se začátkem testu. HTTP server také umožňuje nastavení zabezpečeného *Secure Sockets Layer* (SSL) spojení, velikost HTML stránky a nastavení IPv4 nebo IPv6 adresy, na které bude server naslouchat.

WebGenerator

Je zásuvný modul, který umožňuje generovat webovou stránku s reportem ze statistik sbíraných během testu. Modul během testu automatický sbírá data a ukládá je do souboru, ze kterého následně generuje webovou stránku. Webová stránka je buď generována automaticky s koncem testu (dle předvolby), nebo po stisknutí tlačítka v GUI. Mezi sbírané statistiky patří:

- doba trvání testu,
- úspěšnost requestů,
- jméno vlákna,
- časové značky,
- kódy HTTP odpovědí,
- počet vzorků a chyb,
- latence,
- a další...

ReportGenerator

Je modul, který umožňuje vygenerovat report do souboru HTML. Vstupními daty je například soubor ve formátu XML z modulu Network analyzer. Tato práce se zabývá kompletním znovu vytvořením tohoto modulu. Původní implementace generuje

HTML stránku spolu s JavaScriptem, který zajišťoval generování grafů i navigaci na stránce. Tato implementace neumožňovala snadné rozšíření reportu o nové moduly, a také nebylo jednoduché tuto stránku přegenerovat na PDF soubor, na který se díky možnosti digitálního podepsání klade větší důraz. Původní implementace byla označena jako „deprecated“, ale v modulu byla ponechána.

2 Teoretický rámec pro implementaci

Tato kapitola slouží jako teoretický podklad k následujícím kapitolám o implementaci generátoru reportů ze zátěžového testování a jeho následnému digitálnímu podepsání. V první části zmiňuje základní informace o využitých nástrojích (jazyk HTML, Java, PDF dokument). Následně nastiňuje jakými pravidly se řídí tvorba přídavných modulů do programu Apache JMeter. V neposlední řadě se věnuje principům elektronického podpisu a jeho legislativnímu zakotvení, kryptografickým funkcím a principům potřebným k vytvoření digitálního podpisu. Závěr kapitoly se věnuje ukládání certifikátů, soukromých klíčů, jejich generování a vkládání digitálního podpisu do PDF dokumentu.

2.1 Značkovací jazyk HTML

HTML je značkovací jazyk, který slouží primárně pro vykreslování stránek ve webových prohlížečích. Struktura dokumentu se skládá ze značek (tagů), které mohou obsahovat různé atributy. Atributy specifikují a upravují chování značky při jeho vykreslování v prohlížeči. Značky mohou být buď strukturní nebo vizuální. Ty strukturní upravují logickou strukturu dokumentu – např. rozdělení textu na odstavce – kapitoly, přičemž jeho vzhled není změněn. Typickým příkladem strukturní značky je `<p>Odstavec</p>`. Vizuální značky naopak upravují vizuální podobu dokumentu, například velikost písma, tučnost, sklon písma, barvu nebo třeba font. Typickým příkladem vizuální značky je `<b>Tučný text</b>` [2].

Jak je zřetelné z příkladů v předchozím odstavci, značky se dělí na otevírací `<p>` a zavírací `</p>`. Právě vysázením otevírací a uzavírací značky vzniká jeden HTML element. Mezi oběma značkami je typicky text (hodnota), která má být tímto elementem upravena. Zároveň tento element nemusí mít přímo uzavírací značku, místo ní je element uzavřen přímo v otevírací značce použitím lomítka před koncovým znakem značky. Typickým příkladem je `<img src="..." />`. Celková struktura dokumentu je tvořena zanořováním jednotlivých HTML elementů. Soubory HTML jsou textové soubory využívající přípony „*.html“ [2].

2.2 PDF dokument

PDF je standard [14] souboru umožňující uživatelům spolehlivě a jednoduše číst, upravovat, vytvářet a sdílet dokumenty nezávisle na použitém zařízení a příslušné čtečce těchto souborů. S jeho specifikací a prvotním vývojem přišla firma Adobe Systems Incorporated na počátku roku 1993. Pokračovala v tom až do roku 2007 kdy byla představena norma ISO 32000-1, jenž specifikovala PDF ve verzi 1.7. Aktuální je

verze 2.0 upravována normou ISO 32000-2 [14]. Jeho struktura je založena na základě vylepšení programovacího jazyka PostScript, který byl taktéž vyvinut společností Adobe Systems Incorporated [14].

Poskytované funkce

Standard PDF poskytuje uživatelům spoustu užitečných funkcí, které splňují požadavky na moderní elektronické dokumenty. Některými z nich jsou [14]:

- nezávislost dokumentu na použitém zařízení nebo softwaru,
- spojování více obsahu do jednoho souboru - webových stránek, tabulkových programů, fotografií, grafiky a dalších,
- digitální podpisy zajišťující integritu a autenticitu dokumentu,
- přístupnost dokumentu osobám se zdravotním hendikepem,
- vytváření a vyplňování elektronických formulářů,
- a další...

PDF procesory

Aby bylo dokument možné přečíst, vytvářet nebo upravovat, je nutné mít příslušný software, který poskytuje funkcionalitu v souladu s normou ISO 32000-2 [14]. Obecně se těmto programům říká PDF procesory (PDF processors). Ty se pak dále dělí na PDF čtečky (PDF readers), které slouží ke čtení (resp. vizualizaci) dokumentů a extrakci dat z nich, a na PDF zapisovače (PDF writers), které, jak již z názvu vyplývá, dokumenty vytváří [14].

Struktura PDF dokumentu

Struktura PDF je uložena v binárním souboru s příponou „*.pdf“. Základní struktura dokumentu se dělí na 4 části [14]:

- **Objekty (Objects)** jsou základní stavební jednotkou PDF dokumentu. Mohou sestávat z několika datových typů, např.: *boolean, numeric, string, name, array, dictionary.*
- **Struktura souboru (File structure)** specifikuje, jakým způsobem jsou objekty ukládány, upravovány a jak k nim může být přistupováno, nezávisle na účelu nebo druhu objektu.
- **Struktura dokumentu (Document structure)** definuje, jakým způsobem jsou základní typy objektu použity pro vyjádření součástí PDF dokumentu (stránky, typ písma, ...).

- **Obsahové proudy (Content streams)** jsou objekty, které určují výslednou grafickou podobu jednotlivých komponent PDF dokumentu po jeho vyrenderování. To znamená, že dokument má oddělenou strukturu od vizuální podoby - obě skupiny jsou reprezentovány jinými typy objektů.

Objekty typu string (řetězec) se dále dělí na dva typy [14]:

- **Literal**, kde hodnota je reprezentovaná standardními znaky uzavřená mezi levou a pravou závorku. Tento typ může obsahovat jakékoliv znaky kromě závorek a zpětného lomítka – pro reprezentaci těchto tří znaků se musí použít speciální escape sekvence.
- **Hexadecimal**, kde je hodnota reprezentovaná ASCII znaky v šestnáctkové soustavě. Používá se výhradně pro přidání binárních dat do dokumentu (například digitální podpis). Oproti typu Literal, hodnota se zde uzavírá do ostrých závorek.

2.3 Programovací jazyk Java

Java je velmi rozšířený objektově orientovaný programovací jazyk. Jeho první verze vyšla v roce 1991 a z počátku byl vyvíjen firmou Sun Microsystems, kterou později odkoupila společnost Oracle. Jedná se o multiplatformní jazyk, tudíž jeden kód je možný spustit na jakémkoliv hardwaru nebo softwaru (koncept „write once, run anywhere“ [15]), který má nainstalovaný *Java Virtual Machine* (JVM) umožňující spouštět zkompilovaný bytový kód. Jeho syntaxe je založená na jazycích jako je C nebo C++ [16].

Java má velmi široké využití a patří mezi nejpoužívanější programovací jazyky na světě. Škála využití začíná u Internetu věcí (*Internet of Things* (IoT)), pokračuje velmi velkým využitím v mobilním operačním systému Android, dále také tvorbou desktopových aplikací a her pro všechny operační systémy, umožňuje práci s umělou inteligencí a v neposlední řadě možností vytvářet robustní webové aplikace [16].

Mezi hlavní výhody Javy patří její jednoduchost a hardwarová, resp. softwarová, nezávislost. Jelikož je Java striktně objektově orientovaný jazyk, každý prvek v ní je považován za objekt a může dotvářet a rozšiřovat strukturu jazyka. Java je také velmi dobře zabezpečený jazyk díky dvoufázovému vykonávání. To znamená, že kód je nejprve přeložen pomocí překladatele (kompilátoru) do bytového kódu, který je následně vykonáván (interpretován) v JVM. Jazyk také podporuje více-vláknový běh, což má využití ve velkém objemu aplikací, včetně zátěžového testování [16].

2.4 Tvorba přídavných modulů do nástroje Apache JMeter

Nástroj Apache JMeter [1] umožňuje tvorbu přídavných pluginů, které umožňují rozšířit základní sadu nástrojů o libovolné komponenty. Jelikož celý zdrojový kód aplikace JMeter je psán v jazyce Java, je nutné i tyto přídavné moduly vyvíjet v něm. K tomuto účelu má JMeter nachystané API, které tvorbu přídavných modulů zjednodušuje a sjednocuje.

JMeter poskytuje jako součást dokumentace stručný návod [17], jakým způsobem pluginy vytvářet. Dalším nezbytným zdrojem je JavaDoc dokumentace API nástroje JMeter [18], která poskytuje seznam všech dostupných tříd s popisem a seznamem proměnných a metod.

Základem pro implementaci jsou abstraktní třídy pro GUI jednotlivých komponent, ze které musí dědit třída, která bude obsluhovat vytvářený plugin. Příkladem je třída `AbstractVisualizer`, která se chová jako komponenta Listener nebo případně třída `AbstractSamplerGui`, která by sloužila jako Sampler. Každá z těchto abstraktních tříd obsahuje několik abstraktních metod, které musejí potomci implementovat. Jedná se například o metody [17, 18]:

- `public String getLabelResource()` – obsahující název proměnné, která je použita v konfiguračním souboru s překlady názvů komponent (může avšak vracet přímo daný název a konfigurační soubor nemusí být použit),
- `public String getStaticLabel()` – vracející název komponenty v aktuálním jazyce získaný z předchozí metody,
- `public void configure()` – zajišťující získání dat z TestElementu do GUI a jejich aktualizaci,
- `public void modifyTestElement()` – přesný opak metody configure(). Zajišťuje, aby data vložená do GUI správně nastavila parametry testu,
- `public TestElement createTestElement()` – metoda vytvářející novou instanci třídy TestElement, která zajišťuje jakési „pojítko“ mezi třídami obsluhujícími GUI a třídami zajišťující běh testu.

Nespornou výhodou těchto abstraktních tříd / metod je, že jejich volání si JMeter zajišťuje sám interně, tudíž vývojář vyvíjející plugin se již nemusí starat o to, kdy má jakou z těchto metod zavolat, aby byly ve správný čas přeneseny data mezi GUI a třídami obsluhujícími test samotný [17].

Mimo jiné musí také tyto třídy zajišťovat vykreslení příslušných GUI komponent. Apache JMeter vyžívá pro vykreslení uživatelského rozhraní Java framework Swing [19]. Metody zajišťující správné a požadované vykreslení GUI by se měly volat přímo z konstruktoru třídy, jelikož JMeter API pro tento účel neposkytuje žádné abstraktní metody z rodičovských tříd [17].

2.5 Elektronický podpis

Elektronický podpis jsou data, která jsou logicky připojená ke zprávě (souboru, dokumentu, emailu, ...) a zajišťují autenticitu, integritu a nepopiratelnost[1] dané zprávy. Jinými slovy, jsou to důkazy o tom, že danou zprávu vytvořila konkrétní osoba či subjekt a tato zpráva po jejím podepsání nebyla změněna. Formou elektronického podpisu může být například digitální podpis (viz kapitola 2.10) [23].

Jelikož elektronický podpis umožňuje spolehlivě ověřit pravost dokumentu, je v dnešní době již ve spoustě státech legislativně zakotven a je stavěn na úroveň vlastnoručního podpisu. V Evropské unii upravuje problematiku elektronického podpisu *nařízení Evropského Parlamentu a Rady č. 910/2014 o elektronické identifikaci a důvěryhodných službách pro elektronické transakce na vnitřním evropském trhu* (eIDAS) [27]. Toto nařízení umožňuje přeshraniční uznávání elektronických podpisů v rámci Evropské unie. Upravuje také požadavky na *zaručené elektronické podpisy* (čl. 26 nařízení eIDAS) [27]:

- je jednoznačně spojen s podepisující osobou,
- umožňuje identifikaci podepisující osoby,
- je vytvořen pomocí dat pro vytváření elektronických podpisů, která podepisující osoba může s vysokou úrovní důvěry použít pod svou výhradní kontrolou,
- je k datům, která jsou tímto podpisem podepsána, připojen takový způsobem, že je možné zjistit jakoukoliv následnou změnu dat.

Dále nařízení zmiňuje *kvalifikovaný elektronický podpis*, což je zaručený podpis, který je vytvořen kvalifikovaným prostředkem pro vytváření elektronických podpisů a který je založen na kvalifikovaném certifikátu pro elektronické podpisy (čl. 3, odst. 12 nařízení eIDAS). Kvalifikovaný elektronický podpis musí být dle tohoto nařízení stavěn na stejnou úroveň jako ručně psaný podpis. Nemůže být například soudem řečeno, že dokument podepsaný elektronickým podpisem nemá svou váhu, protože má elektronickou podobu (čl. 25 nařízení eIDAS) [27]. V Právním řádu České republiky implementuje a doplňuje některé části nařízení eIDAS zákon č. 297/2016 Sb., *Zákon o službách vytvářejících důvěru pro elektronické transakce* [28].

[1] Integrita - lze zjistit, jestli s dokumentem bylo nějakým způsobem po podpisu manipulováno, resp. jestli byl změněn; autenticita - lze ověřit, že dokument skutečně podepsal daný subjekt (díky digitálnímu certifikátu, viz kapitola 2.8); nepopiratelnost - autor podpisu nemůže tvrdit, že jej nevytvořil.

2.6 Jednosměrné (hešovací) funkce

Hešovací funkce jsou funkce, které pro libovolný vzor (dokument, zprávu) **x** vygenerují obraz (otisk) **y**, přičemž v ideálním případě není možné z otisku **y** vypočítat zpět původní zprávu **x**. Není avšak prokázané, že takové ideální funkce existují, pouze se spoléhá na to, že používané funkce tuto vlastnost mají. Hešovací funkce se dále dělí na:

- funkce s pevnou délkou výstupu,
- funkce s volitelnou délkou výstupu.

Jednosměrné funkce se nejčastěji používají v kryptografii v autentizačních kryptosystémech, například pro bezpečné uchovávání hesel [29]. Při registraci uživatele se vypočítá jeho otisk, ten se uloží do databáze a při následném ověřování hesla je vždy vypočítán tento otisk znovu a jsou porovnávány pouze otisky hesel. Tím je zajištěno, že nejsou nikde hesla uložena v čitelném formátu.

Dále se tyto funkce používají v algoritmech digitálního podpisu, přičemž se nejčastěji využívají funkce s pevnou délkou výstupu (například *Secure Hash Algorithm* (SHA256)) [29]. Více v kapitole 2.10.

2.7 Asymetrická kryptografie

Asymetrická kryptografie je typem kryptografie, kde se nevyužívá jeden tajný klíč (jako u symetrické kryptografie), ale pár klíčů. Jeden klíč se nazývá veřejný. Ten, jak již z názvu vyplývá, může být zveřejněn na Internetu a být veřejně dostupný. Používá se pro šifrování zpráv nebo pro ověření digitálního podpisu. Druhý klíč z páru se nazývá soukromý klíč, který musí zůstat v utajení u jeho majitele. Tento klíč slouží pro dešifrování zpráv nebo také pro vytvoření digitálního podpisu. Typickým zástupcem je algoritmus *Rivest-Shamir-Adleman* (RSA) [21] používaný a vhodný typicky pro digitální podpisy a šifrování, nebo také algoritmus Diffie-Hellman [22], což je algoritmus primárně využívaný k ustanovení klíčů symetrické kryptografie [23].

2.8 Digitální certifikát

Jelikož veřejný klíč neposkytuje sám o sobě důkaz o tom, že patří subjektu, který jej poskytl (může být podvržený), je potřeba jej určitým způsobem certifikovat. K tomuto účelu slouží digitální certifikáty. Jsou vydávané třetími stranami, tzv. certifikačními autoritami (CA), které se považují za důvěryhodný subjekt [23, 24].

Certifikáty jsou datové struktury, které obsahují osobní údaje o držiteli veřejného klíče, informace o certifikační autoritě, pořadové číslo certifikátu a veřejný klíč subjektu žádající o certifikaci. Certifikaci certifikátu provede CA tím způsobem, že

certifikát digitálně podepíše svým soukromým klíčem. Certifikační autority musejí mít také své certifikáty, které se následně používají při ověřování certifikátů. Při ověření musí příjemce nalézt ve svém úložišti důvěryhodných certifikátů certifikát dané CA, z něj extrahuje veřejný klíč CA, kterým následně ověří digitální podpis na certifikátu od veřejného klíče, který přišel s ověřovanou zprávou. Pokud je ověření certifikátu úspěšné, pak je jisté, že veřejný klíč není podvržený a digitální podpis na přijaté zprávě je pravý. Tím je zajištěna integrita, autenticita a nepopiratelnost podpisu. Podepisování certifikátu CA se používá pouze u certifikátů určené pro digitální podpisy [23].

Pro získání certifikátu je nutné odeslat tzv. žádost o certifikát. V případě podpisových certifikátů se jedná o tzv. *Certificate signing request* (CSR). Žádající subjekt si musí nejprve vygenerovat pár klíčů asymetrické kryptografie. Následně vytvoří žádost o podpisový certifikát (nejčastěji dle standardu PKCS#10 [25]). Ta obsahuje identifikační údaje subjektu, jeho veřejný klíč a v neposlední řadě digitální podpis celé této žádosti vygenerovaným soukromým klíčem. Žádost pošle certifikační autoritě, která tento podpis ověří. Pokud je ověření úspěšné, je zajištěno, že žadatel je skutečným vlastníkem soukromého klíče, ke kterému patří veřejný klíč, o jehož certifikaci žádá, jelikož jedině tento soukromý klíč mohl být k podpisu použit. To je jedna ze základních podmínek pro udělení certifikátu [23, 24].

Proč je nutné prokazovat vlastnictví soukromého klíče? Protože kryptografický pár klíčů si každý žadatel generuje sám a je určitá pravděpodobnost (i když velmi malá), že si 2 subjekty vygenerují stejné klíče. Certifikační autority musí udržovat databázi veřejných klíčů, ke kterým již vydala certifikát. Tím pádem, když přijde žádost o certifikaci veřejného klíče, který již byl certifikován, většina certifikačních autorit automaticky tuto žádost zamítá a následně také odvolává platnost již certifikovaného klíče. Když mají 2 subjekty stejné klíče, mohly by si následně podepisovat (falšovat) dokumenty navzájem. Z toho také plyne důvod, proč je nutné prokazovat ono vlastnictví soukromého klíče - veřejné klíče jsou veřejně známé a tím pádem by tuto žádost mohl podat kdokoliv a kdykoliv. Tzn., že pokaždé, kdy přijde tato žádost k CA, tak by automaticky odvolala platnost již vystaveného klíče a v případě, že je služba certifikování zpoplatněna, docházelo by k finančnímu poškozování subjektu, který skutečně vlastní soukromý klíč [23, 24].

2.9 Digitální certifikát standardu X.509

V Internetu nejpoužívanější variantou certifikátů je standard X.509 v jeho aktuálně nejaktuálnější verzi 3. Je definovaný standardem RFC 5280 [26] a jeho pozdějšími aktualizacemi. Má velmi široké využití napříč Internetem - od SSL/TLS, přes zabezpečenou a ověřenou komunikaci pomocí protokolu HTTPS, digitální podepisování

až po elektronické občanské průkazy.

Certifikát obsahuje spoustu položek, které umožňují identifikovat subjekt, jemuž patří, a také certifikační autoritu, která jej vydala [23]:

- verze certifikátu – verze standardu X.509 (1, 2 nebo 3),
- pořadové číslo certifikátu – kladné celé číslo, které umožňuje jednoznačně identifikovat certifikát v rámci certifikační autority,
- algoritmus podpisu – musí vždy obsahovat 2 algoritmy použité pro vytvoření digitálního podpisu certifikátu, jeden pro vytvoření otisku (např. SHA256), druhý pro šifrování otisku (např. RSA),
- platnost – vymezuje začátek a konec platnosti certifikátu,
- vydavatel – vydavatel certifikátu, jímž je certifikační autorita,
- předmět – informace o držiteli certifikátu,
- veřejný klíč – obsah informace o algoritmu ke kterému je veřejný klíč vytvořen a veřejný klíč samotný.

Dále je možné, kromě základních položek, přidávat k certifikátu také rozšíření. Může se jednat například o položky jako je platnost soukromého klíče, použití klíče (pouze pro podpis, pouze pro šifrování, apod.), alternativní jméno předmětu (například doménové jména) a další.

2.10 Digitální podpis

Digitální podpis je formou elektronického podpisu. Je to kryptografická matematická operace, která zajišťuje pravost podepisované zprávy, dokumentu nebo softwaru. Digitální podpis funguje na základě asymetrické kryptografie s využitím jednosměrných funkcích (otisk, hash). Asymetrická kryptografie využívá páru soukromého klíče (private key) a veřejného klíče (public key). Autor podpisu využívá právě soukromého klíče k vytvoření podpisu. Tento soukromý klíč musí zůstat v utajení. Ověření pravosti digitálního podpisu je poté možné pomocí veřejného klíče, který je veřejně dostupný. To znamená, že všichni jsou schopni pomocí veřejného klíče podpis ověřit, ale pouze autor (vlastník privátního klíče) je schopen podpis vytvořit [20, 23].

V současné době se pro digitální podpis nejčastěji používá algoritmus RSA. Jméno je odvozeno od příjmení tří vynálezců této šifry (Rivest-Shamir-Adleman). Řadí se mezi asymetrické kryptografické algoritmy typu IF (Integer Factorization). Tyto algoritmy fungují na principu obtížnosti rozložit velké číslo na součin prvočísel. Rozložení není nemožné, ale je časově a výpočetně velmi náročné. Bezpečnost spočívá hlavně ve zvolení dostatečně velkého čísla $\mathbf{n}$, jež je součinem prvočísel $\mathbf{p}$ a $\mathbf{q}$. Pak je velmi náročně, ne-li nemožné, tento součin rozložit [29]. Podle doporučení institutu *National Institute of Standards and Technology* (NIST) z roku 2019 by se

měly pro digitální podpisy algoritmem RSA používat kryptografické klíče s minimální délkou 2048 bitů [30].

Postup vytvoření digitálního podpisu

Předpokládejme, že podepisující má již existující pár klíčů asymetrické kryptografie. Vytvoření digitálního podpisu sestává z několika kroků:

1. Vytvoření otisku dokumentu.
2. Podepsání (zašifrování) otisku soukromým klíčem odesílatele.
3. Vzniklý podpis se přiloží k podepisovanému dokumentu.

Následně je dokument i s podpisem připraven k odeslání příjemci. Příjemci se také musí odeslat i veřejný klíč (lépe digitální certifikát), aby byl schopen podpis ověřit [23].

Postup ověření digitálního podpisu

Příjemce musí učinit několik kroků, aby mohl digitální podpis ověřit:

1. Vytvoří otisk přijatého dokumentu.
2. Dešifruje přijatý podpis.
3. Ověří, zda-li se vytvořený a dešifrovaný otisk rovnají.

Pokud se oba otisky rovnají, znamená to, že digitální podpis je validní, jelikož pouze vlastník soukromého klíče mohl tento podpis vytvořit [23]. Avšak pokud veřejný klíč není doručen spolu s digitálním certifikátem a nebyl doručen důvěryhodnou cestou, takto ověřený podpis může být během přenosu kompromitovaný. Stačí, aby útočník podstrčil příjemci svůj veřejný klíč a byl schopen odposlouchávat a upravovat mezilehlou komunikaci. Poté již může přenášený dokument libovolně upravovat a podepisovat svým soukromým klíčem, přičemž příjemce nemusí nic poznat. Zabránit tomuto typu útoku se dá právě digitálním certifikátem (kapitola 2.8), který umožňuje ověřit a zaručit identitu vlastníka veřejného klíče.

2.11 Soubor PKCS#12

Soubor typu PKCS#12 je kryptografický standard pro uchovávání digitálních certifikátů a privátních klíčů. Je to rozšířený standard, který se užívá ve spoustě operačních systémů a webových prohlížečů. Je také známý pod jménem key store (přeloženo jako úložiště klíčů). Soubor má jasně definovanou strukturu. Struktura je tvořená kontejnery různých datových typů. Hlavní kontejner má datový typ AuthenticatedSafe a umožňuje dva režimy ověřování (authentication modes), respektive režimy ochrany integrity:

1. *Ochrana integrity pomocí veřejného klíče (public-key integrity mode)* – ochrana pomocí digitálního podpisu, podpis je vytvořen pomocí soukromého klíče odesílatele a je ověřen příjemcem pomocí jeho veřejného klíče.
2. *Ochrana integrity pomocí hesla (password integrity mode)* – integrita je zajišťována pomocí *Message Authentication Code* (MAC). Ten je odvozen z hesla, které si uživatel zvolil.

AuthenticatedSafe kontejner dále může obsahovat sekvenci ContentInfo kontejnerů. Ty umožňují další dva možné režimy, tentokrát režimy ochrany tajemství, respektive klíčů a certifikátů v nich uložených:

1. *Ochrana tajemství pomocí veřejného klíče (public-key privacy mode)* – tajemství (obsah PKCS#12 souboru) je zašifrováno pomocí veřejného klíče u odesílatele a u příjemce může být rozšifrováno pomocí soukromého klíče.
2. *Ochrana tajemství pomocí hesla (password privacy mode)* – obsah souboru je zašifrován pomocí symetrického klíče, který sestává z hesla a uživatelského jména.

ContentInfo kontejnery tedy poskytují ochranu před nežádoucím odcizením, případně odposlechnutím soukromého klíče. Tyto kontejnery dále mohou obsahovat sekvence kontejnerů datového typu SafeContents. V těch jsou pak uloženy privátní klíče, certifikáty nebo *Certificate revocation lists* (CRLs) opět v dalších sekvencích datového typu SafeBag. Tyto kontejnery mohou obsahovat atribut *friendlyName*, který slouží ke snazší identifikaci dané kryptografické entity, obzvlášť v případě, kdy soubor obsahuje více privátních klíčů a certifikátů, což jeho struktura umožňuje. Tento *friendlyName* atribut se také v některých nástrojích nazývá *alias*. V grafických rozhraních umožňuje pro uživatele daleko snazší výběr páru certifikát-privátní klíč, který chce použít například pro digitální podpis. Načítání a výběru asymetrického páru ze souboru PKCS#12 se věnuje kapitola s implementací v programovacím jazyce Java, viz kapitola 4.3 [24, 35].

Vytvářet PKCS#12 soubory je možné programem OpenSSL, více v kapitole 2.12.

2.12 Knihovna OpenSSL

OpenSSL [31] je kryptografická knihovna vybavená širokou škálou nástrojů pro vytváření a manipulaci s kryptografickými prvky. Podporuje například:

- generování privátních a veřejných klíčů,
- vytváření X.509 certifikátů,
- šifrování a dešifrování,
- výpočet otisků zpráv,
- a další.

Je dostupná pod svobodnou licencí Apache 2.0 [33], která umožňuje komerční i nekomerční užití knihovny při dodržení licenčních podmínek. Knihovna je vhodná pro generování vlastnoručně podepsaných certifikátů a privátních klíčů, které lze použít pro digitální podepsání generovaného reportu.

2.13 Generování certifikátu a privátního klíče pomocí OpenSSL

Pokud uživatel nemá k dispozici digitální certifikát podepsaný certifikační autoritou, je možné vygenerovaný PDF report podepsat pouze s pomocí tzv. self-signed certifikátu. Jedná se v podstatě o certifikát podepsaný svým držitelem (položky „Předmět“ a „Vydavatel“ certifikátu dle standardu X.509 jsou stejné). Takový certifikát má tu nevýhodu, že v případě ověřování podpisu na zařízení, které nemá tento certifikát nainstalovaný mezi důvěryhodnými, bude software ověřující podpis hlásit problém s ověřením certifikátu. Pokud víme, že tento certifikát je opravdu důvěryhodný, pravý a byl doručen důvěryhodnou cestou, lze jej v operačním systému nainstalovat mezi důvěryhodné certifikáty.

Self-signed certifikát je možný vygenerovat následujícím způsobem. Nejdříve je potřeba vygenerovat privátní klíč a certifikát. Využijeme pro to nástroj ***req***, který je součástí OpenSSL:

Výpis 2.1: Generování soukromého klíče a certifikátu

```
1 openssl req -x509 -newkey rsa:4096 -keyout privateKey.pem -out
  ↪  certificate.pem -days 365 -nodes
```

Parametr *-newkey* určuje algoritmus pro vytvoření klíčů a jejich bitovou délku. V našem případě to je algoritmus RSA s délkou 4096 bitů. Dále parametry *-keyout* a *-out* určují jména výstupních souborů pro privátní klíč a certifikát. Parametrem *-days* se nastavuje doba platnosti certifikátu ve dnech. Jako poslední přepínačem *-nodes* určíme, že nechceme v tomto kroku privátní klíč zašifrovat žádným heslem. Po vložení příkazu do terminálu bude uživatel postupně vyzván, aby zadal osobní údaje o držiteli certifikátu. Tyto údaje se následně objeví v informacích o certifikátu v aplikaci na ověřování digitálního podpisu [32].

Nyní je potřeba vygenerovat PKCS#12 soubor (viz kapitola 2.11). To opět provedeme pomocí OpenSSL knihovny, tentokrát s využitím nástroje ***pkcs12***. Celý příkaz lze nalézt ve výpisu 2.2.

Výpis 2.2: Export certifikátu a privátního klíče do PKCS#12 souboru

```
openssl pkcs12 -export -out keyStore.p12 -inkey privateKey.pem -in
↪ certificate.pem
```

Přepínač *-export* říká knihovně, že chceme soubor PKCS#12 generovat a ne parsovat, jelikož nástroj umožňuje načíst existující soubor a přidat do něj další certifikáty a privátní klíče. Parametr *-out* definuje jméno výstupního souboru a jeho příponu (.p12). Dále pomocí přepínače *-inkey* definujeme vstupní soubor privátního klíče, který jsme vygenerovali pomocí příkazu ve výpisu 2.1. To samé se týká certifikátu, akorát je vstupní soubor definovaný přepínače *-in* [34]. Po zadání příkazu bude uživatel vyzván na zadání hesla, které zašifruje privátní klíč v tomto úložišti klíčů. Heslo bude potřeba pokaždé, kdy bude uživatel potřebovat k privátnímu klíči přistoupit.

Vygenerovaný soubor PKCS#12 lze následně použít pro vygenerování digitálně podepsaného reportu z modulu Report generátor. Stačí jej vložit do příslušného pole v grafickém uživatelském rozhraní, viz kapitola 4.2.

2.14 Java KeyStore

Keystore je nástroj pro uchovávání a práci se soukromými, veřejnými klíči, certifikáty a dalšími tajemstvími. V obecném měřítku je možné říci, že i soubor dle standardu PKCS#12 (viz kapitola 2.11) je keystore [37].

Aby bylo možné (a i jednoduché) pracovat s těmito úložišti klíčů v Javě, obsahuje základní balíček *java.security* třídu KeyStore, která poskytuje API pro jednoduché načítání, čtení a modifikaci obsahu keystore souborů. Tato třída obsahuje několik typů údajů, které všechny implementují rozhraní `KeyStore.Entry` – jsou to [36, 37]:

- `KeyStore.PrivateKeyEntry`,
- `KeyStore.SecretKeyEntry`,
- `KeyStore.TrustedCertificateEntry`.

Každý tento údaj je identifikovaný svým aliasem (jakým způsobem je alias implementován v souboru PKCS#12 zmiňuje kapitola 2.11) a definuje typ údaje (klíč, certifikát, ...), který může reprezentovat. Také je třeba myslet na skutečnost, že keystore může být zabezpečen heslem, které je nutné vložit při inicializaci Java objektu. Poté je již jednoduché pomocí dalších metod extrahovat potřebný soukromý klíč a certifikát pro využití při tvorbě například digitálního podpisu [36]. Více v kapitole 4.3.

2.15 Digitální podpis PDF souboru

PDF procesory podporují digitální podpisy dokumentů dle své specifikace v normě ISO 32000-2:2020 [14]. Výhodou onoho standardizování je, že vytváření a ověřování podpisů není závislé na konkrétním PDF prohlížeči a tím pádem může být nezávislé na použité platformě (ostatně jako celé PDF). K podpisu v PDF souboru je pouze nutné uložit informaci, jaký ovladač podpisu („signature handler“) byl pro podepsání použit a následně by měl být použitý k ověření. Příkladem takového ovladače je *Adobe.PPKLite*. Kromě ovladače je také nutné definovat, jakým způsobem (resp. standardem) bude podpis uložen. Velmi častým způsobem ukládání certifikovaných podpisů je standard PKCS#7 [38], který PDF dokumenty podporují. Následně je nutné k dokumentu přiložit sadu povinných a volitelných informací, například certifikát podpisu, data podpisu, jméno podepisujícího a další [14].

Data ve PDF souborech jsou často ukládána a reprezentována tzv. slovníky (dictionaries), což jsou objekty obsahující páry klíč-hodnota. Pro digitální podpis je definován speciální slovník (signature dictionary), který obsahuje data nutná ke správnému zpracování a validování podpisu. Slovníky obsahují několik povinných parametrů [14]:

- **Filter** – jméno ovladače podpisu (signature handler), který byl zmíněn v předchozím odstavci,
- **Contents** – data digitálního podpisu. V případě podpisů za využití veřejného klíče toto pole obsahuje podpis ve tvaru standardu; PKCS#7 [38],
- **TransformMethod** – jméno transformační metody, která slouží k pozdějšímu validování dokumentu. Standardně po vytvoření digitálního podpisu není možné dokument změnit, jelikož by byl podpis nevalidní (viz kapitola 2.10). Právě tyto metody ale umožňují definovat, které části dokumentu mohou po podepsání být změněny při zachování validity podpisu. Toto pole může obsahovat názvy metod *DocMDP*, *UR* (od PDF 2.0 již nepoužívané) a *FieldMDP*,
- **ByteRange** – Rozsah bajtů obsahu dokumentu, které se mají využít pro výpočet podpisu. Správně by měl tento rozsah pokrývat celý obsah dokumentu, včetně podpisového slovníku, ale bez hodnoty podpisu (tzn. bez hodnoty pole **Contents**). Problémem je, že velikost podpisu je těžko předvídatelná, proto musí být velikost pole **Contents** předem alokovaná na dostačující velikost a v případě, že bude podpis zabírat méně bajtů, musí být tato velikost dorovnána přidáním nul za data podpisu (tato poznámka je klíčová při implementaci digitálního podpisu v jazyce Java, viz kapitola 4.5).

Dále mohou podpisové slovníky obsahovat volitelné parametry, případně podmíněně volitelné, například [14]:

- **SubFilter** (volitelný) – standard, ve kterém je digitální podpis uložen, respek-

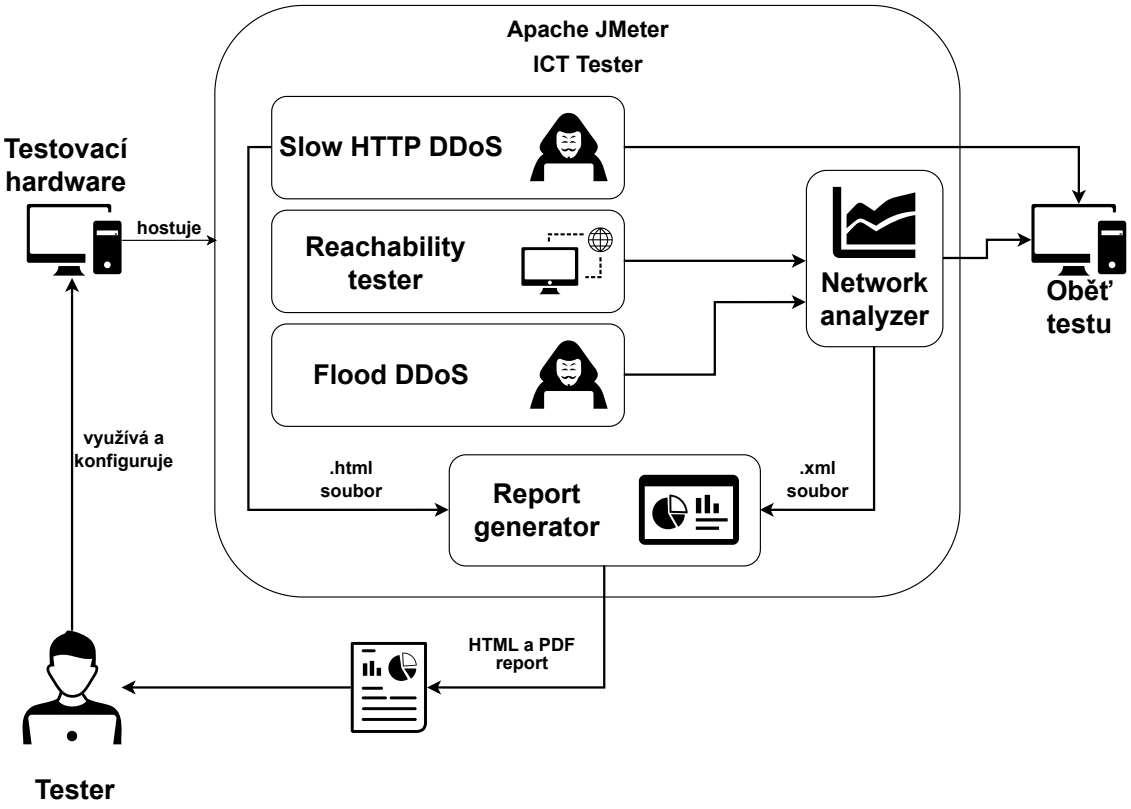
tive zakódován. Aktuálně nejčastěji používaným standardem je *adobe.pkcs7.detached*, ostatní jako *adbe.x509.rsa_sha1* a *adbe.pkcs7.sha1* by se od verze PDF 2.0 již používat neměly,

- **Cert** (podmíněně volitelný) – certifikát veřejného klíče, který byl použit při podpisu dokumentu. Toto pole by se mělo použít pouze v případě, že **SubFilter** obsahuje standard *adbe.x509.rsa_sha1*. V obou ostatních případech je certifikát již součástí pole **Contents**,
- **Name** (volitelný) – jméno subjektu, který podepsal dokument. Toto pole by mělo být použito pouze v případě, že není možné získat jméno podepisující z certifikátu;
- **M** (volitelný) – datum a čas, kdy by dokument podepsán. Tuto hodnotu je také možné opět získat přímo z digitálního podpisu, obzvlášť když je použit standard PKCS#7. V takovém případě by tento parametr neměl být použit,
- **Location** (volitelný) – fyzické místo podepsání dokumentu, případně jméno počítače, ze kterého byl vytvořen podpis (host name),
- **Reason** (volitelný) – důvod podepsání dokumentu, např. „Souhlasím s podmínkami danými v tomto dokumentu, . . . “,
- **ContactInfo** (volitelný) – kontakt na podepisujícího, například telefonní číslo nebo email,
- **DigestMethod** (podmíněně volitelný) – hashovací algoritmus, který byl použit při výpočtu digitálního podpisu. Tato hodnota může být specifikována také přímo v certifikátu.
- a další. . .

3 Implementace generátoru reportů

Tato kapitola pojednává o implementaci generátoru reportů (modulu Report generátor do nástroje Apache JMeter) z výsledků zátěžového testování. Zadanými požadavky na výsledný modul bylo, aby uměl zpracovat výstupní XML soubor z modulu Network analyzer, který analyzuje síťový provoz na lokálních síťových rozhraních a vytížení systémových prostředků při zátěžovém (DDoS) testování, a také z testů Slow DDoS útoků [4], které jsou součástí modulu DDoS (více v kapitole 1.5). Hodnoty vyčtené z tohoto modulu jsou následně přepracovány do HTML výstupu, ze kterého je nakonec vygenerován PDF soubor. Výsledný report obsahuje přehledné grafy hodnot jednotlivých veličin a výpisy základních naměřených hodnot a statistik z jednotlivých testů. Vizualizaci průběhu testování a následného generování reportu je možné vidět na obrázku 3.1.

Kapitola nejprve popisuje ovládací prvky grafického uživatelského rozhraní, následně se zaměří na veličiny, které výsledná zpráva o testování obsahuje. Poté přiblíží algoritmus, který zajišťuje výběr jednotlivých testů do reportu a jejich mazání z výstupních souborů ostatních modulů. V neposlední řadě je v kapitole 3.4 popsáno, jakým způsobem funguje generování HTML souboru a na konec transformace tohoto souboru do formátu PDF.



Obr. 3.1: Diagram průběhu testování a generování reportu

3.1 Grafické uživatelské rozhraní modulu

Jak již bylo zmíněno v kapitole 1.4, Apache JMeter poskytuje možnost vytvářet různé druhy modulů. Jelikož vytvářený modul Report generator slouží jako vizualizace výsledků testů, je nejvhodnější jej koncipovat jako Listener. Modul je určen pouze pro použití v GUI JMeteru, nikoliv v jeho konzolové verzi. Po nainstalování pluginu lze modul přidat do test plánu pravým klikem na záložku Test Plan → Add → Listener → Report generator. Po následném kliknutí na přidaný modul ve stromu je uživateli zobrazen v příslušném okně GUI modulu Report generator.

Rozhraní obsahuje několik základních komponent. Těmi nejzákladnějšími jsou 3 pole, ve kterých je možné definovat vstupní soubory – nejprve pro XML soubor z modulu Network analyzer a HTML soubor (případně ZIP, v případě testů z více zdrojových adres, viz kapitola 5.3) ze Slow DDoS modulů. Třetím polem je možné vybrat výstupní složku, do které budou uloženy výsledné HTML a PDF reporty. Všechna pole je možné vyplnit pouze pomocí průzkumníku souborů, který se otevře po stisknutí příslušného tlačítka **Browse...**. Dalším polem je **Threshold for reachability**, kde je možné zadat procentuální hodnotu. Toto pole ve výsledném reportu ovlivňuje označení cíle jako dostupný či nedostupný, pokud je cíl (oběť útoku, testu) nedostupný více jak x % času testu. Následně rozhraní obsahuje část pro nastavení digitálního podpisu, o které se více zmiňuje kapitola 4.2.

Poslední část obsahuje tlačítka **Load data** a **Generate report**. První z nich načte data z testových souborů, které byly definovány výše a zobrazí přehledně v tabulce testů, jaké testy tyto soubory obsahují. Uživatel si v této tabulce může vybrat, které testy chce zahrnout do výsledného reportu. Může také ke každému testu přidat textový komentář, který je taktéž v reportu vypsán. Nechybí také možnost tyto testy z testových souborů smazat. Po zaškrtnutí příslušné buňky ve sloupci **Delete test record** a stisknutí tlačítka **Remove from xml file** bude záznam testu nadobro vymazán. Po stisknutí tlačítka **Generate report** dojde k vygenerování reportu z testu (testů) podle výše vybraných kritérií.

Základní třídou pro vykreslování a sestavení GUI modulu Report generátor je třída `ReportGUI`, která dědí ze třídy `AbstractVisualizer`, jelikož se jedná o Listener. Nástroj Apache JMeter používá pro vykreslování uživatelského rozhraní Java knihovnu Swing [19], více v kapitole 2.4.

3.2 Veličiny obsažené v reportu

Aby měl výsledný report určitou informační hodnotu, musí obsahovat několik základních údajů o proběhlých testech. Patří mezi ně:

- čas začátku a konce testu,

- délka testu,
- minimální, maximální a průměrný počet přijatých a odeslaný bytů na sledovaném rozhraní (v případě testů sledovaných Network analyzerem),
- minimální, maximální a průměrný počet přijatých a odeslaných bytů za sekundu (v případě testů sledovaných Network analyzerem),
- test dostupnosti cíle (oběti),
- počet spojení (v případě Slow DDoS testů).

Dále pro uživatelskou přívětivost a snazší orientaci v naměřených hodnotách musí report obsahovat přehledné grafy vizualizující naměřené hodnoty. Grafy samozřejmě nelze vytvářet ze všech naměřených hodnot, proto jsou grafy tvořeny z následujících hodnot:

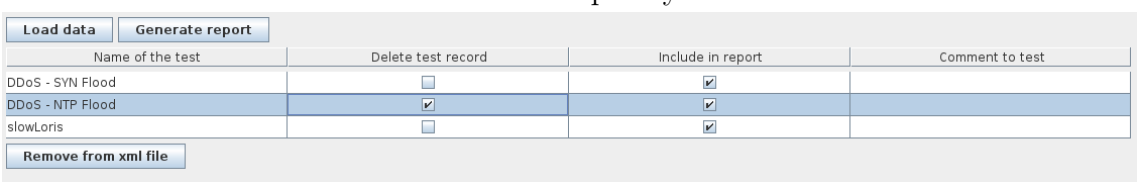
- Minimální, maximální a průměrný počet přijatých a odeslaný bytů na sledovaném rozhraní,
- minimální, maximální a průměrný počet přijatých a odeslaných bytů za sekundu,
- test dostupnosti oběti,
- a počet spojení v případě Slow DDoS útoků.

Graf testu dostupnosti oběti je součástí všech výše zmíněných grafů, aby bylo možné v čase sledovat, kdy cíl přestal být dostupný. Následně každý test obsahuje i koláčový graf znázorňující procentuální dostupnost, respektive nedostupnost oběti z celkové časové délky testu. Příklad výsledného reportu včetně všech veličin a grafů je možné vidět v příloze B.

3.3 Výběr testů pro generování reportu

Jak již bylo v předchozí kapitole zmíněno, modul Report generator umožňuje vybrat, ze kterých aktuálně načtených testů bude vygenerován report. Testy je možné vybrat v příslušné tabulce v uživatelském rozhraní. Existující záznamy testů se z vložených souborů načtou po stisknutí tlačítka **Load data**. Podobu tabulky v GUI ukazuje obrázek 3.2.

Obr. 3.2: Tabulka pro výběr testů



Samotná tabulka je vytvořená pomocí komponenty JTable z Java knihovny

Výpis 3.1: Mapování XML souboru na Java třídu TestResults

```
XmlMapper xmlMapper = new XmlMapper();
xmlMapper.disable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
try {
    testResults = xmlMapper.readValue(file, TestResults.class);
} catch (IOException e) {
    log.error(e.getMessage());
}
```

Swing [19]. Třída `ReportStructure`, která zajišťuje načítání a mazání dat testů z Network analyzeru z příslušného XML souboru, obsahuje dvě hlavní metody:

- `public void loadReportXml(...)` sloužící pro deserializování, resp. namapování vstupního XML souboru na třídu `TestResults`, která obsahuje seznam vzorků (třída `Sample`). Pro mapování XML souborů na Java třídy je použita knihovna Jackson Dataformat XML [39]. Více ve výpisu 3.1. Načtené testy jsou posléze uloženy do instanční proměnné `TestResults testResults;`,
- `public void removeEntriesFromXml(...)` sloužící pro smazání záznamů testů z XML souboru. Ze všech existujících předem načtených testů jsou vyfiltrovány vzorky pouze těch testů, které jsou v tabulce označeny ke smazání. Tyto vzorky jsou posléze upraveny ve třídě `TestResults`, která je následně, opět pomocí knihovny Jakcson Dataformat XML [39], serializována do podoby XML souboru a předchozí soubor je nahrazen novým, upraveným.

Obě tyto metody jsou volány ze třídy `ReportGUI` z příslušných listenerů, které zajišťují příslušné volání v kódu na stisk konkrétního tlačítka.

3.4 Generování HTML reportu

Základním prvkem vizualizace dat získaných během testování je report ve formátu HTML. Tvoří „předstupeň“ před výsledným reportem ve formátu PDF. Struktura HTML se skládá pouze z čistého HTML, *Cascading Style Sheets* (CSS) a přiložených obrázků (grafy, schéma útoku, ...).

Generování reportu je spuštěno se stiskem tlačítka **Generate report**. Základní komponentou pro generování je třída `ReportGeneration` obstarávající a volající nejdůležitější akce nutné pro samotné vygenerování reportu. Metoda `generateReport` nejprve zavolá příslušné části kódu, které vytvoří všechny potřebné složky:

- kořenový adresář pro uložení reportu, jeho název je shodný s názvem výsledného souboru s reportem a je volený uživatelem v GUI,
- adresář **css** pro uložení kaskádových stylů,
- adresář **images** pro uložení obrázků, které budou součástí reportu.

Všechny tyto operace jsou vykonány pomocí výchozích knihoven jazyka Java z balíčku `java.nio.file`. Následně jsou překopírovány všechny potřebné statické zdroje – loga, kaskádové styly, schéma útoku – ze složek zdrojového kódu do cílové složky s budoucím vygenerovaným reportem. Posléze jsou připraveny veškeré podklady a nic nebrání započatí algoritmu pro generování reportu.

Generování začíná v metodě `public void generateReportHTML(...)`, která si jako parametry bere záznamy z obou typů útoků a seznam testů vybraných pro generování z GUI tabulky. Vzorky z Network analyzeru jsou v podobě třídy `TestResults`, které je následně předána do metody `DataMapWrapper.generateMultiHashMap`. V té proběhne filtrace veškerých vzorků (hodnoty jako vytížení procesoru nebo paměti nesledujeme) a následně jejich restrukturalizace a setřídění do datové struktury `Map<String, Map<String, Map<String, List<Sample>>>>`, která v překladu obsahuje pro každý test (1. Map) data ze sledovaných rozhraní (2. Map), které sestávají ze vzorků pro každou sledovanou veličinu (3. Map). Data z modulů pro Slow DDoS útoky jsou z HTML souboru vyextrahována pomocí regulárního výrazu a opět vyfiltrována jen potřebná (počet spojení). Následně jsou data ze všech testů předána do metod, které vytváří již samotnou HTML strukturu pomocí šablon.

3.4.1 HTML šablony

HTML šablony jsou úryvky HTML kódu, které jsou uloženy do Java proměnných a po provedení určitých operací z nich vznikne výsledný HTML report. Každý z těchto úryvků obsahuje unikátní řetězec, který tvoří jakousi proměnnou, místo které může být vložen další HTML kód a tímto postupně vytvářena kompletní struktura celého reportu.

Třída `TemplateSubstituteStrings` obsahuje statické proměnné, kdy každá obsahuje řetězec znaků představující substituční proměnnou v HTML kódu ve tvaru *substitute-**, viz výpis 3.2. Tato třída také obsahuje právě fragmenty HTML kódu (viz proměnná `REACHABILITY` ve výpisu 3.2). Na řádku 11 lze také vidět právě onu substituční proměnnou, která bude později nahrazena dalším fragmentem HTML kódu a posléze bude celý tento fragment vložen do jiného nadřazeného fragmentu. Tímto principem se dynamicky vytvoří výsledný HTML soubor dle vybraných testů, ze kterých má být report generován.

Po stisknutí tlačítka **Generate report** jsou volány dvě metody:

Výpis 3.2: Příklad proměnné obsahující substituční řetězec

```
public static String REACHABILITY_TEXT =
↪  "substitute-reachability-text";
public static String REACHABILITY = """
            <div class="titles_div">
                <h3 class="titles">Reachability result</h3>
            </div>
            <div class="transfer_graph">
                <div class="transfer_graph_box">
                    <img class="graph_img_reach"
                        src="substitute-graph-reachability">
                </div>
            substitute-reachability-text
            </div>
            """.indent(16);
```

- `public String generateXmlTestResultTemplate` – pro testy z Network analyzeru,
- `public String generateSlowAttackTemplate` – pro testy z Slow DDoS modulů.

Obě tyto metody obdobným způsobem procházejí všechny testy vybrané ke generování, které jsou jim předané jako parametry, a postupně vytvářejí výsledný HTML soubor skládáním HTML fragmentů a nahrazováním substitučních proměnných, jak bylo popsáno výše. Pro nahrazování substitučních proměnných daty v šablonách slouží jednoduchá metoda, viz výpis 3.3. Během generování jsou také volány příslušné metody pro vytváření grafů. K tomu je určena třída `ChartOperations`. Každá z metod generující graf vrací jméno obrázkového souboru, do kterého je graf uložen a toto jméno je poté vloženo jako cesta do příslušných HTML tagů, aby byl obrázek správně vykreslen.

Po vytvoření celé HTML struktury ze všech testů jsou ještě dodatečně přidána loga, verze programu Apache JMeter a například tabulka s přehledem všech provedených testů. Následně je celá struktura uložena do jednoho souboru s příponou *.html.

Výpis 3.3: Metoda pro nahrazování substitučních proměnných daty

```
1 public String substituteInTemplateString(String template, String
  ↪ stringTobeReplaced, String data) {
2     return template.replaceFirst(stringTobeReplaced, data);
3 }
```

3.5 Generování PDF reportu

Generování PDF reportu obstarává metoda `generatePDF()`, která se nachází ve třídě `ReportGeneration`. V první fázi si metoda načte z diskového úložiště HTML report, který byl vygenerován v předchozím kroku (viz kapitola 3.4). Následně je za pomoci Java HTML parseru, knihovny jsoup [40], převeden na formát *Extensible Hypertext Markup Language* (XHTML)[1].

Poté je již začíná fáze generování PDF souboru. K té je využita knihovna Flying Saucer, což je knihovna sloužící pro renderování nejen PDF dokumentů ze XML vstupů [41]. Knihovna je šířena pod licencí GNU LGPL 2.1 [42], která umožňuje i komerční použití. Nejprve je provedeno vytvoření instance třídy `OutputStream` definující výstupní soubor. Následně v bloku *try-with-resources* je inicializována instance třídy `ITextRenderer`, která se postará o vyrenderování obsahu PDF dokumentu [44].

Příklad výsledného PDF reportu lze nalézt v příloze B. Útoky vyobrazený v tomto reportu je prováděn vůči serveru s operačním systémem Ubuntu nacházejícím se na lokální síti. Prvním z útoků je útok typu SYN Flood, druhý útok je typu Slow Loris. Ten je prováděn oproti webovému serveru běžícím na tomto stroji. Oba útoky byly vyhodnoceny jako Unreachable (nedostupné), jelikož cíl byl nedostupný více než 3 % z celkového času testu. Tento parametr je možné volit při generování reportu.

[1] XHTML je obdoba HTML jazyka, jehož syntaxe je přísnější než u standardního HTML a je více podobné standardnímu XML jazyku. Mezi hlavní rozdíly patří, že například tagy jako `<head>`, `<body>` jsou povinné, všechny tagy musí být uzavřené a další... [43].

4 Implementace digitálního podpisu

Tato kapitola se věnuje popisu implementace digitálního podpisu výsledného reportu ve formátu PDF. Nejdříve jsou shrnuty aktuálně dostupné knihovny, které je možné použít k digitálnímu podpisu. Následně se kapitola věnuje implementaci grafického rozhraní pro nastavení digitálního podpisu a shrne načítání soukromého a veřejného klíče v prostředí jazyka Java. V neposlední řadě je popsána funkce a důležitost využití kryptografické knihovny Bouncy Castle a samotná implementace podpisu v jazyce Java.

4.1 Výběr knihovny pro digitální podpis PDF

Pro manipulaci s PDF dokumenty je v programovacím jazyku Java k dispozici spousta knihoven. Ovšem pro implementaci digitálního podpisu generované zprávy z testování musí splňovat hlavní podmínku – musí být dostupná pod open-source licencí, která umožňuje i komerční použití bez poplatků, což výběr dostupných knihoven značně omezuje.

Jednou z možností je knihovna Apache PDF Box [45]. Umožňuje komplexní manipulaci s PDF dokumenty – vytváření dokumentů, editaci a extrahování obsahu dokumentů a také digitální podpisy. Je navíc zveřejněna pod open-source licencí Apache License 2.0 [33], která umožňuje i komerční využití bez jakýchkoliv poplatků.

Další dostupnou možností je knihovna OpenPDF [46], která je odnoží placené knihovny iText, a je zveřejněná pod svobodnou licencí GNU LGPL version 2.1 [42]. Svými možnostmi se velmi podobá předchozí zmíněné knihovně Apache PDF Box.

Obě tyto knihovny obsahují API pro vložení digitálního podpisu do struktury PDF dokumentu. Tento podpis musí (může) být vytvořen externím kryptografickým poskytovatelem (knihovnou), například za vyžití Bouncy Castle (viz kapitola 4.4). Jelikož by obě knihovny splňovaly požadavky na licenci a funkčnost, bylo již na rozhodnutí autora, kterou z nich vybere.

Pro účely této práce byla nakonec vybrána knihovna OpenPDF [46], která má z pohledu autora přehlednější, jasnější a jednodušší API a dokumentaci. Navíc je tato knihovna již použita v rámci balíčku Flying Saucer [41], který slouží ke konverzi formátu HTML do PDF (viz kapitola 3.5). Tím pádem nemusela být pro digitální podpis přidávána další závislost na jiné knihovně.

4.2 Uživatelské rozhraní pro digitální podpis

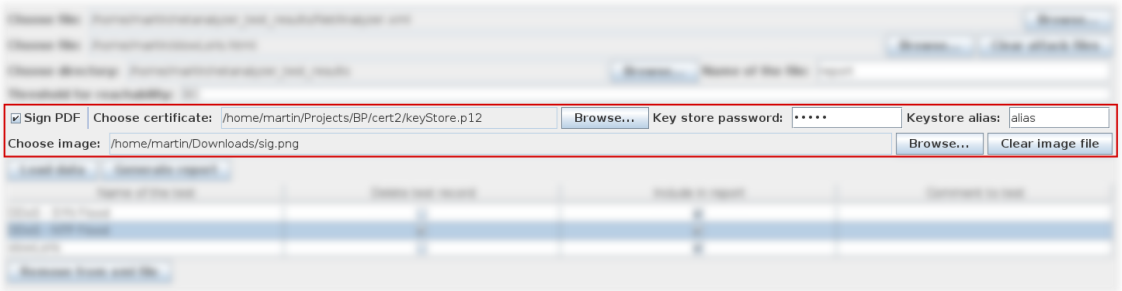
Grafické uživatelské rozhraní pro nastavení digitálního podpisu je součástí GUI pluginu Report generátor, viz kapitola 3.1.

Skládá se z několika důležitých komponent, které jsou pro správné nastavení digitálního podpisu nezbytné (viz obrázek 4.1):

- přepínač **Sign PDF** – kterým se aktivuje funkce digitálního podpisu,
- pole **Choose certificate** (povinné) – je pole, do kterého je nutné vyplnit cestu k souboru PKCS#12 ve formátu *.p12. Pro snazší výběr je možné využít průzkumník souborů, který se otevře po stisknutí tlačítka **Browse...** vedle pole,
- pole **Keystore password** (volitelné) – pole pro vložení hesla souboru PKCS#12,
- pole **Keystore alias** (volitelné) – pole pro jméno, pod kterým jsou v souboru PKCS#12 uložené požadované kryptografické prvky, viz kapitola 2.11,
- pole **Choose image** (volitelné) – pole, do kterého je možné vyplnit cestu k obrázku zobrazujícího se na konci dokumentu, pokud je podpis vytvořen. Tento obrázek může být například logo firmy, případně nějaký text oznamující, že dokument je digitálně podepsán. Stejně jako u výběru souboru PKCS#12 je možné použít průzkumník souborů pomocí tlačítka **Browse...**, případně smazat cestu k obrázku pomocí tlačítka **Clear image file**.

V případě, že je aktivováno tlačítko *Sign PDF*, digitální podpis je vytvořen automaticky s vygenerováním reportu (stiskem tlačítka *Generate report*, viz kapitola 3.1). Podepsaný PDF dokument je uložen na stejné vybrané místo jako ten nepodepsaný, nicméně je mu přidána koncovka „_signed“ k jeho jménu. V případě, že není tlačítko *Sign PDF* zapnuté, všechna pole a tlačítka, týkající se digitálního podpisu, jsou pro uživatelskou přehlednost deaktivována.

Obr. 4.1: Grafické uživatelské rozhraní pro nastavení digitálního podpisu



4.3 Inicializace privátního klíče a certifikátu do Java KeyStore

Jak již bylo zmíněno v teoretické části (kapitola 2.14), Java KeyStore je třída z balíčku *java.security*, která slouží jako objekt reprezentující úložiště kryptografických klíčů a certifikátů. V rámci implementace digitálního podpisu je využita právě pro načtení souboru PKCS#12 a následnou práci s hodnotami v něm uloženými jakožto s objekty v jazyce Java.

Inicializace KeyStore a načtení klíčů, resp. certifikátů, je implementována ve třídě `SignOperation`, která obsahuje privátní metodu *loadKeystore*, viz výpis 4.1. Metoda je volaná přímo z konstruktoru třídy a obsahuje tři parametry, přičemž všechny jsou datovými reprezentacemi vstupů od uživatele z grafického rozhraní:

- `String certFilePath` – cesta k souboru formátu PKCS#12,
- `char[] keyStorePass` – heslo, kterým může být soubor PKCS#12 zašifrován,
- `String alias` – jméno, pod kterým jsou v souboru uloženy požadované kryptografické prvky (certifikát, privátní klíč).

Pro načtení souboru PKCS#12 se na řádku 2 ve výpisu 4.1 využívá statická metoda `KeyStore.getInstance(...)`, která vrací instanci třídy `KeyStore`. Jejími parametry jsou odkaz na soubor PKCS#12 a heslo ke KeyStore. Dále jsou načteny všechny aliasy do datové struktury `Enumeration<T>`. Na řádku 8 následuje cyklus, který prochází všechny aliasy obsažené v této struktuře. Cyklus je ukončen tehdy, pokud aktuální alias se rovná aliasu specifikovanému uživatelem nebo pokud projde všechny elementy. V případě, že uživatel žádný alias nespecifikuje, bere se první nalezený. Naopak, pokud není alias nalezen (nebo keystore žádný neobsahuje), je vytvořena výjimka (`KeyStoreException`), která je propagována do předchozích metod, odkud je uživatel informován smysluplnou chybovou hláškou.

Jako poslední metoda obstarává získání certifikátu a soukromého klíče z instance KeyStore ukryté pod konkrétním aliasem. Oba získané prvky přetypuje na datové typy kompatibilní s metodami, které se následně používají při tvorbě podpisu, a následně uloží do instančních proměnných `certificate` a `privateKey`.

4.4 Knihovna Bouncy Castle

Bouncy Castle je knihovna, která rozšiřuje jazyk Java (případně C#) o kryptografické API umožňující snadnou práci s kryptografickými funkcemi. Patří mezi ně například generování X.509 certifikátů, PKCS#12 souborů, generátory a procesory pro správu časových razítek, digitálních podpisů, šifrování a dešifrování a práci s infrastrukturou veřejných klíčů. Slouží v programovacím jazyce Java také jako tzv.

Výpis 4.1: Implementace metody loadKeystore ve třídě SignOperation

```
1  private void loadKeystore(String certFilePath, char[] keyStorePass,
   ↪  String alias) {
2      KeyStore keyStore = KeyStore.getInstance(new File(certFilePath),
       ↪  keyStorePass);
3      Enumeration<String> aliases = keyStore.aliases();
4      if (aliases == null) {
5          throw new KeyStoreException("KeyStore does not contain any
           ↪  alias");
6      }
7      String chosenAlias = null;
8      while (aliases.hasMoreElements()) {
9          String currentAlias = aliases.nextElement();
10         // If alias is not specified, get first available alias
11         if (alias.isEmpty() || alias.equals(currentAlias)) {
12             chosenAlias = currentAlias;
13             break;
14         }
15     }
16     if (chosenAlias == null) {
17         throw new KeyStoreException("Keystore does not include
           ↪  specified alias");
18     }
19     certificate = (X509Certificate)
       ↪  keyStore.getCertificate(chosenAlias);
20     certificate.checkValidity();
21     privateKey = (PrivateKey) keyStore.getKey(chosenAlias,
       ↪  keyStorePass);
22 }
```

„poskytovatel bezpečnosti“ (security provider) [47].

V implementaci zásuvného modulu Report generator slouží Bouncy Castle jako „poskytovatel“ digitálního podpisu. Jinými slovy vytvoří data digitálního podpisu ve tvaru *Abstract Syntax Notation number One* (ASN.1)[1] z poskytnutého obsahu, soukromého klíče a certifikátu, která následně mohou být vložena do objektu PDF dokumentu.

Implementace vytváření dat digitálního podpisu je součástí třídy `SignOperation` v metodě `private byte[] getSignature(byte[] pdfContent)`, viz výpis 4.2. Základním prvkem je instance třídy `CMSSignedDataGenerator` sloužící ke generování digitálního podpisu ve formátu standardu PKCS#7 [38]. Do něj jsou postupně vkládány pomocí instancí několika tříd další informace, jako je algoritmus použitý k podpisu (SHA256withRSA – SHA256 pro vytvoření hashe a algoritmus RSA pro vytvoření zašifrované zprávy – podpisu, viz. kapitola o digitálním podpisu 2.10), soukromý klíč a certifikát získaný z instančních proměnných objektu třídy `SignOperation`.

Zavoláním metody `dataGenerator.generate(data)` je na řádku 15 dle předchozích vložených parametrů vypočítán digitální podpis a uložen do objektu třídy `CMSSignedData`. Jelikož s tímto datovým typem nemohou přímo pracovat PDF dokumenty, je nutné jej převést na typ ASN.1 [48], což se udělá zavoláním metody `getEncoded()`, která vrátí pole bajtů právě v tomto formátu.

4.5 Vložení digitálního podpisu do PDF dokumentu

Samotné vytvoření digitálního podpisu PDF dokumentu obstarává veřejná metoda `public void sign()`, která je součástí třídy `SignOperation`. Výpis kódu se nachází v příloze A.1. Algoritmus podpisu sestává z několika částí.

První částí (řádek 1 až 5) je načtení PDF souboru, který se bude podepisovat a zároveň vytvoření objektu `FileOutputStream`, který umožňuje zápis do souboru po bytech. Následně jsou vytvořeny instance objektů `PdfReader`, který slouží pro načtení PDF dokumentu (parametrem je cesta k načítanému dokumentu), a `PdfStamper` sloužící pro přidání dalšího doplňujícího obsahu do struktury PDF dokumentu [49]. V našem případě bude sloužit pro přidání digitálního podpisu. Tento objekt se vytváří pomocí statické metody, která jako parametry přebírá dříve vytvořený `PdfReader`, `FileOutputStream` a verzi nově vytvořeného PDF dokumentu. Pokud se vloží znak `'\0'`, tak se použije verze původního dokumentu. Obě tyto metody pocházejí již z knihovny OpenPDF, o které se zmiňuje kapitola 4.1.

[1] ASN.1 je univerzální standard specifikující abstraktní datový typ, který umožňuje „zakódovat“ přenášená data bez ohledu na konkrétní aplikaci nebo použitém programovacím jazyce. Může se jednat o jakýkoliv typ informace – audio, video nebo obecná data [48].

Výpis 4.2: Implementace metody getSignature ve třídě SignOperation

```
1  private byte[] getSignature(byte[] pdfContent) {
2      CMSSignedDataGenerator dataGenerator = new
       ↪  CMSSignedDataGenerator();
3      ContentSigner contentSigner = new
       ↪  JcaContentSignerBuilder("SHA256withRSA")
4          .build(this.privateKey);
5
6      dataGenerator.addSignerInfoGenerator(
7          new JcaSignerInfoGeneratorBuilder(
8              new JcaDigestCalculatorProviderBuilder()
9                  .setProvider(CRYPTO_PROVIDER).build())
10             .build(contentSigner, this.certificate));
11
12     dataGenerator.addCertificate(new
       ↪  X509CertificateHolder(certificate.getEncoded()));
13
14     CMSProcessableByteArray data = new
       ↪  CMSProcessableByteArray(pdfContent);
15     return dataGenerator.generate(data).getEncoded();
16 }
```

Dále na řádku 7 se volí metoda `appendImage(...)`, která se sestává z vložení obrázku ve formátu *Portable Network Graphics* (PNG), který může sloužit jako určité zviditelnění digitálního podpisu v přímo v obsahu dokumentu. Podmínka zajišťuje, že obrázek je vložen pouze v případě, že uživatel vyplní příslušné pole v GUI. Jelikož je žádané, aby se obrázek zobrazil vždy na poslední stránce dokumentu, je nutné nejdříve získat obsah této stránky, aby bylo možné do něj obrázek vložit. Ten se získá pomocí metody `stamper.getOverContent(pdfReader.getNumberOfPages())`. Parametrem je index stránky, jejichž obsah chceme získat.

Další část na řádcích 9 až 20 se zabývá vytvořením podpisového slovníku a jeho vložením do struktury PDF dokumentu. Podpisovým slovníkům se věnuje kapitola 2.15. Nejdříve je potřeba získat objekt třídy `PdfSignatureAppearance`, který se stará o správné zformování výsledného podpisu ve struktuře dokumentu. Instance je již vytvořena uvnitř objektu `PdfStamper`, takže stačí použít příslušný getter pro její získání. Následně je potřeba vytvořit podpisový slovník. K jeho reprezentaci slouží třída `PdfSignature`. Jako parametry konstruktoru se předávají hodnoty do slovníkových proměnných *Filter* a *Subfilter* (viz kapitola 2.15). Následně je pak ještě potřeba do slovníku nastavit aktuální datum podpisu. Slovník se poté vloží do objektu `PdfSignatureAppearance`. Jako poslední věc v této části je nutné vytvořit HashMapu, která jako klíč obsahuje PDF parametr *Contents* a jako hodnotu maximální bytovou velikost, jakou může následný digitální podpis zabírat. Následně je nutné `SignatureAppearance` tzv. „před-uzavřít“ pomocí metody `preClose()`, která si jako parametr bere onu vytvořenou HashMapu s alokovanou délkou podpisu. Z aktuálního obsahu dokumentu se následně bude vytvářet digitální podpis.

Poslední částí algoritmu podpisu je samotné vložení podpisu do PDF dokumentu. Nejdříve je potřeba získat bytovou hodnotu obsahu dokumentu, který se bude podepisovat. Ta se získá přímo z objektu `signatureAppearance` pomocí metody `getRangeStream()`, která vrací instanci třídy `InputStream`, ze které již stačí pomocí metody `readAllBytes()` získat bytové pole. S tímto polem jako parametrem se zavolá metoda `this.getSignature(byte[] pdfContent)`, která vrací samotnou hodnotu digitálního podpisu (více v kapitole 4.4). Následně je potřeba tuto hodnotu vložit do `signatureApperance` pomocí metody `setExternalDigest(...)`, kde se jako první parametr vloží hodnota podpisu, druhý parametr se nastaví na null (slouží pro vkládání extra dat do struktury PKCS#7, které zde nejsou potřeba) a poslední parametr se musí nastavit buď na RSA nebo DSA podle použitého šifrovacího algoritmu při podpisu – v případě této implementace se jedná o RSA. Ještě je potřeba stamperu říci, jaký „poskytovatel kryptografie“ byl využit – v našem případě se jedná o knihovnu Bouncy Castle (viz kapitola 4.4), tím pádem se proměnná nastaví na hodnotu „BC“.

Následně je potřeba vytvořit PDF slovník, který obsahuje klíč Contents s hod-

notou rovnou hodnotě digitálního podpisu v hexadecimálním tvaru. Jelikož ale byl v předchozích fázích rezervován určitý prostor pro hodnotu podpisu a tento prostor musí být zcela zaplněn, je tedy potřeba rozdíl mezi délkou tohoto prostoru a délkou digitálního podpisu vyplnit nulovými byty. K tomuto účelu slouží metoda `SignOperation.addContentPadding(byte[] signedContent)`. Poslední volanou metodou je `signatureAppearance(pdfDictionary)`, která vloží slovník s digitálním podpisem do struktury PDF dokumentu a následně tento dokument uloží na disk.

5 Testování a oprava chyb v ICT testeru

V rámci projektu, ve kterém je celý ICT tester [3] vyvíjen, proběhlo důkladné testování, ve kterém bylo objeveno několik nedostatků, které bylo potřeba opravit a vylepšit. Tyto nedostatky spočívaly jak v opravě chyb, tak i v doplnění určitých chybějících funkcí. Jednalo se o nedostatky jak v modulu Report generátor vyvíjeným v rámci této práce, tak i v dalších modulech, které byly součástí celého projektu.

5.1 Zamrznutí GUI při generování Stairs thread group grafu

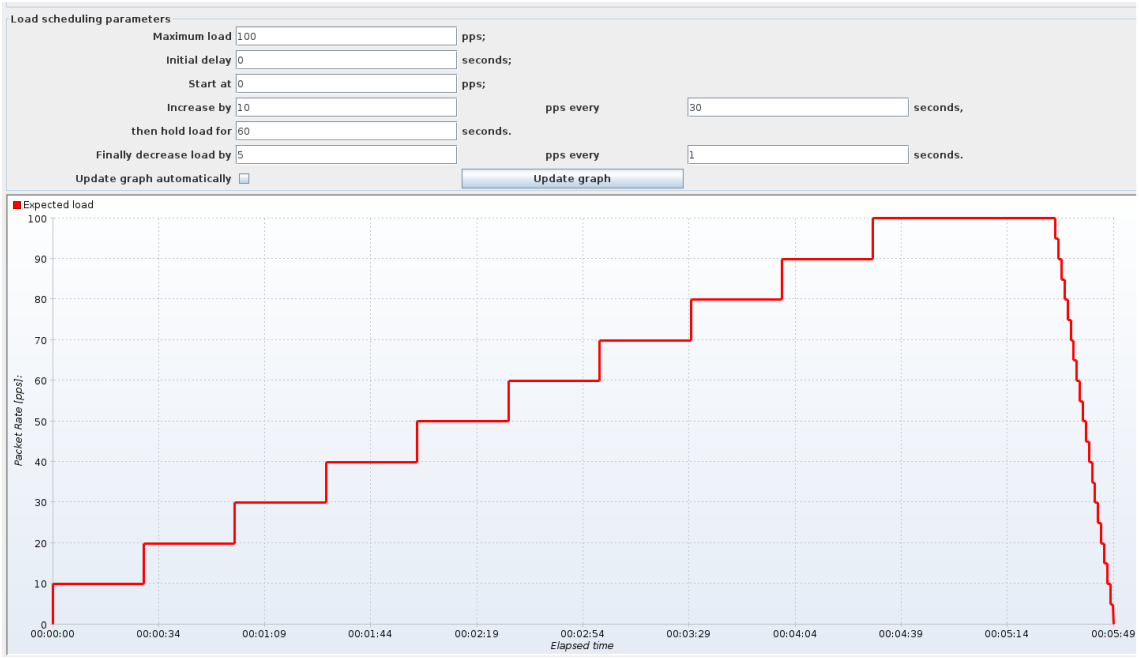
Stairs Thread group je thread group modul umožňující nastavení schodovitého průběhu testu – jinými slovy umožňuje nastavit jak často bude zátěž zvyšována, jak dlouho bude udržována, a poté jakým tempem bude snižována. Tento schodovitý průběh zobrazuje graf, jenž je součástí GUI tohoto modulu (viz obrázek 5.1). Jelikož zátěž se může pohybovat až v milionech paketů za sekundu (pps), může výpočet tohoto grafu zabrat hodně výpočetního času. Původní implementace tento výpočet prováděla na stejném vlákně jako běží i GUI aplikace JMeter, tím pádem tento přístup způsoboval zamrznutí i na 15 vteřin, dokud nebyl výpočet hodnot dokončen, což je z pohledu uživatelské přívětivosti nepřijatelné.

Tato chyba byla opravena způsobem, že se generování grafu přesunulo na vedlejší vlákno, které neovlivňuje fungování samotného GUI JMeteru. Dále bylo uživatelské rozhraní doplněno o zaškrtávací políčko (checkbox), které určuje, zda-li má být graf přegenerován automaticky po změně hodnot v kterémkoliv poli, nebo jestli má být graf znovu vygenerován až po stisknutí tlačítka „Update graph“. Jelikož z pohledu výpočetní náročnosti je preferovaná varianta s tlačítkem, tak byla nastavena jako výchozí.

5.2 Chybějící čas začátku a konce testu v reportu

Jelikož testy (útoky) typu Slow HTTP [4] využívají pro generování provozu jiný testovací nástroj, struktura a způsob generování reportů je lehce odlišná. Nástroj třetí strany, *slowhttptest* [50], generuje své vlastní HTML reporty, které se poté v Report generátoru parsují a vytahují se z nich potřebná data pro výsledný report. Avšak tento HTML report neobsahuje potřebná data (časy začátku a konce testu) a nástroj *slowhttptest* [50] sám o sobě neposkytuje žádnou možnost přidání těchto hodnot do reportu.

Obr. 5.1: Graf ve Stairs Thread Group modulu



Řešením tedy bylo doimplementovat funkcionalitu do DDoS modulu, který po skončení testu načte příslušný vygenerovaný HTML soubor (jméno a umístění je zadáváno uživatelem, takže je známé), načíst ho pomocí knihovny jsoup [40] a obě hodnoty doplnit do tabulky k ostatním datům. Jelikož Report generátor již hodnoty z této tabulky dříve parsoval, bylo poměrně jednoduché report doplnit o tyto dvě hodnoty.

5.3 Implementace reportu z útoků z více zdrojových adres

Moduly, které poskytují Slow HTTP útoky – SlowLoris, SlowPost, SlowRead, neumožňovaly spustit útok z více zdrojových IP adres, což by umožňovalo simulovat typické chování DDoS útoků. Tato funkcionalita byla implementována již dříve jako součást bakalářské práce [51], která ovšem byla zanechána na starší větvi ve verzovacím systému GIT, tím pádem po sloučení větve s nejnovějšími změnami nebyl modul kompatibilní s posledními funkcemi Report generátoru.

5.3.1 Úprava Slow HTTP modulů

Jak již zmiňovala předchozí podkapitola, Slow útoky využívají nástroj *slowhttptest* [50], který generuje vlastní HTML reporty. Hlavním problémem po sloučení

kódu bylo spouštění velkého množství paralelních vláken v Sampleru, daleko přesahující počet zvolených zdrojových IP adres. Problémem bylo, že možnost spuštění nového vlákna nebyla uzamčena před dokončením vlákna předchozího. Tato chyba byla opravena za pomocí jednoduché proměnné, která slouží jako zámek proti spuštění více souběžných vláken – zámek je uzamčen po spuštění testu a odemčen, jakmile je test ukončen nebo *slowhttptest* dokončí útok. Tento problém navíc způsoboval, že všechny paralelní vlákna navzájem přepisovaly jeden soubor s reportem, tím pádem informační hodnota výsledného reportu byla nulová.

Při spuštění útoku z více zdrojových adres je spuštěno paralelně tolik paralelních vláken, kolik zdrojových adres je uživatelem zvoleno. Zde opět nastával obdobný problém s přepisováním jednoho souboru s reportem. Utilita *slowhttptest* obsahuje parametr, do kterého se vpisuje výsledná cesta k HTML reportu. Tento problém je vyřešen tím způsobem, že pro každé spuštěné vlákno se generuje samostatný HTML report, přičemž každý tento report je očíslován od 1 do N, kde N je počet zdrojových IP adres. Do každého reportu je následně vložena informace s příslušnou zdrojovou IP adresou.

Poslední úpravou Slow DDoS modulů bylo rozšíření možnosti útoků z více zdrojových adres také na SlowPost a SlowRead moduly.

5.3.2 Úprava zpracování reportu

Dále bylo potřeba upravit modul Report generátor, aby dokázal zpracovat větší množství HTML reportů. Jelikož vkládání více HTML reportů by nebylo moc uživatelsky přívětivé, byl zvolen přístup zabalení všech jednotlivých HTML reportů do jednoho ZIP souboru, který je následně možné vložit do Report generátoru. Byla zachována i zpětná kompatibilita pro vkládání jednotlivých HTML reportů v případě, že je útok veden pouze z jedné IP adresy, nebo uživatel chce vygenerovat report pouze pro jednotlivý dílčí útok. Tento ZIP soubor je následně Report generátorem rozbalen. Jednotlivé HTML soubory jsou načteny a rozparsovány a nakonec sloučeny do jednoho výsledného celkového reportu. Údaje o síle útoku (počet spojení) jsou sčítány.

Pro správnou funkci generování ZIP souboru bylo nutné doimplementovat ve Slow DDoS modulech správné čekání na ukončení všech paralelních vláken, které obsluhují testy z jednotlivých IP adres. To zahrnuje pole, do kterého jsou postupně vkládány reference na jednotlivé vlákna při spouštění testu. Po ukončení testu se toto pole automaticky projde a program se ujistí, že jsou všechny vlákna ukončena – až poté je vygenerován výsledný ZIP soubor se všemi reporty. Pokud by toto čekání nebylo implementováno, mohla by nastat situace, při které by ZIP soubor obsahoval nekompletní a nekonzistentní data.

5.4 Rozšíření Reachability testeru o ICMP protokol

Modul Reachability tester slouží k testování dostupnosti cíle útoku. Výstup z toho modulu může uživateli (testerovi, útočníkovi) říct, zda-li byl jeho útok úspěšný a podařilo se cílový server shodit. Výsledky z Reachability testeru jsou sbírány v průběhu testu Network analyzerem, který je spojuje dohromady do jednoho výsledného XML souboru.

Předchozí implementace umožňovala testovat dostupnost oběti pouze za použití protokolu HTTP. Jelikož spousta útoků (SYN Flood, UDP Flood) může být mířena i na servery fungující na jiných protokolech (například firewally, směrovače, atp. . .), nebylo tím pádem možné tímto modulem možné měřit jejich dostupnost v průběhu testu. Proto bylo nutné doplnit chybějící funkcionalitu, a to rozšířit o možnost testování dostupnosti pomocí ICMP protokolu. K implementaci byla využita třída `InetAddress` ze základního balíčku `java.net`. Tato třída obsahuje metody `isReachable(String timeout)` vracející boolean hodnotu s údajem, jestli je server dostupný (viz výpis 5.1).

Celá implementace, včetně původní HTTP strategie, byla refaktorována tak, aby byla snadno rozšířitelná o další metody testování dostupnosti služeb. Pro přidání nové strategie stačí vytvořit třídu, která obsluhuje posílání testovacích požadavků na cílový server, a přidání záznamu do enumeration `ReachabilityStrategy` s odkazem na vytvořenou třídu. Přidáním záznamu do této enumeration se nové rozšíření automaticky zobrazí v uživatelském rozhraní.

5.5 Změna zdroje reachability hodnot u reportu ze Slow HTTP testů

Jak již bylo zmíněno v předchozích kapitolách, Slow HTTP testy využívají ke generování provozu program *slowhttptest* [50]. Tento program má také vestavěnou funkci testování dostupnosti cíle, přičemž tyto hodnoty ukládá do svého HTML reportu, které byly poté využity v Report generátoru. Toto řešení má ovšem dvě hlavní nevýhody. První je, že není možné zvolit rozhraní, ze kterého jsou požadavky na dostupnost odesílána, druhou nevýhodou může být ne-konzistence reportu s Flood útoky, kde jsou tato data sbírána přes Reachability tester, respektive Network analyzer. Proto bylo generování reportu ze Slow HTTP testů upraveno tak, aby využívalo data z Reachability testeru.

Jelikož jsou data o počtu spojení stále sbírána pomocí HTML reportu, bylo potřeba spárovat data z toho reportu s daty z Reachability testeru. Data z Reachability testeru jsou sbírána Network analyzerem, který vyžívá pro předávání dat mezi

Výpis 5.1: Implementace ICMP protokolu v Reachability testeru

```
@Override
public boolean isReachable() {
    boolean isReachable;
    try {
        InetAddress targetAddress = InetAddress.getByName(
            sampler.getPropertyAsString(TARGET_IP)
        );
        isReachable = targetAddress.isReachable(
            sampler.getPropertyAsInt(TIMEOUT)
        );
    } catch (UnknownHostException e) {
        isReachable = false;
        log.error("Target hostname not found: " + e);
    } catch (IOException e) {
        log.error("Error while pinging target: " + e);
        isReachable = false;
    }
    return isReachable;
}
```

moduly (název testu, veličiny, ...) vlastnosti (angl. properties) poskytované přes API JMeteru. Tyto properties jsou sdíleny s celou aktuální běžící instancí aplikace a nastavují se pomocí příkazu `JMeterUtils.setProperty(...)`. Jednou z těchto property sbíranými Network analyzerem je jméno testu, které právě bylo využito pro spárování obou zdrojů dat. Jméno testu bylo proto nastaveno na řetězec ve tvaru „slowAttack:<UUID testu>“. UUID testu je generováno při započetí každého jednotlivého testu a je vkládáno do výstupních souborů jak NetAnalyzeru, tak Slow testů. Tato metoda umožňuje spárovat oba proudy dat při generování reportů v Report generátoru.

Následně při nahrání XML souboru z NetAnalyzeru s daty o dostupnosti do Report generátoru nejsou v tabulce testů zobrazena ta data, která začínají řetězcem „slowAttack:“, aby nedošlo k záměně se záplavovými testy. Záznam je v tabulce zobrazen až když je nahrán příslušný HTML report ze Slow testu.

Dále bylo potřeba se vypořádat s problémem časové synchronizace vzorků z obou zdrojů dat, jelikož ve výsledku jsou součástí jednoho grafu. Data o spojení z HTML reportu jsou vzorkována každou vteřinu, přičemž každý vzorek obsahuje pouze relativní čas ve vteřinách od začátku testu. Na druhou stranu data z NetAnalyzeru obsahují časové razítko (timestamp) ve formátu UNIX millis. Výsledná implementace bere v potaz, že test začal ve stejnou dobu, tím pádem první vzorek z NetAnalyzeru se rovná nulté vteřině testu. Následně pro každou další hodnotu počtu spojení ze Slow reportu je odvozen vzorek dostupnosti z NetAnalyzeru. Jelikož NetAnalyzer umožňuje nastavovat vzorkovací frekvenci, je možné, že uživatel nastaví vzorkovací frekvenci jinou než 1 vzorek za vteřinu. Pokud by nebylo možné hodnoty spárovat kvůli chybějícím hodnotám dostupnosti v dané vteřině, zkopíruje se poslední spárovaná hodnota. Tímto způsobem se vyrovnávají vzorkovací frekvence obou modulů.

Nakonec se vygeneruje report, který je vizuálně srovnatelný s reportem ze záplavových testů, včetně obrázku se schématem testu, který v předchozí verzi chyběl a nedával smysl, protože rozhraní, ze kterého byly posílány Reachability dotazy bylo stejné, jako rozhraní, ze kterého byly spouštěny testy.

Závěr

Cílem bakalářské práce bylo vytvořit zásuvný modul do nástroje Apache JMeter, který generuje report z výsledků testů poskytnuté dalšími moduly vyvíjenými na Fakultě elektrotechniky a komunikačních technologií při Vysokém učení technickém v Brně.

Nejprve se práce věnovala přiblížení možností nástroje Apache JMeter a teorii o zátěžovém testování. Následně byly popsány již existující moduly, které byly vyvinuty v rámci projektu na FEKT VUT v Brně.

Další část práce se věnovala teoretickému podkladu pro následnou implementaci, tzn. obecné teorii o elektronickém, resp. digitálním podpisu, co jsou digitální certifikáty a jak s nimi pracovat v jazyce Java a v neposlední řadě popisu a možnostech digitálního podepisování PDF dokumentů.

Hlavní částí práce byla tvorba samotného modulu Report generator, který sestává z automatického generování HTML a PDF souboru na základě vstupních dat z ostatních modulů ve formátu XML nebo HTML. Report je možné vygenerovat z modulů sloužících pro zátěžové testování aplikací pomocí útoků typu Flood DDoS nebo Slow HTTP DDoS. Funkčnost modulu byla otestována na vhodných scénářích v rámci lokálního počítače.

Předposlední část se věnovala automatickému vytváření digitálního podpisu na základě vloženého digitálního certifikátu do modulu Report generator. Takto vytvořený digitální podpis je následně vložen do generovaného reportu ve formátu PDF. Ověření funkčnosti digitálního podpisu bylo následně provedeno v několika volně dostupných programech pro čtení PDF.

Závěrečná kapitola práce se věnovala testování a opravě chyb v rámci celého projektu ICT testeru, který probíhal na FEKT VUT v Brně. V rámci opravy chyb bylo doplněno několik chybějících funkcí, jako report z testů typu Slow HTTP z více zdrojových adres, opravy zamrzávání grafického uživatelského rozhraní, nebo například změna modulu pro testování dostupnosti oběti během testu u Slow útoků. Proběhla také oprava menších chyb a nedostatků z modulu Report generátor, který byl součástí této práce.

Literatura

[1] APACHE SOFTWARE FOUNDATION. *Apache JMeter*. Online. C1999-2023. Dostupné z: `https://jmeter.apache.org/index.html`. [cit. 2023-10-03].

[2] *HTML basics*. Online. MDN Web Docs. C 1998-2023. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics`. [cit. 2023-10-20].

[3] *Technologie pro testování kyberbezpečnosti ICT*. Online. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. 2023. Dostupné z: `https://www.vut.cz/vav/projekty/detail/33356`. [cit. 2024-04-21].

[4] *What is a low and slow attack?* Online. Cloudflare. 2024. Dostupné z: `https://www.cloudflare.com/learning/ddos/ddos-low-and-slow-attack/`. [cit. 2024-04-07].

[5] *Selenium*. Online. 2023. Dostupné z: `https://www.selenium.dev/`. [cit. 2023-10-18].

[6] *Cypress*. Online. 2023. Dostupné z: `https://www.cypress.io/`. [cit. 2023-10-18].

[7] *JMeter Testing: Everything You Need to Know*. Online. PERFORCE SOFTWARE, INC. BlazeMeter. C 2023. Dostupné z: `https://www.blazemeter.com/resources/jmeter-testing`. [cit. 2023-10-18].

[8] *How To Do Security Testing With JMeter*. Online. BlazeMeter. C 2023. Dostupné z: `https://www.blazemeter.com/blog/security-testing-jmeter#Site%20Spidering`. [cit. 2023-10-18].

[9] Experience with performance testing of software systems: issues, an approach, and case study. Online. *IEEE Transactions on Software Engineering*. 2000, roč. 26, č. 12, s. 10. ISSN 1939-3520. Dostupné z: `https://doi.org/10.1109/32.888628`. [cit. 2023-10-03].

[10] *Performance Testing Guidance for Web Applications*. Microsoft Press, 2007. ISBN 978-0735625709.

[11] BLACK, Ryan. *Soak testing*. Online. TECHTARGET. Software Quality. Dostupné z: `https://www.techtarget.com/searchsoftwarequality/definition/Soak-testing`. [cit. 2023-10-17].

[12] APACHE SOFTWARE FOUNDATION. *Component Reference*. Online. Apache JMeter: User's Manual. C1999-2023. Dostupné z: `https://jmeter.apache.org/usermanual/component_reference.html`. [cit. 2023-10-29].

[13] *Trafgen - a fast, multithreaded network packet generator*. Online. CANONICAL LTD. Ubuntu Manpage Repository. C2019. Dostupné z: `https://manpages.ubuntu.com/manpages/lunar/en/man8/trafgen.8.html`. [cit. 2023-11-01].

[14] PDF ASSOCIATION. ISO 32000, *Document management - Portable document format (PDF 2.0)*. Second edition. 2020. Dostupné také z: `https://pdfa.org/resource/iso-32000-pdf/`.

[15] *What is Java?* Online. IBM. 2023. Dostupné z: `https://www.ibm.com/topics/java`. [cit. 2023-12-04].

[16] OZANICH, Athena. *What Is Java: The Beginner's Guide to the Java Programming Language*. Online. HUBSPOT, INC. HubSpot. C2023. Dostupné z: `https://blog.hubspot.com/website/what-is-java`. [cit. 2023-10-28].

[17] APACHE SOFTWARE FOUNDATION. *How to write a plugin for JMeter*. Online. Apache JMeter. C1999-2023. Dostupné z: `https://jmeter.apache.org/usermanual/jmeter_tutorial.html`. [cit. 2023-10-31].

[18] *Overview (Apache JMeter dist API)*. Online. APACHE SOFTWARE FOUNDATION. Apache JMeter. Dostupné z: `https://jmeter.apache.org/api/index.html`. [cit. 2023-10-31].

[19] *Package javax.swing*. Online. ORACLE. Java® Platform, Standard Edition & Java Development Kit Version 17 API Specification. C1993-2023. Dostupné z: `https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/javax/swing/package-summary.html`. [cit. 2023-10-31].

[20] GILLIS, Alexander S.; LUTKEVICH, Ben a BRUNSKIL, Vicki-Lynn. *Digital signature*. Online. TechTarget. 2000 - 2023. Dostupné z: `https://www.techtarget.com/searchsecurity/definition/digital-signature`. [cit. 2023-10-07].

[21] COBB, Michael. *RSA algorithm (Rivest-Shamir-Adleman)*. Online. TechTarget. C2000-2023. Dostupné z: `https://www.techtarget.com/searchsecurity/definition/RSA`. [cit. 2023-12-04].

[22] *Diffie-Hellman key exchange (exponential key exchange)*. Online. TechTarget. C2000-2023. Dostupné z: `https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange`. [cit. 2023-12-04].

[23] DOSTÁLEK, Libor; VOHNOUTOVÁ, Marta a KNOTEK, Miroslav. *Velký průvodce infrastrukturou PKI a technologií elektronického podpisu*. 2. aktualizované vydání. Computer Press, 2009. ISBN 978-80-251-2619-6.

[24] *Introduction to Public Key Infrastructures*. Online. Springer, 2013. ISBN 978-3-642-40657-7. Dostupné z: https://link-springer-com.ezproxy.lib.vutbr.cz/book/10.1007/978-3-642-40657-7. [cit. 2023-10-25].

[25] INTERNET ENGINEERING TASK FORCE. *PKCS #10: Certification Request Syntax Specification*. V1.7. Dostupné také z: https://datatracker.ietf.org/doc/html/rfc2986.

[26] REQUEST FOR COMMENTS. RFC 5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. 2008. Dostupné také z: https://www.rfc-editor.org/info/rfc5280.

[27] EVROPSKÁ UNIE. NAŘÍZENÍ EVROPSKÉHO PARLAMENTU A RADY (EU) č. 910/2014 ze dne 23. července 2014 o elektronické identifikaci a službách vytvářejících důvěru pro elektronické transakce na vnitřním trhu a o zrušení směrnice 1999/93/ES: eIDAS. In: . Brusel, 2014. Dostupné také z: https://eur-lex.europa.eu/legal-content/CS/TXT/PDF/?uri=CELEX:32014R0910.

[28] ČESKÁ REPUBLIKA. Zákon č. 297/2016 Sb., o službách vytvářejících důvěru pro elektronické transakce. Online. In: *Zákony pro lidi.cz*. AION CS, c2010-2023. Dostupné také z: https://www.zakonyprolidi.cz/cs/2016-297.

[29] BURDA, Karel. *Aplikovaná kryptografie*. Brno: Nakladatelství VUTIUM, 2013. ISBN 978-80-214-4612-0.

[30] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. NIST Special Publication 800-131A, *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Revision 2. Dostupné také z: https://doi.org/10.6028/NIST.SP.800-131Ar2.

[31] *OpenSSL Cryptography and SSL/TLS Toolkit*. Online. OpenSSL Cryptography and SSL/TLS Toolkit. 2023. Dostupné z: https://www.openssl.org/. [cit. 2023-10-16].

[32] *OpenSSL -req manual*. Online. OpenSSL. 1999 - 2023. Dostupné z: https://www.openssl.org/docs/man1.0.2/man1/openssl-req.html. [cit. 2023-10-07].

[33] *Apache License, Versions 2.0*. Online. The Apache Software Foundation. 2023. Dostupné z: https://www.apache.org/licenses/LICENSE-2.0. [cit. 2023-10-16].

[34] *OpenSSL -pkcs12 manual*. Online. OpenSSL. 1999 - 2023. Dostupné z: https://www.openssl.org/docs/man1.0.2/man1/pkcs12.html. [cit. 2023-10-07].

[35] INTERNET ENGINEERING TASK FORCE. RFC 7292, *PKCS #12: Personal Information Exchange Syntax*. V1.1. Dostupné také z: https://www.rfc-editor.org/rfc/rfc7292#ref-21.

[36] *Class KeyStore*. Online. ORACLE. Java® Platform, Standard Edition & Java Development Kit Version 17 API Specification. C1993-2023. Dostupné z: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/security/KeyStore.html. [cit. 2023-11-01].

[37] *KeyStore*. Online. IBM CORPORATION. IBM Documentation. C2005-2023. Dostupné z: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=warnings-keystore. [cit. 2023-11-01].

[38] NETWORKING WORKING GROUP. RFC 2315, *PKCS #7: Cryptographic Message Syntax, RFC 2315*. Version 1.5. 1998. Dostupné také z: https://www.rfc-editor.org/info/rfc2315.

[39] *Jackson Dataformat XML*. Online. GitHub, Inc. C2023. Dostupné z: https://github.com/FasterXML/jackson-dataformat-xml. [cit. 2023-11-26].

[40] *Jsoup: Java HTML Parser*. Online. C2009-2023. Dostupné z: https://jsoup.org/. [cit. 2023-11-29].

[41] *Flying Saucer*. Online. GitHub. 2023. Dostupné z: https://github.com/flyingsaucerproject/flyingsaucer. [cit. 2023-11-17].

[42] *GNU Lesser General Public License, version 2.1*. Online. FREE SOFTWARE FOUNDATION, INC. GNU Operating System. C1996-2023. Dostupné z: https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html. [cit. 2023-11-17].

[43] *HTML Versus XHTML*. Online. W3Schools. C1999-2023. Dostupné z: https://www.w3schools.com/html/html_xhtml.asp. [cit. 2023-11-29].

[44] *HTML to PDF Using OpenPDF*. Online. Baeldung. 2023. Dostupné z: https://www.baeldung.com/java-html-to-pdf. [cit. 2023-11-29].

[45] THE APACHE SOFTWARE FOUNDATION. *Apache PDFBox® - A Java PDF Library*. Online. C2009-2023. Dostupné z: `https://pdfbox.apache.org/`. [cit. 2023-11-17].

[46] *OpenPDF is an open source Java library for PDF files*. Online. GitHub. Dostupné z: `https://github.com/LibrePDF/OpenPDF`. [cit. 2023-11-17].

[47] LEGION OF THE BOUNCY CASTLE INC. *Bouncy Castle*. Online. C2013. Dostupné z: `https://www.bouncycastle.org/`. [cit. 2023-11-21].

[48] *Introduction to ASN.1*. Online. International Telecommunication Union. C2023. Dostupné z: `https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx`. [cit. 2023-11-21].

[49] *OpenPDF 1.3.17 API*. Online. C2020. Dostupné z: `https://librepdf.github.io/OpenPDF/docs-1-3-17/`. [cit. 2023-11-22].

[50] SHEKYAN, Sergey. *Slowhttptest*. Online. GitHub, Inc. 2023. Dostupné z: `https://github.com/shekyan/slowhttptest`. [cit. 2024-03-28].

[51] ŠRÁMEK, Michal. *Testování zranitelností cloudových řešení útokem Slowloris*. Online, Bakalářská práce, vedoucí Pavel Šeda. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2023. Dostupné z: `http://hdl.handle.net/11012/212614`. [cit. 2024-03-28].

[52] *Archived OpenJDK General-Availability Releases*. Online. ORACLE CORPORATION. Production and Early-Access OpenJDK Builds, from Oracle. C2024. Dostupné z: `https://jdk.java.net/archive/`. [cit. 2024-05-08].

[53] OLIVERA, Fernando Rodriguez. *MvnRepository*. Online. C2006-2024. Dostupné z: `https://mvnrepository.com/`. [cit. 2024-05-08].

Seznam symbolů a zkratek

API	Application Programming Interface
ASN.1	Abstract Syntax Notation number One
CRL	Certificate revocation list
CSR	Certificate signing request
CSS	Cascading Style Sheets
CSV	Comma-separated values
DoS	Denial of Service
DDoS	Distributed Denial-of-Service
GUI	Graphical User Interface
eIDAS	nařízení Evropského Parlamentu a Rady č. 910/2014 o elektronické identifikaci a důvěryhodných službách pro elektronické transakce na vnitřním evropském trhu
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secured
ICMP	Internet Control Message Protocol
IMAP	Internet Message Access Protocol
IoT	Internet of Things
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
PDF	Portable Document Format

PNG	Portable Network Graphics
POP3	Post Office Protocol
REST	Representational State Transfer
RSA	Rivest-Shamir-Adleman
SHA256	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Seznam příloh

A Úryvky kódu implementace digitálního podpisu **65**
A.1 Metoda SignOperation.sign() 65

B Ukázkový report ze zátěžového testování **67**

C Obsah elektronické přílohy **73**
C.1 Sestavení modulu 73
C.2 Instalace modulu do Apache JMeter 73

A Úryvky kódu implementace digitálního podpisu

A.1 Metoda SignOperation.sign()

```
1  public void sign() throws IOException, CertificateEncodingException,
   ↪  OperatorCreationException, CMSException {
2      log.info("PDF signing started");
3      try (FileOutputStream outputStream = new
       ↪  FileOutputStream(getSignedPDFPath())) {
4          PdfReader pdfReader = new
           ↪  PdfReader(fileForSign.getAbsolutePath());
5          PdfStamper stamper = PdfStamper.createSignature(pdfReader,
           ↪  outputStream, '\0');
6
7          appendImage(pdfReader, stamper);
8
9          PdfSignatureAppearance signatureAppearance =
           ↪  stamper.getSignatureAppearance();
10
11         PdfSignature signature = new
           ↪  PdfSignature(PdfName.ADOBE_PPKLITE,
           ↪  PdfName.ADBE_PKCS7_DETACHED);
12         signature.setDate(new PdfDate());
13
14         signatureAppearance.setCryptoDictionary(signature);
15
16         // Preallocate space for signature
17         Map<PdfName, Integer> sizes = new HashMap<>();
18         sizes.put(PdfName.CONTENTS, SIGNATURE_LENGTH * 2 + 2);
19
20         signatureAppearance.preClose(sizes);
21
22         byte[] signedData = this.getSignature(
23             signatureAppearance.getRangeStream().readAllBytes()
24         );
```

```
25          signatureAppearance.setExternalDigest(signedData, null,
            ↪ "RSA");
26          signatureAppearance.setProvider(CRYPTO_PROVIDER);
27
28          PdfDictionary pdfDictionary = new PdfDictionary();
29          try (ByteBuffer byteBuffer =
            ↪ this.addContentPadding(signedData)) {
30              pdfDictionary.put(PdfName.CONTENTS, new
                ↪ PdfString(byteBuffer.toByteArray())
31                  .setHexWriting(true));
32          }
33
34          signatureAppearance.close(pdfDictionary);
35          log.info("PDF signing finished");
36      }
37  }
```

B Ukázkový report ze zátěžového testování

Overview of testing

For testing was used ICT tester in version 6.0.0 and connected by 10Gb/s link.
the table shows performed test scenarios.

Name of performed test	Time	Average speed	Target	Was reachable
DDoS - SYN Flood	1m 40.271s	62 Mb/s	192.168.0.105	Unreachable
slowLoris	1m 30s	Not relevant	192.168.0.105	Unreachable

If the target is unavailable for 3 % of the total attack time, the target is marked as unreachable.

Report of the testing

DDoS - SYN Flood

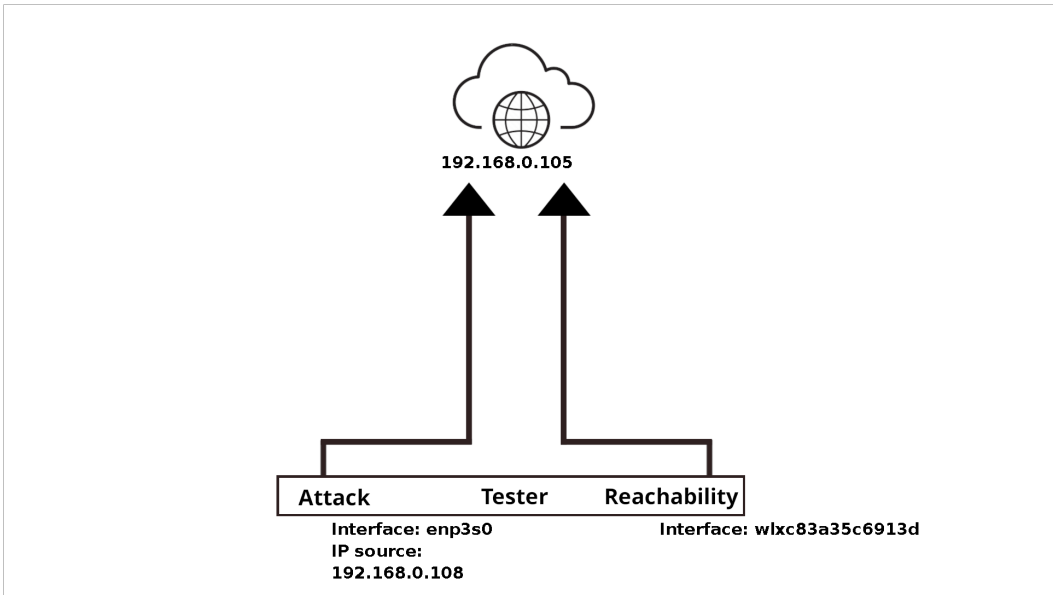
Time

Start of the test	End of the test	Time
Sat Apr 27 17:56:18 CEST 2024	Sat Apr 27 17:57:58 CEST 2024	1m 40.271s

Comment

SYN Flood against Ubuntu server using 100 Mb line

Connection scheme



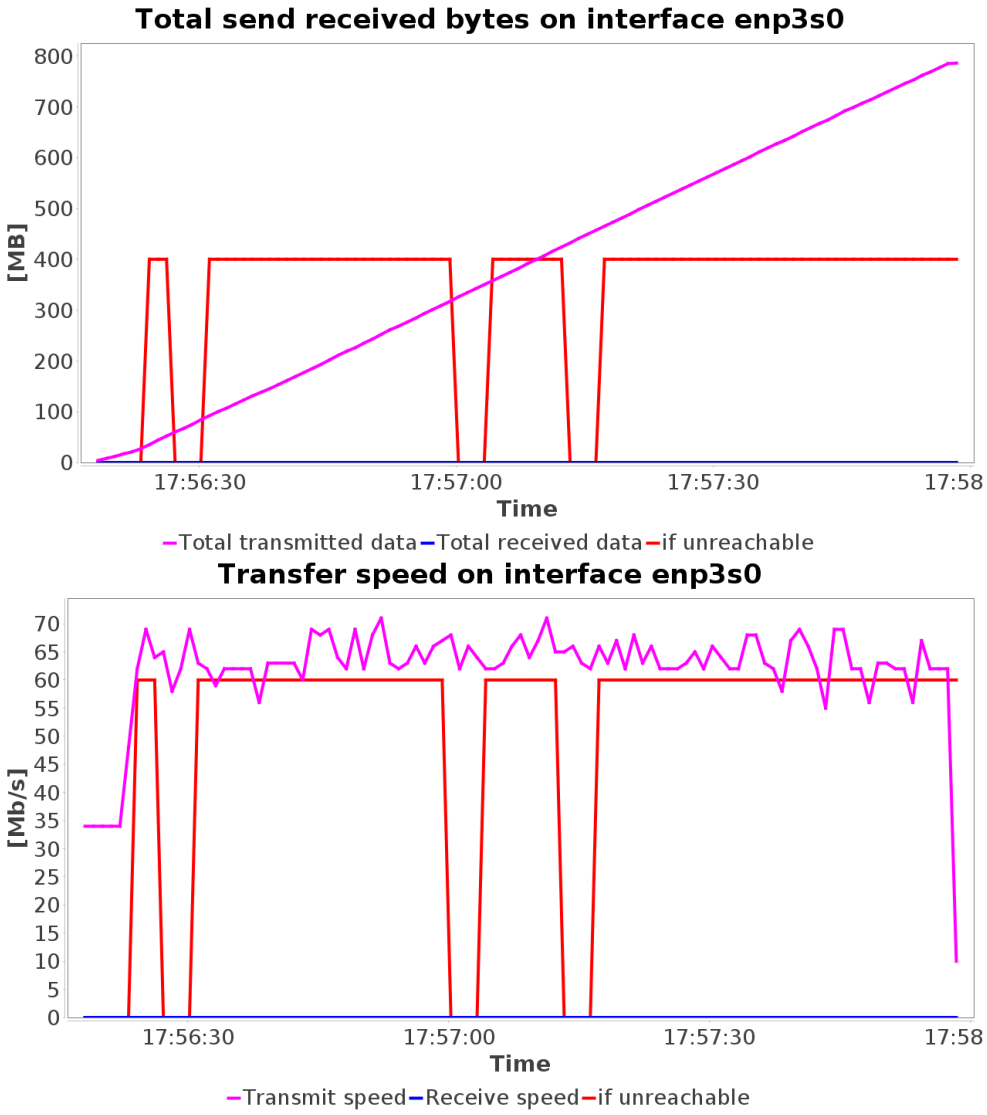
enp3s0

Tx Total [MB]	
Min:	4
Max:	786
Average:	391

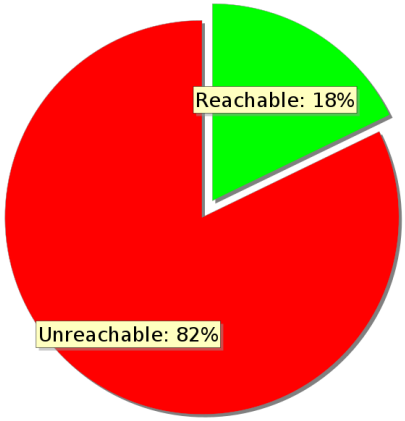
Rx Total [kB]	
Min:	0
Max:	34
Average:	12

Tx [Mb/s]	
Min:	10
Max:	71
Average:	62

Rx [kb/s]	
Min:	0
Max:	79
Average:	2


Total send received bytes on interface enp3s0
800
700
600
500
400
300
200
100
0
[MB]
17:56:30
17:57:00
17:57:30
17:58
Time
Total transmitted data
Total received data
if unreachable
Transfer speed on interface enp3s0
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
[Mb/s]
17:56:30
17:57:00
17:57:30
17:58
Time
Transmit speed
Receive speed
if unreachable

Reachability result



This graph show if the target was unreachable. Metric to evaluate reachability is response time set by user.

slowLoris

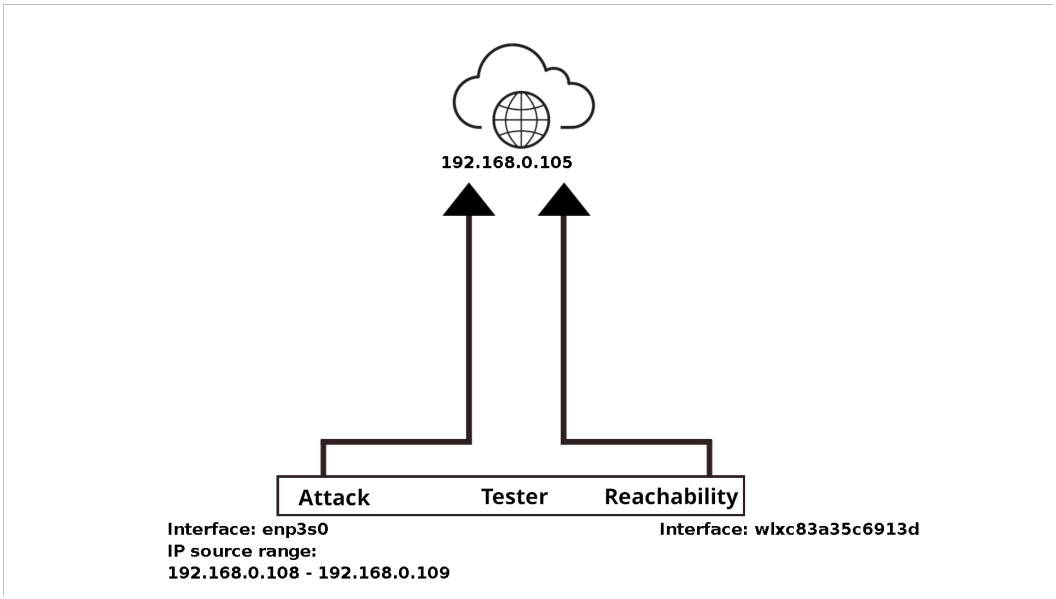
Time

Start of the test	End of the test	Time
Sat Apr 27 17:41:05 CEST 2024	Sat Apr 27 17:42:35 CEST 2024	1m 30s

Comment

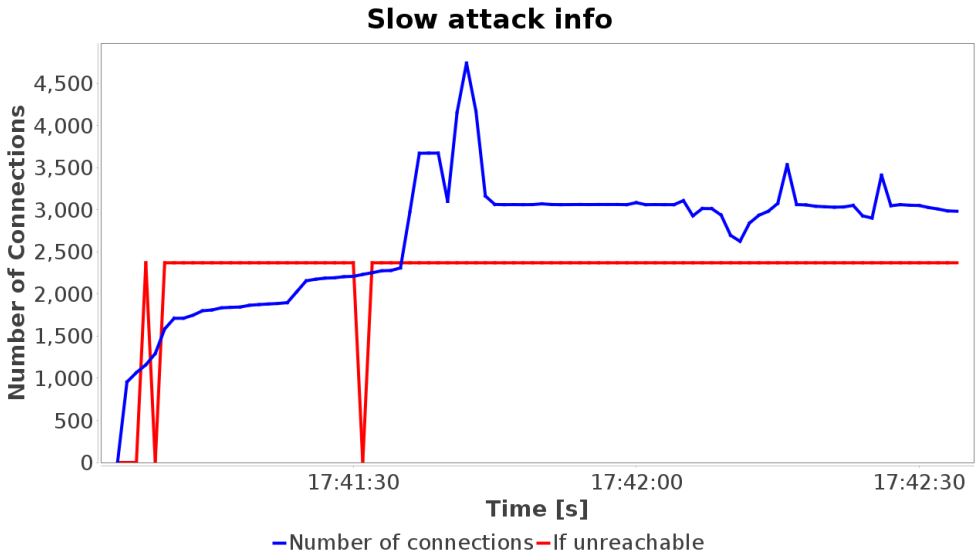
Slowloris against Nginx Web server

Connection scheme

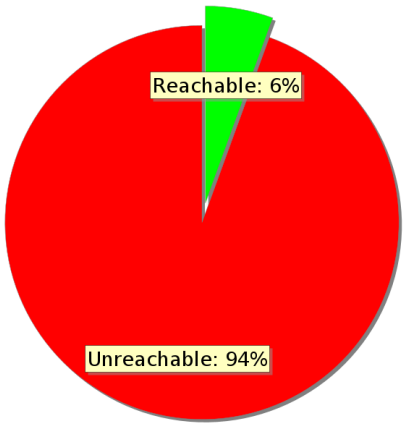


Slow attack details

This attack is SLOW HEADERS type and uses 40000 connections for testing. The test creating 1000 connection per second.



Reachability result



This graph show if the target was unreachable. Metric to evaluate reachability is response time set by user.

C Obsah elektronické přílohy

Součástí elektronické přílohy je soubor *jmeter-reportgenerator-plugin.zip* obsahující zdrojové kódy zásuvného modulu Report generátor, jehož tvorba byla primární náplní této práce.

V případě, že je modul instalován jako součást celého projektu ICT Tester [3], je jeho sestavení a instalace provedena automatickým scriptem. Níže je popsán postup pro případ, že je modul instalován samostatně.

C.1 Sestavení modulu

ZIP repozitář obsahuje pouze nesestavené zdrojové kódy. Pro sestavení je nutné mít nainstalovaný programovací jazyk Java minimálně ve verzi 17 [52]. Dále je nutné provést následující kroky:

1. Rozbalit ZIP soubor do samostatné složky.
2. Otevřít tuto složku v terminálu a spustit sestavení pomocí `./gradlew build`.
3. Po úspěšném sestavení by se měl vygenerovat soubor *ReportGenerator.jar* ve složce *jmeter-reportgenerator-plugin/build/libs*.

C.2 Instalace modulu do Apache JMeter

K provedení tohoto postupu je potřeba mít nainstalovaný program Apache JMeter [1]. Poté je potřeba provést následující kroky:

1. Přejít do adresáře, ve kterém je Apache JMeter nainstalovaný.
2. Přejít do podsložky *jmeter/lib/ext*.
3. Do této podsložky zkopírovat sestavený zdrojový kód *ReportGenerator.jar* dle předchozí podkapitoly.
4. Stáhnout a do stejné podsložky vložit všechny závislosti, které modul vyžaduje. Tyto závislosti jsou k nalezení v souboru *build.gradle* ve složce se zdrojovým kódem v sekci „dependencies“. Tyto závislosti (.jar soubory) je možné najít ke stažení například na stránce MvnRepository [53].