VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2017 Bc. Marek Sikora

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

DETEKCE SLOW-RATE DDOS ÚTOKŮ
DETECTION OF SLOW-RATE DDOS ATTACKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Marek Sikora

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Petr Blažek

BRNO 2017

ABSTRAKT

Diplomová práce je zaměřena na problematiku detekce a obrany před útoky typu Slow DoS a DDoS pomocí analýzy provozu v počítačových sítích. Čtenář je uveden do základní problematiky této specifické kategorie sofistikovaných útoků a jsou mu objasněny charakteristiky několika konkrétních útoků. Dále je zde navrhnuta sada metod pro detekci a ochranu před těmito útoky. Navržené metody jsou použity pro implementaci vlastního systému prevence průniku, který je nasazen na hraniční filtrační server počítačové sítě za účelem ochrany webových serverů před útoky z internetu. Vytvořený systém je následně testován v laboratorní síti. Prezentované výsledky testování prokazují, že vytvořený systém je schopen detekovat útoky typu Slow GET, Slow POST, Slow Read a Apache Range Header a následně ochránit webové servery před ovlivněním poskytovaných služeb.

KLÍČOVÁ SLOVA

Pomalé DoS útoky, detekce, obrana, Slow GET, Slow POST, Apache Range Header, Slow Read, analýza síťového provozu, IPS – systém prevence pruniku

ABSTRACT

This diploma thesis is focused on the detection and protection against Slow DoS and DDoS attacks using computer network traffic analysis. The reader is introduced to the basic issues of this specific category of sophisticated attacks, and the characteristics of several specific attacks are clarified. There is also a set of methods for detecting and protecting against these attacks. The proposed methods are used to implement custom intrusion prevention system that is deployed on the border filtering server of computer network in order to protect Web servers against attacks from the Internet. Then created system is tested in the laboratory network. Presented results of the testing show that the system is able to detect attacks Slow GET, Slow POST, Slow Read and Apache Range Header and then protect Web servers from affecting provided services.

KEYWORDS

Slow DoS attacks, detection, protection, Slow GET, Slow POST, Apache Range Header, Slow Read, network traffic analysis, IPS – Intrusion Prevention Systems

SIKORA, Marek. *Detekce slow-rate DDoS útoků*. Brno, 2017, 65 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Blažek

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Detekce slow-rate DDoS útoků“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové 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(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) 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(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Petru Blažkovi za odborné vedení, konzultace a podnětné návrhy k práci. Poděkování patří také mé přítelkyni a rodině, zejména rodičům, za veškerou podporu při studiu.

Brno ....

podpis autora(-ky)

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM VÝPISŮ

ÚVOD

Diplomová práce se zabývá problematikou počítačových útoků cílených na zamítnutí služby (DoS a DDoS) a obranou vůči nim. DoS útoky patři mezi jedny z nejčastějších útoků v počítačových sítích a to zejména díky jejich jednoduchosti a relativně snadné dostupnosti nástrojů pro jejich realizaci. Tyto útoky jsou zpravidla cíleny na webové servery, kde se projeví nedostupností webových stránek. Situace v této oblasti kybernetické bezpečnosti se neustále vyvíjí a proto je nutné její neustále zabývat. Objevují se stále nové metody útoku, které využívají zatím neobjevených zranitelností systémů. Vůči těmto útokům existuje několik obraných opatření, avšak ne všechny jsou natolik robustní, aby dokázaly odvrátit sofistikovanější typy útoků a reagovat na změny jejich strategie. Tato práce je zaměřena převážně na obranu před poměrně novou skupinou sofistikovanějších útoků, tzv. pomalých DoS útoků (Slow DoS). Tyto útoky se zaměřují především na protokoly aplikační vrstvy modelu ISO/OSI a vyznačují se velmi malým provozem, díky čemuž jsou hůře rozpoznatelné od legitimního provozu.

Cílem práce je návrh a realizace systému prevence průniku (anglicky: Intrusion Prevention Systems, zkráceně IPS) pro praktické využití na hraničním filtračním serveru počítačové sítě za účelem ochrany webových serverů před útoky z internetu. Tento systém je vytvořen v jazyce C na základě detekčních metod, navržených v teoretické části této práce. Vytvořený program je dále rozvinut schopností blokovat probíhající útoky a chránit webové servery před ovlivněním poskytovaných služeb. Výsledný program je následně testován v laboratorní síti. Výsledkem je funkční obranný systém, který je schopen detekovat útoky typu Slow GET, Slow POST, Slow Read a Apache Range Header a následně ochránit webové servery před ovlivněním poskytovaných služeb.

První část práce poskytuje čtenáři nutný teoretický základ k pochopení obsahu této práce. Jsou zde objasněny principy a funkčnost komunikace v počítačových sítích a hlavně samotného protokolu HTTP, který je využíván pro realizaci pomalých DoS útoků.

Druhá část práce rozebírá aktuální situaci v oblasti DoS útoků, popisuje známé typy těchto útoků a dále se zaměřuje na specifickou a relativně novou kategorii pomalých DoS útoků (Slow DoS). Je zde vysvětlena podstata těchto útoků, jejich principy a dopad na cílový systém.

Třetí část práce se zabývá obranou vůči těmto hrozbám. Popisuje základní preventivní opatření, jenž napomáhají odrazit méně nebezpečné hrozby, nebo alespoň znesnadňují provedení úspěšného útoku na cílový server. Dále jsou zkoumány možné metody odhalení probíhajících pomalých DoS útoků na základě jejich charakteristických rysů a slabin.

Následující část práce se zabývá návrhem samotného systému prevence průniku. Popisuje sestavení a funkčnost programu pro zachytávání síťové komunikace, jenž tvoří základní stavební kámen detektoru DoS útoků. Následně je zde řešeno zpracování analyzovaných dat síťové komunikace a realizace navržených metod detekce jednotlivých útoků. Poté je popsána realizace obranných mechanismů, díky kterým je zjištěný útok odražen.

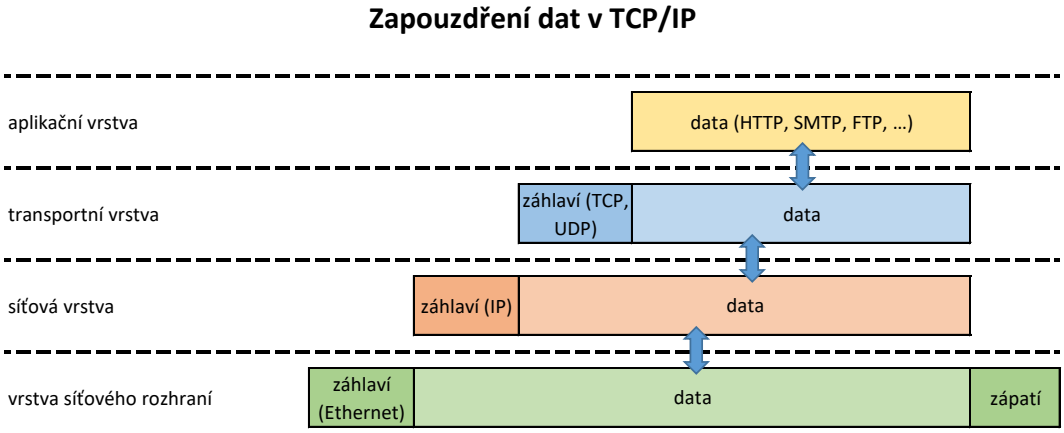
V další části práce je popsáno, jak lze vytvořený systém zprovoznit a používat. Jsou zde uvedeny všechny potřebné knihovny a příkazy pro instalaci v operačním systému linuxového typu. Dále je zde vysvětleno, jak lze přizpůsobit citlivost detektoru pro konkrétní webový server a typ provozu.

Poslední část práce se zabývá testováním vytvořeného systému. Je zde popsáno testovací prostředí a nástroj pro generování útoku. Nejprve je otestován samotný generátor a jeho funkčnost. Následně se testuje vytvořený systém prevence průniku, jeho účinnost a vliv na dostupnost cílového webového serveru. Výsledky jsou zobrazeny v grafech. V poslední fázi je zkoumáno chování detekčního systému pro různé scénáře probíhajícího útoku.

1 CHARAKTERISTIKA SÍŤOVÉHO PROVOZU

Komunikace mezi zařízeními v počítačových sítích se řídí podle pravidel definovaných referenčním modelem ISO/OSI, anebo v praxi častějším TCP/IP. Referenční model obsahuje několik vrstev síťové komunikace, kde každá vrstva obsahuje sadu protokolů zpracovávajících konkrétní úkol či rozšiřují služby komunikační sítě. Při probíhající síťové komunikaci postupují data přes jednotlivé vrstvy, které zpracovávají např. směrování, zabezpečení a zajištění spolehlivosti, až ke konkrétním uživatelským procesům.

Síťový provoz obsahuje kromě uživatelských dat také hlavičky protokolů z každé vrstvy referenčního modelu, proto je při analýze síťového provozu důležité tyto principy dobře znát.



Obr. 1.1: Zapouzdření dat v modelu TCP/IP

Data, vytvořená uživatelským procesem, musí před odesláním projít přes všechny vrstvy referenčního modelu, kde se jim v každé vrstvě přidá hlavička s daty konkrétního protokolu (obr. 1.1) [1]. Na straně příjemce pak data prochází přes vrstvy v opačném pořadí, kdy si protokoly jednotlivých vrstev přečtou a odeberou hlavničku, kterou jim předal protokol téže vrstvy na straně odesílatele. Data se poté bez této hlavičky předají vyšší vrstvě, která provede zpracování stejným způsobem, dokud se původní uživatelská data nedostanou až k uživateli.

Nejnižší vrstva modelu TCP/IP je vrstva síťového rozhraní. Ta umožňuje přístup k přenosovému médiu. Na této vrstvě je definováno hned několik standardů, avšak pro služby sítě Internet se typicky používá technologie Ethernet. Ethernet zajišťuje komunikaci a adresaci mezi sousedními zařízeními na základě MAC adres, dále nabízí funkce pro detekci a opravu chybně přijatých dat.

Síťová vrstva zajišťuje především adresaci v sítích a mezi sítěmi. Nejznámějším

protokolem síťové vrstvy je protokol IP, který provádí směrování dat (paketů) na základě IP adres síťových rozhraní. Strukturu záhlaví IP znázorňuje obrázek 1.2 [2].

Záhlaví protokolu IP

bity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0 - 31	verze				délka záhl.				typ služby								celková délka															
32 - 63	identifikátor IP datagramu																příznaky			posunutí fragmentu od začátku											
64 - 95	doba života (TTL)								protokol vyšší vrstvy								kontrolní součet IP záhlaví															
96 - 127	zdrojová IP adresa																															
128 - 159	cílová IP adresa																															
160 - 191	volitelné položky																															
160/192 +	DATA																															

Obr. 1.2: Záhlaví protokolu IP

Transportní vrstva poskytuje vyšším vrstvám transportní spojení s požadovanou spolehlivostí a také adresuje data jednotlivým uživatelským procesům na základě přiděleného čísla portu. Pro aplikace, jenž nevyžadují minimální zpoždění přenosu, obvykle zajišťuje přenos protokol TCP, který poskytuje spolehlivý přenos dat a předávání dat ve správném pořadí na základě pořadových čísel jednotlivých segmentů. Záhlaví protokolu TCP znázorňuje obrázek 1.3 [3].

Záhlaví protokolu TCP

bity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0 - 31	zdrojový port																cílový port															
32 - 63	pořadové číslo odesílaného bajtu																															
64 - 95	pořadové číslo přijatého bajtu																															
96 - 127	posun dat				rezerva						příznaky						velikost okna															
128 - 159	kontrolní součet TCP zahlaví																ukazatel naléhavých dat															
160 - 191	volitelné položky																															
160/192 +	DATA																															

Obr. 1.3: Záhlaví protokolu TCP

Na nejvyšší aplikační vrstvě pracuje celá řada protokolů, které představují koncové uživatelské funkce, nástroje či aplikace. Mezi tyto protokoly se řadí např. HTTP, DNS, FTP, SSH, Telnet a SMTP.

1.1 HTTP protokol

HTTP je internetový protokol určený pro přenos hypertextových dokumentů a informací. Funguje na principu dotaz – odpověď mezi klientem a serverem. Existuje několik typů dotazů, nejčastěji se však používají dotazy typu GET a POST, které se zároveň stávají častým nástrojem pro DoS útoky na aplikační vrstvě.

Dotaz typu GET se používá pro sdělení serveru, aby klientovi poslal požadovaný soubor, objekt či dokument (obvykle webovou stránku). Validní GET požadavek by měl obsahovat typ požadavku (tedy GET), verzi protokolu a URL požadovaného objektu. Jednotlivé položky jsou odděleny znaky `\r\n`, tedy odřádkováním a celý požadavek je ukončen dvojitým odřádkováním (znaky `\r\n\r\n`). Kromě těchto povinných atributů může požadavek obsahovat ještě další nepovinné položky [4].

Požadavek typu POST slouží k odesílání dat klienta na sever, typicky při odesílání formuláře vyplněného na webových stránkách. Mezi povinné atributy POST požadavku patří typ požadavku, URL objektu, kterému mají být data předána a dále informaci o velikost přenášených dat. Tyto položky v hlavičce dotazu jsou opět odděleny odřádkováním a ukončeny dvojitým odřádkováním. Samotná data se poté nacházejí hned za ním [4].

Server na tyto požadavky odpovídá zprávami s návratovým kódem v hlavičce (trojciferné dekadické číslo) a případně s dalšími daty v těle odpovědi. Návratové kódy upřesňují, jakým způsobem byl požadavek zpracován – jestli byl vyřízen, zamítnut, či zda došlo k chybě.

Příklad požadavku a odpovědi protokolu HTTP při požádání zobrazení webové stránky je zde:

```
GET / HTTP/1.1\r\n
Host: www.seznam.cz\r\n
Connection: keep-alive\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) Chrome/54 ... \r\n
Accept: text/html,application/xhtml+xml,application/xml; ... \r\n
Accept-Encoding: gzip, deflate, sdch\r\n
Accept-Language: cs-CZ,cs;q=0.8\r\n
\r\n

HTTP/1.1 302 Moved Temporarily\r\n
Server: nginx\r\n
Date: Mon, 05 Dec 2016 13:41:45 GMT\r\n
Content-Type: text/html\r\n
Content-Length: 154\r\n
```

```
Connection: keep-alive\r\n
Location: https://www.seznam.cz/\r\n
\r\n
<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

2 DOS A DDOS ÚTOKY

V počítačových sítích existuje celá řada hrozeb, jejichž cílem může být krádež důvěrných dat, identity, odposlouchávání komunikace, nebo zamezení dostupnosti webových, poštovních, databázových, či jiných serverů. Útoky typu Denial of Service (DoS) a Distributed Denial of Service (DDoS) jsou specifické skupiny počítačových útoků, které se snaží způsobit nedostupnost služeb legitimním uživatelům, či snížit kvalitu, popřípadě rychlost poskytovaných služeb [5]. Tyto útoky patři mezi jedny z nejčastějších útoků v počítačových sítích. Zpravidla jsou cíleny na webové servery, kde se projeví nedostupností prezentovaných webových stránek.

Úspěšný DoS útok tedy nesmí uniknout pozornosti uživatelů na rozdíl od jiných počítačových útoků, které cílí např. na narušení integrity a důvěrnosti dat, jako je krádež citlivých údajů, odposlouchávání komunikace, popřípadě neautorizovaný vnik do systému oběti, kdy se útočník snaží o co nejmenší pozornost [5]. Zároveň však musí být příčiny nedostupnosti co nejlépe skryty mezi legitimním provozem, aby se řízený útok nedal jednoduše identifikovat a odvrátit.

2.1 Typy DoS útoků

Existují dva základní typy DoS útoků, a to útoky využívající zranitelnost systému a záplavové (flood) útoky. Tyto útoky je možné dále dělit do podkategorií, jako např. útoky na protokol, aplikační útoky, nebo útoky na zdroje [5].

Pokud je útok veden ke společnému cíli z mnoha často zneužitých stanic, je takový útok řazen do skupiny DDoS, neboli distribuované zamítnutí služby. V takovém případě je dosaženo zesílení účinku útoku a zároveň se stává obtížnější pro detekci a identifikaci zdroje útoku, jelikož provoz z těchto počítačů se může v mnoha ohledech jevit jako legitimní.

Útoky zaměřené na zneužití zranitelnosti hledají nedokonalosti, chyby či neošetřené situace, které se mohou vyskytovat v jednotlivých protokolech nebo i v celé implementaci cílových systémů. Tyto útoky jsou velmi nebezpečné, jelikož k úspěšnému útoku obvykle stačí jen několik málo paketů.

Záplavové útoky fungují tak, že cíl útoku je zaplaven požadavky, které oběť nestíhá zpracovávat. Dojde k vyčerpání zdrojů jako je procesor, paměť nebo propustnost sítě, a příchozí provoz se začne zahazovat. V některých případech může přístroj oběti zcela zkolabovat. Při útoku není obvykle nutné nijak modifikovat pakety, důležité je pouze to, aby jich bylo zasláno v jednu chvíli co největší množství. Tyto útoky jsou snadno implementovatelné a dostupné, proto patří mezi ty nejčastěji používané [5].

V případě útoků na protokol je využíváno nedokonalostí, či chyb implementaci konkrétního protokolu, díky čemuž je možné narušit fungování celého systému např. výměnou nestandardních nebo nekompletních dat. Útoky na konkrétní aplikace se snaží vyčerpat maximální počet dotazů, které je aplikace schopná za určitý čas zpracovat. Při útocích na zdroje se útočník snaží dosáhnout přetížení jednoho zdroje a tím dosáhnout vyřazení celého systému.

Útoky lze také klasifikovat na základě vrstvy síťového modelu ISO/OSI nebo TCP/IP, na kterou se daný útok zaměřuje. Dříve se útoky soustředili spíše na nižší vrstvy, hlavně na síťovou (např. ICMP flood, Smurf, Teardrop, Ping flood), transportní (např. TCP SYN flood, UDP flood, Land attack), případně na linkovou vrstvu (MAC flood). Jejich cílem bylo vyčerpání zdroje, či šířky pásma, díky existenci limitů protokolů na těchto vrstvách [4] [6]. Avšak s rostoucí mírou povědomí o těchto útocích roste také míra zabezpečení těchto systémů. Útočníci stále přicházejí s novými typy útoků, které se nyní stále více zaměřují na nejvyšší aplikační vrstvu modelu ISO/OSI. V těchto případech se tak cílem nestává vyčerpání dostupné šířky pásma, nýbrž vyčerpání zdrojů systému, nebo konkrétní služby. Zásadní je také fakt, že při těchto útocích není potřeba takového počtu paketů, jako pro zahlcení sítě. Díky tomu se průběh útoku více podobá běžnému provozu a je obtížnější ho detekovat.

2.2 Pomalé DoS útoky

Pomalé DoS útoky (Slow DoS) tvoří poměrně novou kategorii DoS útoků, která kombinuje charakteristiky obou základních kategorií (útoky záplavové a využívající zranitelnosti). Tyto útoky jsou cíleny na aplikační vrstvu síťového modelu ISO/OSI a vyznačují se velmi malým provozem. Díky malému množství a nízké rychlosti zasílání dat a validnímu využití protokolů nižších vrstev modelu ISO/OSI lze obejít klasické detekční systémy, jelikož se provoz lépe zamaskuje mezi provoz běžných uživatelů.

Cílem pomalých DoS útoků je zaplnit vstupní fronty webového serveru, čímž dojde k přetížení serveru, zahazování dalších příchozích požadavků a server se stane nedostupným. Stejného výsledku lze dosáhnout i záplavovým DDoS útokem. Jenže v takovém případě je zvýšená pravděpodobnost detekce a odražení útoku. Pro útočníka je tedy výhodné využít strategií zasílání paketů pomalou rychlostí.

Tyto útoky zpravidla není potřeba spouštět distribuovaně. K provedení úspěšného útoku (vyřazení webového serveru) stačí jen několik málo paketů z jedné stanice.

Při spuštění pomalého DoS útoku musí nejprve dojít k navázání kompletního TCP spojení. Na nižších vrstvách je tedy komunikace stejná jako u legitimních

uživatelů a proto je tento útok schopen vyhnout se detektorům distribuovaných DoS útoků, přičemž dosáhne stejného výsledku.

Pro úspěšný slow DoS útok je důležité, aby navázané spojení mezi útočníkem a obětí zůstalo obsazené a nebylo možné dále přiřazovat obsazené výpočetní prostředky serveru legitimním uživatelům. Během navázaného spojení zpravidla nedochází k přenosu dat. Spojení web serveru je tedy obsazené, ale nevyužité. Dále je nutné, aby útočník komunikoval se serverem v dostatečné míře, aby nedošlo k uvolnění spojení. Útočník proto musí po navázání spojení zasílat tzv. udržovací požadavky (sloužící k udržení komunikačního kanálu se serverem), jejichž časový odstup závisí především na ochranných časovačích nastavených na webovém serveru. Snahou je zasílat udržovací požadavek těsně před vypršením časového limitu pro dané spojení. Po udržovacím paketu je spojení opět nečinné, ale prostředky serveru jsou stále obsazené.

Jelikož útok neprobíhá za pomoci nepřetržitého toku dat, je možné jej rozdělit na dvě fáze. A to na dobu zasílání dat a fázi neaktivity útočníka (kdy data zasílána nejsou). Z toho důvodu je průběh útoku podobný běžné komunikaci, tedy v síťovém provozu hůře rozpoznatelný. Tímto způsobem jsou postupně obsazovány všechny volné obslužné kapacity oběti do doby, než jsou prostředky serveru plně obsazeny a server začne veškerý další příchozí provoz zahazovat.

Díky tomu, že nejsou pomalé útoky příliš náročné na výkon stroje útočníka, mohou být jako zdroje útoku využita i chytrá mobilní zařízení, jejichž míra zneužití k DoS útokům v posledních letech stoupá.

Pomalé DoS útoky lze dělit do tří kategorií [5]:

- DoS útoky s čekajícími požadavky;
- DoS útoky s dlouho trvající odpovědí;
- vícevrstvé DoS útoky.

Útoky s čekajícími požadavky využívají vlastnosti HTTP protokolu, která umožňuje rozprostírat HTTP data do více TCP segmentů a odesílat tak pakety s nekompletními požadavky. Pro úspěšný útok je však nutné, aby server alokoval zdroje pro obsloužení dotazu již v době, kdy je přijat i nekompletní požadavek. Tyto útoky jsou tedy charakteristické odesíláním nekompletních HTTP požadavků, které cílový server zahltí aniž by došlo k jejich obsloužení [5]. Příkladem takového útoku může být například útok s názvem Slow GET (Slowloris), který funguje na principu zasílání nekompletního HTTP požadavku GET.

Útoky s dlouhotrvající odpovědí se vyznačují zasláním validního HTTP požadavku, avšak velmi pomalým zasíláním přidružených dat. Server očekává příjem dat určité velikosti, ale útočník zasílá data po velmi malých dílech. Tím je server donucen stále naslouchat na otevřeném spojení a čekat na příjem všech dat, což při větším počtu takových spojení vede k vyčerpání obslužných kapacit serveru. Charakteris-

tickým rysem těchto útoků je tedy velmi pomalý přenos dat. Příkladem takového útoku může být například útok Slow POST, který využívá HTTP požadavek POST.

2.2.1 Slow GET

Útok Slow GET, známý také jako Slow Headers nebo Slowloris, je útok zaměřující se na protokol HTTP pomocí podvržení požadavku GET. Útočník naváže TCP spojení a pošle cílovému web serveru HTTP požadavek GET, který však není validně ukončen, tedy neobsahuje dvojité odřádkování `\r\n\r\n`. Příklad takového požadavku je zde:

```
GET / HTTP/1.1\r\n
Host: 10.0.0.41\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; ... MSOffice 12)\r\n
Content-Length: 42\r\n
```

Útočník se následně odmlčí, zatímco web server stále čeká na dokončení požadavku. V této chvíli se na web serveru spustí časovač, který čeká na další část požadavku. Po jeho vypršení web server prohlásí spojení za mrtvé, uzavře jej a systémové prostředky přidělí dalšímu příchozímu požadavku. Útok Slow GET je však navržen tak, aby těsně před vypršením tohoto časovače odeslal na web server další část nekonečného požadavku (tzv. udržovací požadavek), aby web server tento časovač resetoval a spojení tak mohlo zůstat dále otevřené. Příklad obsahu udržovacího požadavku:

```
X-a: b\r\n
```

Tímto způsobem útočník inicializuje velké množství spojení, dokud nedojde k úplnému vyčerpání zdrojů web serveru a ten přestane odpovídat na všechny legitimní požadavky. Udržovací paket obvykle obsahuje náhodně vygenerovaný a nesmyslný obsah o minimální velikosti (zpravidla jen pár znaků). Při spuštění útoku je důležité, aby byla správně nastavena frekvence zasílání udržovacích paketů, která se postupně přizpůsobí časovači na web serveru testováním před samotným útokem.

Útok Slow GET není efektivní na všech typech webových serverů. Neovlivní např. IIS6, IIS7, lighttpd, Squid, Nginx, ale postihuje nejvíce rozšířený Apache 1.x a Apache 2.x [7].

Tento typ útoku poprvé prezentoval Robert Hansen (zvaný RSnake), který jej implementoval v jazyce Perl jako skript s názvem Slowloris [5].

2.2.2 Slow POST

Útok Slow POST využívá HTTP požadavek POST, který se standardně používá k odesílání dat na webový server (typicky data z webového formuláře). Na rozdíl od útoku Slow GET je záhlaví požadavku ukončeno standardním dvojitým odřádkováním. Záhlaví obsahuje pole `Content-length`, ve kterém je specifikována velikost přenášených dat, které mají následovat v datové části požadavku. Při útoku je v tomto poli záměrně vyplněna enormně vysoká hodnota, zatímco velikost přenášených dat je minimální. Tímto způsobem je server donucen stále čekat na příchod dalších dat o celkové velikosti specifikované v položce `Content-length`. Spojení tedy zůstává stále otevřené. Úvodní POST požadavek tohoto útoku vypadá následovně:

```
POST /form_action.php HTTP/1.1\r\n
Host: 10.0.0.41\r\n
Connection: keep-alive\r\n
Content-Length: 100000000\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; ... )\r\n
\r\n
firstname=
```

Následuje odmlčení útočníka, který čeká do doby těsně před vypršením časovače pro zrušení spojení. Poté útočník odešle další malou část dat, jenž představují další část neodeslaného formuláře, aby resetoval časovač a spojení mohlo být nadále otevřené [8]. Obsah těchto udržovacích požadavků bývá zpravidla náhodný a do nekonečna se opakující. Ve většině případů jde pouze o jeden znak.

Tímto způsobem útočník inicializuje další a další spojení, dokud neobsadí všechny dostupné zdroje web serveru a ten se poté stane pro ostatní uživatele nedostupný.

Útok typu Slow POST využívá nástroj zvaný R.U.D.Y. (zkráceně Are You Dead Yet?).

2.2.3 Apache Range Header

Útok typu Apache Range Header využívá parametru `byte range` v záhlaví HTTP požadavku, který je používán ke specifikaci rozsahu dat, které chceme ze serveru získat. Při útoku útočník zasílá požadavky, které požadující velké množství rozsahů dat. Očekává se, že si server alokuje místo v paměti pro každý z rozsahů zvlášť, čímž dojde k brzkému vyčerpání operační paměti. Díky dalšímu parametru `Accept-Encoding: gzip` v útočníkově HTTP požadavku útočník vynucuje odpověď v komprimované podobě, což vede mimo jiné také k vytížení procesoru [8].

Příklad podvržené hlavičky útoku typu Apache Range Header vypadá následovně:

```
HEAD / HTTP/1.1
Host: [...]
Range: bytes=0-,5-0,5-1,5-2,[ ... ],5-1296,5-1297,5-1298,5-1299
Accept-Encoding: gzip
Connection: close
```

Využití tohoto mechanismu k vedení DoS útoku je možné ze dvou důvodu. Díky chybě v návrhu samotného HTTP protokolu musí server odpovědět na každý požadovaný rozsah samostatně a v pořadí v jakém jsou v hlavičce zapsány. Server tak zasílá velké množství stejných dat opakovaně jen na základě jediného dotazu útočníka. Druhým důvodem je neefektivní práce s pamětí u serveru Apache, který provede alokaci prostor v operační paměti pro každý požadovaný rozsah z HTTP požadavku samostatně, i když se rozsahy vzájemně překrývají nebo jsou ve více dotazech požadována stejná data [5].

Útok tohoto typu je znám jako skript v jazyce Perl s názvem *killapache.pl*, který byl zveřejněn uživatelem známým pod přezdívkou Kingcope [8].

Tato zranitelnost by měla být u serveru Apache opravena již od verze 2.2.20 [5].

2.2.4 Slow READ

Při útoku Slow READ útočník odešle kompletní HTTP požadavek, kterým požaduje zaslání nějakého souboru (např. obrázku), nebo obsahu webové stránky. Server na tento požadavek odpovídá zasláním požadovaných dat. Technikou omezení okna (WIN) v protokolu TCP je však dosaženo toho, že je server donucen odpověď odesílat po částech.

Velikost okna stanovuje počet bajtů, které může odesílatel odeslat bez čekání na potvrzení příjmu od příjemce. Velikost okna je stanovena při navazování TCP spojení a během přenosu dat se může měnit v závislosti například na spolehlivosti přenosu v síti nebo rychlosti čtení dat.

Útočník během přenosu zneužije tuto funkci k tomu, aby co nejvíce snížil velikost okna, čímž přinutí server postupně odesílat jen malou část původních dat. Tímto je dosaženo čtení dat pomalým způsobem (odtud Slow READ) [5]. Útočník se zároveň snaží navázat takových spojení co nejvíc, aby postupně obsadil všechny volné prostředky web serveru. Tím je způsobena nedostupnost serveru legitimním uživatelům.

Jedním z nástrojů pro Slow READ útok je SlowHTTPTest, implementovaný Sergeyem Shekyanem [8].

3 OBRANA PROTI DOS ÚTOKŮM

Vzhledem k tomu, že nové útoky neustále vznikají, neexistuje žádné univerzální obranné řešení. Útočníci stále vymýšlí nové metody a útoky, které využívají zatím neobjevených zranitelností síťových protokolů a mechanismů, k úspěšnému útoků. Proto je důležité neustále sledovat situaci a vývoj kybernetické bezpečnosti ve světě a v neposlední řadě udržovat aktuálnost veškerého používaného software [6].

Rostoucí riziko DoS útoků je způsobeno hlavně díky snadnému ovládání nástrojů, které jsou relativně dobře dostupné. Tím se zvyšuje počet potencionálních útočníků [7].

Obrana vůči některým útokům může být velmi obtížná, jelikož mnoho z nich je problematické identifikovat a odfiltrovat od legitimního provozu. To platí obzvláště u pomalých DoS útoků, u kterých nedochází k enormnímu nárůstu příchozích paketů, ale generovaný provoz je velmi podobný tomu legitimnímu jako např. u klienta s pomalým internetovým připojením. Další komplikací je veliké množství paketů, které je třeba analyzovat, což přináší velké nároky obranných mechanismů.

Ochranou vůči útokům na aplikační protokoly se musí zabývat také vývojáři samotných služeb, v tomto případě webových serverů. V důsledku toho se úroveň zabezpečení liší u jednotlivých řešení webových serverů (Apache, lighttpd, Nginx, IIS, atd.) [7].

V současnosti existuje celá řada řešení pro obranu vůči DoS a DDoS útokům, např. v podobě bezpečnostních modulů a aktualizací pro servery nebo také systémů prevence/detekce průniku. Některá řešení však nemusí být schopna zachytit nově vznikající či více sofistikované útoky. Mnohá bezpečnostní opatření jsou navíc založena pouze na hlídání počtu otevřených spojení z jedné IP adresy, díky čemuž jsou schopny pouze zmírnit rozsah útoku, avšak neubrání se distribuované formě útoku. Z toho důvodu je na místě využití hlubší a specializovanější analýzy síťové komunikace pro detekci možných hrozeb. Existuje také mnoho specializovaných firem zabývajících se komplexní ochranou před útoky z internetu, jenž svá řešení přizpůsobují na míru konkrétním serverům.

Mimo to je zde několik zásad, které by měly být preventivně implementovány na každém serveru, a které mohou zamezit různým nevyžádaným situacím. Ať se jedná o neoprávněný přístup k prostředkům, službám, či o výpadky služeb způsobené různými druhy DoS útoků. Jde především o [6]:

- Používání filtrů provozu s cílem zabránit zahlcení systému nebo průchodu útočících paketů.
- Využít maximální možné výpočetní prostředky serveru, aby zvládl obsloužit co nejvíce uživatelů.
- Použití vyvažování zátěže pro navýšení šířky pásma na kritických místech

v síti, či použití většího množství záložních serverů.
- Využívání klamných serverů, které se snaží nalákat útočníka a odklonit tak jeho pozornost od pravého systému.
- Zakázání nepoužívaných služeb.
- Používaní aktuálního software a posledních bezpečnostních záplat.

3.1 Obrana proti pomalým DoS útokům

Pro obranu vůči pomalým DoS existuje několik základních pravidel. Jelikož se jedná o útoky na aplikační vrstvu s cílem vyčerpat dostupné zdroje, server lze nastavit tak, aby některé z těchto útoků odrazil již v prvopočátku.

Pro co nejvyšší pasivní bezpečnost web serveru by měl být server nastaven tak, aby dokázal obsloužit co největší počet současně připojených uživatelů. Útočník tak bude muset otevřít větší množství spojení. Současně by měl mít server nastavený maximální počet otevřených spojení na jednoho uživatele, dále maximální počet paketů, v nichž lze serveru předat jeden HTTP požadavek a v poslední řadě omezit čas, po který bude server čekat na další část požadavku. U web serverů Apache je obzvlášť důležité změnit výchozí hodnotu časovače pro uzavření neaktivního spojení, protože ve výchozím nastavení dosahuje tento časovač několika minut [9].

Mezi další možné techniky obrany patří i sledování vytížení vlastních zdrojů a předpovězení doby, kdy nastane přetížení nebo také využívání statistik pro porovnávání signatur síťového provozu.

Výše zmíněné metody slouží pro pasivní ochranu webových serverů. Pomocí nich lze docílit obtížnějšího dosažení cíle útoků, znesnadnit jejich nasazení či alespoň způsobit větší zatížení útočníkova počítače. I přes to je však možné některé útoky úspěšně aplikovat, především při pečlivějším nastavení parametrů útoku. Proto jsou níže popsány další metody detekce pomalých DoS útoků, které jsou využity v praktické části této práce. Tyto detekční metody sledují provoz v reálném čase a kontrolují parametry komunikace a validitu požadavků.

Pro detekci útoků typu Slow HTTP GET (Slowloris) je vhodné monitorovat následující parametry:

- Počet aktivních připojení navázaných jedním uživatelem. Klasický uživatel si při prohlížení webu v jednu chvíli otevře přibližně méně než 10 aktivních oken. Rozhodně si jich však neotevře desítky či stovky.
- Počet přijatých paketů na jeden HTTP GET požadavek. U běžných webových stránek se kompletní požadavek vejde do jediného paketu, proto pokud server obdrží hned několik paketů jednoho stále nedokončeného požadavku, může jít útok cílený na vyčerpání zdrojů serveru.

- Časové rozložení mezi příchozími pakety. V běžném provozu přichází pakety s exponenciálním rozložením. Pokud pakety od některého z uživatelů přicházejí s výrazně jiným rozložením, jako např. s konstantním, nebo s rozložením exponenciálním, jehož parametr střední hodnoty je výrazně vyšší, než průměrný parametr střední hodnoty, jedná se pravděpodobně o útočníka [6].
- Velikost, obsah a validita požadavku. Tyto parametry se sice mohou měnit v závislosti na provozované službě, nicméně obecně lze říct, že pokud server obdrží několik po sobe jdoucích a nevalidních požadavků, může jít pravděpodobně o útočníka. Avšak je nutné závěr vyhodnotit s jistou rezervou. Takový provoz může totiž generovat i legitimní uživatel s pomalým internetovým připojením. Podezřelé spojení by tedy mělo být dále sledováno.

Metody pro detekci útoků typu Slow POST vychází z výše zmíněných metod, avšak v některých případech se hlídají i jiné parametry HTTP požadavku. Pro analýzu útoku Slow POST lze tedy navrhnout navíc sledování následujících parametrů provozu:

- Počet souborů nahraných jedním uživatelem. Obyčejný uživatel obvykle nenahraje desítky či stovky souborů denně.
- Pokles přenosové rychlosti. Obyčejný uživatel pravděpodobně nebude soubor posílat po několika málo bitech.
- Časové rozložení mezi jednotlivými částmi souboru. Podobně jako u HTTP GET se očekává exponenciální časové rozložení s průměrně velkým parametrem střední hodnoty [6].
- Obsah a velikost zasílaných dat. Pokud server přijme několik velkých a nečitelných souborů, případně data s náhodným obsahem, může se jednat o útok s cílem vyčerpání zdrojů serveru.

Útok typu Apache Range Header je specifický nevalidní hodnotou položky `Range` v záhlaví HTTP požadavku. Útočící požadavek lze tedy jednoznačně určit na základě těchto kritérií:

- Velikost pole `Range` je enormně vyšší, než u běžných požadavku (např. 200 a více bajtů).
- Pole `Range` obsahuje velké množství požadovaných rozsahů dat, které se navíc kryjí.

Pro detekci útoku Slow Read je vhodné monitorovat tyto parametry:

- Vývoj velikosti okna daného TCP spojení. Pokud velikost okna klesne pod určitou mez, může jít o snahu pozdržet odesílání dat a ponechat spojení otevřené. Tato situace sama o sobě nemusí znamenat útok, pouze slouží jako identifikátor podezřelých spojení. Pro vyhodnocení situace se tato spojení musí nadále podrobněji sledovat.
- Množství otevřených spojení z jedné IP adresy s velmi malou velikostí TCP

okna.

- Pokles přenosové rychlosti. Pokud rychlost odesílání dat klesne k hodnotám blížícím se nule, jde o důvodné podezření na útok Slow Read.
- Ukončení přenosu dat v očekávaném čase.
- Množství přijatých segmentů s velmi malou velikostí TCP okna. Pokud spojení nevykazuje žádné změny a nadále zůstává téměř nehybné, tak naprosto zbytečně obsazuje prostředky serveru.

Tyto metody by měli být dostatečně robustní pro odhalení přítomnosti pomalých DoS útoků typu Slow GET, Slow POST, Apache Range Header a Slow Read. Pokud se podaří odhalit probíhající útok a IP adresy, ze kterých tyto útoky přichází, je v první řadě nutné útočníkovi zamezit další komunikaci s web serverem, abychom včas ochránili web server před přetížením, který tak bude moct nadále obsluhovat legitimní uživatele. Zároveň by mělo neprodleně dojít k uvolnění obsazených prostředků serveru, čímž se anuluje veškerý vliv útoku na cílový server.

4 IMPLEMENTACE VLASTNÍHO IPS

Táto část práce se zabývá tvorbou vlastního systému prevence průniku (IPS). Tento systém má za úkol detekovat pomalé DoS útoky typu Slow GET, Slow POST, Apache Range Header a Slow Read a v případě detekce útoku ochránit webové servery před výpadkem.

Vytvořený systém je určen pro instalaci na samostatný filtrační server s operačním systémem Linux. Systém je implementován jako konzolová aplikace v jazyce C, která je založena na odposlouchávání příchozích paketů. Zachytávání síťového provozu je umožněno díky knihovnám LibPcap [10], poskytující potřebné funkce pro odposlouchávání provozu na síťovém rozhraní.

Detektor postupně rozbaluje záhlaví a data zapouzdřených protokolů příchozího rámce, aby zjistil, zda je provoz validní a nevyznačuje se podezřelými rysy, jenž jsou objasněny v kapitole 3.1. V případě, že v příchozím provozu objeví rysy některého z útoku, je dané TCP spojení považováno za podezřelé a program si uloží informace o tomto spojení a obsahu příchozích rámců do své databáze. S každým zjištěním podezřelého rámce se provádí vyhodnocení situace. V případě, že některá z vlastností provozu přesáhne nastavenou mezní hodnotu, program tuto situaci vyhodnotí jako probíhající DoS útok a následně provede obranné kroky, popsané v dalších kapitolách této práce.

Protože jsou tyto útoky z většiny založeny na pomalém odesílání, či přijmu dat, je nutné, aby program vyhodnocoval situaci s určitou rezervou. V krajních situacích se totiž může jednat pouze o neškodného uživatele s pomalým připojením k internetu, který pouze odesílá požadavky a přijímá data po částech.

Z toho důvodu jsou na začátku programu definovány mezní parametry síťového provozu, které určují detekční práh detektoru. Tyto parametry nejsou jednotné pro veškerý provoz, ale pro každý typ útoku jsou definovány zvlášť. Těmito parametry jsou např. dovolený počet současně otevřených spojení z jedné IP adresy, maximální doba obsluhy jednoho požadavku, minimální datový tok či maximální počet paketů jednoho požadavku. Při překročení těchto hodnot je situace vyhodnocena jako útok. Definované hraniční hodnoty jsou předmětem ladění a přizpůsobování citlivosti detektoru konkrétním webovým serverům a jejich provozu. Při nastavování těchto hodnot ve vytvořeném detektoru se vycházelo z [6] [11] [12]. Hodnoty byly dále upřesněny na základě pozorování běžného provozu a probíhajícího útoku v laboratorní síti.

Výpis 4.1: Definice detekčního prahu detektoru.

```
1 #define GET_CONNECTIONS_LIMIT 20
2 #define GET_REQUESTS_LIMIT 10
3 #define GET_TIME_LIMIT 10
```

```
4
5  #define POST_CONNECTIONS_LIMIT 20
6  #define POST_REQUESTS_LIMIT 10
7  #define POST_TIME_LIMIT 10
8  #define POST_BYTERATE_LIMIT 10
9
10 #define RANGE_CONNECTIONS_LIMIT 20
11 #define RANGE_REQUESTS_LIMIT 5
12 #define RANGE_FIELD_SIZE_LIMIT 200
13 #define RANGE_COUNT_LIMIT 10
14
15 #define SLOWREAD_CONNECTIONS_LIMIT 20
16 #define SLOWREAD_REQUESTS_LIMIT 15
17 #define SLOWREAD_TIME_LIMIT 30
18 #define SLOWREAD_BYTERATE_LIMIT 10
19 #define SLOWREAD_MINIMUM_WINDOW 1460
20 #define SLOWREAD_MINIMUM_WIN_SCALE 0
```

4.1 Zachytávání paketů

Jak již bylo zmíněno, pro odposlouchání síťové komunikace je použita knihovna LibPcap [10], která je například použita ve známém síťovém analyzátoru Wireshark [13]. Tyto knihovny existují v podobě jak pro Linuxové operační systémy, tak i pro OS Windows (v tomto případě se nazývají WinPcap).

V hlavním těle programu (funkce `Main()`) je nejprve inicializováno síťové rozhraní, na kterém bude detektor dále odposlouchávat provoz. Rozhraní (např. lokální smyčka, wi-fi adapter, ...) lze přesně vybrat zadáním jeho názvu do parametrů programu při spouštění systému. V případě, kdy je program spuštěn bez argumentů, je pomocí funkce `pcap_lookupdev(errbuf)` vybráno výchozí komunikační rozhraní. Následuje funkce `pcap_lookupnet(dev, &net, &mask, errbuf)` kterou se získá IP adresa a maska sítě rozhraní určeného pro odposlech. Samotný odposlech se zahájí funkcí `pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf)`, jenž vrací ukazatel na odposlouchávací relaci. První argument funkce `pcap_open_live()` specifikuje, na jakém rozhraní má odposlech probíhat, druhý argument definuje maximální počet bajtů, které mohou být zachyceny, další argument je logického charakteru a definuje, zda má být síťové rozhraní otevřeno promiskuitním režimu. Další argument definuje dobu čtení dat v milisekundách. Poslední parametr slouží k ukládání chybových zpráv. V dalším kroku je specifikován typ hlavičky na linkové vrstvě, která se odvíjí od zvoleného síťovího rozhraní.

Výpis 4.2: Spuštění odposlouchávání komunikace na síťovém rozhraní.

```
1  int main(int argc, char **argv){
2
3    if (argc == 2) {
4       dev = argv[1];
5    }
6    else if (argc > 2) {
7      fprintf(stderr, "error: unrecognized options");
8      exit(EXIT_FAILURE);
9    }
10   else {
11     /* find a capture device if not specified on
12     command-line */
13     dev = pcap_lookupdev(errbuf);
14     if (dev == NULL) {
15       fprintf(stderr, "Couldn't find default device: %s\n",
16               errbuf);
17       exit(EXIT_FAILURE);
18     }
19   }
20
21   /* get network number and mask associated with capture
22   device */
23   if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
24     fprintf(stderr, "Couldn't get netmask for device %s:%s\n"
25     , dev, errbuf);
26     net = 0;
27     mask = 0;
28   }
29
30   /* open capture device */
31   handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
32   if (handle == NULL) {
33     fprintf(stderr, "Couldn't open device %s: %s\n", dev,
34             errbuf);
35     exit(EXIT_FAILURE);
36   }
37
38
39   /* make sure we're capturing on an Ethernet device [2] */
40   if (pcap_datalink(handle) != DLT_EN10MB) {
```

```
41      fprintf(stderr, "%s is not an Ethernet\n", dev);
42      exit(EXIT_FAILURE);
43    }
44
45    ...
46
47  }
```

Poté je filtr zkompilován a spuštěn. V řetězci `filter_exp` se specifikuje provoz, který má být zachytáván. Příklad nastavení:

- `ip` – zachytávají se pouze pakety protokolu IP
- `tcp` – zachytávají se pouze pakety protokolu TCP
- `tcp port 80` – zachytávají se pouze pakety protokolu TCP, které v TCP záhlaví obsahují port 80
- `ip host 10.0.0.3` – zachytávají se pouze pakety, které IP záhlaví obsahují IP adresu 10.0.0.3

Pro účely tohoto detektoru je filtr nastavený tak, aby program zachytával pouze příchozí provoz, který směřuje na port 80, tedy na webový server.

Výpis 4.3: Nastavení filtru pro odposlouchávání komunikace.

```
1   char filter_exp[] = "dst port 80";
2
3   // compile the filter expression
4   if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
5     fprintf(stderr, "Couldn't parse filter %s: %s\n",
6         filter_exp, pcap_geterr(handle));
7     exit(EXIT_FAILURE);
8   }
9
10  // apply the compiled filter
11  if (pcap_setfilter(handle, &fp) == -1) {
12    fprintf(stderr, "Couldn't install filter %s: %s\n",
13        filter_exp, pcap_geterr(handle));
14    exit(EXIT_FAILURE);
15  }
```

Dále následuje volání funkce `pcap_loop()`, která cyklicky snímá příchozí pakety a předává je další funkci `got_packet()` pro zpracování příchozích dat. Tato funkce je jedním ze vstupních parametrů funkce `pcap_loop()`.

Výpis 4.4: Cyklické snímání příchozích dat.

```
1   pcap_loop(handle, num_packets, got_packet, NULL);
```

Funkce `pcap_loop()` je spuštěna v nekonečné smyčce. Proto program obsahuje také obsluhu vstupních událostí. Jakmile uživatel, či jiný proces vyšle signál odpovídající stisku kláves Ctrl + C, dojde k přerušení smyčky `pcap_loop()` a vykonání dílčích příkazů pro korektní uvolnění alokovaných databází a ukončení programu.

4.2 Zpracování zachycených dat

Každý příchozí rámec (proud bajtů) zpracovávaný funkcí `got_packet()` je postupně rozdělen na jednotlivá záhlaví použitých protokolů. Funkce postupně čte jednotlivé bajty a hodnoty zapisuje do připravených struktur. Struktury pro uložení dat ze záhlaví protokolu IP a TCP jsou zde:

Výpis 4.5: Struktura pro ukládání dat ze záhlaví IP protokolu

```
1  /* IP header */
2  struct sniff_ip {
3      u_char  ip_vhl;                 /* version << 4 | header
4                                         length >> 2 */
5      u_char  ip_tos;                 // type of service
6      u_short ip_len;                 // total length
7      u_short ip_id;                  // identification
8      u_short ip_off;                 // fragment offset field
9      #define IP_RF 0x8000            // reserved fragment flag
10     #define IP_DF 0x4000            // dont fragment flag
11     #define IP_MF 0x2000            // more fragments flag
12     #define IP_OFFMASK 0x1fff       // mask for fragmntg bits
13     u_char  ip_ttl;                 // time to live
14     u_char  ip_p;                   // protocol
15     u_short ip_sum;                 // checksum
16     struct  in_addr ip_src,ip_dst;  // src and dst address
17 };
18
19 /* TCP header */
20 struct sniff_tcp {
21      u_short th_sport;               // source port
22      u_short th_dport;               // destination port
23      tcp_seq th_seq;                 // sequence number
24      tcp_seq th_ack;                 // acknowledgement num.
25      u_char  th_offx2;               // data offset, rsvd
26      #define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
27      u_char  th_flags;
28      #define TH_FIN  0x01
```

```
29      #define TH_SYN  0x02
30      #define TH_RST  0x04
31      #define TH_PUSH 0x08
32      #define TH_ACK  0x10
33      #define TH_URG  0x20
34      #define TH_ECE  0x40
35      #define TH_CWR  0x80
36      #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|
37                        TH_ECE|TH_CWR)
38      u_short th_win;                 // window
39      u_short th_sum;                 // checksum
40      u_short th_urp;                 // urgent pointer
41  };
```

Tímto způsobem jsou získány data ethernetového, IP a TCP záhlaví. Pokud se za TCP záhlavím nachází další data přijatého paketu, jedná se již o data aplikační vrstvy, resp. data protokolu HTTP, které detektor dále analyzuje.

Pro ukládání informací o podezřelých paketech a spojeních slouží celkem 4 databáze, každá pro jeden typ útoku. Mimo to je zde také seznam blokovaných IP adres, jenž se plní při blokaci některého z útoku. Tento seznam slouží pro uchování informací o zdrojích útoku, především kvůli přehlednosti aplikovaných firewallových pravidel a pro případné budoucí využití. Všechny tyto databáze i seznam blokovaných IP adres jsou v programu vytvořeny v podobě pole struktur, jejichž velikost je dynamicky alokována dle potřeby. Tím je docíleno efektivního využití paměti systému. Struktura databáze pro uchování informací o útočnících a jejich navázaných spojení je následující:

Výpis 4.6: Struktura databáze pro uchování dat o útočících spojení

```
1  typedef struct {
2      char                clientIP[16];
3      unsigned short      connections_count;
4      clientConnection *  connections;
5  } httpClient;
6
7  typedef struct {
8      unsigned short  port;
9      unsigned int    failRequests;
10     time_t          firstRequest;
11     time_t          lastRequest;
12     tcp_seq         lastTcpAck;
13     tcp_seq         lastTcpSeq;
```

```
14      unsigned int    lastTcpLen;
15      unsigned int    totalData;
16      unsigned int    receivedData;
17      unsigned short  winScale;
18  } clientConnection;
```

Databázi podezřelých uživatelů představuje pole struktur `httpClient`. V každém záznamu je uložena IP adresa potencionálního útočníka, počet otevřených spojení, resp. portů, na kterých komunikuje se serverem. Záznam dále obsahuje pole struktur `clientConnection`, jenž představuje databázi navázaných spojení. Každý záznam otevřeného spojení uchovává informaci o čísle otevřeného portu, počtu přijatých podezřelých paketů na tomto spojení, časy příchodu prvního podezřelého a posledního podezřelého paketu, sekvenční a potvrzovací číslo posledního TCP segmentu, velikost přenášených dat, velikost všech zatím přijatých dat na tomto spojení, velikost dat ohlášených v záhlaví HTTP požadavku a dohodnutá hodnota koeficientu pro výpočet velikosti TCP okna. Tyto záznamy později slouží jako vstupní data algoritmu pro vyhodnocení, zda některý z útoků probíhá.

4.3 Detekce útoků Slow GET

Algoritmus detekce útoku Slow GET při příchodu nového paketu nejprve zkoumá, jestli je příchozí požadavek typu GET ukončený znaky `\r\n\r\n`. Pokud ukončen není, vytvoří nový záznam v databázi a otevřené spojení se nadále sleduje. Pokud jsou přijata pouze data bez záhlaví HTTP požadavku, tak detektor prohledá svou databázi, jestli v minulosti z dané IP adresy a portu nepřišel neukončený HTTP požadavek, resp. požadavek neobsahující ukončovací znaky `\r\n\r\n`.

Pokud algoritmus záznam pro danou IP adresu a port ve své databázi nalezne, znamená to, že server čeká na ukončení již přijaté části záhlaví požadavku. Algoritmus proto v HTTP požadavku nejprve hledá ukončovací znak a v případě, že ho nalezne, spojení je poté korektně uzavřeno, požadavek vyřízen a záznam z databáze je smazán. Tahle situace se vyřeší bez postihu, pokud neukončených požadavků na tomto spojení přišlo jen určité malé množství, což může být způsobeno i pomalým internetovým připojením jinak legitimních uživatelů.

Avšak v případě, že na tomto spojení stále nepřichází ukončená část HTTP GET požadavku a spojení je stále otevřené, je záznam v databázi aktualizován. Hodnota přijatých podezřelých paketů je inkrementována a je zaznamenán čas příchodu této části HTTP požadavku.

Následně proběhne vyhodnocení údajů v databázi. Implementovaný detektor útoku Slow GET porovnává 3 parametry spojení, jenž jsou klíčové pro odhalení

útoku. Tyto parametry jsou:

- počet otevřených spojení dané IP adresy;
- počet přijatých neukončených částí HTTP GET požadavku na daném spojení;
- uplynulá doba od příchodu první části zatím neukončeného HTTP GET požadavku na daném spojení.

Ve výchozím nastavení jsou hraniční hodnoty těchto parametrů nastaveny na maximálně 20 dovolených otevřených spojení z jedné IP adresy, rozdělení požadavku na maximálně 10 částí a časový limit 10 sekund pro dokončení požadavku. Tyto hodnoty byly zvoleny na základě analýzy popsané v teoretické části této práce.

Pokud některý z parametrů překročí svoji hraniční hodnotu, je tato situace vyhodnocena jako probíhající útok a zdrojová adresa je blokována procedurou popsanou v kapitole 4.7.

Celý postup detektoru útoku Slow GET ilustruje vývojový diagram v příloze A.1.

4.4 Detekce útoků Slow POST

Pro detekci útoků typu Slow POST vytvořený algoritmus zkoumá, zda-li je v HTTP požadavku přítomna položka `Content-Length`, která oznamuje celkovou velikost přidružených dat požadavku. Tato položka je nejčastěji využívána v požadavku typu POST, pomocí kterého klient oznamuje zasílání dat, obvykle vyplněného formuláře na webové stránce. Ovšem pro útok není nutné, aby požadavek byl typu POST. Některé sofistikovanější útoky mohou využít požadavky jiného typu, nebo dokonce jako identifikátor typu požadavku použít libovolné či náhodné slovo. Detektor tedy nesmí být závislý na použití konkrétních typů požadavků.

V případě detekce pole `Content-Length` je jeho hodnota porovnána se skutečnou velikostí dat přenášených za záhlavím požadavku. Pokud je zjištěno, že přijaté data jsou jen částí ohlášených dat, tak je dané spojení považováno za podezřelé a nadále je sledován příjem dat od protistrany. Web server totiž v této chvíli data přijme a dále vyčkává na příjem kompletních dat požadavku.

V tento moment je spojení přidáno do databáze a spojení je sledováno. Pokud detektor na tomto spojení přijme další část dat, databáze je opět aktualizována a situace vyhodnocena. Detektor posuzuje následující parametry:

- počet otevřených spojení dané IP adresy;
- počet přijatých části dat požadavku na daném spojení;
- uplynulá doba od příchodu první části dat daného požadavku;
- pokles přenosové rychlosti dat zasílaných uživatelem.

Ve výchozím nastavení jsou hraniční hodnoty těchto parametrů nastaveny na maximálně 20 dovolených otevřených spojení z jedné IP adresy, rozdělení dat poža-

davku na maximálně 10 částí, časový limit 10 sekund pro zaslání kompletních dat a minimální přenosovou rychlost 10 bajtů za sekundu. Pokud je některý z těchto parametrů překročen, je situace vyhodnocena jako útok a zdrojová IP adresa je blokována.

Celý postup detektoru útoku Slow POST ilustruje vývojový diagram v příloze A.2.

4.5 Detekce útoku Apache Range Header

Před útoky typu Range Header je už naprostá většina webových serverů imunní, nicméně je vhodné alespoň monitorovat, zda se někdo o tento útok nepokusí. Detekční systém pak může identifikovat útočníka a hlídat jeho aktivitu. Po neúspěšném útoku je totiž velmi pravděpodobné, že se útočník pokusí útok opakovat odlišnou metodou, zejména pomocí příbuzných útoků Slow GET a Slow POST.

Detekce útoku Range Header je založena na analýze příchozích HTTP požadavků na webový server. V tomto případě mohou být využity požadavky různých typů. Detektor sleduje výskyt atributu `Range` a jeho hodnotu. Na základě této hodnoty je přímo vyhodnocena validita požadavku. Detektor zkoumá:

- velikost pole `Range` v daném požadavku;
- počet rozsahů dat definovaných v poli `Range`.

Hraniční hodnoty pro posuzování validity požadavků na základě těchto parametrů jsou opět nastaveny globálně v rámci detekčního prahu detektoru. Ve výchozím stavu je nastavena maximální velikost pole 200 bajtů a maximálně 10 požadovaných rozsahů dat. Při překročení těchto hodnot detektor ihned označí požadavek jako útok typu Range Header, avšak v tuto chvíli pouze přidá záznam do databáze.

Na základě záznamů v databázi detektor sleduje podobné parametry provozu jako u předchozích útoků, tedy:

- počet otevřených spojení dané IP adresy;
- počet přijatých nevalidních požadavků na daném spojení.

Maximální přípustné hodnoty těchto parametrů jsou ve výchozím nastavení detektoru stanoveny na maximálně 20 spojení z jedné IP adresy a maximálně 5 nevalidních požadavků z daného spojení. Teprve po překročení těchto hodnot detektor provede blokaci protistrany, aby útočníka odřízl od přístupu k web serveru.

Celý postup detektoru útoku Apache Range Header ilustruje vývojový diagram v příloze A.3.

4.6 Detekce útoku Slow Read

Odhalení tohoto útoku je obtížnější, protože využívá také funkcionalit protokolu TCP. Pro odhalení útoku musí detektor sledovat každé TCP spojení od úplného začátku, tedy navázání spojení pomocí tzv. 3-way handshake. V tuto chvíli zatím nelze odhadnout, jaká data budou vlastně přenášena. Sledování navázání spojení je však důležité kvůli zjištění hodnoty `Window scale factor` v záhlaví TCP SYN segmentu. Tato hodnota se vždy stanoví pouze při navazování spojení. Stanice si ji zapamatují a v průběhu spojení již zůstává konstantní. Tato hodnota je nezbytná k výpočtu a následnému sledování velikosti TCP okna, které je pro útok Slow Read využito.

Detektor z úvodního TCP SYN segmentu zjistí hodnotu `Window scale factor`, podle které vypočítá velikost okna. Pokud je velikost okna podezřele nízká, je toto spojení považováno za neobvyklé a je zaznamenáno do databáze. Program dále sleduje příchozí TCP segmenty na tomto spojení a velikosti jejich TCP oken. Detektor má definované minimální přípustné hodnoty velikosti TCP okna a hodnoty `Window scale factor`, pro které lze ještě provoz tolerovat. Ve výchozím nastavení programu je minimální hodnota TCP okna stanovena na 1460 bajtů, což je maximální možná velikost jednoho TCP segmentu. Pro hodnotu `Window scale factor` je jako minimum stanovena hodnota 0, což znamená, že ve výchozím stavu detektoru tato hodnota při detekci útoku nehraje roli. Táhle možnost je tu zde pro možné budoucí přizpůsobení detektoru.

Pokud je velikost okna podezřele nízká, je toto spojení považováno za neobvyklé a je zaznamenáno do databáze. Program dále sleduje příchozí TCP segmenty na tomto spojení a zároveň sleduje následující parametry provozu:

- počet přijatých TCP segmentů s velmi malým TCP oknem na tomto spojení;
- dobu, po kterou má spojení nastavenu velmi malé TCP okno;
- pokles přenosové rychlosti odesílaných dat;
- počet spojení dané IP adresy, ze kterých byl přijat TCP segment s velmi malou velikostí okna.

Ve výchozím nastavení jsou hraniční hodnoty pro tyto parametry nastaveny na: maximálně 20 současně otevřených spojení z jedné IP adresy; maximálně 10 TCP segmentů s velmi malou velikostí TCP okna na daném spojení; nastavení velmi malého TCP okna po dobu maximálně 30 sekund; minimální přenosová rychlost 10 bajtů za sekundu. Po překročení těchto hodnot detektor vyhodnotí situaci jako útok a provede blokaci útočníka.

Celý postup detektoru útoku Slow Read ilustruje vývojový diagram v příloze A.4.

4.7 Zamezení útoku

V navrženém systému prevence průniku probíhá filtrace útoku ve třech krocích. Nejprve je omezeno množství současně otevřených TCP spojení na jednoho uživatele. V dalším kroku je veškerá komunikace uživatele zablokována a v posledním kroku jsou rozeslány TCP segmenty s příznakem RST, jenž mají za úkol uvolnit prostředky web serveru rezervované pro navázaná TCP spojení.

Pro filtraci či omezení síťového provozu je využit firewall s názvem Netfilter, jenž je součástí jádra operačního systému Linux. Netfilter funguje jako paketový/stavový filtr síťové komunikace. Zajišťuje řízení toku paketů, procházejících jádrem operačního systému, na základě mnoha kritérií [14]. Základním nástrojem pro konfiguraci Netfilteru je program s názvem iptables [14], jenž bude v této práci využit pro aplikaci firewallových pravidel.

V prvním kroku je při spuštění navrženého systému na filtračním serveru okamžitě aplikováno nové firewallové pravidlo pomocí programu iptables. Toto pravidlo omezuje všechny IP adresy (klienty a útočníky), přistupující na webový server, stanovením maximálního počtu současně otevřených spojení. V tomto případě na 20 spojení.

```
iptables -I FORWARD -p tcp --dport 80 -m connlimit
--connlimit-above 20 -j DROP
```

Běžný uživatel obvykle v jeden moment neotevře desítky, či stovky spojení, zatímco u útoků je tohle způsob, jak vyčerpat dostupné prostředky serveru. Pokud nějaká IP adresa přesáhne maximální dovolený počet současně otevřených spojení, nová spojení jsou zahazována. Tímto mechanismem je web server chráněn před zahlcením velkým množstvím spojení z jedné IP adresy, avšak neuchrání jej před distribuovanou formou útoku.

Toto preventivní pravidlo je zde z toho důvodu, že samotná analýza aplikačních dat probíhá až poté, co web server přijme požadavek od útočníka a tudíž jeho prostředky jsou již přiděleny útočníkovi. Kdyby byl web server v této fázi zahlcen všemi požadavky, odepření služby nastane okamžitě.

Pokud analýza požadavků zjistí přítomnost některého z útoků pomocí metod popsaných v předcházejících kapitolách, je nutné útočníkovi zamezit další komunikaci s webovým serverem a zároveň uvolnit již navázaná spojení. Proto vytvořený program ihned aplikuje nové firewallové pravidlo blokace všech paketů z útočníkovy IP adresy.

```
iptables -I FORWARD -s "IP_ADRESA_UTOCNIKA" -j DROP
```

Poté se provede uvolnění již navázaných spojení pomocí techniky podvržení paketů. To obnáší sestavení TCP segmentu s příznakem RST (reset) se zdrojovou IP adresou a portem útočníka a také odpovídajícím sekvenčním číslem. Tyto údaje má systém uložen ve své databázi detekovaných útoků.

V operačních systémech se obvykle pro zasílání dat sítí využívají běžné sokety. To znamená, že aplikace předá svá data jádru OS, který automaticky vytvoří záhlaví protokolů TCP, IP a Ethernet a zprostředkuje komunikaci mezi aplikacemi. V situacích, jako je tato, je však potřeba, aby program sestavil celý paket sám s použitím vlastních záhlaví protokolů TCP a IP. K tomu se využívají tzv. raw sockety [15].

V programu je nejprve nutné vložit potřebné knihovny, pomocí kterých může program využívat systémové sockety, a které jsou také využity pro definici struktury záhlaví protokolu TCP a IP.

Výpis 4.7: Vložení knihoven pro práci s raw sockety

```
1 #include <sys/socket.h>
2 #include <netinet/tcp.h>          // Declaration of TCP header
3 #include <netinet/ip.h>           // Declaration of IP header
```

Raw socket se následně vytvoří voláním funkce `socket()`. Specifikací protokolu `IPPROTO_RAW` v parametrech této funkce je řečeno, že veškerý obsah paketu bude sestaven manuálně.

Výpis 4.8: Vytvoření raw socketu

```
1 int s~= socket (AF_INET, SOCK_RAW, IPPROTO_RAW);
```

Program dále inicializuje struktury, jenž představují záhlaví jednotlivých protokolů, které následně vyplní potřebnými daty. Příklad nastavení hodnot záhlaví protokolů TCP a IP zobrazuje následující výpis kódu programu.

Výpis 4.9: Nastavení hodnot záhlaví TCP a IP protokplu

```
1  struct iphdr *iph =     (struct iphdr *) datagram;
2  struct tcphdr *tcph =   (struct tcphdr *) (datagram
3                          + sizeof (struct ip));
4
5  // Fill in the IP Header
6  iph->ihl = 5;
7  iph->version = 4;
8  iph->tos = 0;
9  iph->tot_len = sizeof (struct iphdr) + sizeof (struct tcphdr)
10                 + strlen(data);
11 iph->id = htonl (54321);
12 iph->frag_off = 0;
```

```
13 iph->ttl = 255;
14 iph->protocol = IPPROTO_TCP;
15 iph->check = 0;                        // checksum filled later
16 iph->saddr = inet_addr ( ip_src );     // spoof the src ip addr
17 iph->daddr = sin.sin_addr.s_addr;      // spoof the dst ip addr
18
19 // Fill in theTCP Header
20 tcph->source = htons (port_src);
21 tcph->dest = htons (port_dst);
22 tcph->seq =  htonl(seq_number);
23 tcph->ack_seq = htonl(ack_number);
24 tcph->doff = 5;                        // tcp header size
25 tcph->fin=0;
26 tcph->syn=0;
27 tcph->rst=1;                           // set RST flag
28 tcph->psh=0;
29 tcph->ack=0;
30 tcph->urg=0;
31 tcph->window = htons (0);              // window size
32 tcph->check = 0;                       // checksum filled later
33 tcph->urg_ptr = 0;
```

Následně se dopočítají potřebné kontrolní součty a pomocí funkcí `setsockopt()` a `sendto()` se provede nastavení socketu a odeslání dat.

Výpis 4.10: Odeslání dat

```
1  if (setsockopt (s, IPPROTO_IP, IP_HDRINCL, val, sizeof (one))
2      < 0 )
3  {
4      fprintf(stderr,"ERROR␣Sending␣IP_HDRINCL␣\n");
5      return 0;
6  }
7
8  if (sendto (s, datagram, iph->tot_len ,  0, (struct
9      sockaddr *) &sin, sizeof (sin)) < 0)
10 {
11     fprintf(stderr, "Neporadilo␣se␣odeslat␣TCP␣RST");
12     return 0;
13 } else{
14     //Data send successfully
15     printf("TCP␣RST␣odeslan␣na␣%s:%d,␣seq:␣%u\n", ip_dst,
16             port_dst, seq_number);
```

```
17      return 1;
18  }
```

Pro vložení správných hodnot do jednotlivých záhlaví program využívá detektorem nashromážděné informace v databázi. Do IP záhlaví je dosazena útočníkova IP adresa a do TCP záhlaví je dosazeno číslo otevřeného portu útočníka, vypočítané potvrzovací a sekvenční číslo dat. Příznaky segmentu jsou nastaveny na `RST`, tedy zrušení spojení. Tímto způsobem program vytvoří falešný RST segment, který se jeví jako by jej odeslal útočník. Segment je následně odeslán na web server, který si myslí, že protistrana z neznámého důvodu ukončila spojení. Web server poté spojení uzavře, čímž uvolní své prostředky pro další uživatele.

Výsledkem tohoto procesu je, že útočníkovi se podaří obsadit nepatrnou část výpočetních prostředků web serveru, které jsou ale následně uvolněny díky reakci vytvořeného detektoru, a to ihned po přesáhnutí detekčního prahu. Útočník je poté blokován, takže již další spojení otevírat nemůže. Od tohoto okamžiku je webový server pro útočníka nedostupný, což si útočník může mylně vyložit jako úspěšně provedený útok.

Systém si od odražení útoku až do svého ukončení uchovává seznam blokovaných IP adres pro přehled aplikovaných firewallových pravidel a nebo pro případné budoucí využití.

Program se ukončí stiskem kombinace kláves `Ctrl + C`. Následně je vypsán celý obsah databáze, ve které se nachází všechny zachycené podezřelé IP adresy, jejich otevřené porty a další informace k zachyceným a neukončeným podezřelým spojením. Poté je provedena dealokace všech databází, na základě seznamu blokovaných IP adres jsou odstraněny aplikované firewallové pravidla, čímž je komunikace vrácena do původního stavu a program je korektně ukončen.

4.8 Obrana před distribuovanou formou útoku

V případě distribuované formy útoku proběhne výše popsaný proces blokace pro každou IP adresu, která je zdrojem útoku. Detektor zpravidla jako první detekuje velký počet spojení jednoho uživatele, ze kterých proudí podezřelé požadavky. Útočník by teoreticky mohl přizpůsobit parametry útoku snížením množství navazovaných spojení z každého počítače až pod práh detektoru, tedy na méně než 20 spojení. Tím by docílil, že navázaná spojení nebudou ihned ukončena. Při velkém počtu útočících počítačů by tak mohl každý počítač po malých kouscích beztrestně ukrajovat z volně dostupných prostředků serveru a tím docílit nedostupnosti služby. Nicméně detektor tento provoz sleduje na základě rozpoznání podezřelých rysů a zareaguje

až po přesáhnutí některého z dalších parametrů detekce jako například pokles přenosové rychlosti, nedokončení požadavku v očekávaném čase, nebo příjem většího počtu nedokončených částí požadavku. Poté dojde k výše popsanému procesu blokace. Distribuovaný útok je tedy odražen a prostředky web serveru jsou následně uvolněny.

5 POUŽITÍ VYTVOŘENÉHO SYSTÉMU

Aby mohl být vytvořený systém prevence průniku použit, musí být spuštěn na linuxovém operačním systému, např. Debian, Ubuntu, nebo jiné distribuci. Uvedený postup se vztahuje na distribuci Debian. Nicméně použití tohoto systému je jednoduché a na všech ostatních distribucích bude pravděpodobně totožné.

Jediným požadavkem vytvořeného systému je přítomnost nainstalovaných knihoven Libpcap v operačním systému. V OS Debian lze balíček `lipcap-dev` nainstalovat následujícím příkazem:

```
apt-get install libpcap-dev
```

Poté již stačí vytvořenou aplikaci spustit následujícím způsobem. Pro spuštění je potřeba mít práva superuživatele. Z přiloženého CD disku se na pevný disk počítače zkopíruje soubor `Detektor`. Pomocí terminálu se otevře složka se zkopírovaným souborem a aplikace se spustí příkazem:

```
./Detektor
```

Pokud je potřeba specifikovat síťové rozhraní, na kterém má program odposlouchávat a analyzovat komunikaci, je možné uvést název tohoto rozhraní jako parametr při spouštění programu.

Pro ukončení programu je nutné zmáčknout kombinaci klávec `CTRL + C`. V případě ladění přednastavených hodnot prahu detektoru, musí být upraven zdrojový kód aplikace. Tato verze programu prozatím nemá vytvořený žádný mechanismus pro ovládání citlivosti detektoru. Na disk počítače se z přiloženého CD zkopíruje soubor `main.c` obsahující zdrojový kód. V úvodní části kódu se nacházejí definice prahových hodnot detektoru, které je možné upravovat. Pro detektor útoku Slow GET vypadají následovně:

Výpis 5.1: Definice prahových hodnot detektoru útoku Slow GET

```
1 #define SLOWLORIS_CONNECTIONS_LIMIT 20
2 #define SLOWLORIS_REQUESTS_LIMIT 10
3 #define SLOWLORIS_TIME_LIMIT 10
```

Upravený zdrojový kód je potřeba uložit a znovu zkompilovat. K tomu je možné využít kompilátor `gcc`. Zkompilování kódu se provede následujícím příkazem:

```
gcc -Wall -o Detektor main.c -lpcap
```

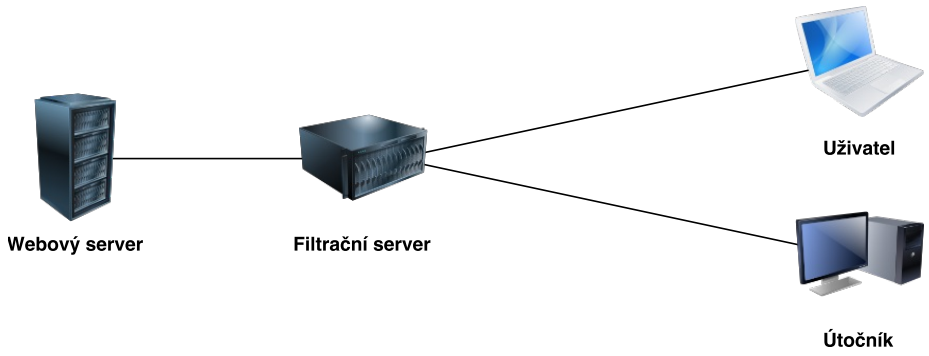
Výsledný soubor `Detektor` už jen stačí spustit stejným příkazem, který byl uveden výše.

6 TESTOVÁNÍ A VÝSLEDKY

V této kapitole jsou prezentovány výsledky testování vytvořeného systému prevence průniku v laboratorní síti. Kapitola je rozdělena do samostatných částí pro každý z testovaných Slow DoS útoků. Výsledky testování jsou znázorněny v grafech, které zobrazují dostupnost webového serveru pro legitimní uživatele (zelená čára) a průběh útoku z pohledu útočníka, tedy množství navázaných (oranžová čára), nenavázaných (červená čára) a zahozených TCP spojení (modrá čára). X-ová osa představuje čas a Y-ová osa vyjadřuje počet spojení.

6.1 Testovací prostředí

Topologie laboratorní sítě je vyobrazena na obrázku 6.1 a tvoří ji webový server, filtrační server, stanice útočníka a stanice běžného uživatele, jenž jsou propojeny dvoubodovými spoji.



Obr. 6.1: Laboratorní síť

Jako webový server byl využit stolní PC s operačním systémem Windows 10, na kterém je v programu VirtualBox spuštěn virtuální linuxový operační systém Ubuntu. V tomto systému je nainstalovaný webový server Apache ve výchozím nastavení s testovacími webovými stránkami, které obsahují také jednoduchý formulář. Parametry počítače s web serverem jsou následující:

- **Hostitelský operační systém:** Microsoft Windows 10 64–bit
- **Virtuální operační systém:** Ubuntu 16.04 64–bit
- **Procesor:** Intel Core i5–4670K (k dispozici 1 jádro)
- **Dostupná paměť RAM:** 2 GB
- **IP adresa:** 10.0.0.57
- **Webový server:** Apache 2.4.18
- **PHP:** 7.0.8

- **Další software:** Wireshark v2.2.2

Počítač s webovým server je fyzicky propojený s notebookem MSI GE 72. Oba počítače se nachází v síti 10.0.0.0/24. Ostatní stanice laboratorní topologie jsou vytvořeny jako virtuální počítače v programu VirtualBox, běžícím na tomto notebooku. Virtuální počítače jsou mezi sebou propojeny virtuální sítí 10.0.1.0/24. Výchozí bránu této virtuální sítě představuje filtrační server. Tomu bylo uzpůsobeno i směrování v této síti. Veškerý provoz ven z této virtuální sítě prochází přes virtuální filtrační server, který jediný je spojen s webových serverem. Tím je zachována logická struktura navržené laboratorní sítě. Filtrační server je tak jediný uzel, přes který se uživatelé dostanou na webový server. Na filtračním serveru je spuštěn implementovaný systém prevence průniku, který zachytává a analyzuje všechny pakety směřující na port 80 webového serveru.

Parametry filtračního serveru jsou:

- **Hostitelský operační systém:** Microsoft Windows 10 64–bit
- **Operační systém:** Debian 8.7 64–bit
- **Procesor:** Intel Core i7–4720HQ (k dispozici 1 jádro)
- **Dostupná paměť RAM:** 2 GB
- **IP adresa:** 10.0.0.51 a 10.0.1.10

Parametry útočníkova stroje jsou:

- **Hostitelský operační systém:** Microsoft Windows 10 64–bit
- **Operační systém:** Ubuntu 16.04 64–bit
- **Procesor:** Intel Core i7–4720HQ (k dispozici 1 jádro)
- **Dostupná paměť RAM:** 2 GB
- **IP adresa:** 10.0.1.20
- **Další software:** Wireshark v2.2.2

Virtuální počítač běžného uživatele je kopií stroje útočníka, má tedy stejné parametry, pouze odlišnou IP adresu. Jediný úkol uživatele při testování je pokoušet se soustavně otevírat webové stránky na web serveru a testovat jejich dostupnost.

U všech útoků bylo nejprve provedeno testování útoku přímo na web server pro ověření účinnosti útoku. Poté útok procházel přes filtrační server, kdy se testovala účinnost navrženého systému prevence průniku. V průběhu testování byl webový server zatěžován pouze požadavky útočníka.

6.2 Generátor útoku

Pro generování útoků byl na stroji útočníka využit program SlowHTTPTest. Tento program je určen pro linuxové operační systémy a spouští se pomocí terminálu. SlowHTTPTest je možné nainstalovat z oficiálních repozitářů distribuce Debian i Ubuntu.

```
apt-get install slowhttptest
```

Nástroj SlowHTTPTest implementuje nejznámější typy DoS útoků na aplikační protokoly. Dokáže simulovat útoky typu Slow GET, Slow POST, Apache Range Header a Slow Read. Po skončení útoku vygeneruje soubory typu HTML a CSV se statistickými daty útoku, jenž jsou využity pro vizualizaci výsledků testů v další části této kapitoly.

Generovaný útok je možné přizpůsobit pomocí velkého množství přepínačů [16]:

- **H**: spuštění útoku typu Slow GET
- **B**: spuštění útoku typu Slow POST
- **R**: spuštění útoku typu Apache Range Header
- **X**: spuštění útoku typu Slow Read
- **g**: po ukončení jsou vygenerovány CSV a HTML soubory se statistikami útoku
- **o**: specifikace jména výstupních souborů
- **c**: celkový počet generovaných spojení během testu
- **r**: množství spojení otevřených za jednu sekundu
- **l**: trvání útoku v sekundách
- **u**: specifikace cíle útoku
- **t**: typ odesílaných požadavků (GET, POST, FAKEWORD, ...)
- **i**: interval mezi odesíláním dat (jen pro útoky Slow GET a Slow POST)
- **s**: specifikace hodnoty pole `Content-Length` při útoku Slow POST
- **x**: maximální velikost dat udržovacích požadavků (Slow GET a Slow POST)
- **p**: interval pro vyčkání odezvy serveru, než je server označen jako nedostupný
- **a**: první hodnota rozsahu pole `Range` při útoku Apache Range Header
- **b**: poslední hodnota rozsahu pole `Range` při útoku Apache Range Header
- **w**: spodní hranice velikosti TCP okna při útoku Slow Read
- **y**: horní hranice velikosti TCP okna při útoku Slow Read
- **z**: množství dat čtených ze vstupního bufferu při útoku Slow Read
- **n**: interval mezi čtením dat při útoku Slow Read
- **k**: množství zapouzdřených požadavků v jednom segmentu (pipeline faktor)

Nastavené hodnoty budou blíže specifikovány u každého útoku samostatně.

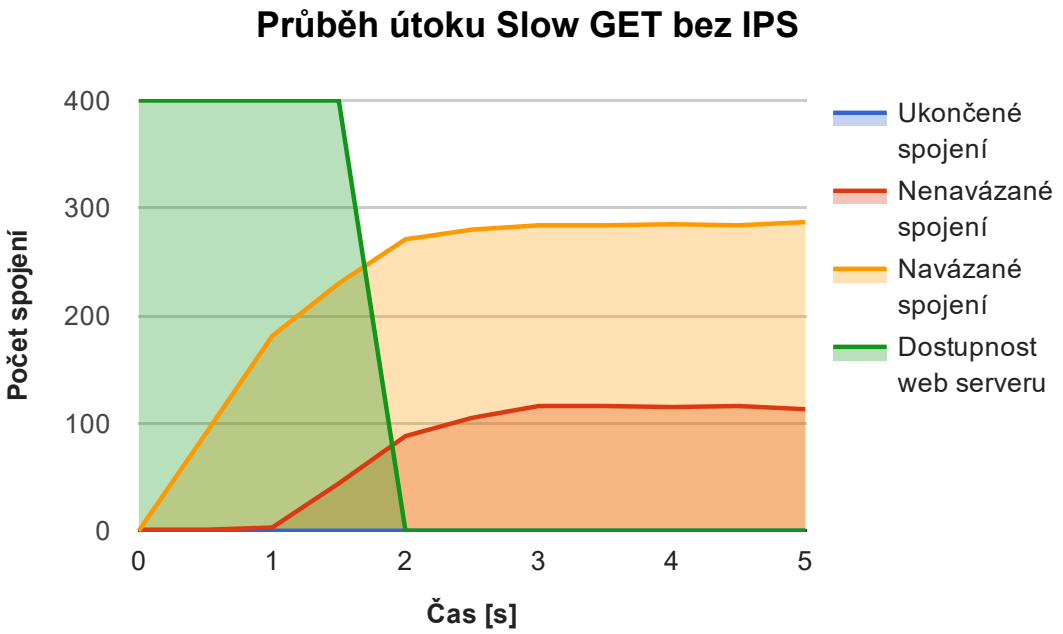
6.3 Slow GET

Pro spuštění útoku typu Slow GET byl použit následující příkaz:

```
slowhttptest -c 400 -H -g -o slow_get -i 10 -r 200 -t GET
-u http://10.0.0.57 -x 24 -p 3 -l 100
```

Parametry útoku typu Slow GET byly nastaveny následovně: 400 TCP spojení o rychlosti navazování 200 spojení za sekundu; maximální velikost udržovacích částí požadavku 24 bajtů; časovač pro kontrolu dostupnosti 3 sekundy; interval mezi zasíláním částí požadavku 10 sekund; trvání útoku 100 sekund.

První sekundy útoku bohatě stačí na to, abychom mohli sledovat zaplnění všech volných prostředků web serveru, jenž stačily obsloužit přibližně 280 spojení. Zbylých 120 spojení zůstalo neobsloužených. K odepření služby došlo ihned po zaplnění těchto prostředků, tedy přibližně po první sekundě útoku. Po celou dobu útoku nedošlo k ukončení žádného z navázaných spojení a výpočetní prostředky web serveru tak zůstaly obsazeny po celou dobu útoku. Průběh útoku vyobrazuje graf na obrázku 6.2.

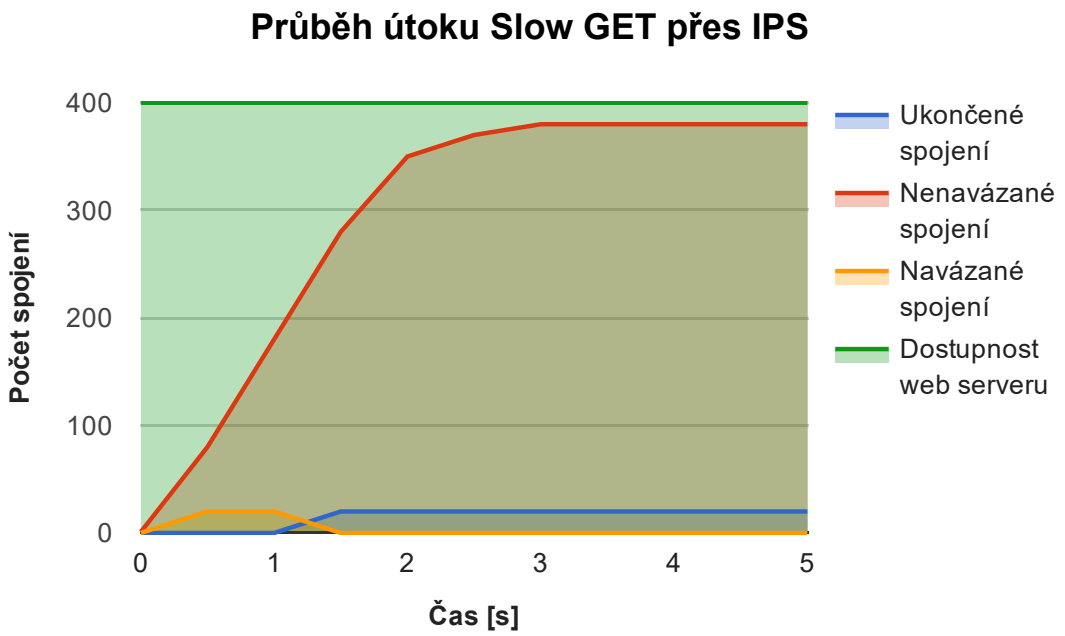


Obr. 6.2: Průběh útoku Slow GET bez IPS

V případě nasazení vytvořeného systému prevence průniku lze z grafu průběhu útoku na obrázku 6.3 pozorovat vliv implementovaného preventivního omezení počtu spojení z jedné IP adresy, jenž nedovolí zahltit web server dalšími požadavky. Došlo tedy k navázání prvních 20 spojení.

Propuštěné požadavky z těchto spojení byly následně analyzovány jako nevalidní požadavky útoku Slow GET na základě chybějících ukončovacích znaků `\r\n\r\n` v GET požadavku. Tato spojení byla přidána do databáze. Díky tomu program zjistil průchod nevalidních požadavků ze všech 20 spojení jedné IP adresy, což znamená překročení detekčního prahu detektoru a situace byla vyhodnocena jako útok. Filtrační server následně zablokoval veškeré další příchozí pakety od útočníka. Přibližně

po první sekundě útočník i web server přijaly od filtračního serveru TCP segment s příznakem RST a adekvátním pořadovým číslem, čímž došlo k ukončení spojení a k uvolnění výpočetních prostředků. Webový server tedy zůstal po celou dobu útoku stále dostupný legitimním uživatelům. Omezení jeho funkčnosti bylo minimální.



Obr. 6.3: Průběh útoku Slow GET s IPS

V případě, že útok generuje méně spojení, než je v detektoru nastaveno jako limit, nedojde k blokaci útočníka, ale útok bude ve velmi omezené míře pokračovat. V závislosti na nastavených parametrech útoku začne útočník po určité době vysílat udržovací pakety, aby udržel spojení obsazené. Detektor tyto pakety zachytí a aktualizuje záznamy ve své databázi. S každým přijatým udržovacím paketem detektor inkrementuje databázový údaj s počtem přijatých a neukončených částí HTTP požadavku pro dané spojení. Jakmile tato hodnota přesáhne stanovený limit (nastaveno na 10), je toto spojení považováno za útočící.

Pokud je v parametrech útoku nastaven velký časový rozestup udržovacích paketů, může naplnění tohoto zásobníku trvat dlouhou dobu. Proto detektor sleduje i čas od příchodu první části HTTP záhlaví. Po přesáhnutí povolené doby čekání na ukončení požadavku je toto spojení považováno za útočící (nastaveno na 10 sekund).

Výše jsou popsány 3 možné situace, kdy detektor zachytí útok. Detektor tak reaguje na různá nastavení vedeného útoku. V situaci, kdy je útok zachycen, detektor tuto informaci zobrazí v terminálu pomocí standardního výstupu a uvede, který z limitů byl překročen.

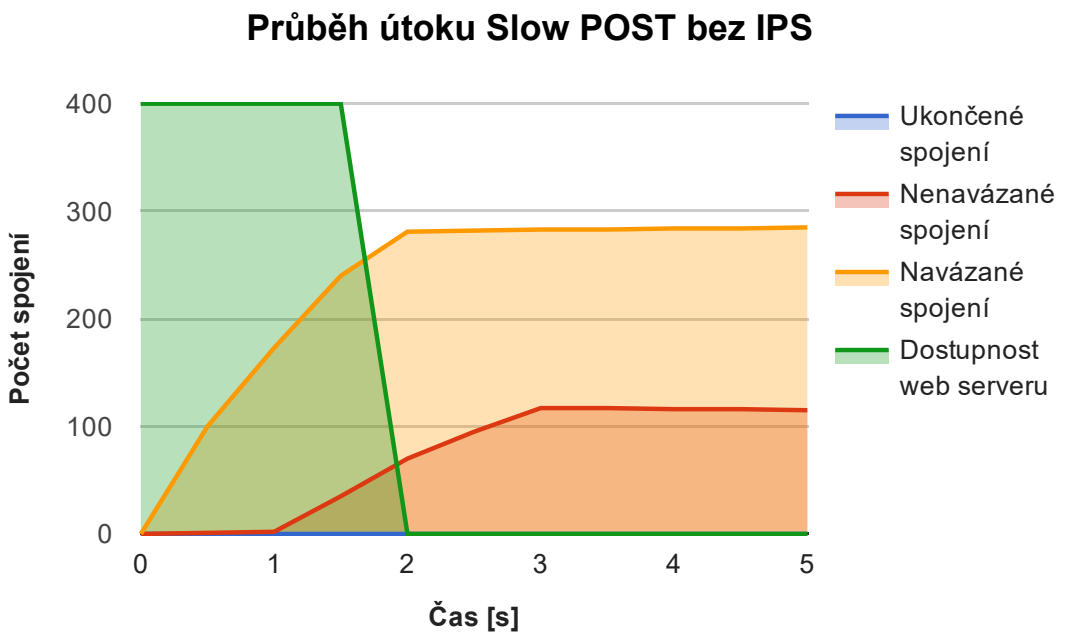
6.4 Slow POST

Útok Slow POST byl spuštěn následujícím příkazem:

```
slowhttptest -c 400 -B -g -o slow_post -i 110 -r 200 -s 8192
-t FAKEVERB -u http://10.0.0.57/form.html -x 10 -p 3 -l 100
```

Parametry útoku typu Slow POST byly nastaveny následovně: 400 TCP spojení o rychlosti navazování 200 spojení za sekundu; maximální velikost udržovacích částí požadavku 22 bajtů; velikost dat v poli `Content-lenght` 8192 bajtů; časovač pro kontrolu dostupnosti 3 sekundy; interval mezi zasíláním částí požadavku 10 sekund; slovo `FAKEVERB` jako identifikátor typu požadavku a trvání útoku 100 sekund.

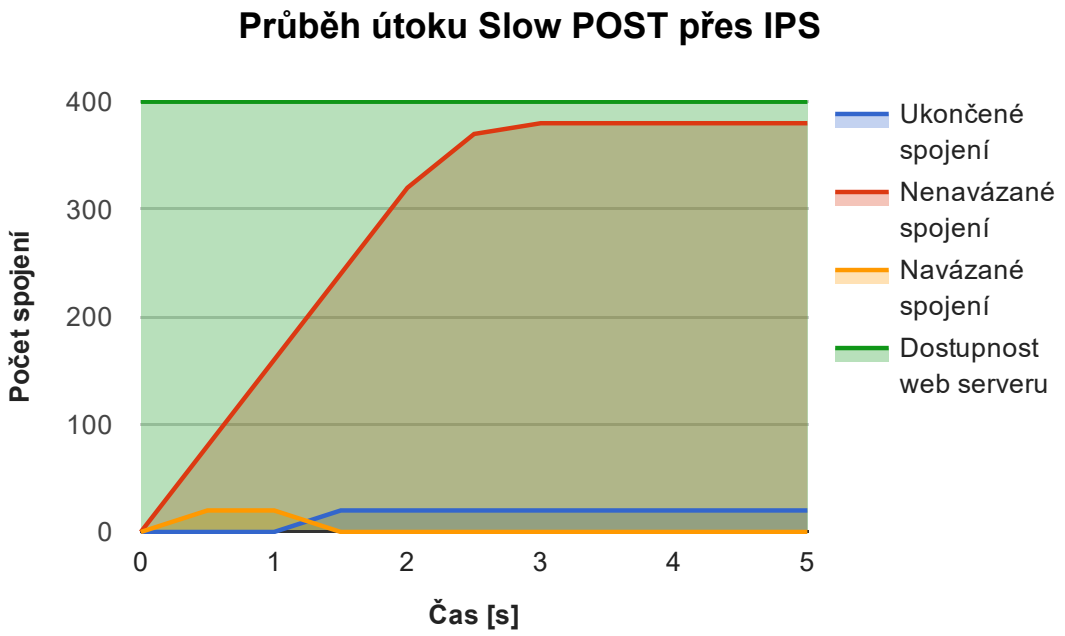
Útok typu Slow POST je v podstatě stejného charakteru jako útok Slow GET, tzn. u obou útočník odesílá data po částech a s velmi malou rychlostí. Útok provedený bez účasti IPS proto vypadá téměř stejně (viz obrázek 6.4). K odepření služby došlo ihned po začátku útoku.



Obr. 6.4: Průběh útoku Slow POST bez IPS

Při útoku s účastí implementovaného IPS postupoval tento systém obdobně jako při útoku Slow GET. Rozdíl byl pouze v identifikaci typu útoku. Systém detekoval pole `Content-lenght` a porovnal jej s velikostí dat požadavku. Jelikož velikosti byly různé, detektor identifikoval přijaté požadavky jako nevalidní požadavky příznačné pro možný útok typu Slow POST. Detektor dále, podobně jako u útoku Slow GET, sledoval počet spojení, ze kterých tyto požadavky přicházejí, dokončení požadavků

v očekávaném čase, množství částí jednotlivých požadavků a v tomto případě detektor rovněž hlídal i aktuální přenosovou rychlost dat. Díky tomu program zjistil průchod nevalidních požadavků ze všech 20 propuštěných spojení jedné IP adresy, což znamená překročení detekčního prahu detektoru a situace byla vyhodnocena jako útok. Systém následně provedl blokaci útočníka a uvolnění prostředků jako u útoku Slow GET. Webový server tedy zůstal po celou dobu útoku stále dostupný legitimním uživatelům. Jak dokazuje graf průběhu útoku na obrázku 6.5, omezení funkčnosti web serveru bylo minimální.



Obr. 6.5: Průběh útoku Slow POST s IPS

6.5 Apache Range Header

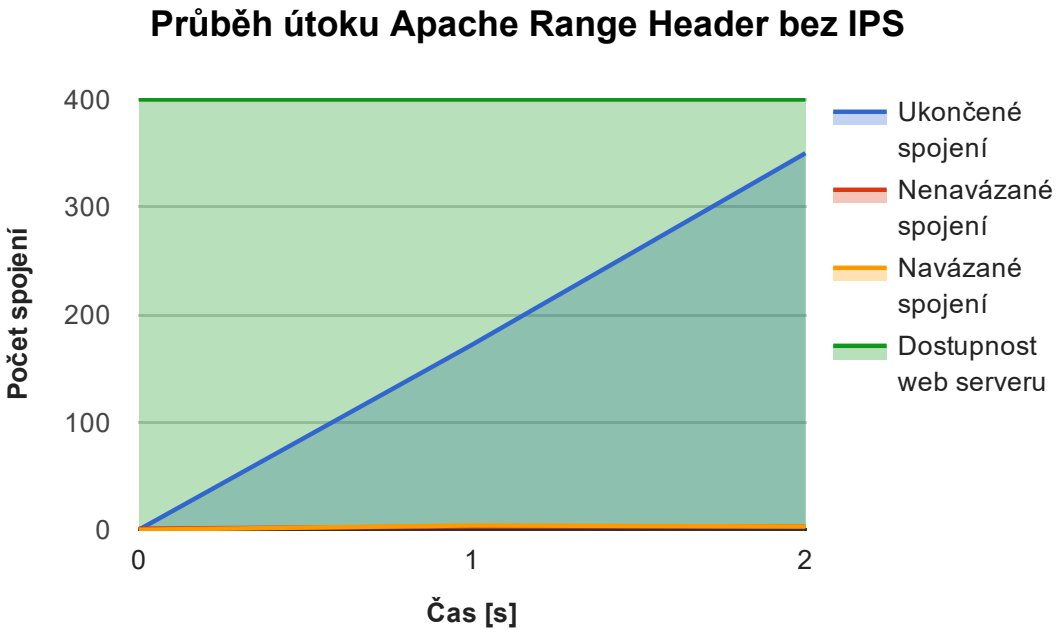
Pro spuštění útoku typu Apache Range Header byl použit následující příkaz:

```
slowhttptest -R -u http://10.0.0.57/ -t HEAD -c 400 -a 10
-b 3000 -r 200 -l 100
```

Parametry útoku byly nastaveny následovně: 400 TCP spojení o rychlosti navazování 200 spojení za sekundu, hranice požadovaných rozsahů dat v požadavku od 10 do 3000 bajtů a trvání útoku 100 sekund.

Jak lze pozorovat z průběhu útoku na obrázku 6.6, web server ukončil každé spojení útočníka ihned po přijetí nevalidního požadavku. Tahle situace nastala díky

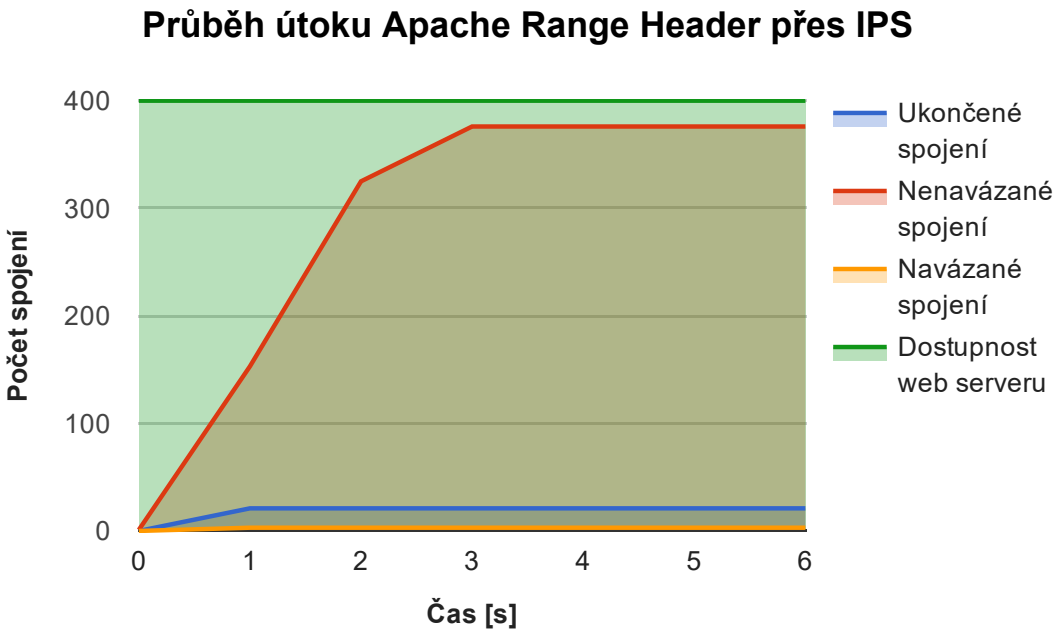
ošetření této zranitelnosti web serveru Apache již od verze 2.2.20. Funkčnost webového serveru tedy zůstala netknuta, protože se dokázal ubránit sám. Po 2 sekundách od spuštění útoku došlo k jeho ukončení z důvodu odmítnutí všech požadavků.



Obr. 6.6: Průběh útoku Range Header bez IPS

V případě nasazení vytvořeného IPS byl útočník opět omezen na navázání pouze 20 spojení. Příchozí požadavky byly následně analyzovány a systém detekoval nevalidní hodnotu pole `Range`. Po přijetí nevalidních požadavků ze všech 20 navázaných spojení, filtrační server cestu k útočníkovi zablokoval a tato spojení uvolnil pomocí rozeslání TCP segmentů s příznakem RST a s adekvátními sekvenčními čísly. Funkčnost webového serveru opět zůstává netknutá. Jediný rozdíl oproti situaci bez IPS je, že omezení na 20 současných spojení z jedné IP adresy, implementované v IPS, zmírnilo nápor nevalidních požadavků na web server a zbývajících 380 požadavků IPS zahodil. Díky tomu se web server již nemusel těmito požadavky zabývat. Tuto situaci zobrazuje graf na obrázku 6.7.

Pokud by byl počet spojení z útočníkovi stanice snížen pod detekční práh, tedy 20 spojení, detektor by sledoval také počet příchozích nevalidních požadavku. Pokud by detektor přijal více nevalidních požadavků, než je dovoleno (nastaveno na 5 požadavků), tak je toto spojení rovněž považováno za útočící.



Obr. 6.7: Průběh útoku Range Header s IPS

6.6 Slow Read

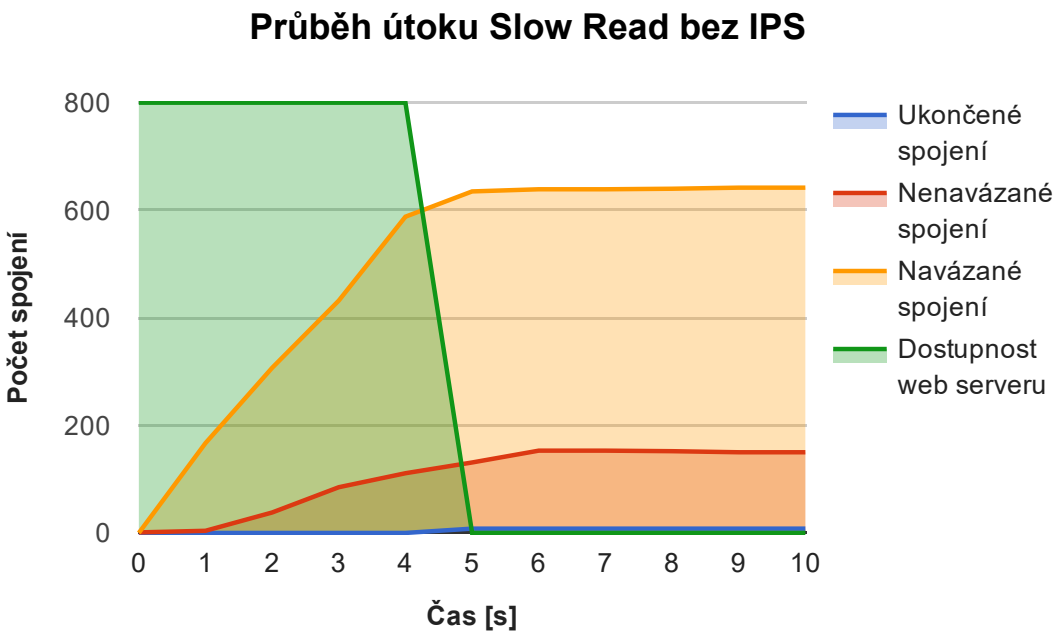
Útok Slow Read byl spuštěn následujícím příkazem:

```
slowhttptest -c 800 -X -r 200 -w 10 -y 100 -n 5
-z 32 -k 3 -u http://10.0.0.57/ -p 3 -l 100
```

Útok Slow Read byl nastaven tak, aby navázal 800 spojení rychlostí 200 spojení za sekundu a útok trval 100 sekund. Velikost okna byla nastavena na rozmezí 10 – 100 bajtů a HTML pipeline faktor pro zapouzdření desíti požadavků do jednoho segmentu.

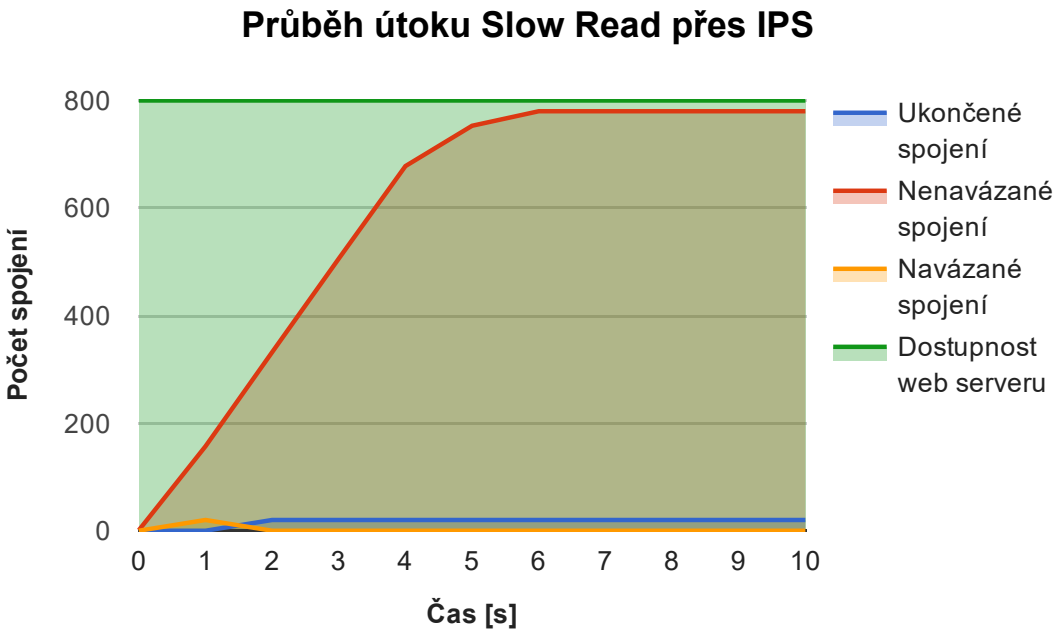
Útok bez využití IPS zobrazuje graf na obrázku 6.8. Web server zvládl obsloužit přibližně 620 požadavků, než se po zhruba 4 sekundách zhroutil pro nedostatek výpočetních prostředků. Došlo tedy k odepření služby.

V případě nasazení vytvořeného IPS (obrázek 6.9) byl útočník schopen navázat 20 spojení pomocí TCP segmentu s příznakem SYN. Detektor zjistil u všech těchto segmentů přítomnost velmi malého TCP okna, což vedlo vyhodnocení, ze se jedná o útok. Do doby, než byl schopen detektor provést analýzu, útočník již s web serverem navázal kompletní TCP spojení a zažádal o velké množství dat hned desíti požadavky na každém spojení. Na tyto požadavky začal web server odpovídat velmi malou rychlostí, jenž byla ustanovena pomocí TCP okna. Po detekci útoku



Obr. 6.8: Průběh útoku Slow Read bez IPS

detektor útočníka zablokoval a rozeslal ukončovací TCP segmenty, čímž uvolnil obsazené prostředky serveru. Omezení funkčnosti serveru bylo po celou dobu útoku minimální.



Obr. 6.9: Průběh útoku Slow Read s IPS

Stejně jako u předchozích útoků, detektor v tomto případě zachytil větší množství nevalidních segmentů, než bylo dovoleno. To vedlo k zjištění přítomnosti útoku. Pokud však útočník otevře méně než 20 spojení, detektor zjistí útok na základě překročení očekávaného času pro přenos dat.

V této situaci útočník mohl přizpůsobit požadavky a velikosti oken tak, aby data přijal do serverem maximálně přípustných třiceti sekund. Útočník je tak donucen posílat více požadavků na menší množství dat. Počet požadavků je ale detektorem limitován. Při přijetí desíti segmentů s velmi malým TCP oknem je situace opět vyhodnocena jako útok. Kromě toho detektor rovněž hlídá pokles přenosové rychlosti, která nesmí klesnout pod 10 bajtů za sekundu.

7 ZÁVĚR

Cílem této práce bylo prostudovat problematiku pomalých DoS útoku a navrhnout metody detekce čtyř nejznámějších typů útoku – Slow GET, Slow POST, Apache Range Header a Slow Read. Cílem praktické části této práce bylo pomocí navržených metod detekce vytvořit program v jazyce C, který bude fungovat jako systém prevence průniku. Tento systém má být schopen zmíněné útoky odhalit a ochránit webový server výpadkem poskytovaných služeb. Celý tento systém má být optimalizován pro reálné využití na mezilehlém filtračním serveru.

V úvodu práce byla provedena stručná charakteristika síťového provozu, zaměřená na popis referenčního modelu TCP/IP a protokolu HTTP. Tyto znalosti představují nutný teoretický základ k pochopení dalších kapitol. Práce dále popisuje současnou situaci kolem DoS útoků se zaměřením především na poměrně novou a nebezpečnou skupinu tzv. pomalých DoS útoků (Slow DoS). Byla zde provedena analýza nejznámějších typů pomalých DoS útoku včetně popisu jejich principu.

Další samostatná kapitola byla věnována obraně před DoS útoky, návrhem základních preventivních opatření pro webové servery a hlavně návrhem metod pro detekci pomalých DoS útoků, jenž jsou následně implementovány v praktické části této práce.

Praktická část popisuje vytvoření aplikace v jazyce C, která je schopna detekovat výše zmíněné DoS útoky za pomocí technik navržených v teoretické části. Tato aplikace funguje na principu zachytávání a analýzy síťového provozu, k čemuž jsou využity knihovny LipPcap. Její další klíčová schopnost je blokovat probíhající útok a zamezit vlivu na poskytované služby webového serveru.

V další části práce je objasněno, jak lze vytvořený systém zprovoznit a používat. Jsou zde uvedeny všechny potřebné knihovny a příkazy pro instalaci v operačním systému linuxového typu. Dále je zde vysvětleno, jak lze přizpůsobit citlivost detektoru pro konkrétní webový server a typ provozu.

V závěrečné části této práce je sestavena testovací síť obsahující počítač útočníka a uživatele, filtrační server s vytvořeným IPS a počítač s webovým serverem. Kapitola popisuje jejich podrobnější specifikaci a dále použitý nástroj pro generování útoku.

První fáze testování prověřila funkčnost všech útoku, kterým se podařilo cílový server vyřadit z provozu. V další fázi testování již útok procházel filtračním serverem s aktivním systémem prevence průniku. Detektor tohoto systému odhalil útoky prakticky okamžitě. Rychlost detekce, resp. citlivost je závislá na nastavených parametrech detektoru. Díky tomu lze detektor přizpůsobit pro individuální nasazení. Systém následně provedl potřebná opatření pro odražení útoku a obnovení plné funkčnosti webového serveru.

Závěrem lze konstatovat, že vytvořený systém prevence průniku splňuje stanovené požadavky a svůj účel. Je dosaženo úspěšné detekce a odražení útoků aniž by byla jakkoliv omezena dostupnost služeb pro legitimní uživatele. Po detekci útoku systém zajistí filtraci veškerého provozu od útočníka a zároveň uvolnění všech útočníkem obsazených výpočetních prostředků webového serveru. Vytvořený systém je optimalizován a testován v reálné síti na mezilehlém filtračním serveru. Lze jej tedy využít k obraně před útoky v praxi, případně využít navržené principy pro konstrukci složitějších aplikačních firewallů.

Předmětem dalšího vývoje může být například doplnění podpory dalších protokolů jako IPv6 a HTTPS či rozšíření schopností detektoru pro rozpoznávání dalších typů útoků.

LITERATURA

[1] BOUŠKA, P. TCP/IP – model, encapsulace, paket vs. rámec. *SAMURAJ–cz* [online]. 16.08.2007 [cit. 26. 10. 2016]. Dostupné z: <http://www.samuraj-cz.com/clanek/tcpip-model-encapsulace-paketu-vs-ramec/>

[2] POSTEL, J. *Internet Protocol* [online]. STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, [cit. 26. 10. 2016]. Dostupné z: <http://www.rfc-editor.org/info/rfc791>

[3] POSTEL, J. *Transmission Control Protocol* [online]. STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, [cit. 26. 10. 2016]. Dostupné z: <http://www.rfc-editor.org/info/rfc793>

[4] LORENC, M. *Transakční a časová detekce odchozích DoS útoků* [online]. Brno, 2015 [cit. 03. 12. 2016]. Dostupné z: <http://is.muni.cz/th/325453/fi_m_a2/DP_text_sajtkgzq.pdf>. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Mgr. Vít Bukač, Ph.D.

[5] DOBEŠ, J. *Analýza nástrojů provádějících pomalé DoS útoky* [online]. Brno, 2015 [cit. 03. 12. 2016]. Dostupné z: <http://is.muni.cz/th/325453/fi_m_a2/DP_text_sajtkgzq.pdf>. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Mgr. Vít Bukač, Ph.D.

[6] JUHAŇÁK, P. *Generování a ochrana proti DOS útoku na aplikační vrstvě* [online]. Brno, 2016 [cit. 03. 12. 2016]. Dostupné z: <https://dspace.vutbr.cz/handle/11012/62107>. Bakalářská práce. VUT Brno, Fakulta informatiky. Vedoucí práce Ing. Petr Musil.

[7] KLOUČEK, L. *Analýza nástrojů pro provádění útoků denial-of-service* [online]. Brno, 2012 [cit. 03. 12. 2016]. Dostupné z: <http://is.muni.cz/th/207385/fi_m/DP.pdf>. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Mgr. Vít Bukač, Ph.D.

[8] CAMBIASO, E.; PAPALEO, G.; AIELLO, M. *Taxonomy of Slow DoS Attacks to Web Applications* v Recent Trends in Computer Networks and Distributed Systems Security, Berlin, 2012.

[9] KRČMÁŘ, P. *Útok Slowloris aneb plíživé nebezpečí pro web servery* 17.5.2011. [online]. [cit. 03. 12. 2016]. Dostupné z: <https://goo.gl/Lccv06>.

[10] Develop a Packet Sniffer with Libpcap. *vic hargrave* [online]. 2012 [cit. 12. 5. 2017]. Dostupné z: <https://vichargrave.github.io/articles/2012-12/develop-a-packet-sniffer-with-libpcap>

[11] BALDWIN, M. Mitigating the Apache Range Header DoS Vulnerability. *Infosec Island* [online]. 2011 [cit. 8. 5. 2017]. Dostupné z: <https://goo.gl/U17PGI>

[12] SHEKYAN, S. Are you ready for slow reading? *Qualys Blog: Security Labs* [online]. 2012 [cit. 7. 5. 2017]. Dostupné z: <https://blog.qualys.com/securitylabs/2012/01/05/slow-read>

[13] SEIFRIED, K. Dissecting network traffic: wireshark, libpcap, packets, packet sniffer. *LINUX MAGAZINE* [online]. 2009(109) [cit. 12. 5. 2017]. Dostupné z: <http://www.linux-magazine.com/Issues/2009/109/Security-Lessons#>

[14] DOČEKAL, M. Správa linuxového serveru: Linuxový firewall, základy iptables. *LinuxEXPRES: opravdový linuxový magazín* [online]. 2010 [cit. 20. 5. 2017]. ISSN 1801-3996. Dostupné z: <https://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-linuxovy-firewall-zaklady-iptables>

[15] Code raw sockets in C on Linux. *BinaryTides* [online]. 2009 [cit. 16. 5. 2017]. Dostupné z: <http://www.binarytides.com/raw-sockets-c-code-linux/>

[16] SHEKYAN, S. slowhttptest(1) - Linux man page. *linux.die.net* [online]. 2012 [cit. 17. 5. 2017]. Dostupné z: <https://linux.die.net/man/1/slowhttptest>

[17] BARNETT, R. Mitigation of 'Slow Read"Denial of Service Attack: ModSecurity Advanced Topic of the Week. *Trustwave: SpiderLabs® Blog* [online]. 2011 [cit. 80. 5. 2017]. Dostupné z: <https://goo.gl/FX7l6s>

[18] DAMON, E.; DALE, J.; LARON, E.; MACHE, J.; LAND, N.; WEISS, R. Hands-on denial of service lab exercises using SlowLoris and RUDY. *Proceedings of the 2012 Information Security Curriculum Development Conference on – InfoSecCD '12* [online]. New York, New York, USA: ACM Press, 2012, 21–29 [cit. 26. 10. 2016]. DOI: 10.1145/2390317.2390321. ISBN 9781450315388. Dostupné z: <http://dl.acm.org/citation.cfm?doid=2390317.2390321>

[19] DURAVKIN, I.; LOKTIONOVA, A.; CARLSSON, A. Method of slow-attack detection. *2014 First International Scientific-Practical Conference Problems of Infocommunications Science and Technology* [online]. IEEE, 2014, , 171-172 [cit. 26. 10. 2016]. DOI: 10.1109/INFOCOMMST.2014.6992341. ISBN 978–1–4799–7342–2. Dostupné z: <http://ieeexplore.ieee.org/document/6992341/>

[20] SHEKYAN, S. How to Protect Against Slow HTTP Attacks? *Qualys Blog: Security Labs* [online]. 2011 [cit. 9. 5. 2017]. Dostupné z: <https://blog.qualys.com/securitylabs/2011/11/02/how-to-protect-against-slow-http-attacks>

[21] Apparatus and method for detecting slow read DoS attack. *Computer Weekly News* [online]. Atlanta, United States: NewsRx, 30. 10. 2014, 5 [cit. 26. 10. 2016]. Dostupné z: <http://search.proquest.com/docview/1615006533?accountid=17115>

[22] Method and Protection System for Mitigating Slow HTTP Attacks Using Rate and Time Monitoring. *Computer Weekly News* [online]. Atlanta, United States: NewsRx, 21. 3. 2013, 6 [cit. 26. 10. 2016]. Dostupné z: <http://search.proquest.com/docview/1316086748?accountid=17115>

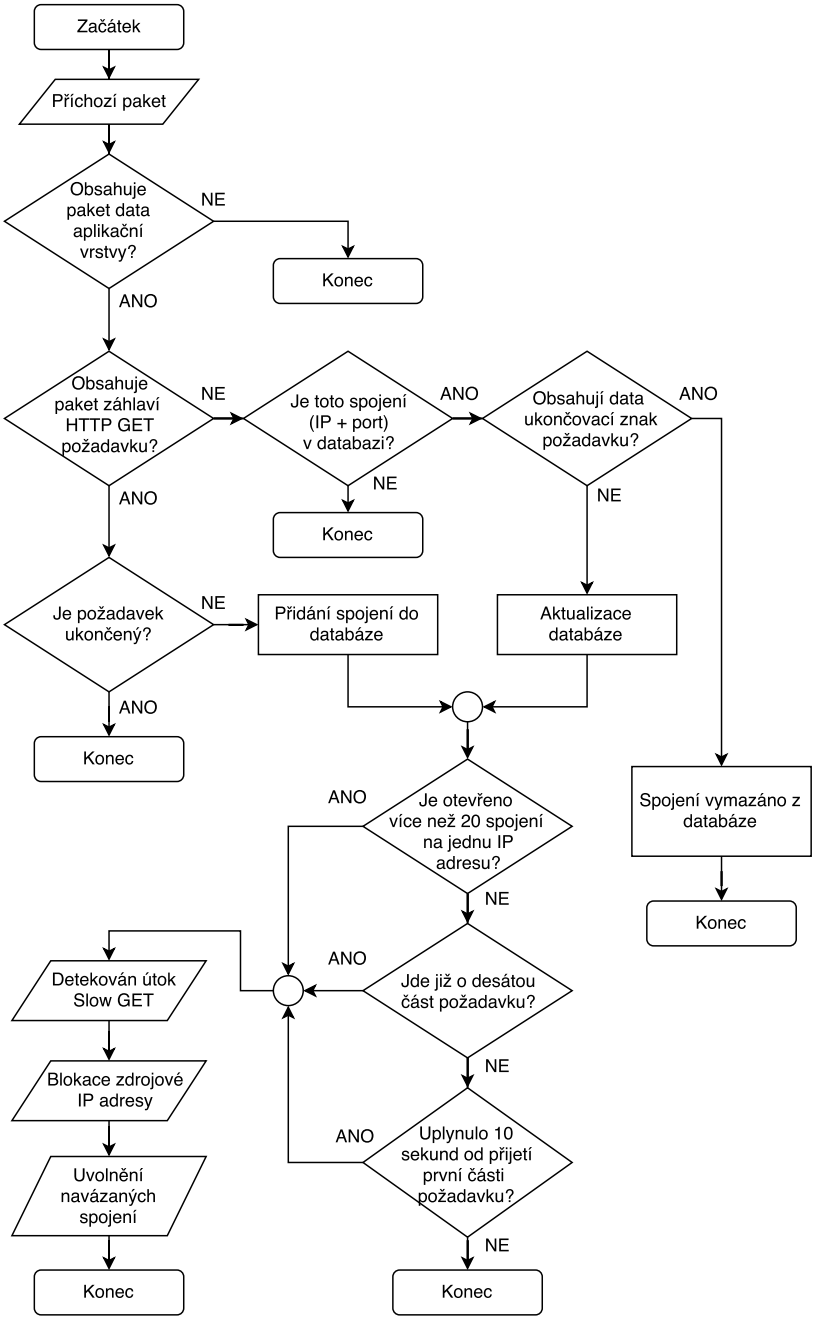
SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

HTTP	Hypertext Transfer Protocol
DoS	Denial of Service
DDoS	Distributed Denial of Service
TCP	Transmission Control Protocol
IP	Internet Protocol
IPS	Intrusion Prevetntion Systems – systém prevence průniku

SEZNAM PŘÍLOH

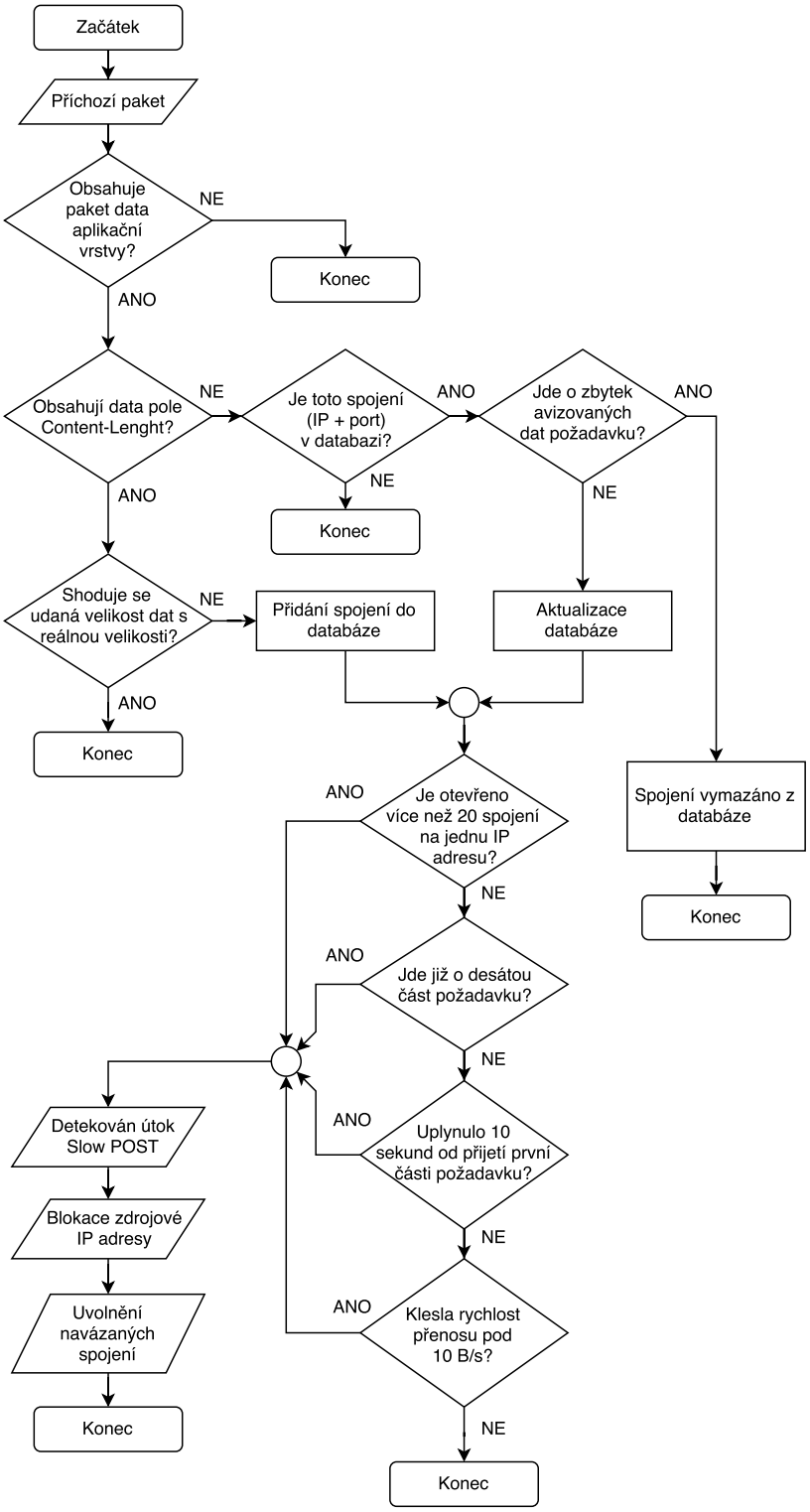
A VÝVOJOVÉ DIAGRAMY DETEKTORU

A.1 Algoritmus detekce útoku Slow GET



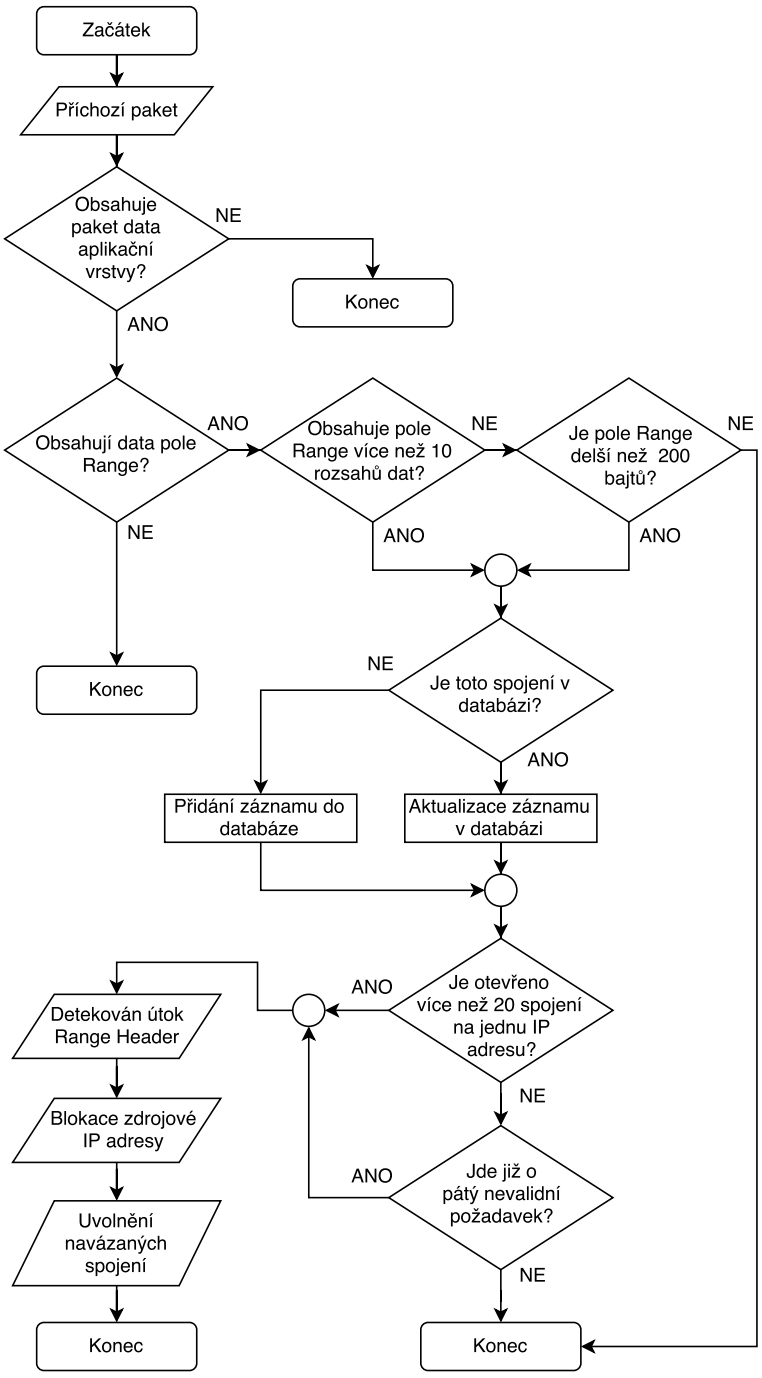
Obr. A.1: Vývojový diagram detektoru útoku typu Slow GET

A.2 Algoritmus detekce útoku Slow POST



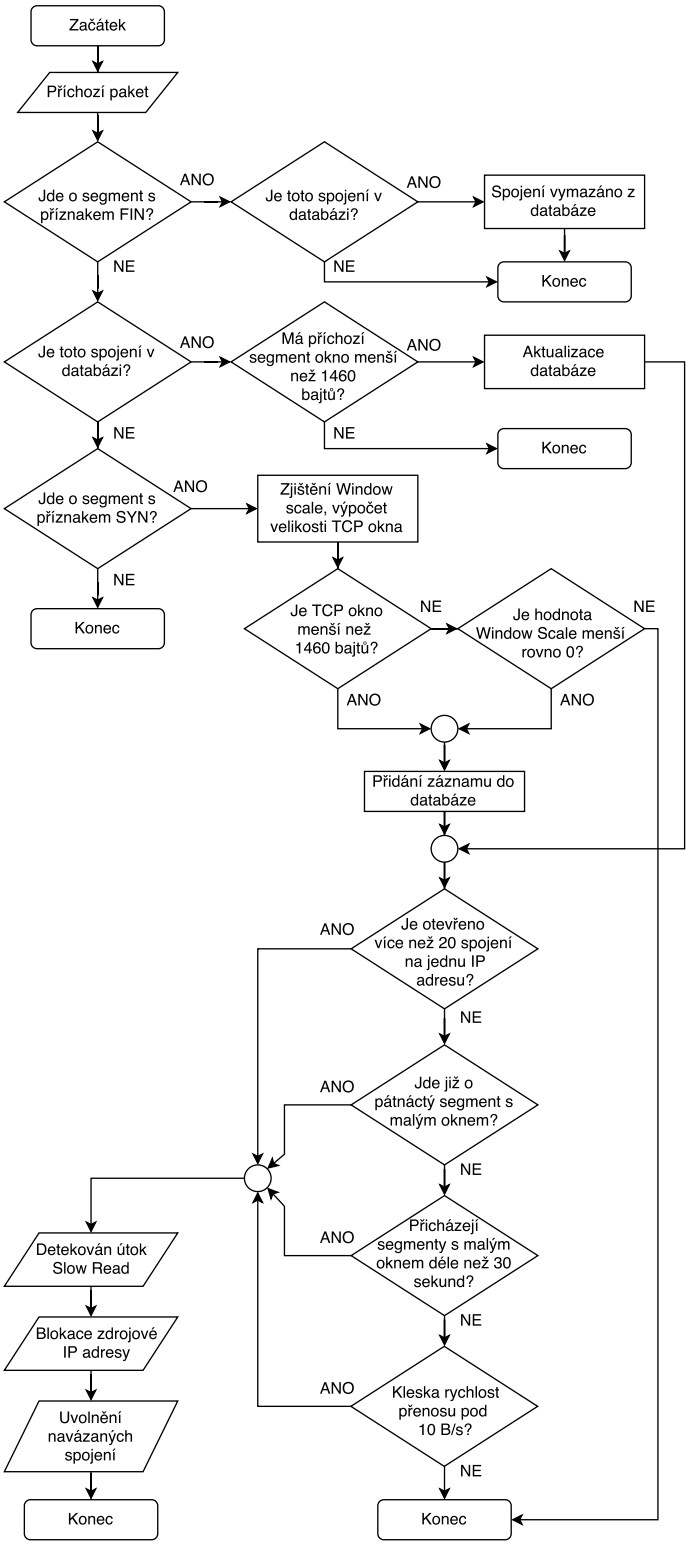
Obr. A.2: Vývojový diagram detektoru útoku typu Slow POST

A.3 Algoritmus detekce útoku Apache Range Header



Obr. A.3: Vývojový diagram detektoru útoku typu Apache Range Header

A.4 Algoritmus detekce útoku Slow Read



Obr. A.4: Vývojový diagram detektoru útoku typu Slow Read

B OBSAH PŘILOŽENÉHO CD

Přiložené CD obsahuje soubory z adresáře projektu vývojového prostředí Code::Blocks IDE 13.12, ve kterém byl systém prevence průniku vytvořen. Adresář obsahuje pracovní soubory a nastavení projektu tohoto vývojového prostředí, dále soubor `main.c` se zdrojovým kódem aplikace. Ten je možné dodatečně upravovat kvůli přizpůsobení prahových hodnot detektoru. Pro spuštění upraveného programu je potřeba kód znovu zkompilovat.

Přiložené CD dále obsahuje samotný spustitelný program s názvem `Detektor`. Zdrojový kód byl zkompilován pomocí překladače GNU GCC Compiler 5.4.0.

Na přiloženém CD se dále nachází elektronická kopie této práce.

```
/ ..... kořenový adresář přiloženého CD
├── bin ..... pracovní soubory vývojového prostředí Code::Blocks
│   └── Debug
│       └── diplomka_1
├── obj ..... pracovní soubory vývojového prosředí Code::Blocks
│   └── Debug
│       └── main.o
├── diplomka_1.cbp ..... projekt vývojového prosředí Code::Blocks
├── diplomka_1.depend .... pracovní soubor vývojového prostředí Code::Blocks
├── diplomka_1.layout .... pracovní soubor vývojového prostředí Code::Blocks
├── main.c ..... zdrojový kód vytvořené aplikace
├── Detektor ..... výsledná spustitelná aplikace
└── DiplomovaPrace.pdf ..... elektronická kopie této práce
```