VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2016 Jiří Zmeškal

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

KYBERNETICKÉ ÚTOKY
CYBER ATTACKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Jiří Zmeškal
AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Číka, Ph.D.
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Jiří Zmeškal ***ID:*** 167729
Ročník: 3 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Kybernetické útoky

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte funkce a možnosti zátěžových testů v TCP/IP. Prostudujte existující open source řešení vhodná pro provádění zátěžových testů. Vytvořte vlastní aplikaci, která bude schopná provést zátěžové testy (HTTP Flood, Slow HTTP a další). Zaměřte se na více vláknové zpracování a taktéž na možnost využití distribuovaných útoků DDoS. Pro testování využijte hardware a infrastrukturu na Vysokém učení technickém v Brně.

DOPORUČENÁ LITERATURA:

[1] MONTE, Matthew. Network Attacks and Exploitation: A Framework. Indianapolis: John Wiley & Sons, Inc., 2015. ISBN 978-1118987124.

[2] YU, Shui. Distributed Denial of Service Attack and Defense (SpringerBriefs in Computer Science). 2014. Melbourne: Springer, 2014. ISBN 978-1461494904.

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 1.6.2016

Vedoucí práce: Ing. Petr Číka, Ph.D.
Konzultant bakalářské práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ANOTACE

Teoretická část práce se věnuje popisu některých základních pojmů síťové komunikace, která v posledních letech zaznamenala obrovský rozmach. Je jí využíváno ve většině každodenních činností, a to od komunikace mezi lidmi přes internetové nákupy a bankovnictví až po vzdálené řízení počítačem řízených strojů. Dále je teoreticky popsáno, jak mohou útočníci zneužít vlastností a nedostatků komunikačních protokolů k páchání nelegálních aktivit, specificky ve formě útoků blokujících veřejně dostupné služby. Je zvoleno několik typů útoků na transportní a aplikační vrstvě, které jsou popsány. Práce také osahuje informace o nalezených open source řešeních. V praktické části jsou následně uvedeny některé možnosti realizace dvou typů útoků, a to HTTP GET Flood a Slow HTTP GET, které jsou součástí vytvořené aplikace. Je zde popsán proces tvorby aplikace, více vláknové zpracování, použité volně dostupné prostředky (například knihovna Boost) a komplikace, které nastaly během programování. Dále jsou také popsány potíže při snaze zprovoznit aplikaci pro operační systém Windows na Linuxu. Aplikace je zamýšlena pouze pro testovací a vzdělávací účely.

Klíčová slova: DoS, DDoS, kybernetický útok, blokování služby, HTTP

ABSTRACT

Theoretical part of this thesis is dedicated to describing basic terms of network communication. Usage of network communication has become a necessity and is used on daily basis for a number of purposes, starting with communication between people, going across internet shopping and banking all the way to remote controlling industrial machinery. Another theoretically described topic is how attackers abuse attributes and shortcomings of communication protocols in order to commit illegal activities, specifically denial of service type of attacks. Finally, theoretical part includes a list of found open source applications capable of launching such attacks. Practical part describes development of application, capable of launching two selected forms of attack. These attacks are HTTP GET Flood, based on sending massive amounts of GET request, and Slow HTTP GET, based on imitating a user with slow internet connection. The development is described step by step and includes multithread processing, used publicly available components (Boost library for example) and challenges encountered during development (such as library limitations and cross platform compatibility).

Keywords: DoS, DDoS, cybernetic attack, denial of service, HTTP

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma „Kybernetické útoky“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících 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.

V Brně dne ……. .....

podpis autora

Poděkování

Děkuji vedoucímu práce Ing. Petru Číkovi, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne …….. .....

podpis autora

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 bakalářské 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

1. Úvod

Lidé měli odjakživa tendence si polepšit nebo se pobavit na úkor druhých. Tento trend se samozřejmě nevyhnul ani kybernetickému prostoru. Naopak, v kombinaci s lidským talentem využívat vynálezy k jiným účelům, než byly původně zamýšleny, a s neznalostí problematiky běžnou populací dává kyberprostor za vznik dokonalému prostředí pro páchání zločinů a špatných vtipů.

Původní myšlenkou globální komunikační sítě bylo zjednodušení komunikace a sdílení informací naskrz celým světem. Nikdo si tou dobou nemyslel, že by něco takového mohlo být někým zneužíváno ke konání kriminální aktivity, což mělo za následek nedostatečné, ba dokonce nulové, zabezpečení této sítě. Tato slabina se projevila již u předchůdce a základního kamene internetu – u *Arpanetu*. V době, kdy *Arpanet* obsahoval asi 60 000 počítačů, se objevil první, samostatně se replikující program, program nazývaný *Moris Worm* [1], který neúmyslně způsobil zahlcení a pád zhruba deseti procent této sítě.

V současnosti je již síť internet rozšířena téměř do všech sebeodlehlejších míst a jejích služeb využívají miliardy lidí po celém světě. Přes internet se začalo dělat téměř vše: od výměny dat a informací, přes telefonování po internetu, sociální služby, nakupování, internetové bankovnictví, až po dálkovou obsluhu strojů. Tyto služby se natolik rozšířily, že mnoho společností není již schopno bez internetu fungovat. Rozvoj síťových technologií musí tedy držet krok hlavně s neustále rostoucí klientelou a požadavky na vyšší přenosovou rychlost, zatímco bezpečnost pokulhává. Tohoto nedostatku si poměrně rychle začali lidé všímat a dali počátek myšlence jak toho zneužít. Motivace těchto lidí je různá a sahá od zábavy přes průmyslovou špionáž až po systematickou likvidaci konkurence.

Cílem práce je seznámit se s *DoS* (*Denial of Service* – zablokování služby [1]**Chyba! enalezen zdroj odkazů.**) a *DDoS* (*Distributed Denial of Service* – distribuované zablokování služby[1][2]) útoky, zneužívají slabin různých komunikačních protokolů k vyčerpání síťových, aplikačních nebo výpočetních zdrojů cíle útoku, kterým nejčastěji bývá internetový server. Jedním z hlavních důvodů zaměření práce na tyto útoky je rapidní nárůst počtu útoků v posledních letech, způsobený hlavně relativní jednoduchostí provedení samotného útoku a složitosti obrany proti němu.

2. Obecné pojmy

2.1. UDP (User Datagram Protocol)

Je jedním z protokolů transportní vrstvy. Poskytuje jednoduchý a rychlý přenos dat bez vytváření jakéhokoli spojení nebo čekání na odezvy. Vzhledem k těmto vlastnostem je UDP ideální pro aplikace, které vyžadují přenos v reálném čase, a u kterých až tak nevadí, pokud se sem tam pár paketů ztratí, zdvojí, přijde mimo pořadí, nebo obsahuje chybu. Nejčastěji je ho využíváno pro multimediální přenosy (IP televize, streamování) a internetovou telefonii (VoIP).

2.2. TCP (Transmission Control Protocol)

Z hlediska přenosu dat se jedná o absolutní opak UDP protokolu. Jediné, co mají společné je zařazení do transportní vrstvy. Jde současně o nejspíše nejrozšířenější protokol této vrstvy.

TCP je spojově orientovaný což znamená, že před vlastním přenosem dat se musí sestavit spojení. K tomu je používán tříkrokový proces zvaný *three-way handshake* [3]. V prvním kroku zašle klient serveru žádost s aktivním příznakem SYN a náhodně vygenerovaným sekvenčním číslem. V druhém kroku odpovídá server klientovi zprávou s příznaky SYN a ACK, kde potvrzovací hodnota je nastavena na hodnotu o jednu vyšší, než původní sekvenční číslo. Nakonec v třetím kroku odesílá klient potvrzení s příznakem ACK a potvrzovací hodnotou opět o jednu vyšší než původní sekvenční číslo. Po obdržení této odpovědi serverem se mohou začít přesouvat data.

Dalším rozdílem mezi TCP a UDP je spolehlivost. U protokolu TCP je prioritou spolehlivost a bezchybnost přenosu. U všech dat je tedy potvrzována správnost dat a jejich doručení. Pokud jsou pakety někde zahozeny, nebo nedojdou v pořádku, je z daného síťového prvku odeslána zpráva pomocí ICMP protokolu, žádající zdroj dat o jejich opětovné zaslání.

Tento protokol je nejčastěji využíván pro přenosy dat, e-mailovou komunikaci a prohlížení internetových webových stránek.

2.3. Adresace v TCP/IP

Důležitou součástí síťové komunikace je způsob identifikace cílové a zdrojové stanice. U protokolového modelu TCP/IP je toho docíleno pomocí adres, a to hned na několika úrovních. Každý paket, obsahující plné záhlaví (existují i komprese záhlaví), si sebou nese informace o zdroji i cíli, a to ve formě tří různých adres.

Na transportní vrstvě je adresování realizováno pomocí čísla portu [4], které je obvykle specifické pro komunikující aplikaci a jeho přidělování se řídí určitými pravidly. Port je šestnácti bitové číslo a dělí se do tří skupin. První skupinou jsou známé porty, jejich rozsah je mezi 0 až 1023 a jsou prvně přiděleny známým službám a jsou využívány servery. Příkladem zde může být port 80 přidělený službě HTTP. Druhou skupinou jsou porty registrované, obsahující skupinu portů v rozmezí 1024 až 49151. Jde o porty aplikací, které byly jejich tvůrci registrovány u organizace IANA (Internet Assigned Numbers Authority), přidělující čísla portů. Obvykle jsou také využívány servery. Třetí skupinou je skupina dynamicky přidělovaných portů. Obsahuje čísla portů 49152 až 65535, které jsou přidělovány klientům. Při navazování spojení klienta se serverem je třeba znát číslo portu kontaktovaného procesu, klient sám ovšem se službou definovanou čísle portu není spojován. Následkem je přítomnost dvou různých čísel portu v jedné komunikaci. Klient kontaktuje server na portu, který je přesně dán aplikací (například 80 pro HTTP) a je obvykle ze skupiny známých nebo privilegovaných portů. Klient sám se ovšem identifikuje porte, který byl jako volný přidělen operačním systémem, a to ze skupiny dynamicky přidělovaných. Na transportní vrstvě je tedy adresován specifický proces.

Další adresace probíhá na síťové vrstvě, a to pomocí IP adres, které jsou pevně svázané se síťovým rozhraním a v současné době jsou používány dvě verze. Hlavním rozdílem mezi verzemi je v délce adresy, kde u verze 4 je používáno 32bitového čísla a u verze 6 je délka 128 bitů. Adresy jsou celosvětově přidělovány organizací IANA, která přiděluje bloky adres pěti regionálním pobočkám. Tyto pobočky následovně přidělují adresy poskytovatelům služeb internetu. Adresu lze rozdělit na dvě části, a to na adresu sítě a adresu zařízení. Rozdělní je často provedeno pomocí masky sítě [4], definující počet bitů využitý pro adresu sítě. Maska je pak tvořena bity s hodnotou 1, reprezentující síťovou část adresy, a bity s hodnotou 0, reprezentující adresu zařízení. Druhé možné dělení je pomocí tříd, které ovšem poskytuje menší flexibilitu z důvodu přesného definování rozsahů pro třídy, čímž může dojít k plýtvání adresami.

Poslední adresace probíhá na linkové vrstvě a je závislé na použité technologii. Například u technologie Ethernet je použito MAC (Media Access Control) adresy [4]. Tyto adresy jsou obvykle pevně přiděleny výrobcem a jsou zapsány v paměti typu ROM (Read Only Memory). Adresa má 48 bitů a je rozdělena na dvě části. První část o délce 24 bitů specifikuje výrobce a je přidělena organizací IEEE (Institute of Electrical and Electronics Engineers). Druhá část o stejné délce je přidělena výrobcem a specifikuje vyrobené zařízení. Každá adresa je tedy unikátní.

2.4. Socket

Pojem socket [4] definuje rozhraní operačního systému pro vstupní a výstupní operace spojené se síťovou komunikací. Jde o rozšíření komunikace mezi procesy pomocí rour (pipe).

Pro úspěšnou komunikaci je zapotřebí dvou socketů, a to jednoho na straně klienta a druhého na straně serveru. Socket nejdříve pomocí IP adresy identifikuje stanici, pro kterou jsou data určena. Dalším krokem je identifikace cílového procesu, probíhající za pomoci čísla portu a používaného protokolu.

Během komunikace TCP prochází socket několika stavy, jejichž přechody jsou řízeny příznaky v záhlaví zasílaných paketů. Prvním ze stavů je čekání na spojení, nachází se v něm server po vytvoření socketu a přidělení portu. Přidělení portu je provedeno pomocí systémového volání nazývaného bind [4]. Toto volání je pro server velmi důležité, umožňuje totiž přidělení specifického portu procesu. Pokud by nebylo provedeno, operační systém by přidělil náhodný port a následná komunikace by selhala. Následuje vyčkávání na příchozí žádost o vytvoření spojení obsahující příznak SYN. Po obdržení takového paketu přechází do stavu sestavování spojení, při kterém odesílá potvrzovací paket, obsahující příznak ACK a současně i vytvoří protisměrný kanál, přidáním příznaku SYN. Po potvrzení příznakem ACK od klienta socket přechází do třetího stavu, ve kterém probíhá samotný přenos dat. Datové segmenty mají nastavený příznak PSH a po ukončení jejich přenosu nastává poslední stav, kterým je ukončení spojení a využívá FIN pro samotné ukončení a ACK pro jeho potvrzení.

2.5. HTTP (Hyper Text Transfer Protocol)

Jedná se o protokol aplikační vrstvy, zabalený v TCP, určený pro sdílení a přenos hypertextových informací, zapsaných v HTML jazyce. Jeho v dnešní době nejznámější a nejrozšířenější využití je prohlížení webových stránek internetu.

Protokol je založen na přímé komunikaci klienta (webový prohlížeč) se serverem (různé web hosting servery), obvykle za pomoci odkazů a URL adres. Klient zasílá žádosti na server, který potvrdí přijetí žádosti (jde o TCP komunikaci) a následně data zašle nebo čeká na jejich přijetí.

V současné verzi HTTP 1.1 [5] je definováno několik metod, určených k provádění specifických činností. První z nich je metoda GET, která oznamuje serveru, že klient vyžaduje zaslání určitých dat, uložených na serveru. Použití této metody, stejně jako u všech následujících, by nemělo mít jiné účinky než ten jeden specifický, v tomto případě doručení vyžádaných dat klientovi. Další metodou, velice podobnou GET, je HEAD. Žádost je opět zaměřena na specifická data na serveru, ale tentokrát nedochází k přenosu samotných dat, přenesou se pouze informace o těchto datech (typ, velikost, datum změny a podobně). Pokud potřebujeme zaslat serveru v žádosti větší objem dat, než je možné u žádosti GET (512 bajtů),

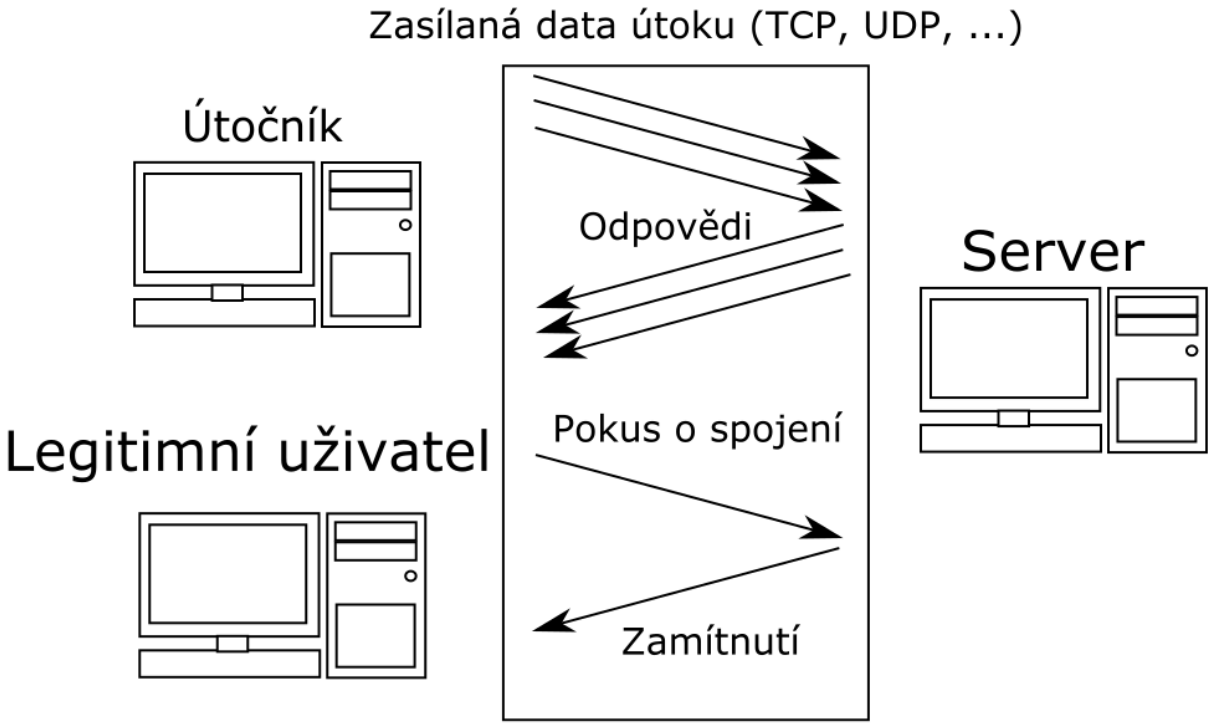
využijeme metody POST, určené především k zasílání objemných formulářů. Výsledek je ovšem stejný a to zaslání požadovaných informací. Nyní se dostáváme k části, kde naopak potřebuji na server nějaká data uložit. K tomuto účelu mi poslouží metoda PUT, která není příliš častá ale je definována a tak určitě stojí za zmínění. Častěji je tento proces prováděn za pomoci FTP přenosu. Také je zde již vyžadována jistá forma autorizace. Další metodou je DELETE, která, jak již název napovídá, maže data uložená na serveru. Pokud chceme zjistit, zda jeden nebo více ze serverů, přes které prochází naše žádost, zprávu nějakým způsobem mění, je zde možnost využití metody TRACE, která po doručení na server zpět odesílateli zašle kopii této zprávy. Další možností je OPTIONS, dotazující se serveru, jaké metody podporuje. Předposlední metodou je CONNECT, umožňující konverzi TCP spojení na tunel, obvykle k vytvoření šifrovaného SSL spojení. Nakonec je zde možnost PATCH, schopna modifikovat existující soubor.

Všechny HTTP servery musí podporovat minimálně metody GET a HEAD, pokud možno i metodu OPTIONS.

3. DoS a DDoS útoky

Hlavním cílem těchto útoků je zablokovat legitimním uživatelům přístup k dané službě. Nástroje vyvinuté harckery k tomuto účelu zpravidla využívají jeden ze dvou přístupů. Prvním je takzvaný *flood* (záplavový) útok, který zaplavuje cíl obrovským množstvím paketů a spoléhá na hrubou sílu. Druhým přístupem je *low and slow* [1] (málo a pomalu), obvykle spoléhající na spojovaný charakter TCP protokolu a jeho nedostatky.

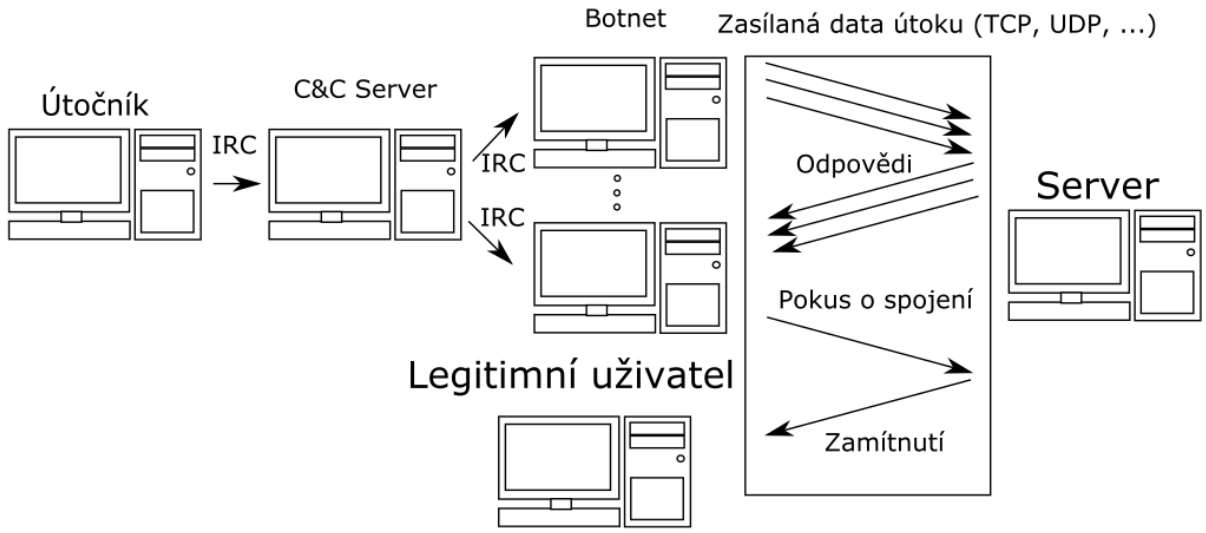
U DoS [1]**Chyba! Nenalezen zdroj odkazů.** útoku se útočník pokouší z jediného počítače, jak e uvedeno na obr. 1 různými způsoby zablokovat ostatním uživatelům přístup k dané službě vyčerpáním síťových nebo výpočetních zdrojů, čímž se jejímu zprostředkovateli snaží většinou způsobit nějakou škodu. Slabinou bývá potřeba vyšší přenosové a výpočetní kapacity než má cíl, tudíž větší servery jsou schopné téměř bez jakýchkoliv komplikací takovýto útok ustát



obr. 1 – Schéma DoS útoku

V případě *DDoS* [1]**Chyba! Nenalezen zdroj odkazů.** je útok distribuovaný, neboli rozložený, ezi vyšší počet počítačů než jeden, jak je naznačeno na obr. 2. Tímto způsobem se s rostoucím počtem účastníků zvyšuje síla útoku a současně se rapidně snižuje jakákoli šance se tomuto útoku ubránit, nestačí již totiž vyhledat a zablokovat jedinou IP adresu, ale tisíce, ne-li miliony. Útočník, obvykle za pomoci nějakého škodlivého kódu, shromažďuje armádu počítačů do takzvaných *botnetů* [1] (Bot – infikovaný počítač, Net – síť). Majitelé těchto počítačů většinou nemají ani tušení, že jejich počítač byl infikován, a že se mohou v okolní době zúčastnit masivního útoku na útočníkem zvolený server. Druhý a zároveň méně častý případ *botnetu* je dobrovolný. V tomto případě si lidé dobrovolně stáhnou do svého počítače software, který útok

provede na příkaz útočníka. U větších *botnetů* obvykle útočník komunikuje s takzvanými *C&C servery* (*Command and Control* [1] – příkazy a řízení), které potom předávají infikovaným nebo dobrovolně přidaným počítačů instrukce k útoku, obvykle komunikující za pomoci *IRC* (*Internet Relay Chat* [1]). Snaha o likvidaci těchto *botnetů* zničením *C&C serverů* nakonec skončila implementací vzájemné komunikace infikovaných počítačů mezi sebou přes veřejné peer-to-peer sítě.



obr. 2 - Schéma DDoS útoku

3.1. Zátěžové testy

Se vznikem kybernetických útoků typu blokování služby vznikla samozřejmě i potřeba ze strany provozovatelů serverů se takovýmto útokům bránit. Při implementaci nových typů útoků, zejména útoky distribuované nebo obsahující IP spoofing je ovšem ochrana velmi složitá. Nelze již jednoduše blokovat útočníky pomocí firewallu a tak muselo být zvoleno jiné řešení.

Jedno z možných řešení spočívá ve vytvoření dostatečně výkonného serveru, který zvládne obsloužit jak požadavky útočníka, tak i legitimních uživatelů, a to ideálně bez zpomalení odezvy. Pro ověření odolnosti bylo potřeba napodobit útok společně s běžně probíhající komunikací. Za tímto účelem začaly vznikat programy, schopné napodobit různé druhy útoků.

Testování obvykle probíhá na dvou úrovních. První bývá sledování chování systému s definovanou zátěží. Po tomto testu je obvykle postupně zvyšována zátěž zesilováním simulovaného útoku až na úroveň, kdy sledovaný systém začne vykazovat nepřijatelné chování. Tato úroveň je pak považována za hranici odolnosti serveru vůči záplavovým útokům.

Jiné typy útoků, využívající slabin aplikací či protokolů, je již třeba testovat sledováním chování dané aplikace během útoku. Ochranou pak může být nalezení a oprava nedostatku v programu.

4. Dělení útoků

Existuje několik různých kritérií, podle kterých lze *DoS* útoky dělit. Hlavní dvě kritéria, podle kterých budou v této práci děleny, jsou na jaké vrstvě vrstvového modelu TCP/IP probíhají a jestli se snaží spotřebovat síťové, výpočetní nebo aplikační zdroje, přičemž hlavním dělení je podle vrstvového modelu.

Vrstvový model TCP/IP **Chyba! Nenalezen zdroj odkazů.** je složen ze čtyř vrstev, z nichž aždá má přesně definovanou funkci a způsob výměny informací s ostatními vrstvami. Jednotlivé vrstvy využívají služeb nižších vrstev a poskytují služby vyšší vrstvě.

První a zároveň nejnižší je vrstva síťového rozhraní (link layer **Chyba! Nenalezen zdroj dkazů.**), zajišťující přístup k přenosovému médiu a přesun dat po něm. Příkladem může být Ethernet nebo PPP (Point-to-Point Protocol). Druhou je síťová vrstva (internet layer [6]), zajišťující hlavně směrování adresaci. Nejčastějším protokolem je IP, pro tuto práci jsou ovšem nejzajímavější ICMP a IGMP. Další vrstva je třetí a je nazývaná transportní (transport layer [6]). Zajišťuje komunikaci mezi dvěma koncovými body, a to jak spojově, tak i nespojově orientovanou, za pomoci jejích dvou hlavních zástupců TCP a UDP. Poslední vrstvou je pak aplikační (application layer [6]), poskytující funkce aplikací uživateli. Příkladem protokolu této vrstvy může být HTTP, RTP, SMTP, SSL a mnohé další.

4.1. Útoky na síťové vrstvě

Nejnižší vrstvou, významnou pro *DoS* útoky je vrstva síťová. Výhodou této vrstvy ze strany útočníků je, že je implementována ve všech síťových prvcích a tudíž na ni lze zaútočit bez ohledu na charakter cíle. Často se lze setkat s útokem za pomoci záplavy ICMP pakety, využitelná jak proti osobním počítačům, tak i proti například HTTP serverům. Vedle ICMP protokolu se k útokům také využívá IGMP.

4.1.1. ICPM flood

ICMP (Internet Control Message Protocol) protokol je využíván naprostou většinou zařízení, připojených k síti, a to obvykle k ověření dostupnosti a zasílání chybových hlášek. Většina síťových prvků by tudíž měla na ICMP pakety reagovat, což z nich dělá ideální prostředek k provedení útoku. Takový útok nezávisí na žádné bezpečnostní chybě nebo slabině v protokolu. Většinou jde o ICMP pakety typu *echo request* [1], neboli o standardně využívanou žádost o zaslání odezvy. Útok od klasické žádosti o odezvu lze rozeznat pouze podle neobvykle vysokého počtu příchozích žádostí. Jde tedy o zaplavení ohromným množstvím těchto paketů, ve snaze spotřebovat síťové zdroje cíle, který se tím stane nedostupným.

Nevýhodou této metody je potřeba vyšší přenosové rychlosti než má cíl. Na druhou stranu, jednoduchost tohoto útoku spočívá ve faktu, že jej lze provést pouze za pomoci příkazu ping v

příkazové řádce. Jsou totiž k dispozici různé modifikátory. Příkladem je parametr -l, nastavující velikost datagramu až na zhruba 65500 bitů z původních zhruba 70. Dále je tu také u systémů Linux parametr -f, neboli flood, který odesílá další ICMP paket jakmile přijme odezvu z předchozího. Z tohoto popisu si již lze představit, že i takto jednoduchý nástroj může být poměrně mocný.

4.1.2. IGMP flood

Protokol IGMP (Internet Group Management Protocol) je využíván běžně ke správě multicastových skupin. Základem tohoto protokolu je zasílání zpráv na multicastovou adresu. Útok má většinou formu zasílání velkého počtu náhodných IGMP zpráv. Účinky jsou podobné jako u ICMP flood [1] a cílem je také spotřebovat celý datový tok připojení.

Společnou vlastností těchto útoků je možnost jejich zesílení nebo znásobení. Využívá se zde broadcastové adresy směrovačů a takzvaného *IP spoofingu* [1] (změna zdrojové IP adresy). Za pomoci těchto dvou možností může útočník poslat velké množství ICMP paketů na broadcastovou adresu směrovače, nebo IGMP paketů na multicastovou adresu skupiny, se zdrojovou IP adresou cíle. Výsledek takto provedené akce je, že ICMP jsou zaslány všem síťovým prvkům dané sítě, IGMP pak celé skupině, a všechny následně posílají odpovědi na zdrojovou IP adresu, zaměněnou za adresu cíle útoku. Počet paketů odeslaný útočníkem je tedy znásoben počtem síťových prvků nebo členů skupiny.

4.2. Útoky na transportní vrstvě

Jak již bylo zmíněno v předchozím popisu, síťová vrstva obsahuje dva hlavní protokoly, a to spojově orientovaný a spolehlivější TCP (Transmission Control Protocol), vyžadující vytvoření spojení před vlastním přenosem dat a kontrolující jejich doručení, a nespojovaný a méně spolehlivý UDP (User Datagram Protocol), odesílající data bez jakéhokoli spojení a bez kontroly doručení.

4.2.1. UDP Flood

Podobně jako u ostatních záplavových útoků zde nejde o využívání bezpečnostních chyb ale o prosté zasypání cíle obrovským množstvím dat, v tomto případě náhodných UDP paketů na náhodné porty, ideálně z vymyšlené zdrojové adresy. Tento server ve snaze odeslat na zdrojovou adresu odpověď (ICMP destination unreachable) spotřebuje zbytek volného datového toku.

Další útoky, probíhající na této vrstvě, využívají slabiny v TCP protokolu a to šesti kontrolních bitů, česky nazývaných příznaky, SYN, ACK, RST, PSH, FIN a URG. Protokol TCP je spojově

orientovaný, což má za následek potřebu otevření obousměrného spojení před zasíláním uživatelských dat. K vytvoření spojení složí takzvaný *three-way handshake* [1], kdy klient zašle serveru TCP paket s příznakem SYN nastaveným na hodnotu jedna, čímž ho žádá o vytvoření spojení. Následuje paket SYN-ACK zaslaný serverem, který napůl otevírá spojení ze strany serveru a je zaslán na zdrojovou adresu přijatého SYN paketu. Třetím a posledním krokem je zaslání paketu ACK klientem, sloužícím jako potvrzení a otevírajícím spojení ze strany klienta. Hacker záměrně zasílá pakety buď ve špatném pořadí, což způsobí spotřebování výpočetních zdrojů ve snaze serveru o porozumění neobvyklých dat.

4.2.2. TCP SYN flood

Tato forma útoků spoléhá na slabinu TCP protokolu, kterou je výše zmíněné tříkrokové navazování spojení. Obvykle využívá *IP spoofingu*, kterým mění zdrojovou adresu SYN paketů, zasílaných serveru v obrovských počtech. Vzhledem ke zfalšované adrese ale klient obvykle neexistuje nebo nevyžaduje od tohoto serveru spojení, zpětně zaslaný SYN-ACK paket je buď zahozen klientem, nebo nějakým síťovým prvkem. Na serveru se mezi tím hromadí vysoký počet napůl otevřených spojení, marně čekajících na konečný ACK paket. Server se pokouší zaslat SYN-ACK pakety znovu a znovu, dokud nenarazí na limit opětovných zaslání a neukončí spojení jako request time-out. Server, jako každé jiné zařízení, má omezený výpočetní výkon a maximální počet aktivních (i když jen napůl) spojení. Cílem útoku je buď vyplnit celou tabulku aktivních spojení, nebo server donutit spotřebovat veškerý výpočetní výkon zpracováváním obrovského počtu žádostí a zasíláním odpovědí. Tento útok je velmi oblíbený a dle průzkumu z roku 2011 tvořil asi 25% všech známých útoků.

4.2.3. TCP RST útok

Příznak RST [9] říká serveru, že by měl ukončit specifické TCP spojení, definované sekvenčním číslem a adresou klienta. Provedení útoku spoléhá na odposlechnutí nebo znalosti IP adresy klienta a zaslání RST paketu serveru. Pro ukončení spojení je ovšem zapotřebí znát ještě sekvenční číslo. Toto číslo se snaží útočník odhadnout náhodným výběrem čísel. V této fázi je obvykle použit botnet, což značně snižuje čas potřebný k odhadu správného sekvenčního čísla. Jakmile je zadáno správné číslo, spojení je ukončeno.

4.2.4. TCP PSH+ACK flood

U TCP spojení byl z důvodu efektivity přenosu dat vytvořen na obou komunikujících stranách buffer (zásobník dat). Tento buffer se postupně plní aplikačními daty, dokud jich neobsahuje dostatek nato, aby byl odeslán paket maximální velikosti. Přestože je tento systém mnohem efektivnější pro odesílání větších souborů, není příliš vhodný pro aplikace pracující v reálném čase. Z tohoto důvodu byla zavedena možnost příznaku PSH [7], který nařídí příjemci

okamžité odeslání dat v bufferu. Tohoto faktu může samozřejmě být zneužito k provedení útoku. Ten spočívá v masivní záplavě TCP pakety s PSH příznakem nastaveným na hodnotu jedna, což znamená právě požadavek na vyprázdnění bufferu a následné zaslání potvrzení ve formě ACK paketu o úspěšném provedení této akce. Vzhledem k obrovskému množství těchto žádostí je buffer serveru zahlcen až do stavu, kdy není schopen další žádosti zpracovávat, natož na ně odpovídat.

4.2.5. Low and slow útoky

Až dosud byly všechny popisované postupy založeny na zasílání masivního počtu paketů a většinou potřebovaly botnety ke splnění svého účelu. Takzvané *Low and slow* [1] útoky obvykle zneužívají specifické chyby nebo nedostatku protokolu a k docílení výsledku potřebují relativně málo paketů, výsledkem potom bývá *DoS* stav nebo pád serveru.

Příkladem těchto útoků může být aplikace Sockstress [1]. Tato aplikace regulérním postupem vytváří spojení pomocí tříkrokové komunikace SYN, SYN-ACK, ACK. V posledním ACK je ovšem nastavena hodnota *window size* [1][9] na nulu. Hodnota *window size*, česky velikost okna, informuje příjemce paketu o velikosti volného místa v příjmovém bufferu. Tento buffer skládá příchozí data, než je zašle aplikaci a existuje pravidlo, že velikost tohoto bufferu je rovna maximálnímu objemu dat, která jsou odeslána klientem před obdržením potvrzení o jejich doručení. Nastavením tohoto parametru na nulu je dáno najevo, že na příjmové straně již není žádné místo k ukládání dat a tudíž by neměli být odesílány další data. Server následně začne v pravidelných intervalech zasílat pakety se žádostí o změnu velikosti okna (zjišťuje, jestli už bylo uvolněno nějaké místo v bufferu). Vzhledem k tomu, že jde o útok, útočník opět zašle velikost okna rovnu nule. Tímto způsobem pokračuje dále a je schopen udržet spojení otevřené až do konce útoku. Cílem útoku je otevřít dostatečný počet takovýchto spojení k úplnému zaplnění tabulky aktivních spojení serveru, která je samozřejmě omezená. Jakmile tento stav nastane, je serverem blokován jakýkoli další pokus o vytvoření spojení (*DoS* stav).

Podobným postupem lze dosáhnout spotřebování operační paměti serveru a jeho následné pádu, a to nastavením velikosti okna na velmi malou, ale nenulovou, hodnotu. Pokud je nastavena velikost například na šest, je server donucen rozdělit data na šesti-bitové kousky a posílat je jednotlivě jeden po druhém, a to až po obdržení potvrzení o doručení. Tímto způsobem se opět vytvoří vysoký počet paralelních spojení, až nastane výše uvedený stav.

4.3. Útoky na aplikační vrstvě

Aplikační vrstva zprostředkovává komunikace mezi samotnými procesy a aplikacemi. Vzhledem k rozmanitosti aplikací a využití sítí existuje nepřeberné množství různých aplikačních protokolů, z nichž každý slouží svému účelu. Útočníci mají tím pádem na výběr z velkého množství cest k provedené útoku, který obvykle spoléhá na nalezené chyby a

nedokonalosti, a zneužívá záležitostí vytvořených pro lidi proti nim. Útoky se obvykle zaměřují buď na samotné aplikace, nebo na výpočetní výkon.

4.3.1. DNS flood

DNS je velmi důležitý protokol využívaný naprostou většinou aplikací, které komunikují po síti. Jeho hlavním úkolem je překlad doménových jmen na IP dresy, a bez něj by navigace v dnešním internetu byla téměř nemožná. Tyto faktory z něj dělají ideální prostředek na útok, který je poměrně jednoduché provést, ale velmi složité detekovat a zastavit. Vzhledem k tomu, že je zabalen v UDP, není potřeba vytvářet spojení, a tudíž jde o rychlou komunikaci. Útok je proveden zasíláním velkého počtu DNS žádostí na DNS server objeti, ten je ve snaze je všechny zpracovat zahlcen a časem spadne. Následuje stav, kdy se klienti, kteří jsou k tomuto DNS serveru připojení nemohou připojit k internetovým stránkám a serverům za pomoci doménových jmen a adres, protože jim je nepřekládá žádný DNS server.

4.3.2. SSL útoky

Protokol SSL (Secure Socket Layer **Chyba! Nenalezen zdroj odkazů.**) je ryptografický protokol, umožňující zašifrování zasílaných dat. Je využívaný k zabezpečení komunikace proti nežádoucímu odposlechu a následnému potenciálnímu zneužití uživatelských dat. Obvykle je implementován při prohlížení webu, zasílání e-mailů a faxů, VoIP telefonii a podobně.

Útočník si může vybrat z řady různých přístupů k útoku. Vzhledem k faktu, že je SSL zabaleno v TCP, může se útočník snažit napadnout tříkrokové vytváření spojení, posílat nesmyslná data SSL serveru, nebo zneužít procesu domluvy zabezpečovacího klíče. Nejúčinnější variantou ale nejspíš je zasílání zašifrovaného datového toku. Tento postup je nazýván nesymetrickým útokem, protože server při snaze zpracovat zašifrovaná data spotřebuje mnohem více výpočetního výkonu, než útočník k zaslání dat. Tento útok je navíc neskutečně složité detekovat.

Skupina hackerů nazývaná *The Hackers Choice* (THC [1]) vytvořila k poukázání na chyby SSL aplikaci THC-SSL-DOS. Tento nástroj legitimním postupem vytvoří SSL spojení a následně okamžitě žádá o vytvoření nového šifrovacího klíče. Tento postup neustále opakuje, dokud nespotřebuje veškerý výpočetní výkon serveru. Využívání SSL *handshaku* je velmi oblíbené vzhledem k faktu, že výpočetní výkon, potřebný k vytvoření spojení za pomoci SSL, je zhruba patnáctkrát vyšší na straně serveru, než na straně útočníka.

4.3.3. HTTP flood

V dnešní době jeden z nejčastěji používanějších protokolů, HTTP (Hyper Text Transfer Protocol) využívá spolehlivosti TCP protokolu a je používán k prohlížení nepřeberného množství webových stránek, dostupných na internetu. Útok je obvykle prováděn z více zdrojů najednou (dobrovolníci anebo *botnet*) a spočívá ve vytvoření normálního spojení se serverem a následné opakované zasílání HTTP GET nebo POST zpráv, a to v obrovském počtu. Cílem tohoto útoku je spotřebovat veškeré aplikační zdroje a tím pádem navodit *DoS* stav, při němž je další příchozí komunikace blokována.

4.3.4. Slow HTTP GET

Je dalším z řady útoků, využívajících oblíbenosti a nedostatků HTTP protokolu. Spočívá v zasílání relativně malého počtu nekompletních GET žádostí. Server pro každou z těchto žádostí vytvoří oddělené spojení a čeká, až dorazí zbytek dat požadavku. Tyto data jsou ovšem zasílána velmi pomalu a v přesně vypočítaných intervalech, aby nedošlo k vypršení časového limitu spojení, a tím k jeho ukončení. Tímto způsobem útočník zaplní celou spojovací tabulku serveru a ten začne odmítat všechna nová připojení, včetně legitimních uživatelů.

4.3.5. Slow HTTP POST

Podobně jako v předchozím případě je zde využíváno slabiny HTTP serverů. Jde opět o *low and slow* útok, který napodobuje chování legitimního uživatele s velice pomalým připojením k internetu, tentokrát je ovšem využita žádost typu POST. Útočník zašle nekompletní POST žádosti a následně zasílá zbytek těchto žádostí bajt po bajtu v maximálním možném časovém rozmezí, aby nedošlo k vypršení časového limitu spojení. Využívanou slabinou je, že jakmile server začne data přijímat, je nucen počkat, dokud nedojdou všechna. Výsledkem opět je zaplnění tabulky aktivních připojení a následné blokování nových pokusů o spojení se serverem.

5. Nástroje k útoku

Postupem času vznikly uživatelské programy, z nichž několik zde bude zmíněno, které byly vyvinuty ať už k testování ochrany proti útokům a odolnosti serverů, nebo k samotnému provádění útoků.

Většina programů původně fungovala pouze pod operačními systémy Linux a Solaris, příkladem těchto programů může být například výše zmíněný *Trinoo*. V dnešní době jsou ovšem nejčastěji používané ty, které fungují na všech platformách. Tyto programy jsou často velmi jednoduše dostupné a kolikrát i open-source.

5.1. Low Orbit Ion Cannon

Známý pod zkratkou LOIC [1], tento poměrně jednoduchý, záplavový open-source nástroj, který je schopný generovat masivní počty TCP, UDP, HTTP paketů a posílat je na cílový server. Původně byl vyvinut firmou Praetox Technologies k testování jejich vlastních serverů pod vysokým zatížením. Hackerská skupina Anonymous převzala tento nástroj a začala jej používat k provádění *DoS* útoků.

Časem program začala vyvíjet a dodala jí její takzvaný *hivemind* [1] prvek. Tento prvek umožňoval připojení LOIC k IRC a následné přesměrování na dálkové ovládání skrz tento IRC server. Velké množství takto ovládaných programů jedním člověkem většinou vyústilo masivním *DDoS* útokem z mnoha LOIC, rozsetých po celém světě, v jedu chvíli.

Slabinou LOIC ovšem byl fakt, že se ani nějak nepokoušel zamaskovat zdrojovou IP adresu, což roku 2011 skončilo zatýkáním mnoha účastníků těchto dobrovolných útoků. Tato událost ukončila období slávy LOIC, když skrz IRC kanály Anonymous informoval o zákazu používání LOIC.

5.2. High Orbit Ion Cannon

Jedná se o nástupce LOIC, který implementuje takzvaný booster script. Tento skript umožňuje útočníkům specifikovat seznam URL adres, mezi nimiž HOIC [1] při svém útoku pravidelně přepíná, což komplikuje blokování útoku.

HOIC byl použit například při útoku na Ministerstvo spravedlnosti v USA. Útok byl reakcí na rozhodnutí o ukončení stránky Megaupload.com.

I když jde o mnohem mocnější nástroj, problém nedostatečné anonymity útočníků stále přetrvává.

5.3. Hping

Vzhledem k nedostatkům LOIC a HOIC v maskování zdroje útoku, hackeři vyvinuli nový nástroj. Tento program se jeví jako klasická příkazová řádka, umí však mnohem více. Hlavní předností, hned vedle zasílání masivního počtu TCP paketů, je možnost *IP spoofingu,* což umožňuje skrýt zdrojovou IP adresu útočníka za generátor náhodných čísel, nebo dokonce za jednu specifickou adresu zadanou uživatelem.

5.4. Slowloris

Tento program implementuje myšlenku *low and slow* útoků. Zasílá řadu legitimních, ale nekompletních HTTP žádostí, a následně odesílá zbytek dat z žádostí bit po bitu v co nejdelších intervalech, těsně před vypršením časového limitu. Postupným vyplněním tabulky aktivních připojení způsobuje blokování nově příchozích žádostí, jak jsem již výše uvedl.

6. Vývoj aplikace

Nejprve byl vybrán a realizován útok typu HTTP GET flood, a to vzhledem k zaměření práce na útoky probíhající na aplikační vrstvě. Dalšími důvody této volby je fakt, že dle statistik z roku 2013 jde o nejčastější formu útoku na aplikační vrstvě a současně druhý nejčastější DoS útok celkově (přes devatenáct procent z všech útoků). Program je napsán v jazyce C++ a tvořen v prostředí Microsoft Visual studia.

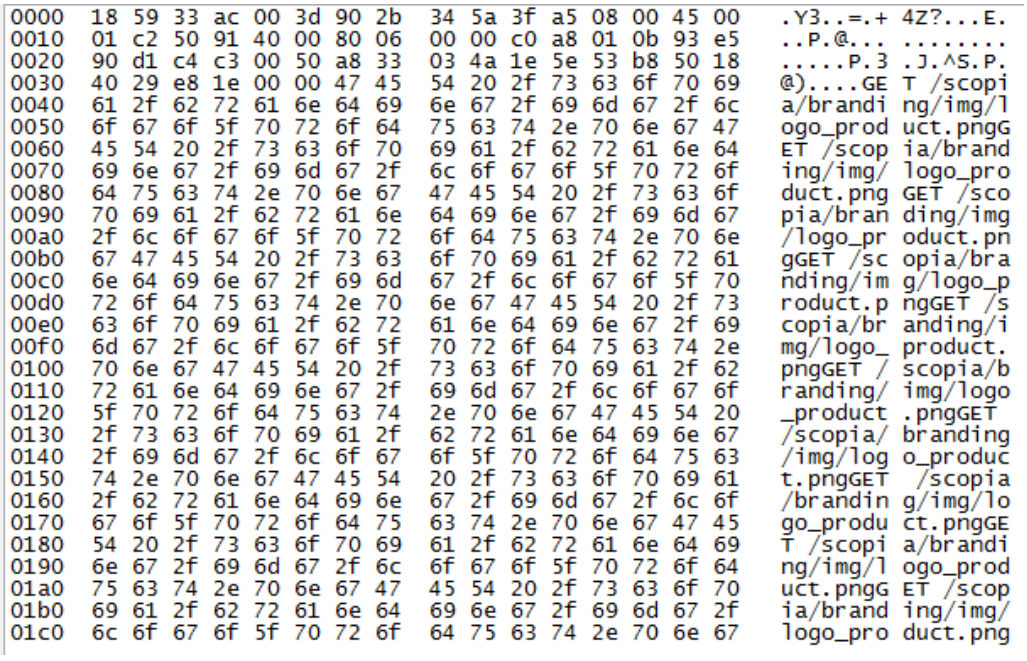
Při tvorbě programu je třeba nejdříve zjistit, co má tento program vlastně postupně dělat a za pomoci jakých metod toho můžeme dosáhnout. Hlavním cílem je zde zasílat data, v tomto případě HTTP GET žádosti, sítí na internetový server, k čemuž poslouží metody ovládání socketu, definované v knihově *winsock2.h*.

Vzhledem k zapouzdření HTTP protokolu v TCP je potřeba před zasláním dat vytvořit spojení, k čemuž je třeba znát IP adresu cíle a port pro komunikaci. Defaultní port pro HTTP je 80 a je v programu napevno zadán, adresa už ale takto zadána být nemůže. V programu jsou pro tento účel vytvořeny dvě metody. Jedna z nich jednoduše převede adresu, zadanou přes příkazový řádek, do formátu, vhodného pro program. Druhá možnost je zadání URL (Uniform Resource Locator) adresy a následné zjištění IP adresy programem za pomoci komunikace s DNS serverem. Vzhledem k potížím, vzniklým kvůli Nagleovo algoritmu (zefektivnění přenosu sítí spojením malých paketů do jednoho velkého), způsobujícím řetězení několika požadavků v jediném paketu až do velikosti MTU (Maximum Transmit Unit), jak je vidět na obr. 3, byly vytvořeny opět dvě možnosti. V první je samotné vytvoření socketu a sestavení spojení přesunuto až k odeslání žádosti. Každá žádost tedy vytvoří a ukončí samostatné spojení. Druhou možností je doplnění každé žádosti na velikost MTU. Toto řešení ovšem přineslo další nečekaný problém, kterým je ukončení spojení ze strany serveru obvykle po zaslání sedmi žádostí. Problém byl vyřešen opětovným navázáním spojení, je-li ukončeno před zasláním požadovaného počtu paketů, jak je ukázáno na obr. 4 z programu Wireshark.

Další částí je zadání cesty k datům, o která chceme server žádat. Text samotné žádosti se skládá ze slova GET, mezery a cesty k datům na serveru. Program je pro zadání cesty opět vybaven dvěma metodami. V první je možnost vložení cesty do příkazové řádky bez slova GET a bez mezery, které jsou doplněny programem, začínat by tedy v tomto případě měla stoupajícím lomítkem. Druhou možností je načtení celého textu žádosti ze souboru, umístěného ve složce společně se spouštěcím souborem a pojmenovaného data.txt, který již musí obsahovat i slovo GET a mezeru. Text odkazu leze jednoduše získat ze samotné stránky jeho zkopírováním, nebo můžeme získat celou žádost odposlechnutím v programu Wireshark. Tato část je sice důležitá při formování správné žádosti, nikoliv však pro splnění účelu tohoto programu. Nejde zde totiž o zpětné zaslání vyžádané informace, nýbrž o zaneprázdnění serveru řešením programem zaslaných žádostí a to ať už regulérních nebo nesmyslných. Server zdroji žádostí totiž stejně musí odpovědět.

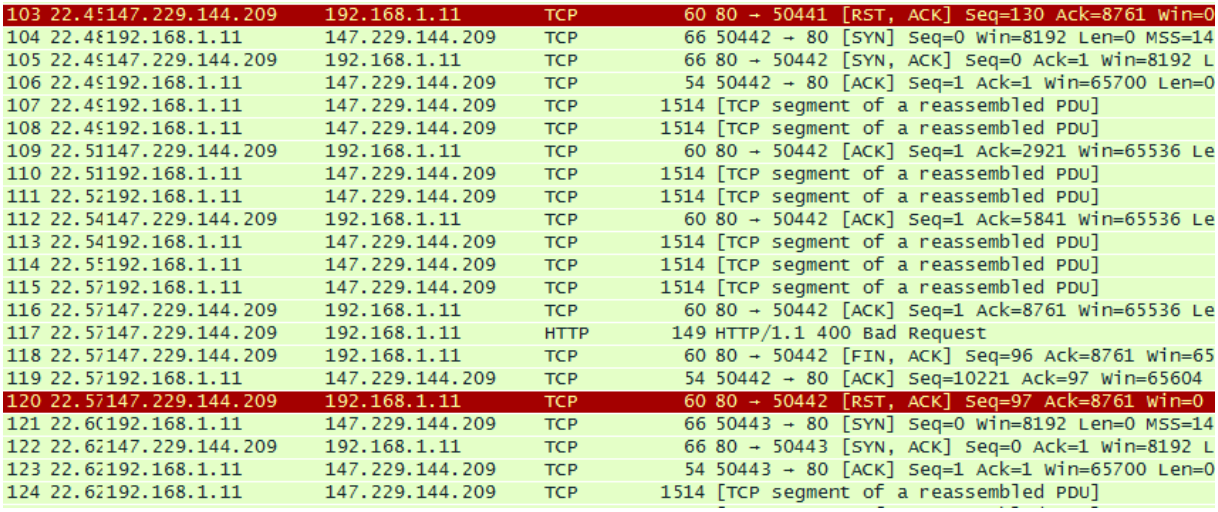
Nyní má program dostatek informací k vytvoření spojení a zaslání dat. Může tedy přistoupit k metodám, které je v konzolovém okně uvedeny jako HTTP Send a HTTP Send mkII. Tyto metody provádí veškeré síťové operace, kromě výše uvedeného DNS dotazu. Po spuštění je uživatel dotázán na dvě informace, kde první je doba čekání v milisekundách mezi zasláním jednotlivých žádostí (je-li zde zadána nula, cyklus vytvoření spojení a odeslání další žádosti bude opakován ihned po ukončení předchozího), druhou pak kolik celkově žádostí má zaslat. Po zadání těchto dvou hodnot je vytvořeno spojení, zaslán zadaný požadavek, počká se zadaný čas a následně je spojení ukončeno. Celý tento proces je opakován, dokud je cílový server dostupný, nebo dokud není odeslán požadovaný počet paketů. V případě nedostupnosti serveru buď program dosáhl svého cíle a server je zahlcen, anebo byla na straně serveru aktivována ochrana, blokující příchozí spojení z používané zdrojové adresy.

Výsledný program je tedy schopen posílat větší množství HTTP GET žádostí na uživatelem zvolený webový server dvěma uvedenými způsoby. Výsledkem dosud provedených testů je, že vytvořenou aplikací při této formě útoku není zasílán dostatek žádostí k úplnému zahlcení serveru, který byl dán k vedoucím práce dispozici pro účely vzdálené testování.



```
0000  18 59 33 ac 00 3d 90 2b  34 5a 3f a5 08 00 45 00   .Y3..=.+ 4Z?...E.
0010  01 c2 50 91 40 00 80 06  00 00 c0 a8 01 0b 93 e5   ..P.@... ........
0020  90 d1 c4 c3 00 50 a8 33  03 4a 1e 5e 53 b8 50 18   .....P.3 .J.^S.P.
0030  40 29 e8 1e 00 00 47 45  54 20 2f 73 63 6f 70 69   @)....GE T /scopi
0040  61 2f 62 72 61 6e 64 69  6e 67 2f 69 6d 67 2f 6c   a/brandi ng/img/l
0050  6f 67 6f 5f 70 72 6f 64  75 63 74 2e 70 6e 67 47   ogo_prod uct.pngG
0060  45 54 20 2f 73 63 6f 70  69 61 2f 62 72 61 6e 64   ET /scop ia/brand
0070  69 6e 67 2f 69 6d 67 2f  6c 6f 67 6f 5f 70 72 6f   ing/img/ logo_pro
0080  64 75 63 74 2e 70 6e 67  47 45 54 20 2f 73 63 6f   duct.png GET /sco
0090  70 69 61 2f 62 72 61 6e  64 69 6e 67 2f 69 6d 67   pia/bran ding/img
00a0  2f 6c 6f 67 6f 5f 70 72  6f 64 75 63 74 2e 70 6e   /logo_pr oduct.pn
00b0  67 47 45 54 20 2f 73 63  6f 70 69 61 2f 62 72 61   gGET /sc opia/bra
00c0  6e 64 69 6e 67 2f 69 6d  67 2f 6c 6f 67 6f 5f 70   nding/im g/logo_p
00d0  72 6f 64 75 63 74 2e 70  6e 67 47 45 54 20 2f 73   roduct.p ngGET /s
00e0  63 6f 70 69 61 2f 62 72  61 6e 64 69 6e 67 2f 69   copia/br anding/i
00f0  6d 67 2f 6c 6f 67 6f 5f  70 72 6f 64 75 63 74 2e   mg/logo_ product.
0100  70 6e 67 47 45 54 20 2f  73 63 6f 70 69 61 2f 62   pngGET / scopia/b
0110  72 61 6e 64 69 6e 67 2f  69 6d 67 2f 6c 6f 67 6f   randing/ img/logo
0120  5f 70 72 6f 64 75 63 74  2e 70 6e 67 47 45 54 20   _product .pngGET
0130  2f 73 63 6f 70 69 61 2f  62 72 61 6e 64 69 6e 67   /scopia/ branding
0140  2f 69 6d 67 2f 6c 6f 67  6f 5f 70 72 6f 64 75 63   /img/log o_produc
0150  74 2e 70 6e 67 47 45 54  20 2f 73 63 6f 70 69 61   t.pngGET  /scopia
0160  2f 62 72 61 6e 64 69 6e  67 2f 69 6d 67 2f 6c 6f   /brandin g/img/lo
0170  67 6f 5f 70 72 6f 64 75  63 74 2e 70 6e 67 47 45   go_produ ct.pngGE
0180  54 20 2f 73 63 6f 70 69  61 2f 62 72 61 6e 64 69   T /scopi a/brandi
0190  6e 67 2f 69 6d 67 2f 6c  6f 67 6f 5f 70 72 6f 64   ng/img/l ogo_prod
01a0  75 63 74 2e 70 6e 67 47  45 54 20 2f 73 63 6f 70   uct.pngG ET /scop
01b0  69 61 2f 62 72 61 6e 64  69 6e 67 2f 69 6d 67 2f   ia/brand ing/img/
01c0  6c 6f 67 6f 5f 70 72 6f  64 75 63 74 2e 70 6e 67   logo_pro duct.png
```

obr. 3 - Nagleho algoritmus a 10 žádostí (program Wireshark)



103	22.45	147.229.144.209	192.168.1.11	TCP	60 80 → 50441 [RST, ACK] Seq=130 Ack=8761 Win=0
104	22.48	192.168.1.11	147.229.144.209	TCP	66 50442 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=14
105	22.49	147.229.144.209	192.168.1.11	TCP	66 80 → 50442 [SYN, ACK] Seq=0 Ack=1 Win=8192 L
106	22.49	192.168.1.11	147.229.144.209	TCP	54 50442 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
107	22.49	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]
108	22.49	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]
109	22.51	147.229.144.209	192.168.1.11	TCP	60 80 → 50442 [ACK] Seq=1 Ack=2921 Win=65536 Le
110	22.51	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]
111	22.52	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]
112	22.54	147.229.144.209	192.168.1.11	TCP	60 80 → 50442 [ACK] Seq=1 Ack=5841 Win=65536 Le
113	22.54	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]
114	22.55	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]
115	22.57	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]
116	22.57	147.229.144.209	192.168.1.11	TCP	60 80 → 50442 [ACK] Seq=1 Ack=8761 Win=65536 Le
117	22.57	147.229.144.209	192.168.1.11	HTTP	149 HTTP/1.1 400 Bad Request
118	22.57	147.229.144.209	192.168.1.11	TCP	60 80 → 50442 [FIN, ACK] Seq=96 Ack=8761 Win=65
119	22.57	192.168.1.11	147.229.144.209	TCP	54 50442 → 80 [ACK] Seq=10221 Ack=97 Win=65604
120	22.57	147.229.144.209	192.168.1.11	TCP	60 80 → 50442 [RST, ACK] Seq=97 Ack=8761 Win=0
121	22.60	192.168.1.11	147.229.144.209	TCP	66 50443 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=14
122	22.62	147.229.144.209	192.168.1.11	TCP	66 80 → 50443 [SYN, ACK] Seq=0 Ack=1 Win=8192 L
123	22.62	192.168.1.11	147.229.144.209	TCP	54 50443 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
124	22.62	192.168.1.11	147.229.144.209	TCP	1514 [TCP segment of a reassembled PDU]

obr. 4 - Opětovné vytvoření spojení po jeho ukončení serverem (program Wireshark)

6.1. Další rozvoj a testování

V rámci rozvoje byla aplikace rozšířena o další typ útoku, více vláknové zpracování a následné otestování vzniklé aplikace na hardware a infrastruktuře VUT v Brně, obsahující cílový server tvořený zařízením Avalanche a linuxový server na samotné provedení útoku.

Nová verze aplikace tedy měla být schopna běhu na distribuci operačního systému Linux, pod kterým funguje server určený pro testování aplikace. Bylo tudíž potřeba použít jiné knihovny než *winsock2.h*, použité u předchozí verze, vzhledem k její nekompatibilitě s jinými operačními systémy než je Windows. Byla proto zvolena knihovna *Boost verze 1.60.0* (v době vývoje nejnovější verze), obsahující třídu *Asio*, která obsahuje metody potřebné k síťové komunikaci. Funkce jednotlivých metod nové knihovny je sice podobná, jejich implementace již stejná ovšem není. Vždy před zasíláním dat sítí musíme provést stejné kroky, a to obzvlášť při komunikaci pomocí protokolu TCP. Vzhledem k spojové orientaci protokolu je potřeba nejdříve vytvořit spojení mezi dvěma stanicemi na specifickém portu.

Prvním rozdílem je hned zadání cílové stanice. U *winsock* tento proces spočíval buď v přímém zadání IP adresy a jejím následném převodu na síťový formát, nebo v zadání URL adresy a následného zjištění IP adresy pomocí komunikace s DNS serverem, následované zadáním portu. Knihovna *Boost* pro tento účel využívá iteračního procesu. Tomuto procesu je zadána IP nebo URL adresa ve formě textového řetězce. Využívané metody *resolver* a *iterator* pak samostatně zjistí, o jakou adresu jde. V případě URL adresy automaticky kontaktuje DNS server za účelem zjištění IP adresy. Následuje iterační proces, který určí množinu koncových bodů. Následuje vytvoření socketu.

Dalším krokem, vzhledem k využívání TCP protokolu, je vytvoření spojení. To je realizované metodou *connect*, vyžadující vytvořený socket a dříve vytvořenou množinu koncových bodů. Metoda následně postupně zkouší vytvořit spojení s jednotlivými koncovými body, až nakonec

uspěje. Toto je další rozdíl oproti *winsock*, kde celý tento proces byl realizován jedním jediným krokem, a to funkcí obdobného jména, vyžadující vytvořený socket a strukturu, obsahující IP adresu, port a specifikaci protokolu.

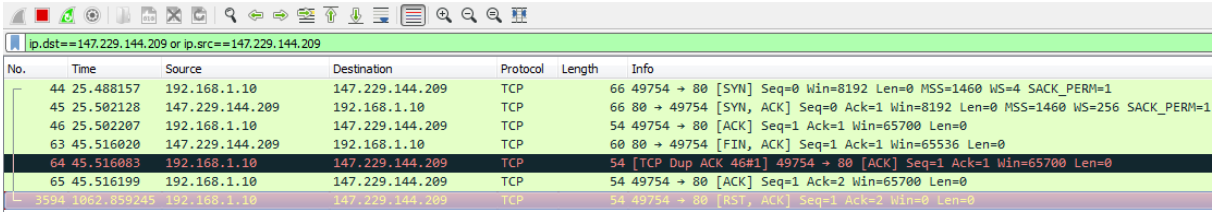
Dalším rozdílem je samotný mechanismus zasílání dat. U předchozí knihovny byl proces realizován funkcí *send*, které byl jako argument předán textový řetězec k zaslání, počet znaků k zaslání a socket. Funkce pak vracela hodnotu rovnou počtu zaslaných znaků, který se nemusel vždy rovnat zadanému počtu. Nová knihovna používá vlastního zásobníku pro data, která do něj mohou být postupně vkládána, což zjednodušuje doplnění od uživatele získané cesty pro HTTP GET žádost doplňujícími. Metodě *write* je pak jako argument předán tento zásobník a socket. Metoda ovšem vždy zašle celý zásobník před pokračováním programu. Pokud tedy nedojde při prvním pokusu k zaslání všech dat, tak se proces zasílání automaticky opakuje, dokud není zasláno vše.

Znovu vytvořená a upravená metoda pro simulaci útoku HTTP GET Flood je tvořena výše zmíněným postupem pro vytvoření spojení TCP a cyklem, umožňujícím opakované zasání stejné žádosti cílovému serveru. Cyklus je záměrně nekonečný a je možné jej ukončit pomocí zapsání znaku „s“ do konzole a následného potvrzení stisknutím klávesy enter. Ukončení je ovšem součástí jiné funkce, která běží paralelně s útokem na jiném vlákně. Součástí cyklu je skládání žádostí, jejich následné odesílání a ošetření situace, kdy cílový server ukončí spojení, spočívající v opětovném navázání spojení. Dále je implementována možnost časové prodlevy mezi jednotlivými pakety, která ovšem u některých serverů způsobuje okamžité ukončení spojení hned po jeho navázání. Pokud tato situace nastane, je zpoždění automaticky nastaveno na nulu. Upravená metoda již ve většině případů nemá problém, kdy bylo zasláno několik žádostí jedním paketem (až do velikosti MTU) jak je vidět na obr. 5.

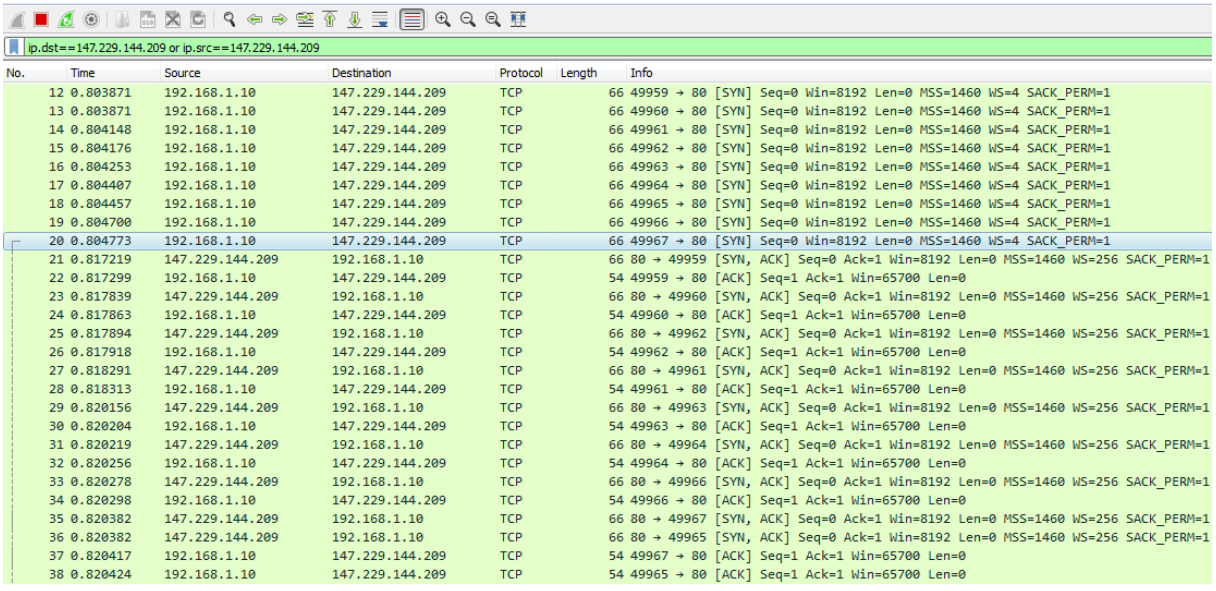
No.	Time	Source	Destination	Protocol	Length	Info
62	1.771239	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
63	1.771293	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
64	1.771428	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
65	1.771448	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
66	1.771514	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
67	1.771588	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
68	1.773472	192.168.1.1	192.168.1.10	TCP	60	80 → 50229 [ACK] Seq=1 Ack=841 Win=16680 Len=0
69	1.773472	192.168.1.1	192.168.1.10	TCP	60	80 → 50228 [ACK] Seq=1 Ack=1051 Win=16470 Len=0
70	1.773625	192.168.1.1	192.168.1.10	TCP	60	80 → 50230 [ACK] Seq=1 Ack=1051 Win=16470 Len=0
71	1.773626	192.168.1.1	192.168.1.10	TCP	60	80 → 50231 [ACK] Seq=1 Ack=946 Win=16575 Len=0
72	1.773638	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
73	1.773750	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
74	1.773837	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
75	1.773919	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
76	1.773968	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
77	1.774055	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
78	1.774188	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
79	1.774254	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
80	1.775525	192.168.1.1	192.168.1.10	TCP	60	80 → 50228 [ACK] Seq=1 Ack=1261 Win=16260 Len=0
81	1.775525	192.168.1.1	192.168.1.10	TCP	60	80 → 50229 [ACK] Seq=1 Ack=1051 Win=16470 Len=0
82	1.775627	192.168.1.1	192.168.1.10	TCP	60	80 → 50230 [ACK] Seq=1 Ack=1261 Win=16260 Len=0
83	1.775628	192.168.1.1	192.168.1.10	TCP	60	80 → 50231 [ACK] Seq=1 Ack=1156 Win=16365 Len=0
84	1.775804	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
85	1.775814	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
86	1.775825	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
87	1.775981	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
88	1.776092	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
89	1.776167	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
90	1.776232	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0
91	1.776330	192.168.1.10	192.168.1.1	HTTP	159	GET /scopia/branding/img/logo_product.png HTTP/1.0

obr. 5 - Zasílání jednotlivých žádostí (Program Wireshark)

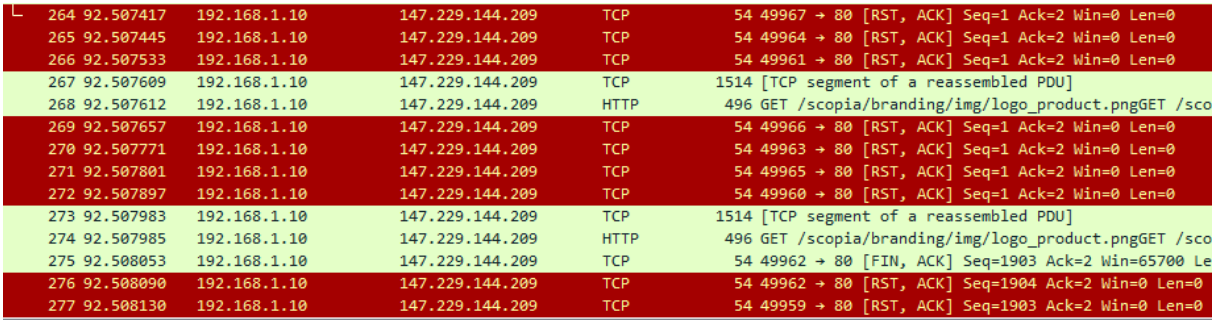
Nově přidanou funkcí aplikace je metoda, snažící se napodobit útok Slow HTTP GET. Cílem útoku je napodobit chování klienta s velmi pomalým připojením k internetu a paralelně na mnoha vláknech vytvořit dostatek aktivních spojení k zaplnění tabulky aktivních připojení. Metoda tento proces napodobuje vytvořením TCP spojením a následujícím pomalým skládáním a postupným zasíláním zprávy, a to opět v nekonečném cyklu. Cílový server spojení z jedné strany po určité době spojení ukončí, z druhé strany ovšem zůstává otevřené až do ukončení cyklu. Tímto procesem ovšem nastává situace, kdy po dlouhou dobu zůstává otevřeno jednosměrné spojení mezi zdrojem a cílem, a dochází tím k blokování části tabulky aktivních TCP spojení serveru. Jedná se ovšem spíše o popis útoku typu SYN flood a metoda tudíž zůstává kombinací těchto typů útoku. Průběh byl zaznamenán programem Wireshark a je zobrazen postupně na následujících třech obrázcích obr. 6, obr. 7 a obr. 8.



obr. 6 - Dlouhodobé spojení jedním vláknem (Program Wireshark)



obr. 7 - Krátkodobý test po 9 vláknech - navázání spojení (Program Wireshark)



obr. 8 - Krátkodobý test po 9 vláknech - ukončování až po zastavení programu (Program Wireshark)

Jedním z hlavních úkolů bylo více vláknové zpracování, umožňující několikrát paralelně spustit stejný kód. Možnost více vláknového zpracování je implementována přímo v knihovně *Boost* ve třídě *Thread*. Funkce, která je volána programem při spuštění jednoho z výše popsaných útoků (HTTP Flood a Slow HTTP), se uživatele nejdříve zeptá na informace potřebné k běhu. Mezi tyto informace patří cílová adresa, zpoždění a hlavně i počet vláken, na kterých útok poběží. Počet vláken byl omezen na maximálně dvě sta, a to z důvodu značné nestability programu při použití vyššího počtu. Na prvním vlákně je pak spuštěna funkce *end*, zajišťující ukončení programu zapsáním specifického znaku. Na zbylém počtu vláken, který je tedy maximálně sto devadesát devět, je paralelně načten kód útoku a následně jsou vlákna spuštěna. Po zavolání funkce *end* je do podmínky nekonečného cyklu útoku dosazena hodnota, znamenající konec cyklu, což má za následek návrat programu do funkce volající vlákna a jejich následné ukončení.

V nové verzi je tedy implementována řada změn. Při spuštění aplikace je uživateli nabídnuto několik možných činností. První dvě možnosti nezaznamenaly velkou změnu a jde o způsoby načtení textu zasílané žádosti, reprezentované možností A nebo B. Při zvolení první možnosti je uživatel vyzván k zadání textu žádosti přímo do příkazové řádky. Druhou možností je načtení textového řetězce ze souboru a již zde nastala změna. V nové verzi již není automaticky načten soubor specifického jména, umístěný ve stejné složce se spustitelným souborem programu. Místo toho dojde k vyzvání uživatele k zadání názvu souboru a cesty k němu. Dojde-li k zadání pouze názvu souboru, program jej bude hledat pouze ve složce se spustitelným souborem, pokud je zadána celá cesta i s názvem souboru, program načte textový soubor podle cesty. Následuje možnost výběru druhu útoku, reprezentované možnostmi C a D. Při volení jedné z těchto možností je programem volána stejná funkce, pouze s jiným parametrem. Účelem volané funkce je zjištění potřebných informací ke spuštění útoku pomocí vyzvání uživatele k jejich zadání a následné spuštění požadované funkce na specifikovaném počtu vláken. Dalším rozdílem je pak ukončení útoku, které již neprobíhá po zaslání zadaného počtu paketů, ale na žádost uživatele. Je realizováno funkcí end a běží po celou dobu průběhu útoku na samostatném vlákně. Poslední možností nabízenou programem je jeho ukončení.

Testy provedené na serverech, které byly dány k dispozici za účelem testování aplikace, je jednoznačně viditelný přínos vice vláknového přístupu. Pokud útok běží přes jediné vlákno (je zadán počet vláken dvě, jedno je pro ukončení) dokáže program zasílat průměrně osm, maximálně deset, paketů za vteřinu. Pokud vláken pro útok použiji více, počet zaslaných paketů se zhruba násobí počtem vláken, rozdíl lze jasně vidět na obr. 9.

obr. 9 - Zasílání po jednom vlákně a zasílání po 199 vláknech (Spirent Avalanche commander)

6.2. Vytvořená aplikace a operační systém Linux

Komplikace začaly zjištěním, že .exe soubor vytvořený programem Visual studio na operačním systému Windows nebude spustitelný v operačním systému Linux. Byla tedy potřeba zdrojový soubor zkompilovat pro Linux. Za tímto účelem byl na server k provedení útoku nainstalován program GCC, schopný příkazem g++ zkompilovat kód napsaný v jazyce C++. Dále bylo třeba nainstalovat verzi *Boost* knihovny, vhodnou pro daný operační systém.

Vzhledem k přístupu k serveru jedině přes příkazovou řádku bylo potřeba dát dohromady příkaz, který kompilátoru poskytne dostatek informací k provedení kompilace. V tuto chvíli se začali objevovat potíže. První byla nekompatibilita příkazů, pracujících s datovým zásobníkem, implementující ochranu proti přetečení zásobníku. Příkladem může být funkce *scanf_s*, využívající standardního vstupu k načtení dat zadaných do konzole a implementující ochranu proti přetečení zásobníku. Bylo tedy potřeba vytvořit odlišnou verzi programu, implementující pouze příkazy kompatibilní s operačním systémem Linux.

Po úpravě použitých funkcí se ovšem objevil další problém. I přes řádné nainstalování knihovny Boost kompilátor GCC nebyl schopný najít potřebné soubory. Bylo proto potřeba pomocí parametru *–I* zadat kompilátoru cestu k adresáři, ve kterém byly potřebné soubory.

Následoval další pokus o kompilaci, který byl ukončen chybou linkeru, a to v každém řádku kódu. Chyby byly o nedefinovaných funkcích a nakonec byly v nedefinování cest k součástem knihovny *Boost*, využívaných linkerem k vložení implementací funkcí. Bylo tedy potřeba přidat další parametry programu GCC, a to o umístění složky *stage/lib* pomocí parametru *-L*, vytvořené při instalaci knihovny, a následně přidání dvou potřebných souborů pomocí parametru *-l*. Po přidání všech parametrů konečně následovala úspěšná kompilace kódu.

Při pokusu o spuštění se ovšem objeví další chyba, informující o nenalezení dynamicky linkovaných knihoven. Řešení této situace spočívá v nutnosti linkovat složku *stage/lib* pomocí příkazu LD_LIBRARY_PATH při každém spuštění. U zkompilovaného souboru je stejnojmenný skript, který příkaz obsahuje, je ovšem potřeba na každém zařízení, ze kterého bude verze pro Linux spouštěna, nejdříve nainstalovat knihovnu *Boost* a následně upravit cestu k linkovanému adresáři podle systému. Všechny provedené pokusy automatizovat tento proces jakýmkoliv způsobem bohužel selhaly. Po nalinkování složky s potřebnými soubory se program spustí a je funkční, až na načítání dat ze souboru.

6.3. Možnost implementace distribuovaných útoků DDoS

Základní definicí distribuovaného útoku DDoS je snaha blokování služby serveru, a to za pomocí více než jednoho zdroje. Tento typ útoku přináší pro útočníky mnohé výhody, dochází totiž nejen k znásobení síly útoku počtem zdrojů, ale i ke znásobení výpočetních a síťových prostředků pro útok, přičemž cílový server je má stále stejné. Další výhodou může být fakt, že útočník sám obvykle pouze řídí ostatní a sám nic cíli nezasílá, jeho identita tedy zůstane skryta.

Pro implementaci schopnosti takových útoků do aplikace by nejdříve bylo třeba rozšířit aplikaci na více zařízení. Útočníci tohoto cíle obvykle dosahují šířením škodlivých kódů, které pak nainstalují a spustí aplikaci, nebo umožní vzdálený přístup. Novodobě se ovšem objevují dobrovolníci, kteří si software úmyslně nainstalují a spustí jej. Vzhledem k vzdělávacímu a testovacímu zaměření aplikace je uvažováno dobrovolné rozšíření a spuštění programu na několika zařízeních.

Dalším krokem je zajištění synchronního spuštění útoku, potřebné ke zvýšení efektivity útoku (budou-li útoky probíhat v různou dobu, nedojde k navýšení účinnosti). K tomuto účelu je třeba určit jeden ze zdrojů jako řídící a ostatní převést do stavu čekání na pokyn k zahájení. Jedna aplikace, ovládaná útočníkem, by tedy musela mít seznam adres všech ostatních. Řídicí stanice

by pak po zadání všech potřebných informací rozeslala tyto informace ostatním, čímž by došlo k synchronnímu spuštění.

Nejlepším řešením tedy je přidat aplikaci dvě nové metody, zajišťující jejich vzájemnou síťovou komunikaci. První z metod by přepnula aplikaci do stavu, kdy čeká na síťově zaslaný příkaz k zahájení útoku (takzvaný bot, součástí botnetu), obsahující všechny potřebné informace. Mezi tyto informace patří IP adresa cíle, časové prodlevy, počet vláken pro útok a zasílaný textový řetězec. Metoda by musela být po celou dobu běhu spuštěna na odděleném vlákně a čekat na následný signál o ukončení útoku. Komunikace mezi aplikacemi by probíhala na specifické portu, na kterém by všechny aplikace čekající na zahájení útoku naslouchaly. K tomu je zapotřebí provést proces *bind*, který přiřadí socketu číslo portu. Bez této operace by socketu bylo přiřazeno náhodné číslo portu z rozsahu dynamicky generovaných portů a komunikace by selhala. Druhá metoda by zajišťovala řídící funkci (takzvaný C&C Server), mající na starost rozeslání potřebných informací k zahájení či ukončení útoku známým stanicím na specifickém portu.

7. Závěr

Práce obsahuje souhrn nalezených informací o nejčastějších a nejběžnějších typech DoS a DDoS útoků. Dále také obsahuje informace o protokolech, jejich vadách a nedostatcích, kterých je k provedení útoků zneužíváno, přičemž byly rozděleny podle vrstvy síťového modelu TCP/IP, na které probíhají. Jsou zde také uvedeny již existující nástroje k realizaci různých druhů útoků, z nichž některé byly původně vyvinuty k testování odolnosti serverů.

Praktická část práce je věnována vývoji vlastní aplikace k provedení DoS útoku, jeho následná demonstrace na přiděleném hardware na VUT v Brně a sledování výsledků pomocí Spirent Avalanche commander. Druhá verze programu byla rozšířena o nový útok a hlavně o vice vláknové zpracování, které umožnilo zesílení síly útoku o pročet použitých vláken. Dále byl program i přes některé potíže, hlavně s odkazováním na knihovny, zprovozněn a otestován na operačním systému Linux. Podle výsledků lze říct, že při provedení implementovaných útoků z jednoho zdroje nejsou dostačující na úplném zahlcení serveru a tím způsobení stavu, kdy cílový server nezvládá zpracovávat požadavky legitimních uživatelů.

Literatura

[1] *DDoS Survival Handbook* [online]. Radware, Ltd, 2013 [cit. 15.12.2015]. Dostupné z: http://security.radware.com/uploadedFiles/Resources_and_Content/DDoS_Handbook/DDoS_Handbook.pdf

[2] IEEE SYSTEMS, Man and Cybernetics Society. *IEEE International Conference on Networking, Sensing and Control, 2008: ICNSC 2008 ; 6 - 8 April 2008, Sanya, China*. Piscataway, NJ: IEEE Operations Center, 2008. ISBN 978-142-4416-868.

[3] GAO, Zhiqiang a Nirwan ANSARI. Differentiating Malicious DDoS Attack Traffic from Normal TCP Flows by Proactive Tests. *IEEE Communications Letters*. 2006, **10**(11), 793-795. DOI: 10.1109/LCOMM.2006.060669. ISSN 1089-7798. Dostupné také z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4020544

[4] Dostálek, L., Kabelová, A. Velký průvodce protokoly TCP/IP a systémem DNS. Computer Press, Brno, 2008. Páté vydání, ISBN: 978-80-251-2236-5.

[5] [IEEE], Man and Cybernetics Society. *International Conference on Telecommunications, 2008: ICT 2008 ; 16 - 19 June 2008, [St. Petersburg, Russia]*. Piscataway, NJ: IEEE, 2008. ISBN 978-142-4420-360.

[6] *Computing, Measurement, Control and Sensor Network (CMCSN), 2012 International Conference on*. S.l.: [s.n.]. ISBN 978-146-7320-337.

[7] *TCP Flags: PSH and URG* [online]. [cit. 15.12.2015]. Dostupné z: http://packetlife.net/blog/2011/mar/2/tcp-flags-psh-and-urg/

[8] SPONSORED BY IEEE COMPUTER SOCIETY, IEEE Communications Society. *Proceedings, IEEE INFOCOM'99: the conference on computer communications : Eighteenth annual joint conference of the IEEE Computer and Communications Societies : the future is now : 21-25 March, 1999, Hotel Sheraton, New York, NY, USA*. Piscataway, NJ: IEEE, 1999. ISBN 07-803-5417-6.

[9] *Layer Seven DDoS Attacks* [online]. [cit. 15.12.2015]. Dostupné z: http://resources.infosecinstitute.com/layer-seven-ddos-attacks/

Seznam použitých zkratek

C&C - Command and Control (řízení, příkazy)

DDoS - Distributed Denial of Service (Distribuované blokování služby)

DNS - Domain Name System

DoS - Denial of Service (Blokování služby)

HOIC - High Orbit Ion Cannon

HTML - HyperText markup Language

HTTP - HyperText Transfer protocol

IANA - Internet Assigned Numbers Authority

ICMP - Internet Control Message Protocol

IEEE - Institute of Electrical and Electronics Engineers

IGMP - Internet Group Management Protocol

IRC - Internet Relay Chat

LOIC - Low Orbit Ion Cannon

MAC - Media Access Control

MTU - Maximum Transmission Unit

PPP - Point to Point Protocol

ROM - Read Only Memory

RTP - Real-time Transfer Protocol

SMTP - Simple Mail Transfer Protocol

SSL - Secure Socket Layer

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

URL - Uniform Resource Locator

VoIP - Voice over IP

Seznam příloh

- DVD obsahující:
  - Elektronickou verzi práce.
  - Vytvořený program pro Windows a potřebné knihovny (Bakalarka.exe).
  - Vytvořený program pro Linux a potřebné komponenty pro spuštění
    - Zkompilovaný program (Bakalarka)
    - Knihovnu Boost, potřeba nainstalovat pro spuštění na Linuxu
    - Skript obsahující příkaz ke spuštění (nutno editovat cestu před spuštěním Bakalarka.sh)