VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO DETEKCI BEZPEČNOSTNÍCH SLABIN WEBOVÉHO SERVERU NGINX

NGINX WEB SERVER SECURITY WEAKNESS DETECTION TOOL

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE **MICHAL WAGNER**
AUTHOR

VEDOUCÍ PRÁCE **Ing. KŘENA BOHUSLAV, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

157111

Ústav: Ústav inteligentních systémů (UITS)
Student: **Wagner Michal**
Program: Informační technologie
Název: **Nástroj pro detekci bezpečnostních slabin webového serveru Nginx**
Kategorie: Bezpečnost
Akademický rok: 2023/24

Zadání:

1. Seznamte se s webovým serverem Nginx a možnostmi jeho nastavení. Zaměřte se při tom především na funkci reverzního proxy serveru a porovnejte ji s dostupnými alternativami.
2. Prostudujte nejčastější bezpečností hrozby pro proxy servery, a to včetně útoků typu Slow DoS. Proveďte rešerši existujících nástrojů pro detekci bezpečnostních slabin reverzních proxy a na základě požadavků firmy Kyndryl navrhněte softwarový nástroj pro demonstraci bezpečnostních útoků na server Nginx.
3. Tento nástroj implementujte a s pomocí vhodného testovacího prostředí a s přiměřenou sadou testů demonstrujte možné slabiny v nastavení Nginx serveru.
4. Na základě získaných zkušeností vytvořte technickou specifikaci (doporučení) pro nastavení reverzního proxy serveru Nginx omezující možnosti jeho napadení.
5. Zhodnoťte dosažené výsledky a diskutujte další možná rozšíření práce.

Literatura:

1. DeJonghe, D. *NGINX CookBook: Advanced Recipes for High-Performance Load Balancing.* O'Reilly Media, 2020. ISBN 978-1492078487.
2. Nedelcu, C. *Nginx HTTP Server.* Third Edition. Packt Publishing, 2015. ISBN 978-1785280337.
3. Tripathi, N., Hubballi, N., Singh, Y. *How secure are web servers? An empirical study of slow HTTP DoS attacks and detection.* 11th International Conference on Availability, Reliability and Security (ARES), IEEE, 2016. DOI: 10.1109/ARES.2016.20.

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Křena Bohuslav, Ing., Ph.D.**
Konzultant: RNDr. Taťána Zítková
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 16.5.2024
Datum schválení: 4.1.2024

Abstrakt

Tato práce se zaměřuje na bezpečnost webového serveru Nginx a jeho konfigurační možnosti, s důrazem na funkci reverzního proxy serveru. Prozkoumává dostupné Nginx alternativy a bezpečnostní hrozby pro reverzní proxy servery. Dále se práce věnuje způsobu odhalení těchto bezpečnostních hrozeb pomocí metod penetračního testování a je provedena rešerše existujících nástrojů pro detekci bezpečnostních slabin reverzních proxy. V praktické části je vyvinutý nástroj pro demonstraci bezpečnostních útoků a detekci bezpečnostních slabin na server Nginx. V závěru práce je funkčnost nástroje otestována a na základě získaných zkušeností vytvořena technická specifikace pro nastavení reverzního proxy serveru Nginx omezující možnosti jeho napadení.

Abstract

This thesis focuses on the security of the Nginx web server and its configuration options, with an emphasis on the reverse proxy server feature. It explores available alternatives to Nginx and security threats to reverse proxy servers. The thesis also delves into methods of detecting these security threats through penetration testing and conducts research on existing tools for detecting security vulnerabilities in reverse proxies. In the practical part, a tool is developed to demonstrate security attacks and detect vulnerabilities on the Nginx server. The functionality of the tool is validated in a suitable testing environment, and based on the gained experience, a technical specification is formulated for configuring the Nginx reverse proxy server to limit its susceptibility to attacks.

Klíčová slova

bezpečnost, hrozba, zranitelnost, nginx, webový server, reverzní proxy

Keywords

security, threat, vulnerability, nginx, web server, reverse proxy

Citace

WAGNER, Michal. *Nástroj pro detekci bezpečnostních slabin webového serveru Nginx*. Brno, 2024. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Křena Bohuslav, Ph.D.

Nástroj pro detekci bezpečnostních slabin webového serveru Nginx

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Bohuslava Křeny, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Wagner
14. května 2024

Poděkování

Chtěl bych poděkovat Ing. Bohuslavu Křenovi, Ph.D. za pedagogické vedení mojí práce a konzultace. Poděkování patří i RNDr. Ing. Pavlovi Šedovi, Ph.D. a RNDr. Taťáně Zítkové za spolupráci a cenné rady.

Obsah

Kapitola 1

Úvod

Webové servery hrají klíčovou roli v dnešní digitální éře, umožňují správu a dostupnost webových aplikací a služeb. Mezi tyto servery patří i Nginx, který se stal jedním z nejoblíbenějších a nejpoužívanějších webových serverů díky své výkonnosti a efektivitě. Nicméně, s neustálým vývojem kybernetických hrozeb se stává bezpečnost webových serverů, včetně Nginx, stále naléhavějším tématem. Bezpečnostní slabiny v těchto serverech mohou představovat vážné riziko pro provozovatele a uživatele webových stránek a aplikací, a to včetně úniku citlivých informací, zneužití serveru nebo odmítnutí služby.

Tato bakalářská práce se zaměřuje na problematiku bezpečnosti webového serveru Nginx, s důrazem na jeho funkci jako reverzního proxy serveru a je motivována potřebami společnosti Kyndryl. Funkce reverzního proxy serveru je klíčová pro zabezpečení a správu komunikace mezi klienty a serverem. Reverzní proxy server funguje jako prostředník mezi klienty a serverem tím, že přijímá žádosti od klientů a přeposílá je na skutečný server. Tímto způsobem umožňuje optimalizaci výkonu, zabezpečení komunikace a správu provozu na straně serveru. Podrobnější informace o této funkci lze nalézt v kapitole 2.2. V rámci této práce budeme zkoumat Nginx jako reverzní proxy server a porovnáme ho s dostupnými alternativami, abychom porozuměli jeho výhodám a omezením v kontextu bezpečnosti.

Bezpečnostní hrozby na internetu jsou stále složitější a rafinovanější, a to platí i pro proxy servery, včetně Nginx. Jednou z klíčových hrozeb jsou útoky typu `Slow DoS`, které mohou zpomalit nebo dokonce zcela zastavit provoz na webovém serveru. V této práci provedeme důkladnou rešerši existujících bezpečnostních hrozeb pro proxy servery a analyzujeme útoky typu `Slow DoS`.

Dalším důležitým aspektem této práce je návrh a implementace softwarového nástroje pro detekci bezpečnostních slabin webového serveru Nginx. Tento nástroj bude sloužit k demonstraci možných bezpečnostních útoků na server Nginx, což umožní získat hlubší pochopení rizik spojených s tímto webovým serverem.

Práce zahrnuje i návrh technické specifikace pro nastavení reverzního proxy serveru Nginx, která pomůže organizacím minimalizovat možnosti jeho napadení. Tím bude zvýšena celková bezpečnost jejich webových aplikací a služeb. Vzniklá technická specifikace na základě simulací bezpečnostních útoků na reverzní proxy server Nginx byla vytvořena pro potřeby společnosti Kyndryl. Slouží na pomoc specialistům při konfiguraci reverzního proxy serveru Nginx.

Závěrem této bakalářské práce zhodnotíme dosažené výsledky a provedeme diskuzi o dalších možných rozšířeních práce. Tato práce přispěje k lepšímu porozumění problematiky bezpečnosti webových serverů a poskytne konkrétní návody a doporučení pro zlepšení jejich zabezpečení.

Kapitola 2

Nginx

Tato kapitola je zaměřena na Nginx, vysoce výkonný webový server a reverzní proxy. Seznámí nás se samotným Nginx webovým serverem. Následně se zaměří na jeho roli jako reverzního proxy serveru. Dále se zabývá ukončením SSL, důležitou funkcí Nginx pro zajištění šifrovaného spojení mezi klienty a servery. Poté základní konfiguraci pro reverzní proxy server Nginx. Závěr kapitoly se zabývá alternativami k Nginx a jejich srovnání.

2.1 Nginx web-server

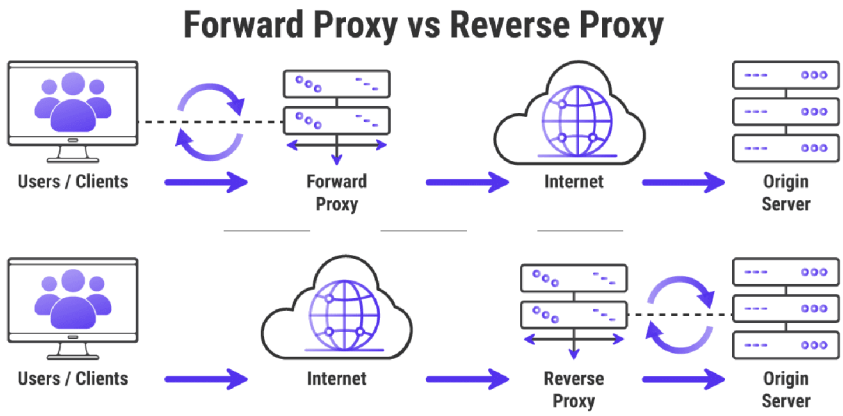
Nginx je open-source webový server a reverzní proxy server, který se stal jedním z nejoblíbenějších a nejrozšířenějších webových serverů a proxy serverů využívaných na celém světě [23]. Nginx byl vytvořen, aby vyřešil takzvaný c10k problém, což znamená, že webový server, který používá vlákna k zpracování požadavků uživatelů, není schopen spravovat více než 10 000 připojení současně [4]. V minulosti mnoho webových serverů dokázalo zpracovat pouze 10 000 připojení současně. Nginx byl vyvinut s důrazem na vysokou výkonost, spolehlivost a škálovatelnost. Tento server je schopen obsluhovat tisíce současných připojení, což ho činí efektivním nástrojem pro zajištění dostupnosti a rychlosti webových stránek [74].

Tento server představuje multifunkční nástroj schopný fungovat jako reverzní proxy, vyvažovač zátěže, vyrovnávací paměť HTTP a mnoho dalšího. Podporuje šifrování prostřednictvím SSL/TLS protokolů, což umožňuje zabezpečenou komunikaci mezi serverem a klientem. Nginx se vyznačuje událostmi řízenou, modulární, asynchronní architekturou, která funguje na jediném vlákně. Tato architektura se mimořádně efektivně škáluje na běžném serverovém hardwaru a dokáže bez problémů pracovat s vícejádrovými systémy. Na rozdíl od jiných webových serverů, jako je například Apache, se Nginx vyznačuje optimalizací využití paměti, CPU a síťového provozu. Tím dosahuje maximálního výkonu na jakémkoli fyzickém nebo virtuálním serveru. To znamená, že Nginx dokáže obsloužit nejméně 10x více (většinou 100-1 000x více) požadavků na server oproti Apache [4, 74].

2.2 Reverzní proxy

Reverzní proxy server je typ proxy serveru, který se nachází mezi klientem a backendovým serverem. Reverzní proxy server přijímá požadavky od klientů a přeposílá je na backendový server. Chrání backend servery před přímým vystavením internetu. Dokáže zachytit a zkontrolovat příchozí provoz, snížit dosah útoku a zmírnit potenciální zranitelnosti. Ke každé

službě se tedy dostane pouze nezbytný provoz. Reverzní proxy server může provádět různé úkoly, včetně [74]:



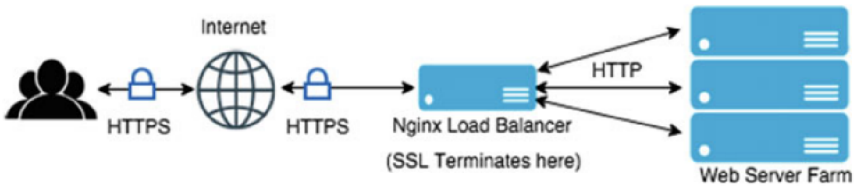
Obrázek 2.1: Reverzní proxy oproti forward proxy

- Ochrana backendového serveru: Reverzní proxy server může chránit backendový server před útoky, jako jsou útoky typu `DoS` nebo `DDoS`. Navíc dokáže skrýt IP adresy backendových serverů.
- Vyvažování zátěže: Reverzní proxy server může rozložit požadavky mezi více backendových serverů, aby se zlepšila výkonnost. Minimalizuje riziko výpadků způsobených přetížením jednoho serveru. Pokud na jednom serveru dojde k problémům, ostatní mohou pokračovat v provozu, čímž se zachová dostupnost webu. Podporuje následující metody vyvažování zátěže:
  - Round Robin: Tato metoda je používaná ve výchozím nastavení. Požadavky jsou distribuovány rovnoměrně mezi servery, přičemž se berou v úvahu váhy serverů (tento parametr je implicitně nastavený na 1).
  - Nejméně spojení: Požadavek je odeslán na server s nejmenším počtem aktivních spojení, opět s ohledem na váhy serveru.
  - IP Hash: Server, na který je požadavek odeslán, je určen z IP adresy klienta. V tomto případě se pro výpočet hodnoty hash použijí buď první tři oktety IPv4 adresy, nebo celá adresa IPv6. Metoda zaručuje, že požadavky ze stejné adresy se dostanou na stejný server, pokud je dostupný.
  - Obecný Hash: Server, na který je požadavek odeslán, je určen z uživatelem definovaného klíče, kterým může být textový řetězec, proměnná nebo jejich kombinace.
  - Náhodný: Každý požadavek bude předán na náhodně vybraný server.
  - Nejméně času: Pro každý požadavek se vybere server s nejnižší průměrnou latencí a nejnižším počtem aktivních připojení.
- Omezení přístupu: Reverzní proxy server může omezit přístup k backendovému serveru na základě IP adresy nebo uživatelského jména a hesla.
- Ukládání do vyrovnávací paměti: Reverzní proxy server může ukládat statické soubory do vyrovnávací paměti za účelem zlepšení rychlosti odezvy.

2.3 Ukončení SSL

SSL používá certifikáty k vytvoření šifrovaného spojení mezi serverem a klientem. To umožňuje bezpečný přenos citlivých informací. Bezpečná vrstva SSL je nezbytná pro každou webovou stránku, která pracuje s citlivými daty. Pokud jde o webový provoz, SSL také přináší dodatečné zpracování na straně serveru, kde pokaždé musí dešifrovat požadavek. Zde vzniká situace, která připomíná dilema: Pokud je odstraněno SSL, server je vystaven riziku útoku, a pokud je použito SSL, server přijde o trochu rychlosti nebo čelí dodatečným nákladům při škálování [74].

Vzhledem k tomu, že Nginx má schopnost fungovat jako vyvažovač zátěže, můžou se mu přidělit další úlohy. Základní myšlenkou ukončení SSL je, že požadavek přichází k vyvažovači zátěže po zabezpečeném kanálu, ale je odesílán na ostatní webové servery bez SSL. Tímto způsobem webový server pracuje rychleji a nakonec jsou požadavky odesílány klientům bezpečným způsobem [74].



Obrázek 2.2: Nginx vyvažování zátěže a SSL ukončení

2.4 Základní konfigurace Nginx

Nginx poskytuje mnoho konfigurací klíčových pro správnou funkcionalitu a optimalizaci pro konkrétní potřeby uživatele. V příloze B lze nalézt popis mnoha užitečných konfigurací pro webový server Nginx a mezi základní konfiguraci patří:

2.4.1 Konfigurační soubor

Konfigurace Nginxu se skládá z dvojic klíč-hodnota nazývaných direktivy. Direktivy rozhodují, která konfigurace se má použít. Tyto direktivy mohou být organizovány a seskupeny do bloků nazývaných kontexty. Kontexty jsou struktury podobné stromu, které mohou být vzájemně vnořené a direktivy mohou být používány pouze uvnitř kontextů [43]. Důležité kontexty (`main`, `server`, `upstream` a `location`) [30]:

- Direktivy v hlavním kontextu se vztahují ke všemu.
- Direktivy v kontextu `server` se vztahují k určitému serveru/portu.
- Direktivy v kontextu `upstream` odkazují na sadu backendových serverů.
- Direktivy v kontextu `location` se vztahují pouze k odpovídajícím webovým umístěním (například "/"a "/images").

2.4.2 Konfigurace reverzní proxy

Předání požadavku na proxy server funguje tak, že Nginx přeposílá požadavek na určený backend server, získává odpověď a posílá ji zpět klientovi.

Pro směrování požadavku na HTTP proxy server se využívá direktivy `proxy_pass` uvnitř kontextu (`location`). Například [53]:

```
location /some/path/ {
    proxy_pass http://www.example.com/link/;
}
```

Výpis 2.1: Reverzní proxy pomocí proxy_pass.

2.4.3 Omezení připojení

Nginx umožňuje omezení počtu připojení na základě předem definovaného klíče, například podle klientovy IP adresy. Je vytvořena oblast sdílené paměti pro uložení metrik připojení a je použita direktiva `limit_conn` k omezení otevřených připojení [11]:

```
http {
    limit_conn_zone $binary_remote_addr zone=limitbyaddr:10m;
    limit_conn_status 429;
    server {
        limit_conn limitbyaddr 30;
    }
}
```

Výpis 2.2: Konfigurace pro omezení počtu připojení.

Tato konfigurace vytváří oblast sdílené paměti pojmenovanou `limitbyaddr`. Předem definovaný klíč je klientova IP adresa v binární formě. Direktiva `limit_conn` má dva parametry: `limit_conn_zone` a počet povolených připojení. Direktiva `limit_conn_status` nastavuje odpověď v případě omezení připojení na status 429, což označuje příliš mnoho požadavků. Direktivy `limit_conn` a `limit_conn_status` jsou platné v kontextu HTTP, server a location [11].

2.4.4 Omezení provozu

Pomocí Nginx můžeme omezit rychlost požadavků předem definovaným klíčem, například podle klientovy IP adresy [11]:

```
http {
    limit_req_zone $binary_remote_addr zone=limitbyaddr:10m rate=1r/s;
    limit_req_status 429;
    server {
        limit_req zone=limitbyaddr burst=10 nodelay;
    }
}
```

Výpis 2.3: Konfigurace pro omezení provozu.

Tato konfigurace vytváří oblast sdílené paměti pojmenovanou `limitbyaddr`. Předem definovaný klíč je klientova IP adresa v binární formě. Direktiva `limit_req` má dva volitelné argumenty: `zone` a `burst`. Argument `zone` určuje, na kterou oblast sdílené paměti omezení žádostí se má odkazovat. Ve chvíli, kdy je počet požadavků pro danou zónu překročen, žádosti jsou odloženy, dokud nedojde k dosažení jejich maximální velikosti `burst`. Direktivy `limit_req_status` a `limit_req` jsou platné v kontextu `HTTP`, `server` a `location`. Direktiva `limit_req_zone` je platná pouze v `HTTP kontextu` [11].

2.4.5 Omezení šířky pásma

Nginx umožňuje omezení šířky pásma stahování pro klienta. Pro omezení provozu požadavků klienta využijeme direktivy `limit_rate` a `limit_rate_after` [11]:

```
location /download/ {
    limit_rate_after 10m;
    limit_rate 1m;
}
```

Výpis 2.4: Konfigurace pro omezení šířky pásma.

Konfigurace `location` bloku stanovuje, že pro URI s předponou download bude rychlost poskytování odpovědi klientovi omezena na 1 megabajt za sekundu po dosažení limitu 10 megabajtů [11].

2.5 HTTP provoz

Nginx je univerzální webový server, který dokáže zpracovat různé typy příchozího provozu. Funguje jako reverzní proxy, vyvažovač zátěže a HTTP vyrovnávací paměť, efektivně zpracovává a poskytuje požadavky HTTP/HTTPS. Dokáže přijímat HTTP požadavky od klientů a rozdělovat je mezi backendové servery, poskytuje funkce jako ukončení SSL, ukládání obsahu do vyrovnávací paměti a směrování požadavků na základě různých kritérií.

Příklad konfigurace pro HTTP provoz:

Lze využít Nginx HTTP modul pro vyvažování zátěže přes HTTP servery za použití `upstream` bloku [11]:

```
upstream backend {
    server 10.10.12.45:80 weight=1;
    server app.example.com:80 weight=2;
}
server {
    location / {
    proxy_pass http://backend;
    }
}
```

Výpis 2.5: Konfigurace pro HTTP provoz

Tato konfigurace vyvažuje zátěž mezi dvěma HTTP servery na portu 80. Parametr `weight` udává, aby se předával dvojnásobný počet připojení druhému serveru. Výchozí hodnota tohoto parametru je 1 [11].

2.6 TCP a UDP provoz

Nginx dokáže zpracovávat také TCP a UDP provoz, obvykle pomocí modulu stream. Dokáže směrovat, vyvažovat zátěž a spravovat TCP a UDP spojení.

Příklad konfigurace pro TCP provoz:

Za pomoci modulu `stream` lze vyvážit zátěž přes TCP servery použitím `upstream` bloku [11]:

```
stream {
    upstream mysql_read {
        server read1.example.com:3306 weight=5;
        server read2.example.com:3306;
        server 10.10.12.34:3306 backup;
    }
    server {
        listen 3306;
        proxy_pass mysql_read;
    }
}
```

Výpis 2.6: Konfigurace pro TCP provoz

Server blok v této konfiguraci říká, aby Nginx naslouchal TCP komunikaci na portu 3306 a vyvážil zatížení mezi dvěmi replikami MySQL databází. Uvádí i další jako zálohu, které bude předán provoz pokud budou primární databáze nedostupné [11].

Příklad konfigurace pro UDP provoz:

V tomto příkladu je využito vyvážení zátěže přes UDP servery za použití `upstrem` bloku definovaného jako udp [11]:

```
stream {
    upstream ntp {
        server ntp1.example.com:123 weight=2;
        server ntp2.example.com:123;
    }
    server {
        listen 123 udp;
        proxy_pass ntp;
    }
}
```

Výpis 2.7: Konfigurace pro UDP provoz

Tato konfigurace vyrovnává zatížení mezi dvěma upstream servery NTP (Network Time Protocol), které využívají protokol UDP. Pokud služba, která slouží pro vyrovnávání zátěže, vyžaduje odesílání více paketů tam a zpět mezi klientem a serverem, tak lze využít parametr `reuseport`. Mezi tyto služby patří OpenVPN, VoIP, DTLS [11].

2.7 Alternativy Nginx

V této části budou probrány alternativy k webovému serveru Nginx. Budou zmíněny jejich přednosti a nedostatky.

2.7.1 Apache HTTP Server

Apache HTTP Server je open-source webový server. Tento server je známý pro svou spolehlivost, flexibilitu, bezpečnost a lehkou škálovatelnost. Jednou z výhod Apache je jeho schopnost zvládat velký objem provozu s minimální konfigurací. Nabízí velké množství modulů, které umožňují uživatelům přizpůsobit konfiguraci podle specifických požadavků. Má modulární architekturu, což umožňuje vybrat a aktivovat pouze potřebné moduly, což může přispět k zvýšení výkonu. Nabízí modul mod_proxy, který mu umožňuje fungovat jako reverzní proxy server pro přesměrování požadavků a využít algoritmů na vyvažování zátěže [18]. Je kompatibilní s různými operačními systémy, což umožňuje jeho nasazení na široké škále platforem. Využívá vláknovou strukturu, což může při vysoké návštěvnosti vést k problémům s výkonem.

2.7.2 Lighttpd

Tento open-source webový server je optimalizovaný pro prostředí kritická z hlediska rychlosti, přičemž zůstává v souladu se standardy. Je navržen tak, aby byl lehký a měl malé nároky na paměť, takže je vhodný zejména pro servery, které trpí problémy s načítáním a úniky paměti. Je schopný zvládat vysoký provoz s malou spotřebou paměti a procesoru. Využívá architekturu řízenou událostmi, díky čemuž je efektivní při zpracování mnoha současných připojení. Nginx je však často vnímán jako bohatší na funkce, nabízí pokročilé funkce vyvažování zátěže a reverzní proxy. Lighttpd je často chválen pro svou jednoduchost a snadnou konfiguraci. Poskytuje modul mod_proxy. Tento modul umožňuje definovat proxy pravidla v konfiguračním souboru a umožňuje distribuovat provoz mezi různé servery pomocí vyvažování zátěže. Dokáže přesměrovat požadavky na různé backendové servery, ale nepodporuje SSL/TLS připojení na backendové servery [70].

2.7.3 Microsoft Internet Information Services (IIS)

Webový server vyvinutý společností Microsoft je určen pro poskytování obsahu na platformě Windows. IIS je dobře integrován s dalšími produkty od společnosti Microsoft a nabízí pokročilou podporu pro technologie jako jsou .NET framework, ASP.NET. Poskytuje vestavěné možnosti autentizace jako Basic, ASP.NET a Windows aut. Funguje jako vícevláknový webový server s řízením událostí, když požadavek dorazí na server IIS, je přidělen do aplikčního poolu, který běží v izolaci. Nabízí modul Application Request Routing (ARR), který umožňuje provádět funkci reverzního proxy. Dokáže směřovat a řídit požadavky mezi uživatelem a backendovými servery na základě hlaviček a algoritmů vyvažování zátěže [29].

2.7.4 LiteSpeed

Komerční webový server, který je založen na Apache HTTP Serveru. Je to velmi výkonný a škálovatelný webový server. Lze jej použít k nahrazení stávajícího serveru Apache, aniž by bylo nutné měnit jakékoli další programy nebo detaily operačního systému. Využívá architekturu řízenou událostmi, spotřebuje méně zdrojů než Apache, tímto zajišťuje lepší výkon s minimálním využitím paměti a procesoru. Může fungovat jako reverzní proxy server. Nabízí moduly i pro vyvažování zátěže a ukládání do vyrovnávací paměti [78].

2.7.5 IBM HTTP Server

Webový server, který je založen na technologii Apache HTTP Server. Jedná se o výkonný a spolehlivý server, který dokáže obsluhovat velké množství požadavků současně. IHS je využíván pro nasazení webových aplikací v podnikovém prostředí a často je integrován s dalšími produkty od IBM (např. WebSphere Application Server). Poskytuje pokročilé zabezpečení a šifrování dat. Stejně jako Apache HTTP Server poskytuje modul mod_proxy pro funkcionalitu reverzní proxy a dokáže provádět přesměrovnání požadavků a vyvažování zátěže [22].

Je to poměrně komplexní server, tudíž jeho konfikurace může být náročná. Navíc je to komerční produkt, takže jeho cena je vyšší. Tento server se hlavně využije na aplikace, které požadují vysoký výkon a škálovatelnost.

2.7.6 Porovnání klíčových vlastností

Následující tabulka porovnává klíčové vlastnosti zmíněných serverů 2.1:

Porovnání klíčových vlastností

Vlastnosti	Apache	Lighttpd	IIS	LiteSpeed	IBM HTTP	Nginx
Arch.	Process-based	Event-based	Process-based	Event-based	Process-based	Event-based
OS	Linux, Windows, Unix	Linux, Unix	Windows	Linux, Windows, Unix	Linux, Unix	Linux, Windows, Unix
Licence	Open Source	Open Source	Patent	Komerční	Komerční	Open Source
Reverzní p.	Ano	Ano	Ano	Ano	Ano	Ano
Vyvaž. z.	Ano	Ano (omezené)	Ano	Ano	Ano	Ano
SSL/TLS	Ano	Ano	Ano	Ano	Ano	Ano
Moduly	Rozsáhlé	Omezené	Rozsáhlé	Rozsáhlé	Omezené	Rozsáhlé

Tabulka 2.1: Porovnání klíčovách funkcí serverů: Apache, Lighttpd, IIS, LiteSpeed, IBM HTTP Server, a Nginx.

Kapitola 3

Penetrační testování

Penetrační testování, známé také jako pentest, je bezpečnostní testovací metoda, která se zaměřuje na aktivní vyhledávání zranitelností v informačním systému či aplikaci. Cílem penetračního testování je zkoumat bezpečnostní postupy, konfigurace a reakce na útoky systému pomocí simulace reálných útoků. Tyto útoky se pokouší získat přístup ke všem zdrojům, které útočník může dostat po úspěšném útoku, včetně citlivých dat. To umožňuje organizacím identifikovat slabá místa v jejich systémech nebo aplikacích, aby mohly být řešeny před skutečným útokem. Je nezbytné shromáždit podrobné informace o systému, což zahrnuje otevřené porty, analýzu certifikátů, zařízení v síti a jejich IP adresy, operační systémy, software a jejich verze [83].

3.1 White box a black box testování

V této části jsou probrány dva typy testování a těmi jsou:

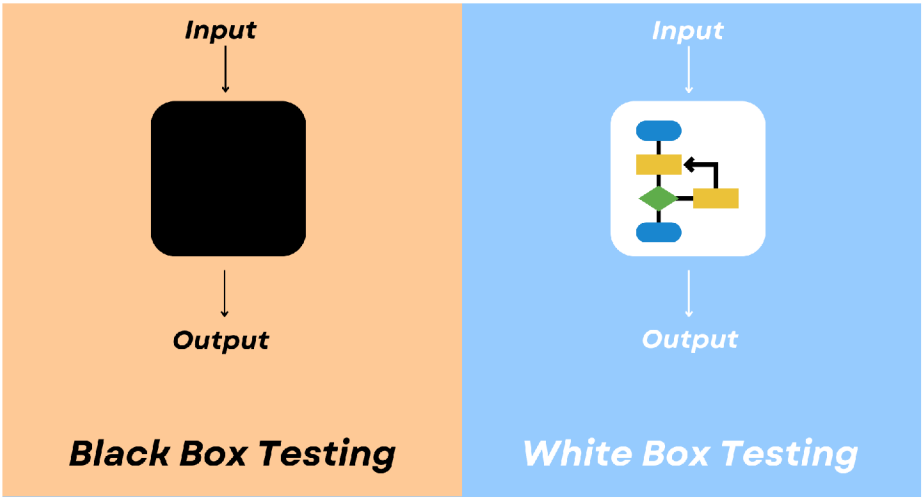
White box testování

Při white box penetračním testování má tester kompletní přehled o systému. Zahrnuje sdílení úplných síťových a systémových informací s testerem, včetně síťových map a přihlašovacích údajů. White box testování je užitečné pro simulaci cíleného útoku uvnitř sítě na konkrétní systém s využitím co největšího počtu útočných vektorů. Toto testování vyžaduje vysoké porozumění základu kódu [24, 72].

Black box testování

Při black box penetračním testování nejsou testerovi poskytnuty žádné informace. Tester v tomto případě simuluje přístup neprivilegovaného útočníka, od počátečního přístupu a spuštění až po zneužití. Tento scénář lze považovat za nejautentičtější a ukazuje, jak by se protivník bez znalostí o struktuře kódu a architektuře systému zaměřoval na organizaci a kompromitoval ji.

Black box testování se týká pouze vstupů a výstupů, protože tyto dvě věci jsou pod kontrolou testera. Bere v úvahu vnější chování systému. Tester vybírá různé proměnné, aby zjistil, jak softwarová aplikace reaguje. Cílem black box testování je předpovědět, jak se testovaný software bude chovat [24, 72].



Obrázek 3.1: Ukázka black box a white box testování

3.2 Typy testování

Typy penetračního testování závisí na rozsahu testování a požadavcích organizace. Zde jsou popsány typy penetračních testů [8, 83]:

Penetrační testování sítě

Objevuje a využívá nejvíce vystavené zranitelnosti v síťové infrastruktuře jako jsou servery, routery, firewall a switche. Toto zahrnuje kontrolu chybné konfigurace, verze softwaru, zbytečné služby, otevřené porty a další potenciálně slabá místa v síťové infrastruktuře.

Penetrační testování bezdrátové sítě

Infrastruktura bezdrátové sítě a zařízení v síti jsou testovány na zranitelnost. Bezdrátové penetrační testování se používá k identifikaci rizik spojených s bezdrátovými sítěmi a snaží se využít zranitelností jako protokolu, špatné konfigurace, slabých nebo výchozích hesel a neautorizovaných přístupových bodů.

Penetrační testování webové aplikace

Tento typ penetračního testováni se používá k nalezení zranitelností ve webových aplikacích. Toto zahrnuje identifikování bezpečnostních slabin v databázích, zdrojovém kódu a backendových sítích webové aplikace. Mezi běžné zranitelnosti zjištěné během penetračního testování webové aplikace patří SQL injection, Cross-site scripting (XSS) a špatná konfigurace. Tyto zranitelnosti budou popsány dále v kapitole 3.4.

Penetrační testování na straně klienta

Tento typ se zaměřuje na nainstalovaný software na klientském počítači. Tento software by mohl zahrnovat například webový prohlížeč, přehrávače médií a další lokálně nainstalovaný software. Útočníci využijí nainstalovaného softwaru k získání přístupu do infrastruktury organizace. Tyto zranitelnosti by mohly být zneužity phishing útoky.

3.3 Fáze penetračního testování

Existuje několik metod pro přístup k bezpečnosti organizace. Tento proces zahrnuje několik fází od shromáždění informací po shrnutí testování ve zprávě o zjištěných zranitelnostech. Tyto fáze jsou [25, 83]:

Příprava a plánování

Tato fáze je úvodním krokem penetračního testování. Je často opomíjena, ale je zásadní, aby tester a organizace objasnili požadavky a očekávané výsledky. Je důležité stanovit rozsah testování, časový plán testování a pravidla závazku. Měly by tedy být jasně určeny cíle testování, oblast testování a podepsané smlouvy.

Shromažďování detailních informací (Reconnaissance)

Tato fáze je soustředěná na shromáždění co nejvíce informací o systému. Cílem je získat relevantní data pro plánované penetrační testování. Průzkum může být aktivní, kde tester pracuje s cílovým systémem, nebo pasivní, kde se získávají veřejně dostupné informace. Aktivní sběr dat zahrnuje sítě, operační systémy, aplikace, uživatelské účty a doménová jména.

Modelování hrozeb (Threat modeling)

Na základě získaných informací testeři tvoří scénáře útoků. Jsou identifikovány potenciální cíle a mapovány způsoby útoků. Fáze spočívá ve skenování a analýze cíle z pohledu útočníka. Tester může získat informace o operačním systému, otevřených portech, protokolech a běžících službách. Cílem je nalézt relevantní zranitelnosti v systému, na které by se útočník mohl zaměřit.

Analýza zranitelností (Vulnerability analysis)

Následně penetrační testeři začínají aktivně hledat zranitelnosti, aby určili, jak úspěšné by mohly být jejich strategie využití. V této fázi testeři provádějí skeny zranitelností a manuální analýzu za účelem identifikace bezpečnostně slabých míst a určení, které útoky mají největší pravděpodobnost úspěchu.

Exploitace

V této fázi jsou spuštěny útoky na zranitelnosti, které testeři objevili pro získání přístupu k zdrojům cíle. Cílem jsou zranitelnosti, které se prokázaly, že jsou přítomné a zneužitelné.

Post-exploitace

Fáze post-exploitace je kritickou částí penetračního testování a začíná po kompromitování jednoho nebo více systémů. Během této fáze sbíráme informace o systému, hledáme zajímavé soubory, pokoušíme se zvýšit naše práva a přistoupit do jiných systémů, získat hesla a další.

Reportování

Reportování je poslední a nejvíce důležitá část penetračního testování. Předáme naši detailní zprávu zjištění zákazníkovi. Tato zpráva obsahuje, co se udělalo, jak se to udělalo

a jak by zákazník měl objevené zranitelnosti opravit. Měla by být rozdělena na technické a netechnické sekce. Netechnická sekce by měla zahrnovat shrnutí testování a technická detailní informace na nalezené zranitelnosti.

3.4 Bezpečností hrozby pro proxy servery

V této části jsou probrány časté bezpečností hrozby pro reverzní proxy servery.

3.4.1 DoS nebo DDoS

Útoky typu `DoS` nebo `DDoS` se snaží přetížit proxy server požadavky, což může vést k jeho selhání. `DDoS` přidává dimenzi "mnoho na jedno", čímž ztěžuje prevenci těchto útoků [26]. Proxy servery mohou být cílem těchto útoků, protože se nacházejí mezi klienty a backendovými servery. Pokud je proxy server přetížen, může to ovlivnit i backendové servery, které se mohou stát nedostupnými [16]. Dva z nejběžnějších DoS útoků jsou [16]:

Ping of Death

V průběhu tohoto útoku útočník odesílá stovky ping žádostí (`ICMP` žádosti o echo) s velkým nebo nelegálním objemem dat na server s cílem jej odpojit nebo udržet tak zaneprázdněným odpovídáním na `ICMP` žádosti o echo, že není schopen obsluhovat své klienty.

TCP SYN Flood

Útok využívá standardní třífázový proces navazování spojení protokolu TCP, přičemž odesílá žádost o připojení s neplatnou návratovou adresou.

3.4.2 Slow DoS

Tento útok si klade za cíl vyčerpat prostředky serveru tím, že udržuje dostatečný počet neúplných požadavků a prodlužuje dobu připojení. Pro provedení tohoto útoku vytvoří útočník několik připojení k webovému serveru a odesílá z nich neúplné HTTP požadavky. Kvůli pomalé interakci těchto připojení se serverem je server ukládá do fronty připojení až do doby, než jsou tato připojení zcela obsloužena. Jakmile je dostupný prostor ve frontě obsazen, server nepřijímá žádná (legitimní) připojení [80].

Útok Slow Header

Tento útok využívá skutečnosti, že HTTP protokol vyžaduje kompletní přijetí hlaviček GET před jejich zpracováním. Pokud server obdrží nekompletní hlavičky GET, předpokládá, že klient pracuje s pomalým internetovým připojením, a tím udržuje své zdroje obsazené čekáním na zbytek hlavičky. Útočník však nikdy neodesílá kompletní hlavičku GET. Místo toho poskytuje falešné pole hlaviček HTTP serveru v pravidelných časových intervalech, aby server resetoval expirační časovač pokaždé, kdy obdrží část hlavičky, a tím udržel připojení aktivní po delší dobu [80].

Útok Slow POST

V tomto útoku útočník odesílá kompletní hlavičku HTTP POST, ale zprávu odesílá velmi malými částmi, čímž nutí server čekat na kompletní tělo zprávy. Pole Content-Length hla-

vičky HTTP udává velikost těla zprávy v bajtech. Útočník k vytvoření tohoto útoku odesílá HTTP zprávy s vyšší hodnotou Content-Length než je skutečná velikost těla zprávy. Když webový server obdrží tuto zprávu, předpokládá, že přijal pouze část těla zprávy. Takto udržuje server své zdroje obsazené čekáním na zbytek zprávy. Útočník vytváří dostatečný počet připojení k webovému serveru a zasílá takové HTTP zprávy z každého z nich, aby obsadil frontu připojení webového serveru [80].

Útok Slow READ

V tomto útoku útočník odešle kompletní HTTP požadavek na server. Útočník, ale oznámí serveru menší velikost okna pro příjem dat ze serveru, než je obvyklé. Tím je server nucen zpomalit svoji odpověď, čímž dochází k postupnému vyčerpaní zdrojů serveru. Útočník se snaží navázat co nejvíce těchto aktivních spojení, aby došlo k úplnému obsazení volných prostředků serveru. Pro vyšší účinnost útoku si útočník vyžádá velký soubor (například obrázek) od serveru, aby spojení udržel co nejdéle. V důsledku toho se legitimní uživatelé nemohou připojit na server [68].

3.4.3 CRLF injection

CRLF (Carriage Return Line Feed) injection je typ zranitelnosti, který cílí na ukončení řádku v HTTP požadavcích a odpovědích. Tento útok umožňuje útočníkovi vložit nežádoucí hlavičky do HTTP komunikace, což může vést k přesměrování uživatele na jinou webovou stránku nebo k zranitelnosti `XSS` [19].

3.4.4 Cross-site scripting (XSS)

Útoky typu XSS využívají chyby v konfiguraci proxy serveru k vložení škodlivého kódu do webových stránek. Proxy servery mohou být cílem těchto útoků, protože často zpracovávají webové stránky. Pokud je proxy server napaden útokem typu XSS, útočník může vložit škodlivý kód na webové stránky, které jsou pak zobrazeny klientům. Tento škodlivý kód může být použit k odcizení cookies, úniku dat uživatelů nebo provádění dalších útoků [76].

3.4.5 Man-in-the-middle (MitM)

Útoky typu MitM umožňují útočníkovi odposlouchávat a měnit komunikaci mezi klientem a proxy serverem. Proxy servery mohou být cílem těchto útoků, protože se nacházejí mezi klienty a backendovými servery. Pokud je proxy server napaden útokem typu MitM, útočník může odposlouchávat komunikaci mezi klientem a backendovým serverem a dokonce ji i měnit [76].

3.4.6 Directory Traversal Attacks

Directory traversal je HTTP útok, který umožňuje útočníkovi přístup k omezeným adresářům a spouštět příkazy mimo kořenový adresář webového serveru. Tento útok může ohrozit citlivá data, která by mohla poskytnou informace požadované pro další poškození systému a konfigurační soubory na serveru. Útočník provádí příkazy tak, že se vydává za uživatele, který je spojen s webem. Vše tedy závisí na tom, k čemu má uživatel webu v systému přístup [1, 76].

3.4.7 Session hijacking attack

Útok, kdy útočník může přebrat kontrolu nad platnou relací uživatele, čímž získává neoprávněný přístup k systému nebo aplikaci. Útok kompromituje token relace tím, že ukradne nebo předpovídá platný token relace, aby získal neoprávněný přístup k webovému serveru. Token relace by mohl být kompromitován různými způsoby, nejběžnější jsou XSS nebo MitM [67, 76].

3.4.8 Špatná konfigurace zapezpečení

Tato chyba se může vyskytovat na jakékoli vrstvě zásobníku aplikací, cloud, sítě, webového serveru a databáze. Špatná konfigurace může poskytnout neoprávněný přístup k citlivým datům nebo být využita k napadení systému [9, 76].

3.5 Existující nástroje

Tyto nástroje jsou významnými pomocníky pro automatizaci testování a objevování známých bezpečnostních zranitelností v reverzních proxy serverech a pomáhají v identifikaci a řešení potenciálních hrozeb pro webové aplikace.

3.5.1 W3af

Tento open-source framework je vyvinut v programovacím jazyce Python a poskytuje prostředky pro identifikaci, testování a vyhodnocování bezpečnostních nedostatků webových aplikací. Je k dispozici na operačních systémech Windows, Linux a MacOS. Poskytuje víc jak 130 pluginů, které identifikují SQL injection, XSS a další [81].

Je to modulární framework, což umožňuje přizpůsobení testování zranitelnosti. Skládá se ze dvou důležitých částí jádra a pluginů. Jádro spouští hlavní proces a koordinuje práci pluginů a výměnu informací mezi nimi. W3af má velké požadavky na procesor, proto mu musí být omezeny zdroje, jinak dokáže využít celý procesor [38].

3.5.2 Nessus

Jedná se o výkonný nástroj, který dokáže identifikovat širokou škálu bezpečnostních zranitelností v síti. Své úkoly plní prostřednictvím více než 1200 kontrol testujících možné útoky, které by mohly ohrozit bezpečnost počítače. Nástroj prochází každý port na cílovém počítači, identifikuje běžící služby a následně testuje tyto služby, aby zajistil, že neobsahují zranitelnosti, které by mohly být zneužity k neoprávněnému přístupu nebo poškození systému. Nepředpokládá konfiguraci serveru a dokáže efektivně testovat zranitelnosti bez ohledu na specifické nastavení [14].

Jeho klient-server architektura činí skeny více škálovatelné a přesné. Mnoho uživatelů si nastavuje nessusd server, který provádí skenování na žádost klienta běžícího na jiném počítači. Servery lze umístit na různé strategické body v síti, což umožňuje provádění testů z různých úhlů pohledu [6].

3.5.3 Nikto

Nikto je open-source webový skener, který provádí komplexní testy proti webovým serverům pro více položek, včetně více než 6 700 potenciálně nebezpečných souborů a programů, kon-

troluje zastaralé verze více než 1 250 webových serverů a problémy specifické pro jednotlivé verze na více než 270 serverech [69].

Kontroluje položky konfigurace serveru, jako je přítomnost více indexových souborů, možnosti HTTP serveru, sken SSL certifikátů a pokusí se identifikovat nainstalované webove servery a software. Dokáže skenovat několik portů na serveru s několika běžícími webovými servery [77].
Identifikuje potenciální problémy a bezpečnostní zranitelnosti, včetně [77] :

- Chyby v konfiguraci serveru a softwaru
- Výchozí soubory a programy
- Nezabezpečené soubory a programy
- Zastaralé verze serverů a programů

3.5.4 OWASP ZAP

ZAP je open-source nástroj navržený pro skenování zranitelností, a jedná se o flexibilní a rozšiřitelný nástroj. V jádře funguje jako tzv. "man-in-the-middle proxy". Tedy se postaví mezi prohlížeč testera a webovou aplikaci, což mu umožňuje zachytávat a kontrolovat zprávy posílané mezi prohlížečem a webovou aplikací. V případě potřeby může upravit obsah těchto zpráv a následně je přeposílat na určené místo. ZAP je navržen s cílem identifikovat možné bezpečnostní nedostatky, které by mohly být využity pro různé útoky, včetně SQL injection, cross-site scripting (XSS), neoprávněný přístup a další [71].

3.5.5 Nmap

Nmap je bezplatný open-source nástroj pro objevování sítí. Je flexibilní a dostupný na mnoho operačních systémech. Nmap využívá IP pakety k určení dostupných hostů v síti, nabízených služeb (název a verze aplikace), běžících operačních systémů (a jejich verzí), typu používaných paketových filtrů/firewalů a mnoha dalších charakteristik. Byl navržen pro rychlé skenování rozsáhlých sítí, ale funguje dobře i při skenování jednotlivých hostů [28].

3.5.6 SSLScan

SSLScan je nástroj příkazového řádku, který vykonává rozsáhlé testy na určeném cíli a poskytuje kompletní seznam protokolů a šifer, které SSL/TLS server akceptuje. Zároveň dodává další užitečné informace pro posouzení zabezpečení. SSLScan může snížit riziko úniku dat tím, že odhaluje chyby v konfiguraci a slabiny v nastavení SSL/TLS cílových systémů [39].

3.5.7 Problémy s nástroji pro penetrační testování

- Pokrytí: Nemusí být schopný detekovat vyžadovanou bezpečnostní hrozbu.
- Složitost: Jsou to komplexní nástroje a poskytují mnoho funkcí, což přináší vysokou složitost.
- Konfigurace: Mohou mít omezené možnosti konfigurace skenů nebo filtrování výsledků.

- Rozšiřitelnost: Nemusí poskytovat další rozšíření funkcionality.
- Kompatibilita: Kompatibilita s dalsími nástroji může být obtížná.
- Cena: Některé z těchto nástrojů nabízejí bezplatnou verzi, plně vybavené obvykle vyžadují licenci.

Kapitola 4

Návrh nástroje

Pro účely demonstrace bezpečnostních rizik a slabin v nastavení serveru Nginx je pro potřeby společnosti Kyndryl požadováno vyvinout softwarový nástroj, který umožní simulovat a demonstrovat možné útoky na tento webový server. Hledání těchto jednotlivých slabin by manuálně byla zdlouhavá práce, proto je vhodné použít automatizovaný program. Jak je popsáno v kapitole 3, penetrační testování může mít mnoho podob. Tato práce si klade za cíl simulovat nejčastější bezpečnostní hrozby pro proxy servery, jejich dopad na server Nginx a prokázat vliv správné konfigurace na výsledky těchto útoků. Dalším cílem je vytvořit technickou specifikaci (doporučení) pro nastavení reverzního proxy serveru Nginx omezující možnosti jeho napadení.

4.1 Požadavky na funkčnost

Zde jsou vypsány základní úkoly, které nástroj musí provést.

- Skenování cíle: Nástroj by měl být schopný skenovat specifický server, což by mohla být IP adresa nebo doménové jméno. Vyhodnocení zda server běží, identifikace otevřených portů a spuštěných služeb.
- Simulace útoků: Nástroj bude schopný provést široké spektrum útoků, které jsou zmíněny v kapitole 3.
- Konfigurace a přizpůsobení: Nástroj by měl poskytovat možnosti pro přizpůsobení, což umožňuje testerovi upravit jeho chování na základě konkrétních požadavků na testování. Tester bude moci nastavit parametry simulovaných útoků, včetně jejich intenzity, časování a cílových míst na serveru Nginx, což umožní detailní testování různých scénářů útoků
- Detailní zpráva výsledků: Nástroj by měl generovat detailní zprávy o výsledcích penetračních testů, což zahrnuje detailní informace o zjištěných bezpečnostních zranitelnostech a jejich možných dopadech na server Nginx. Navíc poskytne informace o počtu spuštěných testů, nalezených bezpečnostních hrozbách, identifikace útoku, jejich typu a cílových místech na serveru Nginx.

4.2 Požadavky nefunkční povahy

- Přenositelnost: Tento nástroj bude možné přizpůsobit na jiný operační systém. Nástroj bude navržen a implementován pro operační systém Linux.
- Modularita: Nástroj bude rozdělen do různých komponent, které spolupracují a tvoří jeden funkční celek. Toto nabízí vysokou flexibilitu, kdy uživatel může přidávat nové moduly nebo upravovat existující moduly bez narušení existující architektury.
- Rozšiřitelnost: Nástroj bude snadné rozšířit o nové funkce nebo vlastnosti. Bude navržen tak, aby umožňoval přidávání nových modulů nebo funkcí bez potřeby velkých úprav stávající architektury.

4.3 Technologie

Jako implementační jazyk pro nástroj pro demonstraci bezpečnostních rizik byl vybrán Python. Toto rozhodnutí bylo učiněno z důvodu rozsáhlého ekosystému knihoven a frameworků, které lze využít pro úkoly související se sítí (Socket, Requests, socket, Scapy). Python je multiplatformní, což znamená, že kód napsaný v Pythonu lze spouštět na různých operačních systémech bez větších úprav. Tato flexibilita je výhodná pro testování a předvádění útoků napříč různými prostředími. Navíc lze snadno integrovat s dalšími bezpečnostními nástroji a frameworky. Toto umožňuje další rozšíření jeho funkčnosti.

Nástroj bude možné spustit z příkazového řádku a neposkytuje jakékoli jiné než textové rozhraní. Během běhu se na standardní výstup budou vypisovat ty nejdůležitější zprávy.

Důležitou součástí implementace je prezentace výsledků. Ve chvíli, kdy nástroj dokončí demonstraci útoku, tak se vygeneruje textový soubor se závěrečnou detailní zprávou o zjištěných zranitelnostech a jejich dopadu na server Nginx. Toto umožní procházení a analýzu dat získaných během simulace útoku, což umožní rychlé a přehledné zobrazení bezpečnostních aspektů serveru.

4.4 Návrh

Nástroj se bude skládat z několika souborů Python zodpovědných za funkčnost nástroje. Pro spuštění tohoto nástroje bude vyžadován vstup na příkazovém řádku ve formě cíle útoku, jako je IP adresa nebo doménové jméno, a také parametry pro konfiguraci skenu. Kromě toho bude možnost připojit na vstup konfigurační soubor pro konfiguraci skenu a definování cíle útoku. Během spuštění nástroje dojde k skenování portů zadaného cíle a vybrané moduly budou spuštěny na běžících službách. Každý modul bude pokrývat specifickou funkci v rámci celého nástroje. Díky schopnosti přidávat vlastní moduly bude nástroj vysoce rozšiřitelný. Uživatel bude moci kombinovat různé parametry a přidávat nové parametry pro spuštění nové funkcionality do již integrovaného nástroje. Výstup spuštěného nástroje bude zpracován a prezentován uživateli v jednotné formě, buď na standardním výstupu, nebo zapsán do textového souboru.

Nástroj bude schopen identifikovat implementované moduly a stanovit pořadí, v němž mají být tyto moduly provedeny. Během provádění modulů bude shromažďovat nezbytné výsledky, které budou tvořit celkový výstup. Každý modul bude pokrývat konkrétní funkčnost v rámci skenování zabezpečení.

Kapitola 5

Implementace

Tato kapitola popisuje proces převodu návrhu do implementace nástroje. Budou zde probírány vybrané technologie pro implementaci nástroje, architektura nástroje a testovací prostředí.

5.1 Nástroj pro demonstraci bezpečnostních útoků

Pro účely testování konfigurace a demonstraci útoků na reverzní proxy Nginx byl implementován nástroj v programovacím jazyce Python3. Programovací jazyk Python3 byl zvolen z důvodu bohatého výběru knihoven a modulů. Cílem experimentů je najít vhodnou konfiguraci webového serveru Nginx, která umožní bránit se a zmírnit dopad útoků. Implementovaný nástroj lze spustit jak ze strany serveru, tak útočníka.

Hlavní částí nástroje je `nginx_scanner.py`, který propojuje a spouští jednotlivé moduly nástroje. Konfigurace skenu lze nastavit pomocí konfiguračního souboru, který obsahuje potřebné parametry pro sken. Nástroj nabízí flexibilitu díky možnosti přizpůsobit parametry skenování přes příkazový řádek nebo v konfiguračním souboru. Nástroj poskytuje podrobné logování o odezvě cíle a zjištěných zranitelnostech ve formátu textového souboru, souboru `json` nebo souboru `csv`, který uživatel může zvolit pomocí argumentů příkazového řádku. Poskytuje různé úrovně logování (`DEBUG`, `INFO`, `WARNING` a `ERROR`), pomocí kterých může uživatel filtrovat logované zprávy.

Nástroj je schopný vyhodnotit, zda je cílový server dostupný, skenovat porty a shromáždit informace o otevřených portech a službách, které na nich běží. Uživatel může pomocí konfiguračního souboru nebo příkazového řádku zadat seznam portů oddělených čárkami, rozsah portů oddělený pomlčkou nebo jeden port. Na základě zadaných portů je ověřeno zda je port otevřený, a pomocí knihovny `nmap` jsou získány informace o službě na daném portu. Tyto informace jsou následně předány uživateli, nebo je hlášeno, že nebyly nalezeny žádné otevřené porty.

Nástroj provádí skenování konfiguračních souborů Nginx na detekci potencionálních zranitelností typu CRLF injection. Tento sken lze použít při spuštění nástroje ze strany serveru zadáním cest k konfiguračním souborům webového serveru Nginx. Využívají se regulární výrazy k identifikaci zranitelných částí v konfiguračním souboru a zaznamenávají se nálezy spolu s jejich kontextem (blok `server` nebo `location`). Pokud nejsou nalezeny žádné zranitelnosti, zaznamená se zpráva, že nebyly zjištěny žádné zranitelnosti.

Nástroj je schopný provádět útoky `Denial of Service`, konkrétně `Slow Header`, `Slow Body`, `Slow Read`, `ICMP Flood`, `SYN Flood` a `UDP Flood`, které jsou popsány v sekci 3.4.

Tímto umožňuje ztížit či znemožnit přístup na webový server. Útoky využívají vláken k vytvoření více připojení a současnou simulaci `Denial of Service` útoků.

Útoky `Slow Header`, `Slow Body` a `Slow Read` jsou rozděleny na dva typy:

- Kontrolovaný útok: Útoky jsou prováděny s nastavenou délkou trvání a posílají nebo čtou data s nastaveným zpožděním mezi každým čtením/posíláním. Obsahují `probe_connection`, které kontroluje, zda připojení k soketu stále reaguje. V případě, že spojení nereaguje tři sekundy, tak je spojení uzavřeno. Vytvoření soketu pro útok je nastaveno, tak aby v případě neúspěšného vytvoření soketu zkoušel nástroj vytvářet soket do doby, než je úšpěšně navázáno spojení.

- Rekurzivní útok: Útoky jsou prováděny s nastavenou délkou trvání, ale narozdíl od kontrolovaného útoku neobsahují `probe_connection`. V případě, že spojení nereaguje, tak se nástroj pokusí uzavřít spojení a rekurzivně vytvoří nové spojení k serveru přes, které nástroj pokračuje v útoku. To znamená, že po celou dobu útoku je udržován počet zadaných připojení. Vytvoření soketu pro útok je nastaveno tak, aby v případě neúspěšného vytvoření soketu zkoušel nástroj vytvářet soket do doby, než je úšpěšně navázáno spojení. Tento útok slouží jako dodatečné stresové testování direktivy, která v kontrolované variantě úspěšně ustála útok.

Útoky `ICMP Flood`, `SYN Flood` a `UDP Flood` využívají knihovnu `scapy` k vytvoření a odeslání přizpůsobených paketů pro každý typ útoku. `Slow Denial of Service` útoky využívají modul `socket` k navazání a udržení mnoha spojení s cílem vyčerpat prostředky serveru.

Popis jednotlivých argumentů nástroje:

- `-t`/`--target`: IP adresa nebo hostname cíle
- `-p`/`--port`: Port cíle
- `-pr`/`--port\_range`: Jeden port, rozsah portů (oddělený pomlčkou: 80-120) nebo list portů (oddělený čárkou: 80,443,8080) pro skenování portů.
- `-a`/`--attack`: Typ útoku (header, post, read, icmp, syn, udp)
- `-s`/`--sockets`: Počet soketů (připojení) pro SlowDoS útok
- `-th`/`--threads`: Počet vláken (připojení) pro DoS útok
- `-du`/`--duration`: Doba trvání útoku
- `-d`/`--data`: Nastavení hodnoty pro hlavičku `Content-Length` v Slow Post útoku
- `-b`/`--bytes`: Počet bajtů poslaných v jedné iteraci Slow Post útoku.
- `-ps`/`--packet\_size`: Velikost paketu odeslaného v DoS útoku
- `-nb`/`--number\_bytes`: Počet bajtů, které mají být načteny z vyrovnávací paměti při každé operaci čtení.
- `-sr`/`--start\_range`: Začátek rozsahu velikosti dat odesílaných v rámci okna TCP
- `-er`/`--end\_range`: Konec rozsahu velikosti dat odeslaných v rámci okna TCP

- `-de/--delay`: Prodleva mezi odesíláním po sobě jdoucích požadavků nebo datových paketů během útoku Slow DoS. Zadaná hodnota má vytvořenou odchylku 30%.
- `-c/--connection\_delay`: Prodleva mezi vytvářením jednotlivých vláken při útoku SlowDoS (počet připojení za sekundu).
- `-r/--reps`: Počet opakování DoS útoku
- `-crlf/--crlf\_check`: Zkontroluje konfigurační soubor na přítomnost CRLF injection poskytnutím cesty k konfiguračnímu souboru.
- `-v/--verbose`: Poskytne více informací během útoku.
- `-lo/--local`: Argument pro aktivaci monitorování při útoku ze strany serveru.
- `--log\_type`: Výstupní formát logování
- `--log\_level`: Nastavení práhové hodnoty logování.

5.2 Skript pro monitorování webového serveru

Pro účely testování a sběru dat byl vytvořen skript v programovacím jazyce Python3, který umožňuje monitorovat webový server Nginx. Skript `server_monitor.py` umožňuje sběr dat pro útoky `ICMP Flood`, `SYN Flood`, `UDP Flood` a `Slow Denial of Service`.

Tento skript vytvoří vlákno a každou sekundu sbírá data o stavu webového serveru Nginx a stavu TCP spojení na IP adrese a portu předané argumenty. Tato data následně ukládá a interpretuje v podobě grafu. Skript běží po dobu stanovenou argumentem `--duration` nebo, pokud není argument zadán, do doby, než uživatel stiskne klávesu `enter`. V případě monitorování `Flood` útoků jsou sbírána data o stavu `ICMP`, `UDP` nebo `SYN` požadavku.

Dostupnost webového serveru Nginx je kontrolována pomocí knihovny `requests` zaslaním `HTTP HEAD` požadavku. Pokud služba neodpoví do nastaveného časového limitu je považována za nedostupnou.

Pravidelně sbíraná data o TCP spojení jsou získána pomocí modulu `subprocess` a nástroje `ss`. Sbíraná data o ICMP a UDP požadavku jsou získána pomocí knihovny `scapy`.

Interpretace grafu je dosažena pomocí knihovny `matplotlib`, `mpld3` a `seaborn`. Výsledná data jsou exportována do souboru `csv` a `html`, a interpretace grafu do souboru `png`. Následně vypočítá dobu výpadku, maximální počet navázaných připojení a celkovou dobu výpadku, a výsledky jsou vypsány do konzole.

Při útoku ze strany serveru (tzn. útok ze strany serveru na stejný server) je možnost použít dříve zmíněný argument `--local` pro aktivaci funkce monitorování v rámci nástroje.

Kapitola 6

Testování

Tato kapitola probírá testovací prostředí, strukturu testování, průběh a výsledky testování nástroje.

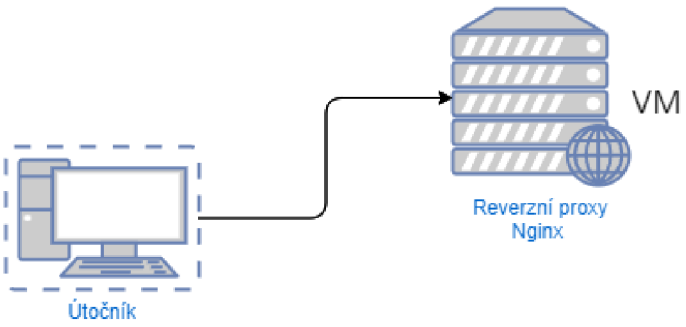
6.1 Testovací prostředí

Původním plánem bylo využít testovací server poskytnutý společností Kyndryl. Testovací server společnosti Kyndryl, ale využívá strategii s několika vrstvami ochrany ještě předtím, než se provoz dostane k testovacímu serveru. Tyto ochrany jsou výhradně zásadami společnosti Kyndryl a bližší podrobnosti nelze specifikovat v rámci této práce. Kvůli zmíněným důvodům bylo možné pouze provést testování útoků z testovacího serveru přímo na testovací server. Nástroj pro toto testování byl přizpůsoben a testování částečně proběhlo, ale bylo usouzeno, že lepší formou testování je síť složená z více zařízení. Z těchto důvodů byly vytvořené virtuální testovací zařízení za pomoci programu VMware Workstation Pro na osobním počítači s operačním systémem Windows 11. Vytvořené zařízení se procesory vyrovnají testovacímu serveru společnosti Kyndryl, ale dostupnou paměť mají dvakrát menší. Testovací prostředí lze vidět na Obrázku 6.1.
Zařízení útočníka, který provádí útoky na server:

- Jádra procesoru: 2x
- RAM[GB]: 6
- Operační systém: Fedora 39

Server, který bude ohrožený útočníkem:

- Jádra procesoru: 2x
- RAM[GB]: 6
- Operační systém: Fedora 39
- Middleware: Nginx, Tomcat



Obrázek 6.1: Ilustrace testovacího prostředí

Bylo využito reverzního proxy serveru Nginx a byla nasazená jednoduchá webová stránka pomocí webového serveru Tomcat. Webový server Tomcat byl ponechán v původním nastavení a pracovalo se pouze s konfigurací Nginx reverzní proxy. Tyto technologie byly zvoleny z důvodu napodobení prostředí na testovacím serveru společnosti Kyndryl. Byly otestovány možné konfigurace reverzní proxy Nginx pro ochranu proti různým variantám `Slow Denial of Service` útoků. Vliv a úspěšnost útoků byla monitorována ze strany serveru a byla interpretována v podobě grafů.

Cílem testování je demonstrace možných slabin v konfiguraci reverzního proxy serveru Nginx. Pomocí simulací jsou otestovány hodnoty některých direktiv v konfiguraci reverzního proxy serveru Nginx. Experimenty se zaměřují na možnosti konfigurace, které lze využít pro zmírnění dopadu útoků. Experimenty slouží jako podklad pro technickou specifikaci pro konfiguraci reverzního proxy serveru Nginx, kterou lze nalézt v příloze B.

Experimenty proběhly pro výchozí konfiguraci a manuálně upravenou konfiguraci reverzního proxy serveru Nginx. Proběhly `Slow Denial of Service` útoky ve dvou možných variantách, které jsou popsány v sekci 5.1. Ve všech simulacích byl měřen dopad útoku na reverzní proxy server Nginx.

6.2 Struktura testování

Následující struktura je použita pro jednotlivé experimenty:

- Popis experimentu
- Použité parametry nástroje pro experiment.
- Použitá konfigurace pro zmírnění dopadu útoku.
- Grafy zobrazující dopad útoku.

6.2.1 Použité parametry pro experimenty

V následujících experimentech byl zvolen počet připojení 15 000 pro kontrolovanou variantu útoků `Slow Header Denial of Service` a `Slow POST Denial of Service`. Vytvořené virtuální zařízení útočníka dokáže udržovat pouze 7 000 aktivních vláken pro útok, ale v kontrolované variantě těchto útoků dochází k postupnému ukončení spojení před dosažením tohoto limitu. Toto ukončení je příčinnou dříve zmíněného `probe_connection` a ukončením pomocí použité direktivy. Z tohoto důvodu byla využita vyšší hodnota pro robustnější testování.

Pro rekurzivní varianty útoků `Slow Denial of Service` byl zvolen počet připojení 7 000. Tato hodnota byla zvolena z důvodu, že rekurzivní varianta útoku v případě, že spojení nereaguje rekurzivně vytváří nové spojení k serveru a pokračuje v útoku. To znamená, že neustále udržuje maximální počet aktivních vláken, které virtuální zařízení může vytvořit.

Pro kontrolovanou variantu útoku `Slow READ Denial of Service` byl zvolen počet připojení 7 000 z důvodu, že virtuální zařízení útočníka nedokáže pracovat s více vlákny pro čtení dat ze serveru. Při tomto útoku je příliš velká zátěž na virtuální zařízení útočníka a spojení nejsou ukončovány stejnou rychlostí, jako při ostatních kontrolovaných útocích.

Počet připojení za sekundu byl omezen na 250 pro kontrolovanou i rekurzivní variantu útoku `Slow Header Denial of Service` a `Slow POST Denial of Service`. Pro kontrolovanou i rekurzivní variantu útoku `Slow READ Denial of Service` byl počet připojení za sekundu omezen na 500. Tato omezení jsou z důvodu postupného otevírání připojení během útoku.

Pro rekurzivní i kontrolovanou variantu útoku `Slow READ Denial of Service` byla omezena velikost okna pro příjem dat na náhodnout hodnotu v rozmezí 512-768 bajtů a rychlost čtení na 24 bajtů každých 6 sekund. Tato omezení byla zavedena pro efektivní omezení zdrojů serveru.

6.2.2 Grafy útoků

Graf zobrazuje počet připojení v průběhu času během simulace útoku na server. Následující data jsou zobrazena na grafu:

- osa x: Uplynulý čas v sekundách od začátku testu.
- osa y: Zobrazuje počet připojení.
- `Total closed`: Celkový počet uzavřených spojení.
- `Closing`: Spojení, která jsou v procesu ukončování. Soket čeká po uzavření spojení na zpracování paketů, které jsou stále v síti.
- `Closed`: Spojení, která byla ukončena.
- `Established`: Spojení, která byla úspěšně navázána a aktivně předávají data mezi klientem a serverem.
- `Service Available`: Označuje, zda je webový server Nginx aktuálně spuštěný a dostupný.

Během rekurzivních variant útoků bylo vypnuto zobrazení celkového počtu uzavřených připojení (`Total closed`). Celkový počet uzavřených připojení v rámci těchto útoků neposkytuje relevantní data.

6.3 Průběh testování

V této části jsou popsány simulace útoku, které byly provedeny na reverzním proxy serveru Nginx a jejich výsledný dopad na stav reverzního proxy serveru Nginx. Otestovány byly různé direktivy, které nabízí reverzní proxy server Nginx, aby se dospělo k nejvhodnější konfiguraci proti `Slow Denial of Service` útokům. Pro simulace byly využity dvě varianty `Slow Denial of Service` útoků (kontrolovaný/rekurzivní útok), které jsou popsány v sekci 5.1. Je porovnán úspěch `Slow Denial of Service` útoku proti výchozí konfiguraci reverzního proxy serveru Nginx oproti konfiguraci obsahující testovanou direktivu. Zmíněná varianta rekurzivních útoků slouží jako dodatečný stresový test direktiv, které byly prohlášeny za úspěšné v obraně proti `Slow Denial of Service` útoku v kontrolované variantě. Všechny použité direktivy v následujících experimentech jsou popsány v příloze B.

6.3.1 Experiment 1

Tato simulace proběhla za použití kontrolované varianty `Slow Header Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv direktivy `client_header_timeout` (viz. B.1.2) s hodnoutou 5 sekund a 10 sekund oproti výchozí konfiguraci reverzního proxy serveru Nginx. Navíce je předvedeno, že omezení počtu připojení podle IP adresy za pomoci modulu `ngx_http_limit_conn_module` (viz. B.1.3) neposkytuje proti tomuto útoku žádnou obranu. Spojení je započítáno do limitu omezení počtu připojení pouze v případě, že celá hlavička požadavku byla načtena.

Parametry použité pro útok lze vidět v Tabulce 6.1:

Typ útoku	`Slow Header Denial of Service`
Varianta útoku	kontrolovaná
Cíl útoku	`http://192.168.237.130/`
Počet připojení (soketů)	15 000
Počet připojení za sekundu	250
Prodleva mezi požadavky [s]	10
Doba trvání experimentu [s]	210
Časový limit pro `probe` připojení [s]	3

Tabulka 6.1: Parametry nástroje použité pro testování kontrolovaného `Slow Header Denial of Service` útoku

Konfigurace direktivy `client_header_timeout`:

```
http {
    client_header_timeout 5s|10s;
}
```

Výpis 6.1: Použitá konfigurace pro experiment 1 - client_header_timeout

Konfigurace modulu `ngx_http_limit_conn_module`:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;
    limit_conn_status 429;
    server {
```

```
        location / {
            limit_conn addr 10;
        }
    }
}
```

Výpis 6.2: Použitá konfigurace pro experiment 1 - limit připojení

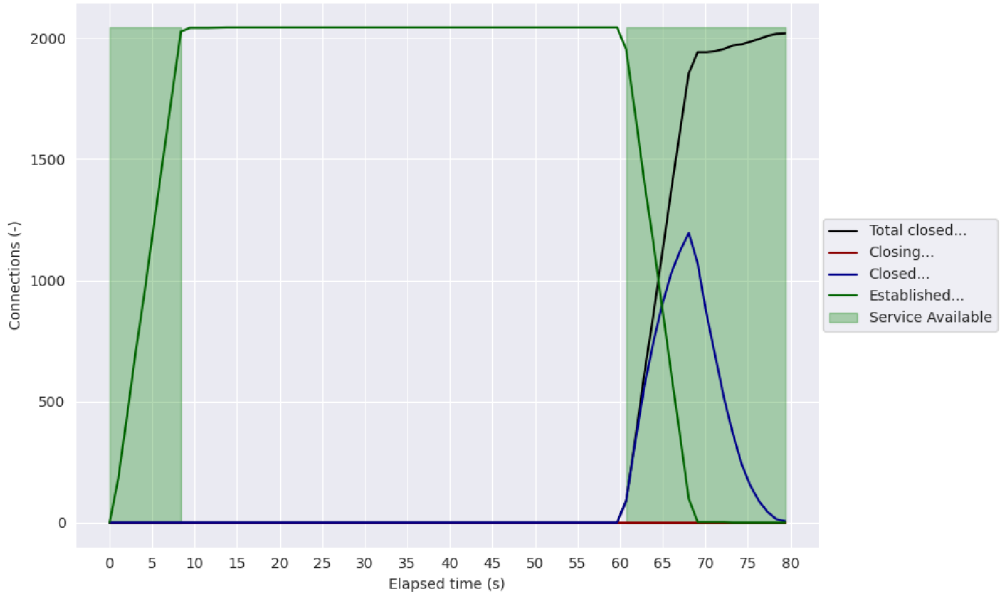
Výsledky testování

Kontrolovaný `Slow Header Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.2. Ve chvíli, kdy počet připojení přesáhl 2 000 aktivních připojení se server stal nedostupným a nepřijímal další připojení. Server se stal znovu dostupným po zásahu direktivy `client_header_timeout`, která ve výchozím nastavení má hodnotu 60 sekund.

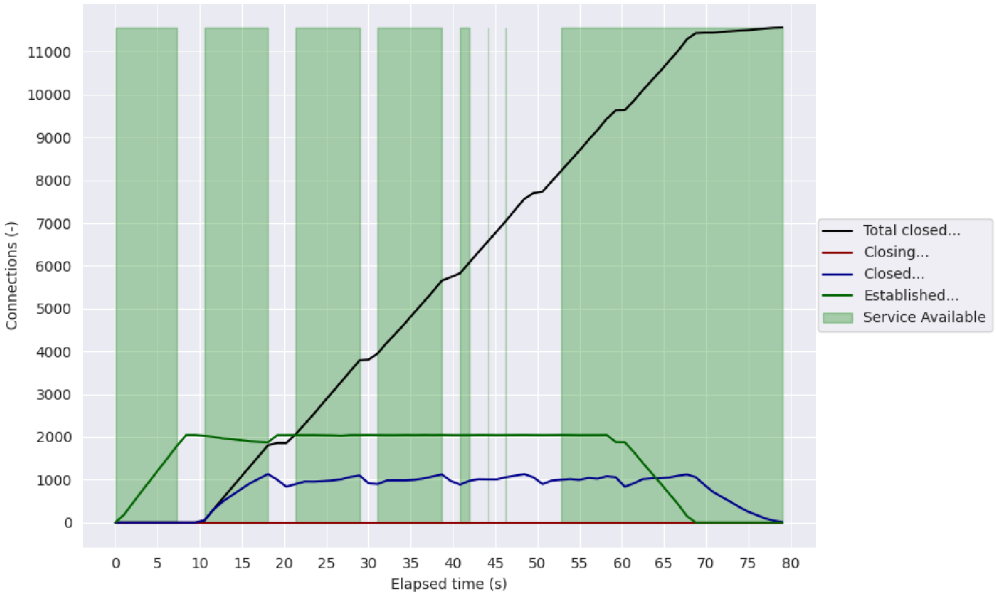
Po využití direktivy `client_header_timeout` s hodnotou 10 sekund byl útok úspěšný, jak lze vidět na Obrázku 6.3, protože reverzní proxy server Nginx nestíhal uzavírat škodlivá připojení a stal se nedostupným.

Útok byl neúspěšný po uvedení direktivy `client_header_timeout` s hodnotou 5 sekund. Reverzní proxy server Nginx zůstal dostupný po dobu útoku, jak lze vidět na Obrázku 6.4. Škodlivá připojení byla každých 5 sekund uzavírána.

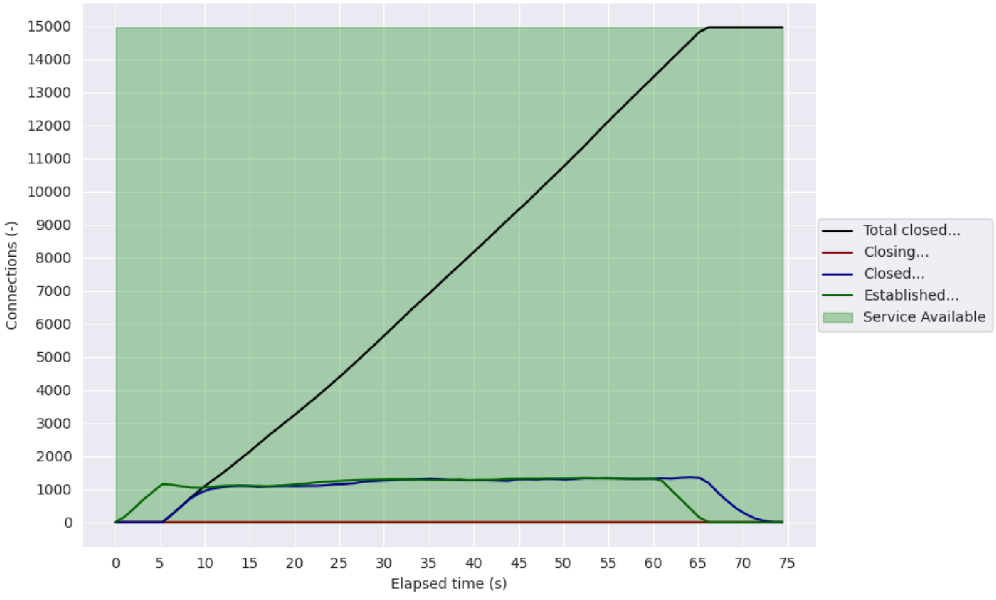
Po omezení počtu připojení, které IP adresa může vytvořit na 10 připojení byl útok proveden úspěšně, jak lze vidět na Obrázku 6.5. Bylo potvrzeno, že modul `ngx_http_limit_conn_module` neposkytuje ochranu proti `Slow Header` útoku. Server se stal znovu dostupným po zásahu direktivy `client_header_timeout` s výchozí hodnotou 60 sekund.



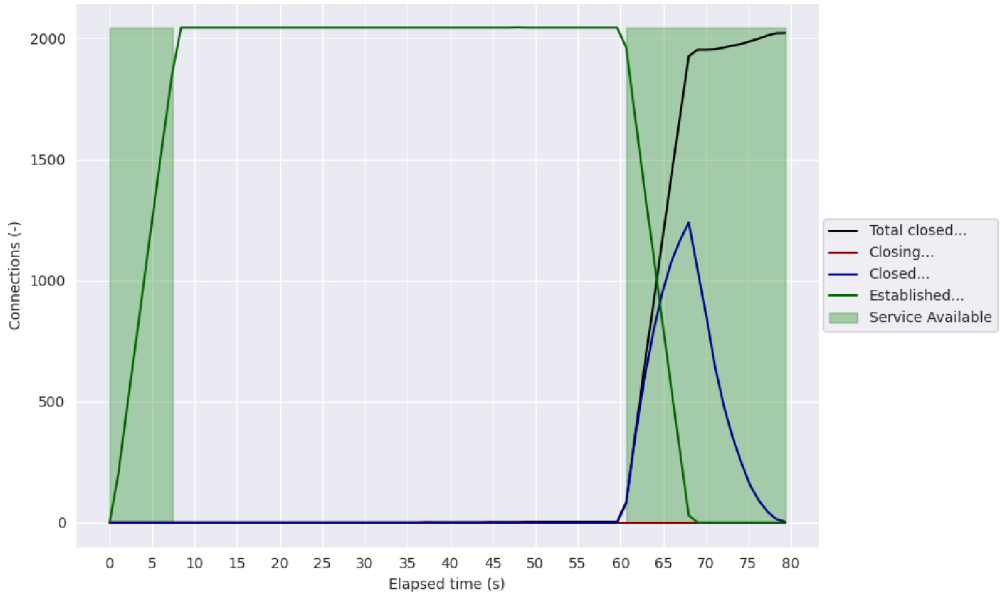
Obrázek 6.2: Reverzní proxy server Nginx pod kontrolovaným `Slow Header Denial of Service` útokem (15 000 připojení) ve výchozí konfiguraci.



Obrázek 6.3: Reverzní proxy server Nginx pod kontrolovaným `Slow Header Denial of Service` útokem (15 000 připojení) po využití direktivy `client_header_timeout` s hodnotou 10 sekund.



Obrázek 6.4: Reverzní proxy server Nginx pod kontrolovaným `Slow Header Denial of Service` útokem (15 000 připojení) po využití direktivy `client_header_timeout` s hodnotou 5 sekund.



Obrázek 6.5: Reverzní proxy server Nginx pod kontrolovaným `Slow Header Denial of Service` útokem (15 000 připojení) po omezení počtu připojení na 10 připojení.

6.3.2 Experiment 2

Tato simulace proběhla za použití rekurzivní varianty `Slow Header Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv direktivy `client_header_timeout` (viz. B.1.2) s hodnoutou 5 sekund a 10 sekund oproti výchozí konfiguraci reverzního proxy serveru Nginx.

Parametry použité pro útok lze vidět v Tabulce 6.2:

Typ útoku	`Slow Header Denial of Service`
Varianta útoku	rekurzivní
Cíl útoku	`http://192.168.237.130/`
Počet připojení (soketů)	7 000
Počet připojení za sekundu	250
Prodleva mezi požadavky [s]	10
Doba trvání experimentu [s]	210

Tabulka 6.2: Parametry nástroje použité pro testování rekurzivního `Slow Header Denial of Service` útoku

Konfigurace direktivy `client_header_timeout`:

```
http {
    client_header_timeout 5s|10s;
}
```

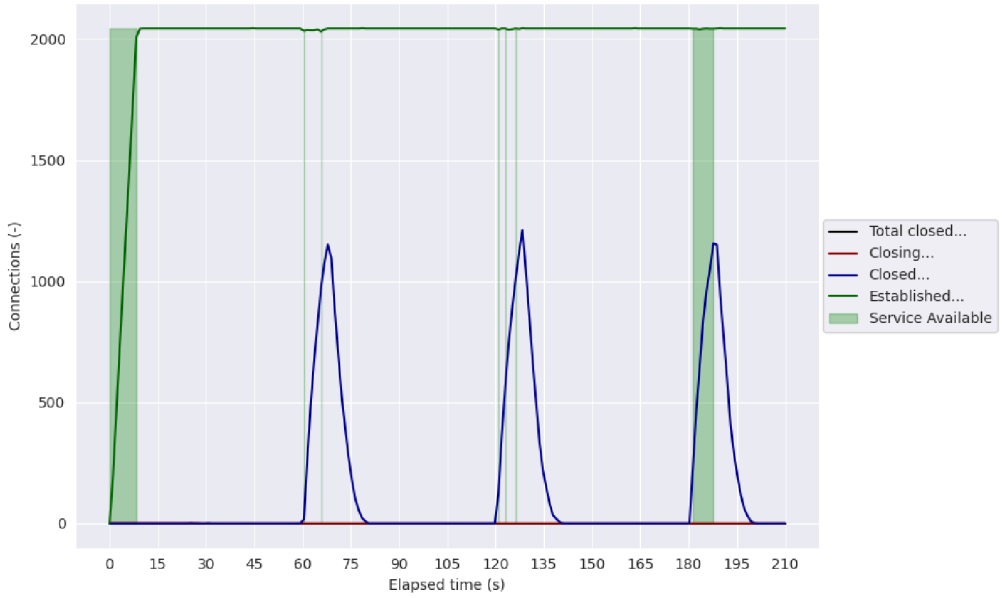
Výpis 6.3: Použitá konfigurace pro experiment 2 - client_header_timeout

Výsledky testování

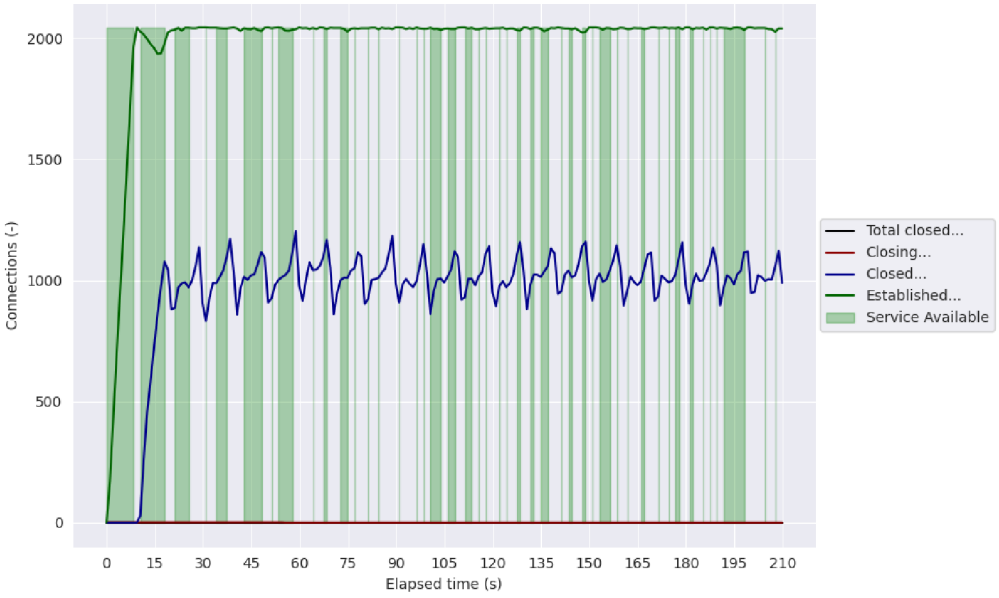
Rekurzivní `Slow Header Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.6. Ve chvíli, kdy počet připojení přesáhl 2 000 aktivních připojení se server stal nedostupným a nepřijímal další připojení. Na rozdíl od experimentu 1 zůstal server nedostupný, protože uzavřená spojení pomocí direktivy `client_header_timeout` s výchozí hodnotou 60 sekund byla ihned znovu otevřena.

Po využití direktivy `client_header_timeout` s hodnotou 10 sekund byl útok úspěšný, jak lze vidět na Obrázku 6.7, protože reverzní proxy server Nginx nestíhal uzavírat škodlivá spojení, která se opětovně vytvářela v průběhu útoku. Server byl po většinu doby útoku nedostupným.

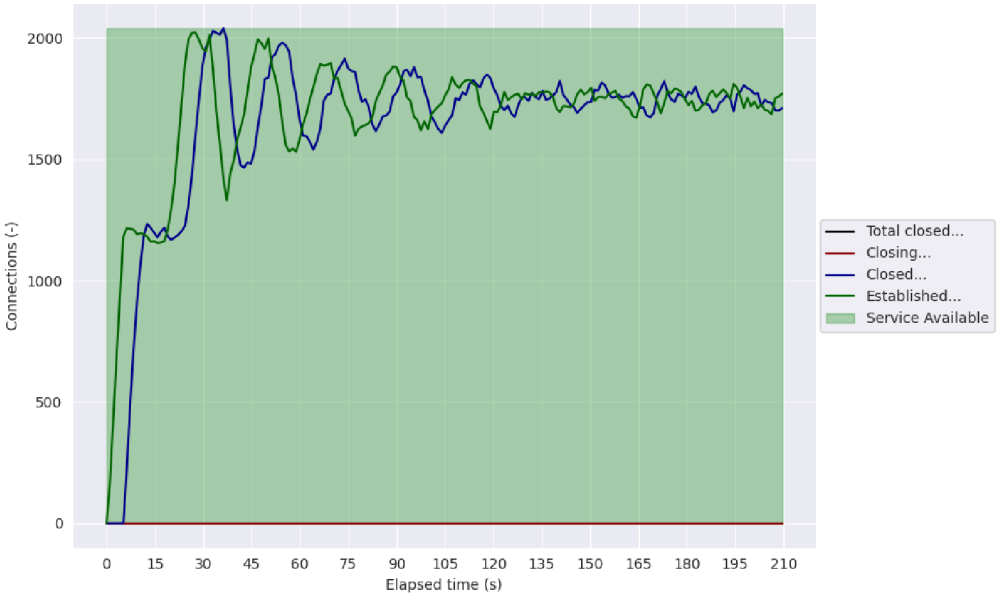
Útok byl neúspěšný po uvedení direktivy `client_header_timeout` s hodnotou 5 sekund. Reverzní proxy server Nginx zůstal dostupný po celou dobu útoku, jak lze vidět na Obrázku 6.8. Škodlivá připojení byla uzavřena a opětovně vytvořena každých 5 sekund.



Obrázek 6.6: Reverzní proxy server Nginx pod rekurzivním `Slow Header Denial of Service` útokem (7 000 připojení) ve výchozí konfiguraci.



Obrázek 6.7: Reverzní proxy server Nginx pod rekurzivním `Slow Header Denial of Service` útokem (7 000 připojení) po využití direktivy `client_header_timeout` s hodnotou 10 sekund.



Obrázek 6.8: Reverzní proxy server Nginx pod rekurzivním `Slow Header Denial of Service` útokem (7 000 připojení) po využití direktivy `client_header_timeout` s hodnotou 5 sekund.

6.3.3 Experiment 3

Tato simulace proběhla za použití kontrolované varianty `Slow POST Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv direktivy `client_body_timeout` (viz. B.1.2) s hod-

noutou 5 sekund a 10 sekund oproti výchozí konfiguraci reverzního proxy serveru Nginx. Následně je porovnán vliv direktivy `client_max_body_size` (viz. B.1.6) s hodnotou 3k a 4k oproti výchozí konfiguraci reverzního proxy serveru Nginx.

Parametry použité pro útok lze vidět v Tabulce 6.3:

Typ útoku	`Slow POST Denial of Service`
Varianta útoku	kontrolovaná
Cíl útoku	`http://192.168.237.130/`
Počet připojení (soketů)	15 000
Počet připojení za sekundu	250
Prodleva mezi požadavky [s]	10
Velikost Content-Length hlavičky [B]	4 096
Velikost posílaných dat [B]	1
Doba trvání experimentu [s]	210
Časový limit pro `probe` připojení [s]	3

Tabulka 6.3: Parametry nástroje použité pro testování kontrolovaného `Slow POST Denial of Service` útoku

Konfigurace direktivy `client_body_timeout`:

```
http {
    client_body_timeout 5s|10s;
}
```

Výpis 6.4: Použitá konfigurace pro experiment 3 - client_header_timeout

Konfigurace direktivy `client_max_body_size`:

```
http {
    server {
        location / {
            client_max_body_size 3k|4k;
        }
    }
}
```

Výpis 6.5: Použitá konfigurace pro experiment 3 - client_max_body_size

Výsledky testování

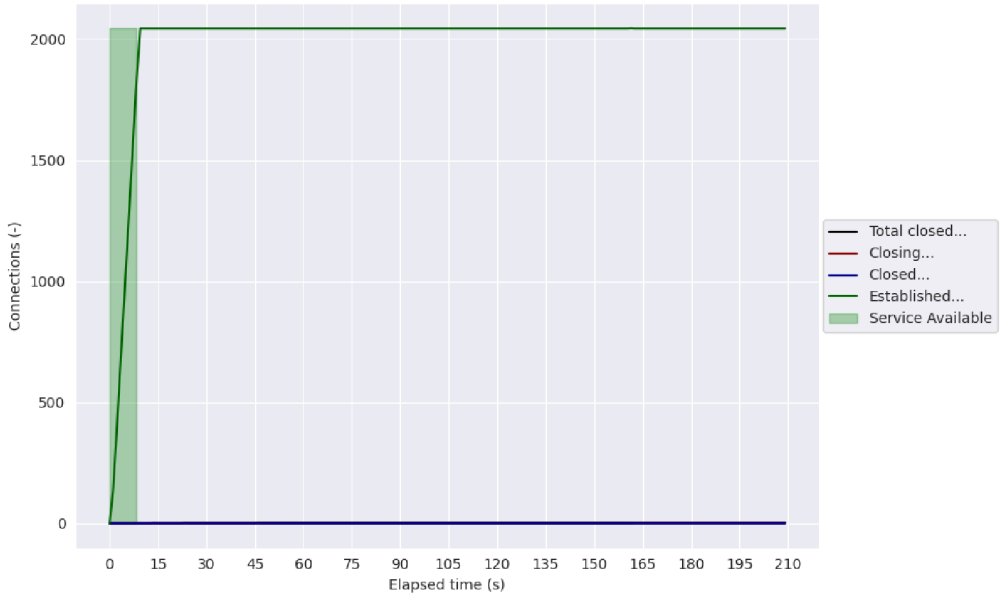
Kontrolovaný `Slow POST Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.9. Ve chvíli, kdy počet připojení přesáhl 2 000 aktivních připojení se server stal nedostupným a nepřijímal další připojení. Server zůstal nedostupný po celou dobu útoku, protože ho žádná direktiva neomezila. Výchozí hodnota direktivy `client_body_timeout` je 60 sekund, ale tento časový limit je nastaven pouze pro dobu mezi dvěma po sobě jdoucími operacemi čtení požadavku, nikoli pro přenos celého těla požadavku. Data byla zasílána s prodlevou 10 sekund takže tento limit nebyl aplikován. Výchozí hodnota direktivy `client_max_body_size` je 1 m takže tento limit taky nebyl aplikován.

Po využití direktivy `client_body_timeout` s hodnotou 10 sekund byl útok úspěšný, jak lze vidět na Obrázku 6.10, protože reverzní proxy server Nginx nestíhal uzavírat škodlivá spojení a stal se nedostupným.

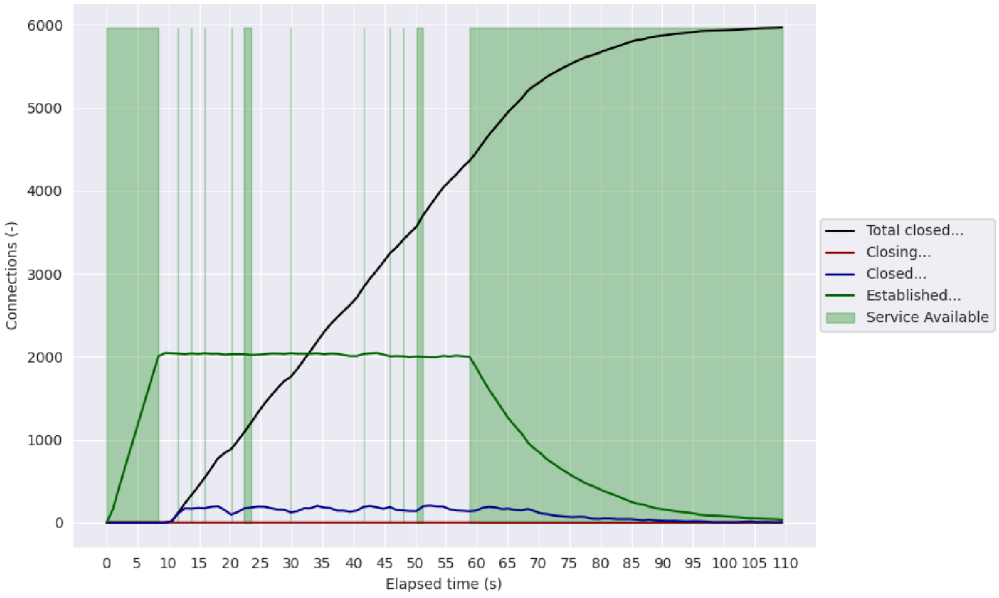
Útok byl neúspěšný po uvedení direktivy `client_body_timeout` s hodnotou 5 sekund. Reverzní proxy server Nginx zůstal dostupný po dobu útoku, jak lze vidět na Obrázku 6.11. Škodlivá připojení byla každých 5 sekund uzavírána.

Po zavedení direktivy `client_max_body_size` s hodnotou 4k byl útok na reverzní proxy server Nginx opět úspěšný, jak lze vidět v Obrázku 6.12. Útok byl úspěšný, protože hlavička `Content-Length` s hodnotou 4 096 nepřekročí limit zadaný direktivou `client_max_body_size`.

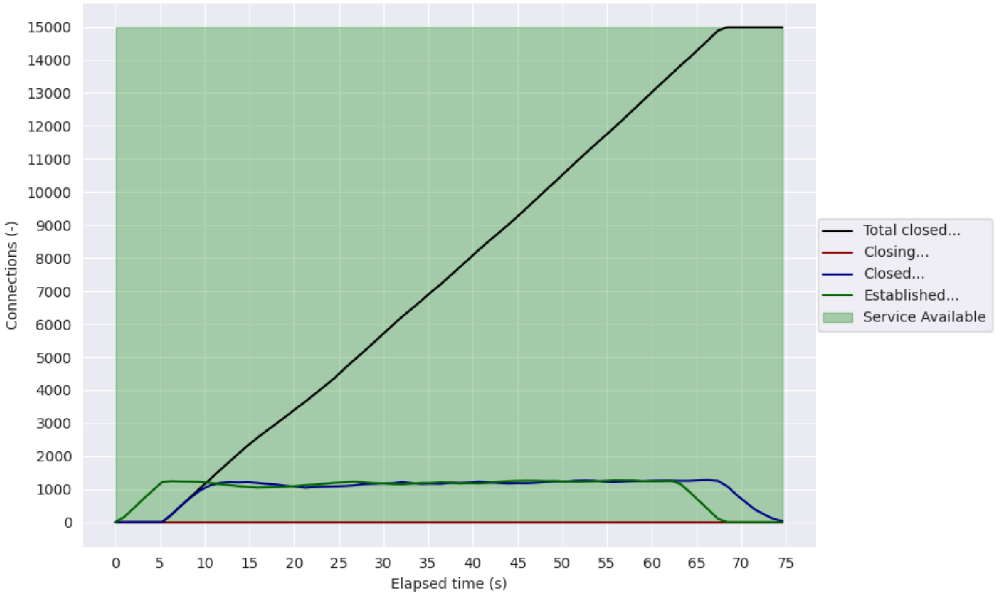
Útok byl neúspěšný po zmenšení hodnoty direktivy `client_max_body_size` na 3k, jak lze vidět v Obrázku 6.13. Útok byl neúspěšný, protože hlavička `Content-Length` s hodnotou 4 096 překročila limit zadaný direktivou `client_max_body_size`.



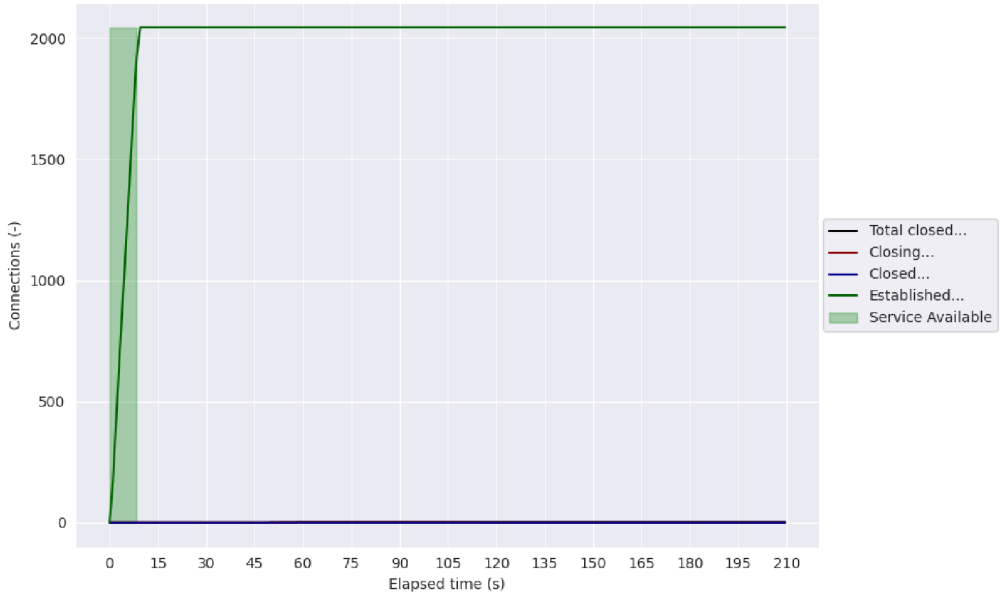
Obrázek 6.9: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) ve výchozí konfiguraci.



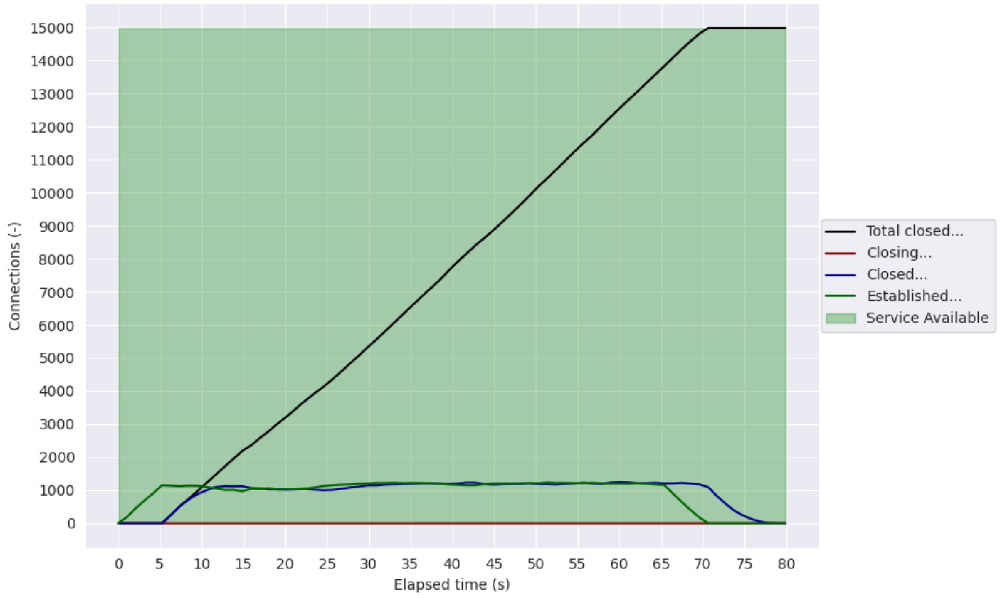
Obrázek 6.10: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) po využití direktivy `client_body_timeout` s hodnotou 10 sekund.



Obrázek 6.11: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) po využití direktivy `client_body_timeout` s hodnotou 5 sekund.



Obrázek 6.12: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) po využití direktivy `client_max_body_size` s hodnotou 4k.



Obrázek 6.13: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) po využití direktivy `client_max_body_size` s hodnotou 3k.

6.3.4 Experiment 4

Tato simulace proběhla za použití rekurzivní varianty `Slow POST Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv direktivy `client_body_timeout` (viz. B.1.2) s hod-

noutou 5 sekund a vliv direktivy `client_max_body_size` (viz. B.1.6) s hodnotou 3k oproti výchozí konfiguraci reverzního proxy serveru Nginx.

Parametry použité pro útok lze vidět v Tabulce 6.4:

Typ útoku	`Slow POST Denial of Service`
Varianta útoku	rekurzivní
Cíl útoku	`http://192.168.237.130/`
Počet připojení (soketů)	7 000
Počet připojení za sekundu	250
Prodleva mezi požadavky [s]	10
Velikost Content-Length hlavičky [B]	4 096
Velikost posílaných dat [B]	1
Doba trvání experimentu [s]	210

Tabulka 6.4: Parametry nástroje použité pro testování rekurzivního `Slow POST Denial of Service` útoku

Konfigurace direktivy `client_body_timeout`:

```
http {
    client_body_timeout 5s;
}
```

Výpis 6.6: Použitá konfigurace pro experiment 4 - client_body_timeout

Konfigurace direktivy `client_max_body_size`:

```
http {
    server {
        location / {
            client_max_body_size 3k;
        }
    }
}
```

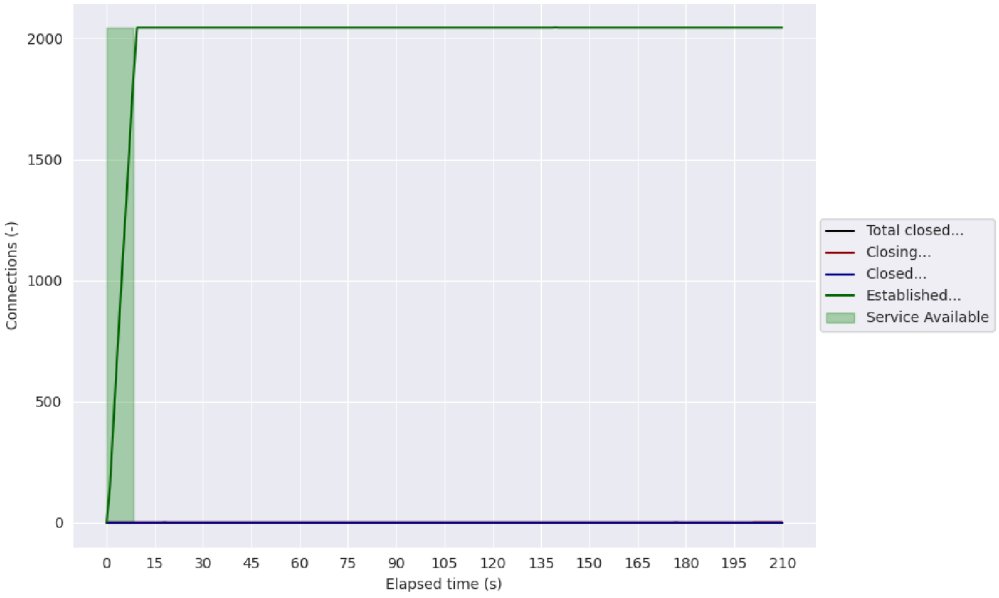
Výpis 6.7: Použitá konfigurace pro experiment 4 - client_max_body_size

Výsledky testování

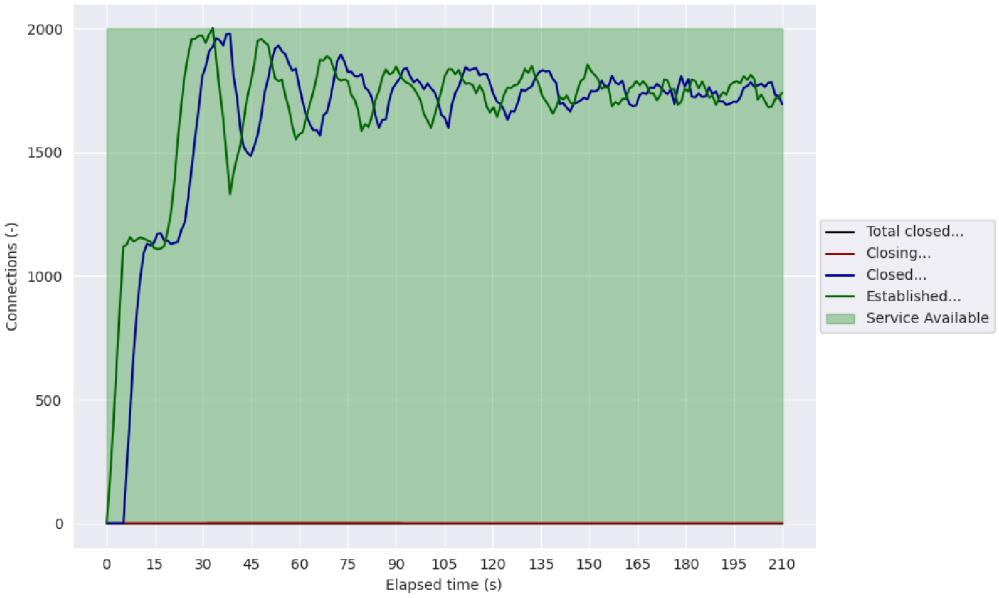
Rekurzivní `Slow POST Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.14. Ve chvíli, kdy počet připojení přesáhl 2 000 aktivních připojení se server stal nedostupným a nepřijímal další připojení. Server zůstal nedostupný po celou dobu útoku ze stejného důvodu, jako v experimentu 3 (viz. 6.3.3).

Útok se stal neúspěšným po zavedení direktivy `client_body_timeout` s hodnotou 5s, jak lze vidět v obrázku 6.15. Škodlivá připojení byla uzavřena a opětovně vytvořena každých 5 sekund.

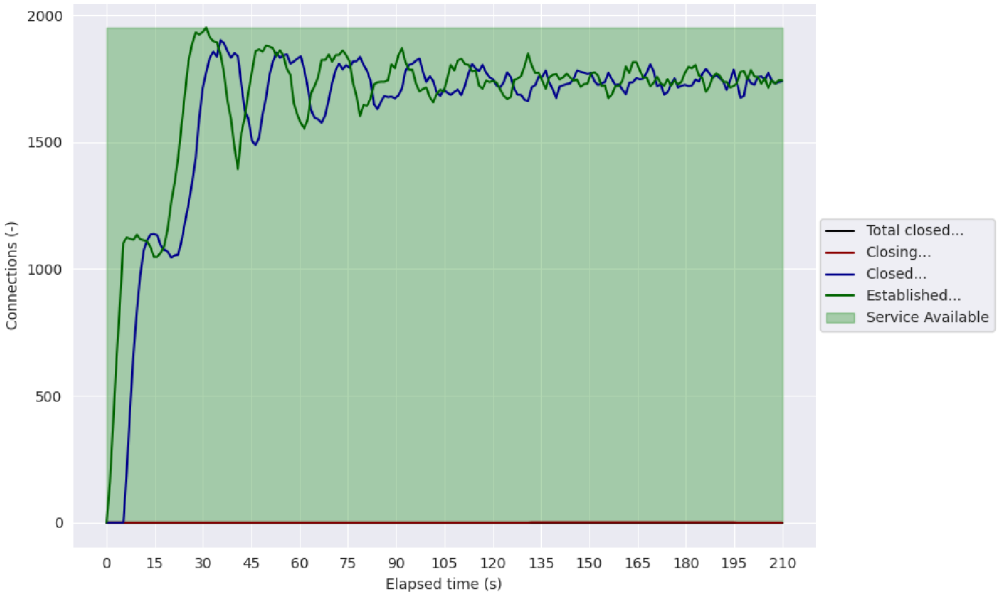
Po zavedení direktivy `client_max_body_size` s hodnotou 3k byl útok opět neúspěšný, jak lze vidět v obrázku 6.16. Škodlivá připojení byla uzavřena a opětovně vytvořena, protože hlavička `Content-Length` s hodnotou 4 096 překročila limit zadaný direktivou `client_max_body_size`.



Obrázek 6.14: Reverzní proxy server Nginx pod rekurzivním `Slow POST Denial of Service` útokem (7 000 připojení) ve výchozí konfiguraci.



Obrázek 6.15: Reverzní proxy server Nginx pod rekurzivním `Slow POST Denial of Service` útokem (7 000 připojení) po využití direktivy `client_body_timeout` s hodnotou 5 sekund.



Obrázek 6.16: Reverzní proxy server Nginx pod rekurzivním `Slow POST Denial of Service` útokem (7 000 připojení) po využití direktivy `client_max_body_size` s hodnotou 3k.

6.3.5 Experiment 5

Tato simulace proběhla za použití kontrolované varianty `Slow POST Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv omezení počtu připojení, které IP adresa může vytvořit na 30 připojení pomocí modulu `ngx_http_limit_conn_module` (viz. B.1.3) a omezení provozu na 5 požadavků za sekundu s hodnotou `burst` 10 pomocí modulu `ngx_http_limit_req_module` (viz. B.1.5) oproti výchozí konfiguraci reverzního proxy serveru Nginx.

Parametry použité pro útok lze vidět v Tabulce 6.5:

Typ útoku	`Slow POST Denial of Service`
Varianta útoku	kontrolovaná
Cíl útoku	`http://192.168.237.130/`
Počet připojení (soketů)	15 000
Počet připojení za sekundu	250
Prodleva mezi požadavky [s]	10
Velikost Content-Length hlavičky [B]	4 096
Velikost posílaných dat [B]	1
Doba trvání experimentu [s]	210
Časový limit pro `probe` připojení [s]	3

Tabulka 6.5: Parametry nástroje použité pro testování kontrolovaného `Slow POST Denial of Service` útoku

Konfigurace modulu `ngx_http_limit_conn_module`:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;
    limit_conn_status 429;
    server {
        location / {
            limit_conn addr 30;
        }
    }
}
```

Výpis 6.8: Použitá konfigurace pro experiment 5 - ngx_http_limit_conn_module

Konfigurace modulu `ngx_http_limit_req_module`:

```
http {
    limit_req_zone $binary_remote_addr zone=addr_limit:10m rate=5r/s;
    limit_req_status 429;
    server {
        location / {
            limit_req zone=addr_limit burst=10 nodelay;
        }
    }
}
```

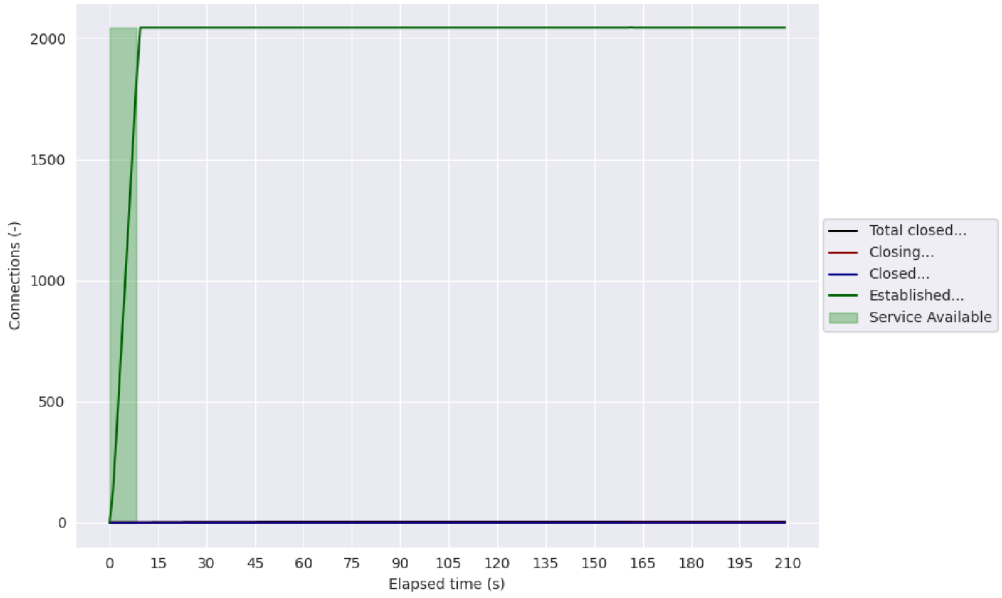
Výpis 6.9: Použitá konfigurace pro experiment 5 - ngx_http_limit_req_module

Výsledky testování

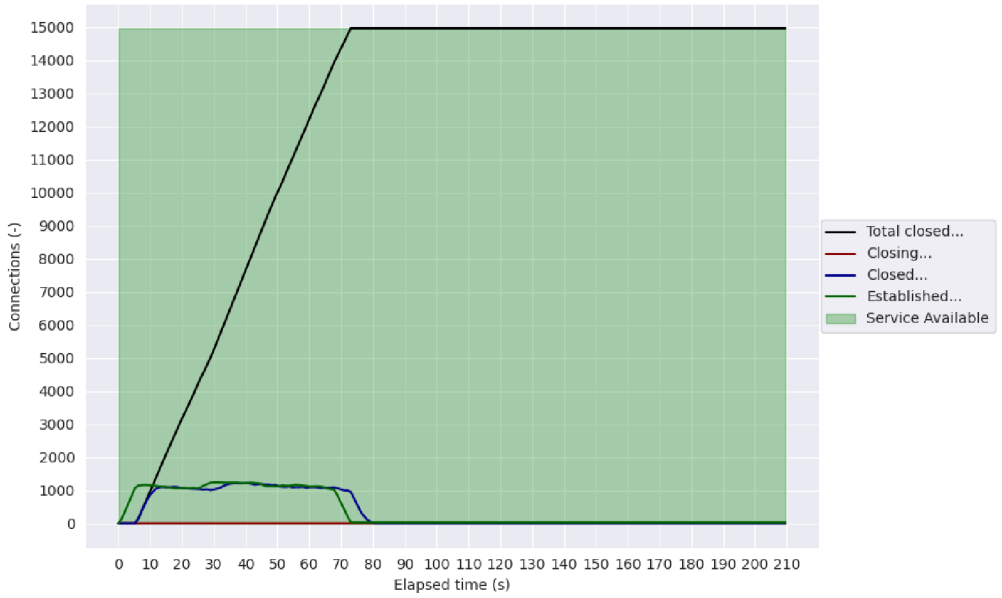
Kontrolovaný `Slow POST Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.17. Ve chvíli, kdy počet připojení přesáhl 2 000 aktivních připojení se server stal nedostupným a nepřijímal další připojení. Server zůstal nedostupný po celou dobu útoku ze stejného důvodu, jako v experimentu 3 (viz. 6.3.3).

Po omezení počtu připojení, které IP adresa může vytvořit pomocí modulu `ngx_http_limit_conn_module` byl útok neúspěšný, jak lze vidět na Obrázku 6.18. Počet připojení je postupně omezen na 30 aktivních připojení, která zůstavají aktivní po celou dobu útoku a server zůstává dostupný.

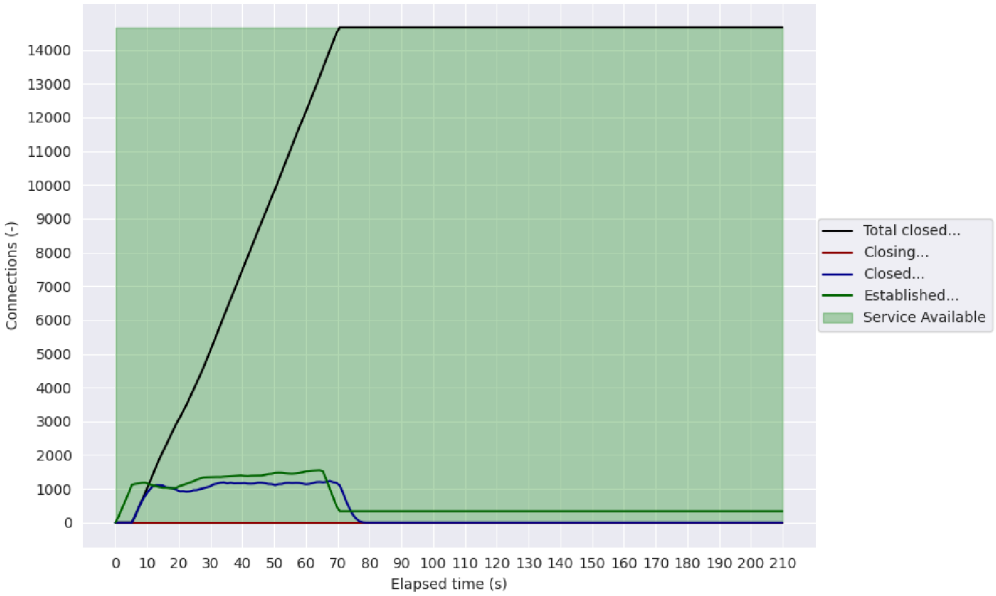
Útok byl neúspěšný i po omezení požadavků za sekundu, které IP adres může vytvořit pomocí modulu `ngx_http_limit_req_module`, jak lze vidět na Obrázku 6.19. Došlo k úspěšnému omezení příchozích požadavků, čímž bylo zabráněno zahlcení serveru.



Obrázek 6.17: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) ve výchozí konfiguraci.



Obrázek 6.18: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) po omezení počtu připojení na 30 připojení



Obrázek 6.19: Reverzní proxy server Nginx pod kontrolovaným `Slow POST Denial of Service` útokem (15 000 připojení) po omezení požadavků na 5 r/s s hodnotou `burst` 10.

6.3.6 Experiment 6

Tato simulace proběhla za použití rekurzivní varianty `Slow POST Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv omezení počtu připojení, které IP adresa může vytvořit na 30 připojení pomocí modulu `ngx_http_limit_conn_module` (viz. B.1.3) a omezení provozu na 5 požadavků za sekundu s hodnotou `burst` 10 pomocí modulu `ngx_http_limit_req_module` (viz. B.1.5) oproti výchozí konfiguraci reverzního proxy serveru Nginx.

Parametry použité pro útok lze vidět v Tabulce 6.6:

Typ útoku	`Slow POST Denial of Service`
Varianta útoku	rekurzivní
Cíl útoku	`http://192.168.237.130/`
Počet připojení (soketů)	7 000
Počet připojení za sekundu	250
Prodleva mezi požadavky [s]	10
Hodnota Content-Length hlavičky	4 096
Velikost posílaných dat [B]	1
Doba trvání experimentu [s]	210

Tabulka 6.6: Parametry nástroje použité pro testování rekurzivního `Slow POST Denial of Service` útoku

Konfigurace modulu `ngx_http_limit_conn_module`:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;
```

```
    limit_conn_status 429;
    server {
        location / {
            limit_conn addr 30;
        }
    }
}
```

Výpis 6.10: Použitá konfigurace pro experiment 6 - ngx_http_limit_conn_module

Konfigurace modulu `ngx_http_limit_req_module`:

```
http {
    limit_req_zone $binary_remote_addr zone=addr_limit:10m rate=5r/s;
    limit_req_status 429;
    server {
        location / {
            limit_req zone=addr_limit burst=10 nodelay;
        }
    }
}
```

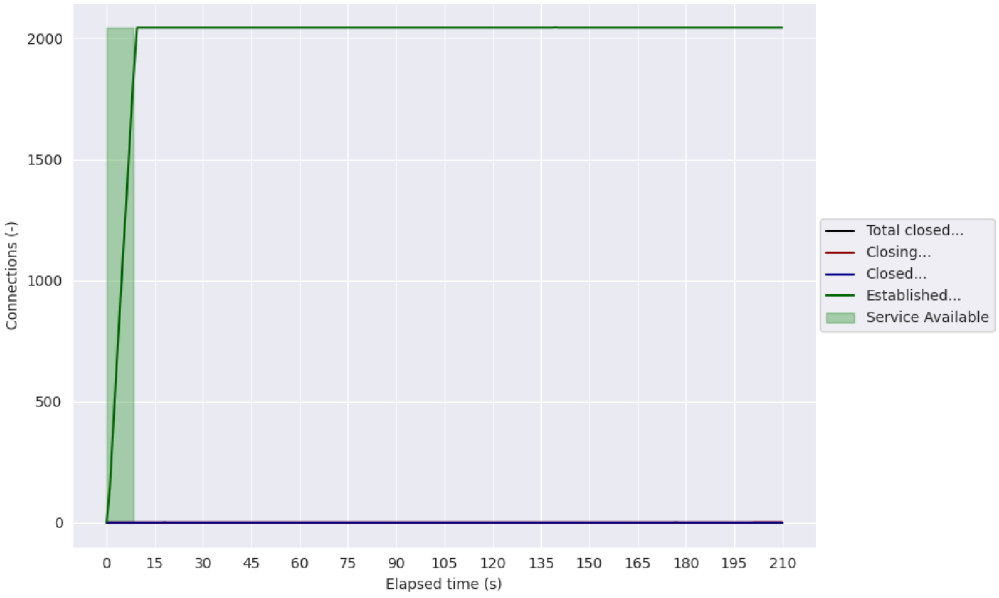
Výpis 6.11: Použitá konfigurace pro experiment 6 - ngx_http_limit_req_module

Výsledky testování

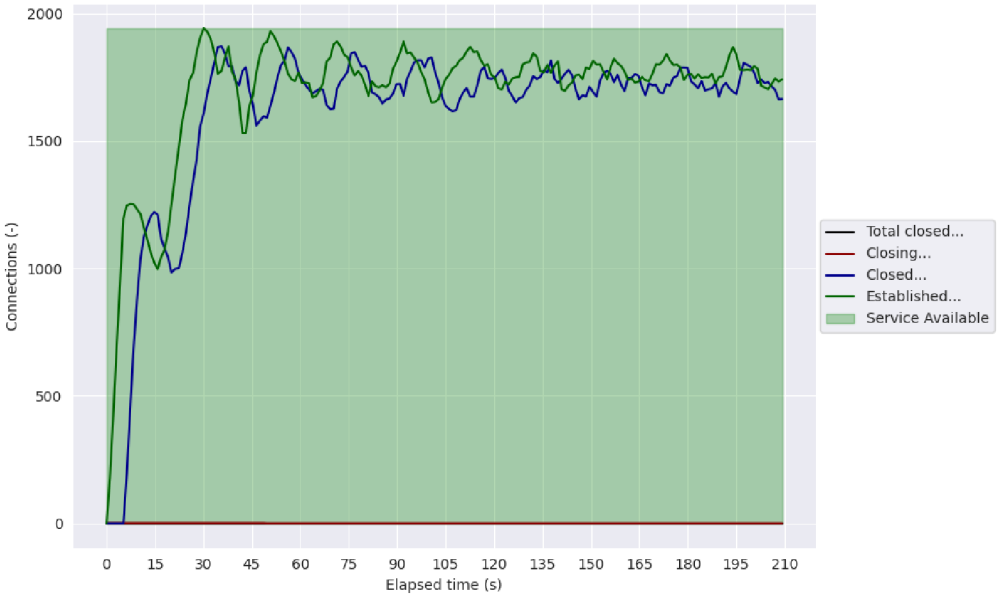
Rekurzivní `Slow POST Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.20. Ve chvíli, kdy počet připojení přesáhl 2 000 aktivních připojení se server stal nedostupným a nepřijímal další připojení. Server zůstal nedostupný po celou dobu útoku ze stejného důvodu, jako v experimentu 3 (viz. 6.3.3).

Po omezení počtu připojení, které IP adres může vytvořit pomocí modulu `ngx_http_limit_conn_module` byl útok neúspěšný, jak lze vidět na Obrázku 6.21. Aktivní připojení, která jsou nad limit počtu připojení, které IP adres může vytvořit jsou postupně uzavírána a opětovně vytvářena v průběhu útoku. Server zůstal po celou dobu útoku dostupným.

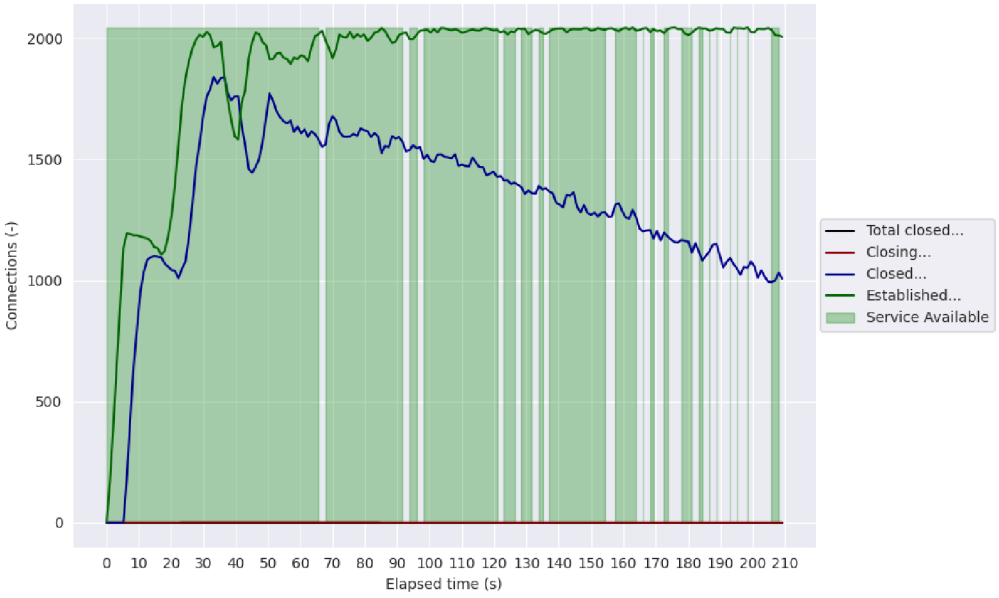
Po zavedení modulu `ngx_http_limit_req_module` se stal útok úspěšným narozdíl od předchozího kontrolovaného útoku na konfiguraci s modulem `ngx_http_limit_req_module` (viz. 6.3.5). Příchozí požadavky se postupně uzavíraly a opětovně vytvářeli, čímž postupem času útoku došlo k krátkým výpadkům dostupnosti serveru.



Obrázek 6.20: Reverzní proxy server Nginx pod rekurzivním `Slow POST Denial of Service` útokem (7 000 připojení) ve výchozí konfiguraci.



Obrázek 6.21: Reverzní proxy server Nginx pod rekurzivním `Slow POST Denial of Service` útokem (7 000 připojení) po omezení počtu připojení na 30 připojení



Obrázek 6.22: Reverzní proxy server Nginx pod rekurzivním `Slow POST Denial of Service` útokem (7 000 připojení) po omezení požadavků na 5 r/s s hodnotou `burst` 10.

6.3.7 Experiment 7

Tato simulace proběhla za použití kontrolované varianty `Slow READ Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv omezení počtu připojení na 30 připojení pomocí modulu `ngx_http_limit_conn_module` (viz. B.1.3) a omezení provozu na 5 požadavků za sekundu s hodnotou `burst` 10 pomocí modulu `ngx_http_limit_req_module` (viz. B.1.5) oproti výchozí konfiguraci reverzního proxy serveru Nginx.

Parametry použité pro útok lze vidět v Tabulce 6.7:

Typ útoku	`Slow READ Denial of Service`
Varianta útoku	kontrolovaná
Cíl útoku	`http://192.168.237.130/img.jpg`
Počet připojení (soketů)	7 000
Počet připojení za sekundu	500
Prodleva mezi požadavky [s]	6
Velikost okna pro příjem dat [B]	512-768
Rychlost čtení dat	24B/6s
Doba trvání experimentu [s]	210
Časový limit pro `probe` připojení [s]	3

Tabulka 6.7: Parametry nástroje použité pro testování rekurzivního `Slow READ Denial of Service` útoku

Konfigurace modulu `ngx_http_limit_conn_module`:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;
    limit_conn_status 429;
    server {
        location / {
            limit_conn addr 30;
        }
    }
}
```

Výpis 6.12: Použitá konfigurace pro experiment 7 - ngx_http_limit_conn_module

Konfigurace modulu `ngx_http_limit_req_module`:

```
http {
    limit_req_zone $binary_remote_addr zone=addr_limit:10m rate=5r/s;
    limit_req_status 429;
    server {
        location / {
            limit_req zone=addr_limit burst=10 nodelay;
        }
    }
}
```

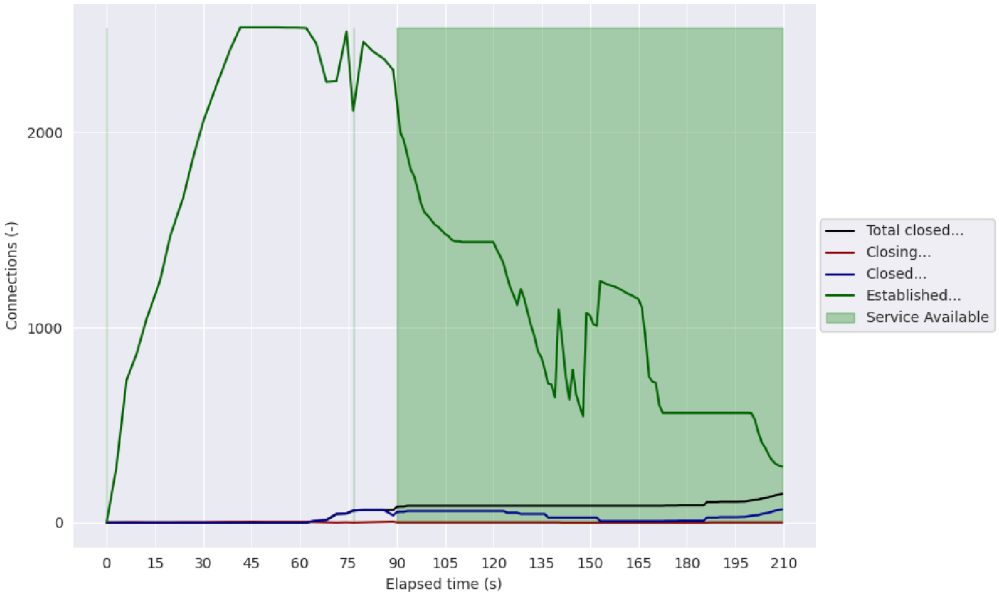
Výpis 6.13: Použitá konfigurace pro experiment 7 - ngx_http_limit_req_module

Výsledky testování

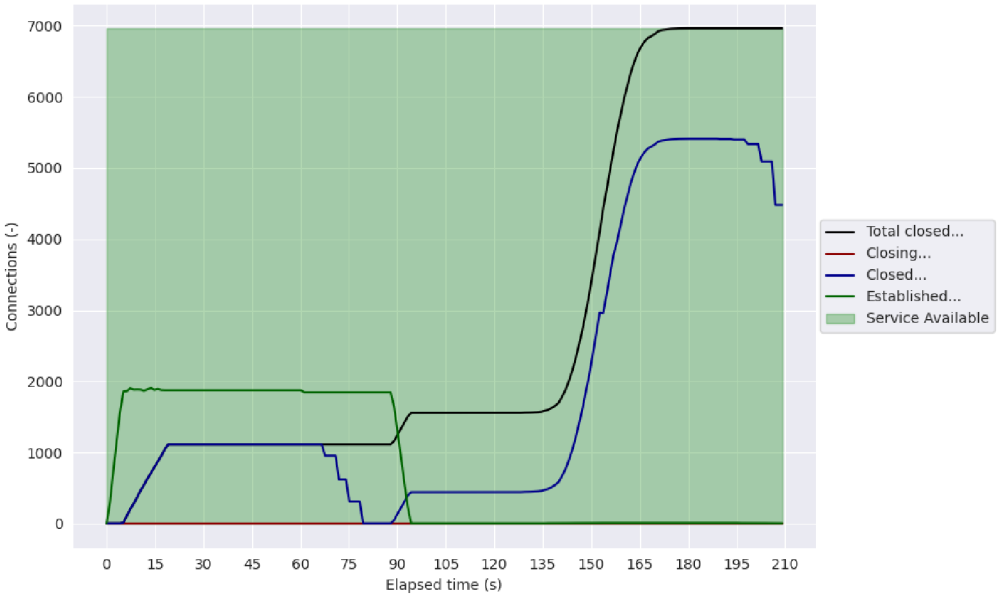
Kontrolovaný `Slow READ Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.23. Server se stal nedostupným po začátku útoku. Ve chvíli, kdy server začal postupně uzavírat škodlivá připojení se opět stal dostupným.

Po omezení připojení, které IP adresa může vytvořit pomocí modulu `ngx_http_limit_conn_module` byl útok neúspěšný, jak lze vidět na Obrázku 6.24. Počet připojení je postupně omezen na 30 aktivních připojení, která zůstávají aktivní po celou dobu útoku a server zůstává dostupný. Přebytečné připojení jsou po omezení počtu připojení ihned uzavírány.

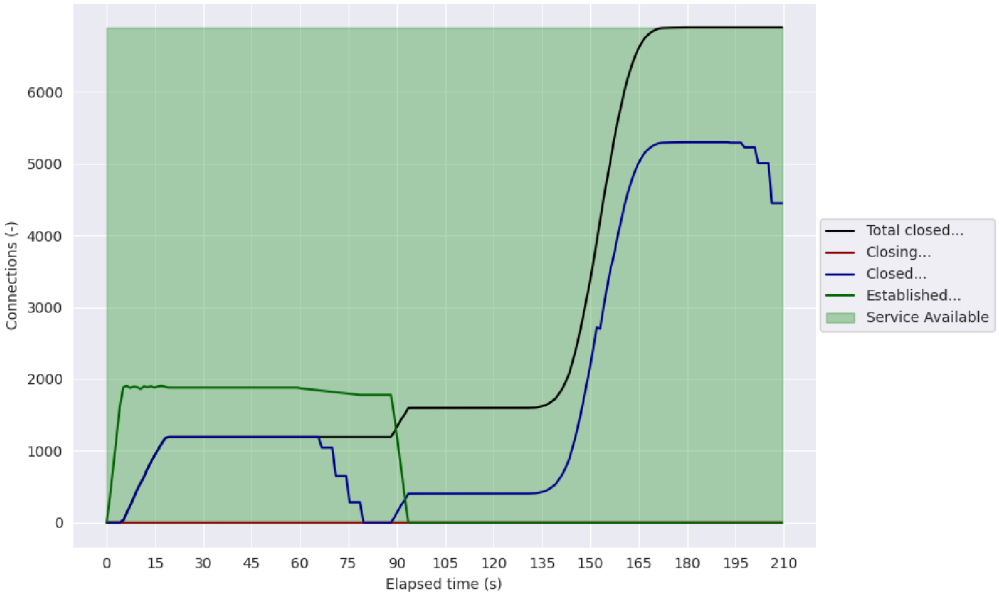
Útok byl neúspěšný i po omezení požadavků za sekundu, které IP adres může vytvořit pomocí modulu `ngx_http_limit_req_module`, jak lze vidět na Obrázku 6.25. Došlo k úspěšnému omezení příchozích požadavků, čímž bylo zabráněno zahlcení serveru. Přebytečné požadavky jsou po omezení ihned uzavírány.



Obrázek 6.23: Reverzní proxy server Nginx pod kontrolovaným `Slow READ Denial of Service` útokem (7 000 připojení) ve výchozí konfiguraci.



Obrázek 6.24: Reverzní proxy server Nginx pod kontrolovaným `Slow READ Denial of Service` útokem (7 000 připojení) po omezení počtu připojení na 30 připojení



Obrázek 6.25: Reverzní proxy server Nginx pod kontrolovaným `Slow READ Denial of Service` útokem (7 000 připojení) po omezení požadavků na 5 r/s s hodnotou `burst` 10.

6.3.8 Experiment 8

Tato simulace proběhla za použití rekurzivní varianty `Slow READ Denial of Service` útoku (viz. 3.4.2). Je porovnán vliv omezení počtu připojení, které IP adresa může vytvořit na 30 připojení pomocí modulu `ngx_http_limit_conn_module` (viz. B.1.3) a omezení provozu na 5 požadavků za sekundu s hodnotou `burst` 10 pomocí modulu `ngx_http_limit_req_module` (viz. B.1.5) oproti výchozí konfiguraci reverzního proxy serveru Nginx.

Parametry použité pro útok lze vidět v Tabulce 6.8:

Typ útoku	`Slow READ Denial of Service`
Varianta útoku	rekurzivní
Cíl útoku	`http://192.168.237.130/img.jpg`
Počet připojení (soketů)	7 000
Počet připojení za sekundu	500
Prodleva mezi požadavky [s]	6
Velikost okna pro příjem dat [B]	512-768
Rychlost čtení dat	24B/6s
Doba trvání experimentu [s]	210

Tabulka 6.8: Parametry nástroje použité pro testování rekurzivního `Slow READ Denial of Service` útoku

Konfigurace modulu `ngx_http_limit_conn_module`:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;
    limit_conn_status 429;
    server {
        location / {
            limit_conn addr 30;
        }
    }
}
```

Výpis 6.14: Použitá konfigurace pro experiment 8 - ngx_http_limit_conn_module

Konfigurace modulu `ngx_http_limit_req_module`:

```
http {
    limit_req_zone $binary_remote_addr zone=addr_limit:10m rate=5r/s;
    limit_req_status 429;
    server {
        location / {
            limit_req zone=addr_limit burst=10 nodelay;
        }
    }
}
```

Výpis 6.15: Použitá konfigurace pro experiment 8 - ngx_http_limit_req_module

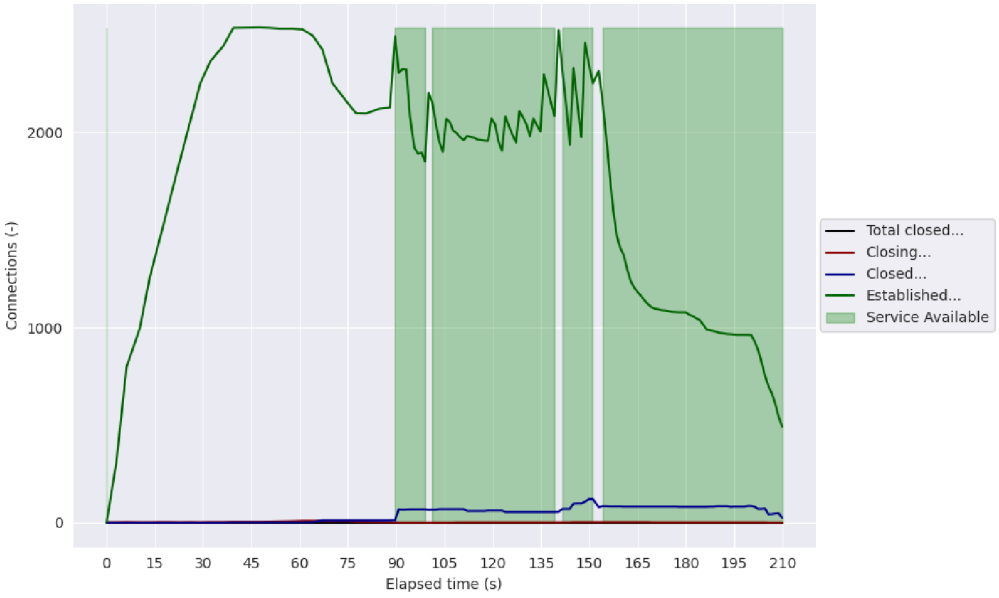
Výsledky testování

Rekurzivní `Slow READ Denial of Service` útok na reverzní proxy server Nginx byl úspěšný proti výchozí konfiguraci, jak lze vidět na Obrázku 6.26. Server se stal nedostupným po začátku útoku. Ve chvíli, kdy server začal postupně uzavírat škodlivá připojení se opět stal dostupným. Uzavřená připojení byla opětovně vytvářena, ale bylo dosaženo pouze krátkých momentů, kdy server byl opět nedostupný.

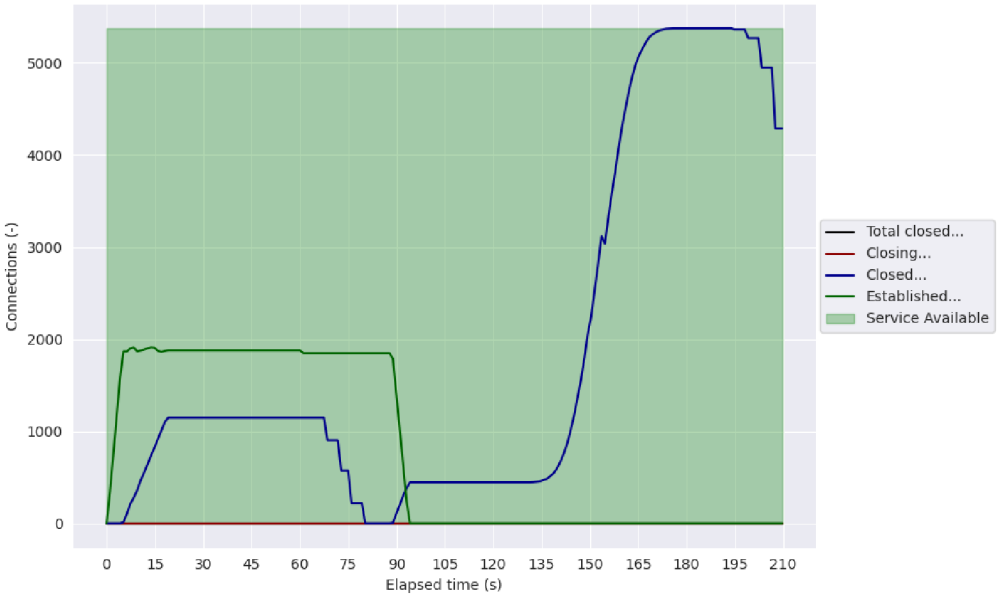
Po omezení připojení, které IP adresa může vytvořit pomocí modulu `ngx_http_limit_conn_module` byl útok neúspěšný, jak lze vidět na Obrázku 6.27. Počet připojení je postupně omezen na 30 aktivních připojení, která zůstávají aktivní po celou dobu útoku a server zůstává dostupný. Přebytečná připojení jsou po omezení počtu připojení ihned uzavírána.

Útok byl neúspěšný i po omezení požadavků za sekundu, které IP adres může vytvořit pomocí modulu `ngx_http_limit_req_module`, jak lze vidět na Obrázku 6.28. Došlo k úspěšnému omezení příchozích požadavků, čímž bylo zabráněno zahlcení serveru. Přebytečné požadavky jsou po omezení ihned uzavírány.

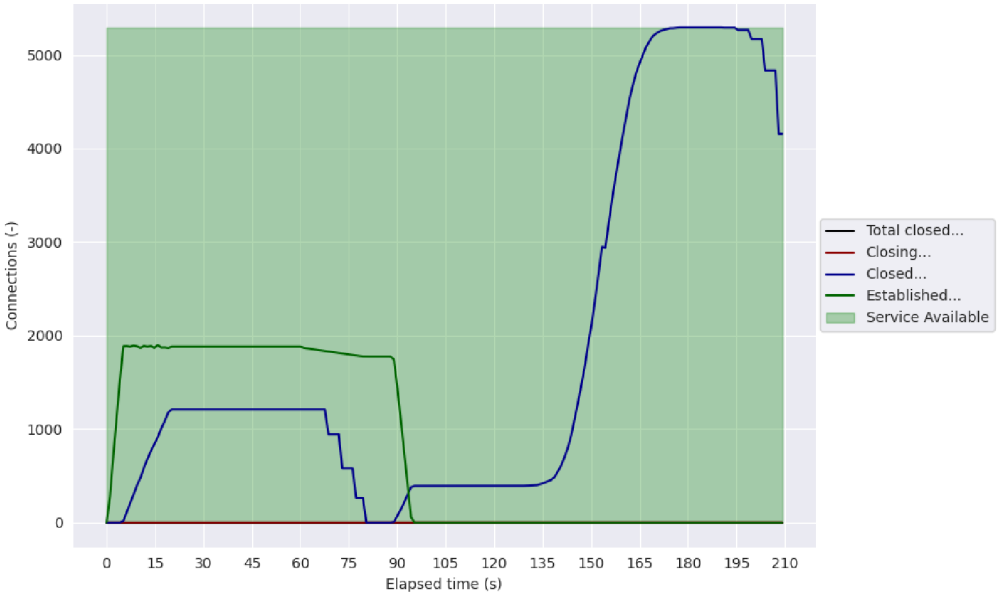
Průběh experimentu po zavedení direktiv měl stejný průběh, jako předchozí experiment provedený pomocí kontrolované varianty útoku `Slow READ Denial of Service`.



Obrázek 6.26: Reverzní proxy server Nginx pod rekurzivním `Slow READ Denial of Service` útokem (7 000 připojení) ve výchozí konfiguraci.



Obrázek 6.27: Reverzní proxy server Nginx pod rekurzivním `Slow READ Denial of Service` útokem (7 000 připojení) po omezení počtu připojení na 30 připojení



Obrázek 6.28: Reverzní proxy server Nginx pod rekurzivním `Slow READ Denial of Service` útokem (7 000 připojení) po omezení požadavků na 5 r/s s hodnotou `burst` 10.

6.4 Tabulka úspěšnosti útoků

Následující tabulka znázorňuje úspěšnost provedených útoků. Hodnoty v tabulce udávají zaokrouhlenou celkovou dobu, po kterou byl server nedostupný během útoku.

Typ útoku:	`Slow Header`		`Slow Post`		`Slow READ`	
Varianta (kontrolovaná/rekurzivní):	kon.	rek.	kon.	rek.	kon.	rek.
Počet připojení:	15 000	7 000	15 000	7 000	7 000	7 000
Doba trvání experimentu:	210	210	210	210	210	210
Výchozí konfigurace	52 s	189 s	201 s	201 s	87 s	90 s
`client_header_timeout 5s;`	0 s	0 s	-	-	-	-
`client_header_timeout 10s;`	19 s	98 s	-	-	-	-
`client_max_body_size 3k;`	-	-	0 s	0 s	-	-
`client_max_body_size 4k;`	-	-	201 s	-	-	-
`client_body_timeout 5s;`	-	-	0 s	0 s	-	-
`client_body_timeout 10s;`	-	-	38 s	-	-	-
Omezení počtu připojení (30 připojení)	52 s	-	0 s	0 s	0 s	0 s
Omezení provozu (5 r/s, 10 burst)	-	-	0 s	35 s	0 s	0 s

Tabulka 6.9: Úspěšnost útoků proti použitým direktivám.

Kapitola 7

Analýza výsledků a diskuze

V této kapitole jsou zhodnoceny výsledky experimentů z předchozí kapitoly 6.

7.1 Dopad útoků na reverzní proxy server Nginx

V experimentu 1 a 2 byla předvedena direktiva `client_header_timeout` pro omezení `Slow Header Denial of Service` útoku. Vzhledem k tomu, že tento útok cílí na vyčerpání zdrojů serveru pomalým odesíláním hlavičky požadavku, tak nastavením správného časového limitu pro příjem hlavičky požadavku se tento útok stal neúčinným vůči reverznímu proxy serveru Nginx. Vzhledem k povaze této direktivy byla použita pouze pro `Slow Header Denial of Service` útok, protože ostatní útoky nevyužívají zranitelností v hlavičce požadavku. Příliš nízké nastavení této direktivy by mohlo mít negativní dopad na legitimní uživatele s pomalým připojením.

V experimentu 1, 5, 6 a 7 byl předveden modul `ngx_http_limit_conn_module` sloužící k omezení počtu připojení, které jedna IP adresa může vytvořit. Tento modul byl využit k omezení `Slow Denial of Service` útoků. Omezení počtu připojení je základní ochranou proti `Slow Denial of Service` útokům, protože tyto útoky jsou založeny na vytvoření co nejvíce souběžných připojení. Nalezením správného limitu omezení počtu připojení poskytne serveru schopnost kompletně znemožnit, či zmírní vliv `Slow Denial of Service` útoku. Použití modulu `ngx_http_limit_conn_module` se projevilo úspěšným proti `Slow POST` a `Slow READ Denial of Service` útoku. Experiment 1 sloužil jako ukázka, že `ngx_http_limit_conn_module` modul nelze použít proti `Slow Header Denial of Service` útoku. Důvodem je, že spojení je započítáno do limitu omezení pouze v případě, že celá hlavička požadavku byla načtena a požadavek je zpracovaný serverem. To znamená, že `Slow Header Denial of Service` útok do tohoto limitu nikdy nebude započítán. V případě, že útočník využije více IP adres pro útok, může limity nastavené touto direktivou obejít.

V experimentu 3 a 4 byly předvedeny direktivy `client_body_timeout` a `client_max_body_size`, které byly použity proti útoku `Slow POST Denial of Service`. Vzhledem k tomu, že tento útok cílí podobně jako již zmíněný `Slow Header Denial of Service` útok na vyčerpání zdrojů serveru pomalým odesíláním těla požadavku. Z tohoto důvodu po nastavení správného limitu pro čtení těla požadavku pomocí direktivy `client_body_timeout` se tento útok stal neúčinným vůči reverznímu proxy serveru Nginx. `Slow POST Denial of Service` zneužívá pole `Content-Length` hlavičky HTTP zadáním větší hodnoty do pole `Content-Length`, než je skutečná velikost těla zprávy. Pomocí di-

rektivy `client_max_body_size` byla zavedena maximální povolená velikost těla požadavku klienta. Tímto se stal útok neúčinným vůči reverznímu proxy serveru Nginx, protože server vrátí klientovi chybu v případě překročení této velikosti. Nginx kontroluje velikost pole `Content-Length` v hlavičce požadavku a porovná tuto velikost s direktivou `client_max_body_size`. Příliš nízké nastavení direktivy `client_body_timeout` by mohlo mít negativní dopad na legitimní uživatele s pomalým připojením. Nízká hodnota direktivy `client_max_body_size` může představovat problém v případě, že je nutné podporovat odeslání větších souborů.

V experimentu 5, 6, 7 a 8 byl předveden modul `ngx_http_limit_req_module`, který slouží k omezení rychlosti zpracovávání požadavků, které jedna IP adreas může vytvořit. Tento modul byl využit k omezení `Slow POST` a `Slow READ Denial of Service` útoků. Omezení rychlosti zpracovávání požadavků je velmi důležité pro obranu proti `Slow Denial of Service` útokům, protože jsou založeny na co nejvíce souběžných připojení. Pokud tato souběžná připojení pochází ze stejné IP adresy, tak modul `ngx_http_limit_req_module` velmi omezí rozsah `Slow Denial of Service` útoku, protože útočník bude moci posílat pouze omezený počet požadavků za sekundu. Nalezením správného limitu omezení rychlosti zpracovávání požadavků poskytne serveru schopnost kompletně znemožnit, či zmírní vliv `Slow Denial of Service` útoku. V případě, že útočník využije více IP adres pro útok, může limity nastavené touto direktivou obejít.

Dodatečný rekurzivní útok, který byl proved na direktivy, které úspěšně podstoupili první kontrolovaný útok složil pro dodatečný stresový test, jak již bylo zmíněno v kapitole 6

7.2 Diskuze

Provedené experimenty vůči reverznímu proxy serveru Nginx dokazují, že i s minimální konfigurací je schopný se vypořádat s robustními `Slow Denial of Service` útoky. Bylo dokázáno, že v případě velmi špatné konfigurace reverzního proxy serveru Nginx je tento server náchylný `Slow Denial of Service` útokům.

Možným rozšířením práce je přidání více útoků a funkcí do vytvořeného nástroje, jako je například `Man-in-the-middle` útok. Dalším možným rozšířením práce je sbírání dat, které by interpretovala zdroje (CPU, paměť, doba odezvy) serveru v průběhu `Slow Denial of Service` útoků. Experimenty by jako další rožšíření práce mohly být otestovány ve více reálném prostředí, aby se zajistlo, že konfigurace poskytuje správné řešení pro ochranu proti `Slow Denial of Service` útokům. Dalším možným rozšířením práce je simulace útoku na aplikaci s legitimními klienty, což by ukázalo vliv útoku a použitých direktiv na skutečný provoz.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo nastudovat problematiku nejčastějších hrozeb pro reverzní proxy servery se zaměřením na `Slow Denial of Service` útoky a implementovat nástroj pro demonstraci bezpečnostních útoků, který sloužil pro testování zranitelostí reverzního proxy serveru Nginx ve vytvořeném testovacím prostředí.

Na základě experimentů následně proběhla analýza, která sloužila jako podklad pro vytvoření technické specifikace, která je vytvořena za účelem pomáhat specialistům ve společnosti Kyndryl s nastavením reverzního proxy serveru Nginx. Úvodní kapitola obsahuje seznámení s cílem práce. V následující druhé kapitole 2 proběhlo prostudování webového serveru Nginx a jeho konfigurací. Webový server Nginx byl porovnán s dostupnými alternativami. Třetí kapitola byla zaměřena na proces penetračního testování a možné bezpečnostní hrozby pro proxy servery včetně `Slow Denial of Service` útoků. Následně byly prostudovány existující nástroje pro detekci bezpečnostních slabin reverzních proxy serverů. Kapitola čtyři a pět byla věnována návrhu a implementaci nástroje na základě požadavků společnosti Kyndryl, který slouží pro demonstraci možných slabin v konfiguraci serveru Nginx. Nástroj byl otestován v prostředí vytvořeném pomocí VMWare Workstation PRO. Pro testování byl využitý operační systém Linux, reverzní proxy server Nginx a webový server Tomcat pro nasazení webové stránky. Kapitola šest popisuje průběh testování a provedené experimenty, které slouží jako podklad pro vytvořenou technickou specifikaci pro společnost Kyndryl. Provedeny byly útoky typu `Slow Denial of Service`, jejichž průběh byl monitorován a interpretován pomocí grafů. Na základě experimentů proběhla analýza výsledků a diskuze dalších možných rozšíření práce v kapitole sedm. Příloha B poskytuje kompletní technickou specifikaci pro server Nginx, kde jsou popsány možné konfigurace.

Literatura

[1] Acunetix. *Directory Traversal*. 2023 [cit. 2023-12-21]. Dostupné z: https://www.acunetix.com/websitesecurity/directory-traversal/.

[2] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M. et al. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, s. 5–17. ISBN 9781450338325.

[3] Aivaliotis, D. *Mastering NGINX*. Packt Publishing, 2016. ISBN 9781785283765. Dostupné z: https://books.google.cz/books?id=jM2qDQAAQBAJ.

[4] Art. *C10K Problem: Understanding and Overcoming the 10,000 Concurrent Connections Challenge*. 2023 [cit. 2023-12-22]. Dostupné z: https://webhostinggeeks.com/blog/c10k-problem-understanding-and-overcoming-the-10000-concurrent-connections-challenge/.

[5] Banach, Z. *How the BEAST Attack Works*. 2024 [cit. 2023-04-22]. Dostupné z: https://www.invicti.com/blog/web-security/how-the-beast-attack-works/.

[6] Beale, J., Meer, H., Walt, C. van der a Deraison, R. *Nessus Network Auditing: Jay Beale Open Source Security Series*. Elsevier Science, 2004. ISBN 9780080479620. Dostupné z: https://books.google.cz/books?id=gUKveBFIP6wC.

[7] Blog, M. T. *Finding the Nginx gzip_comp_level Sweet Spot*. 2015 [cit. 2023-04-22]. Dostupné z: https://mjanja.ch/2015/03/finding-the-nginx-gzip_comp_level-sweet-spot/.

[8] BrightSec. *Penetration Testing Types*. 2022 [cit. 2023-12-21]. Dostupné z: https://brightsec.com/blog/penetration-testing-types/.

[9] BrightSec. *Security Misconfiguration*. 2022 [cit. 2023-12-21]. Dostupné z: https://brightsec.com/blog/security-misconfiguration/.

[10] Cloudflare. *Cipher Suite Recommendations*. 2024 [cit. 2023-12-23]. Dostupné z: https://developers.cloudflare.com/ssl/reference/cipher-suites/recommendations/.

[11] DeJonghe, D. *NGINX Cookbook*. O'Reilly Media, 2024. ISBN 9781098158392. Dostupné z: https://books.google.cz/books?id=2TjxEAAAQBAJ.

[12] Documentation, N. *Ngx_http_limit_conn_module documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_limit_conn_module.html.

[13] DreamHost. *The most important steps to take to make an Nginx server more secure*. 2024 [cit. 2023-12-22]. Dostupné z: https://help.dreamhost.com/hc/en-us/articles/222784068-The-most-important-steps-to-take-to-make-an-nginx-server-more-secure.

[14] Dwendlan. *Nessus - The World's Most Popular Vulnerability Scanner*. 2024 [cit. 2023-12-22]. Dostupné z: https://www.cs.cmu.edu/~dwendlan/personal/nessus.html.

[15] Elie Steinbock. *Nginx worker_rlimit_nofile*. 2019 [cit. 2023-04-22]. Dostupné z: https://stackoverflow.com/questions/37591784/nginx-worker-rlimit-nofile.

[16] Elleithy, K. M., Blagovic, D., Cheng, W. K. a Sideleau, P. Denial of service attack techniques: analysis, implementation and comparison. 2005.

[17] Fjordvald, M. a Nedelcu, C. *Nginx HTTP Server: Harness the power of Nginx to make the most of your infrastructure and serve pages faster than ever before, 4th Edition*. Packt Publishing, 2018. ISBN 9781788621977. Dostupné z: https://books.google.cz/books?id=CJdMDwAAQBAJ.

[18] Foundation, T. A. S. *Mod_proxy – Apache HTTP Server Version 2.4*. 2024 [cit. 2023-12-20]. Dostupné z: https://httpd.apache.org/docs/2.4/mod/mod_proxy.html.

[19] Hasan, M. a Rahman, M. M. Minimize Web Applications vulnerabilities through the early Detection of CRLF Injection. *ArXiv preprint arXiv:2303.02567*. 2023.

[20] Helme, S. *Content Security Policy Cheat Sheet*. 2024 [cit. 2023-04-26]. Dostupné z: https://scotthelme.co.uk/csp-cheat-sheet/.

[21] Heroku. *Increasing Application Performance with HTTP Cache Headers*. 2023 [cit. 2023-04-01]. Dostupné z: https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers.

[22] IBM. *How to configure IBM HTTP Server as a reverse proxy for Rational DOORS Web Access*. 2020 [cit. 2023-12-20]. Dostupné z: https://www.ibm.com/support/pages/how-configure-ibm-http-server-reverse-proxy-rational-doors-web-access.

[23] Inc., N. *Now World's #1 Web Server, NGINX Looks Forward to Even Brighter Future*. 2022 [cit. 2023-04-29]. Dostupné z: https://www.nginx.com/blog/now-worlds-1-web-server-nginx-looks-forward-to-even-brighter-future/.

[24] Jorgensen, P. *Software Testing: A Craftsman's Approach, Third Edition*. CRC Press, 2013 [cit. 2023-12-22]. ISBN 9781439889503. Dostupné z: https://books.google.cz/books?id=UhTSBQAAQBAJ.

[25] Kennedy, D., O'Gorman, J., Kearns, D. a Aharoni, M. *Metasploit: The Penetration Tester's Guide*. No Starch Press, 2011 [cit. 2023-12-21]. ISBN 9781593274023. Dostupné z: https://books.google.cz/books?id=T9HKgEOCYZEC.

[26] Lau, F., Rubin, S. H., Smith, M. H. a Trajkovic, L. Distributed denial of service attacks. In: IEEE. *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0.* 2000, sv. 3, s. 2275–2280.

[27] Linuxize. *Nginx Reverse Proxy*. 2019 [cit. 2023-04-22]. Dostupné z: https://linuxize.com/post/nginx-reverse-proxy/.

[28] LLC, I. *Nmap - Free Security Scanner for Network Exploration & Security Audits*. 2023 [cit. 2023-12-22]. Dostupné z: https://nmap.org.

[29] Microsoft. *Application Request Routing*. 2023 [cit. 2023-12-20]. Dostupné z: https://www.iis.net/downloads/microsoft/application-request-routing.

[30] Miller, E. *NGINX Module Writing Guide*. 2024 [cit. 2023-01-5]. Dostupné z: https://www.evanmiller.org/nginx-modules-guide.html.

[31] Mozilla. *SSL Configuration Generator*. 2024 [cit. 2023-12-23]. Dostupné z: https://ssl-config.mozilla.org/.

[32] Mozilla Developer Network. *Content Security Policy*. 2024 [cit. 2023-04-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy.

[33] Mozilla Developer Network. *Referrer-Policy*. 2024 [cit. 2023-04-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy.

[34] Mozilla Developer Network (MDN). *Permissions-Policy*. 2023 [cit. 2023-12-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy.

[35] Mozilla Developer Network (MDN). *X-Content-Type-Options*. 2023 [cit. 2023-12-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options.

[36] Mozilla Developer Network (MDN). *X-Frame-Options*. 2023 [cit. 2023-12-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options.

[37] Mozilla Developer Network (MDN). *X-XSS-Protection*. 2023 [cit. 2023-12-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection.

[38] Murashka, U. *W3af - Web Application Attack and Audit Framework*. 2016 [cit. 2023-12-22]. Dostupné z: https://www.scanforsecurity.com/vulnerability-scanners/w3af.html.

[39] Najera Gutierrez, G. a Ansari, J. *Web Penetration Testing with Kali Linux: Explore the methods and tools of ethical hacking with Kali Linux, 3rd Edition*. Packt Publishing, 2018. ISBN 9781788623803. Dostupné z: https://books.google.cz/books?id=yulODwAAQBAJ.

[40] Nelson, R. *Tuning NGINX*. 2014 [cit. 2023-04-22]. Dostupné z: https://www.nginx.com/blog/tuning-nginx/#worker_connections.

[41] Nginx. *Compression in NGINX Web Server*. 2024 [cit. 2023-04-22]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/web-server/compression/.

[42] NGINX. *Controlling Access to Proxied HTTP Resources*. 2024 [cit. 2023-03-30]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/security-controls/controlling-access-proxied-http/.

[43] NGINX. *Managing Configuration Files*. 2024 [cit. 2023-01-5]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/basic-functionality/managing-configuration-files/.

[44] NGINX. *Module ngx_http_proxy_module*. 2024 [cit. 2023-12-23]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_proxy_module.html.

[45] NGINX. *Module ngx_http_ssl_module*. 2024 [cit. 2023-12-22]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_ssl_module.html.

[46] NGINX. *Module ngx_http_upstream_module*. 2024 [cit. 2023-04-29]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_upstream_module.html.

[47] NGINX. *NGINX Core Module Documentation*. 2024 [cit. 2023-04-22]. Dostupné z: https://nginx.org/en/docs/ngx_core_module.html.

[48] NGINX. *NGINX Documentation*. 2024 [cit. 2023-03-30]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_core_module.html.

[49] NGINX. *NGINX ngx_http_gzip_module documentation*. 2024 [cit. 2023-04-22]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_gzip_module.html.

[50] NGINX. *Ngx_http_rewrite_module*. 2024 [cit. 2023-12-23]. Dostupné z: http://nginx.org/en/docs/http/ngx_http_rewrite_module.html.

[51] NGINX. *Rate Limiting in NGINX*. 2024 [cit. 2023-03-30]. Dostupné z: https://www.nginx.com/blog/rate-limiting-nginx/.

[52] NGINX. *Request Processing*. 2024 [cit. 2023-04-01]. Dostupné z: https://nginx.org/en/docs/http/request_processing.html.

[53] NGINX. *Reverse Proxy*. 2024 [cit. 2023-01-5]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/.

[54] NGINX, I. *NGINX Configuring HTTP Basic Authentication Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-http-basic-authentication/.

[55] NGINX, I. *NGINX Configuring Subrequest Authentication Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-subrequest-authentication/.

[56] NGINX, I. *NGINX Dynamic Modules - GeoIP Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/dynamic-modules/geoip/.

[57] NGINX, I. *NGINX Headers More Module Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://www.nginx.com/resources/wiki/modules/headers_more/.

[58] NGINX, I. *NGINX ngx_http_access_module Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_access_module.html.

[59] NGINX, I. *NGINX ngx_http_auth_basic_module Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_auth_basic_module.html.

[60] NGINX, I. *NGINX ngx_http_auth_request_module Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_auth_request_module.html.

[61] NGINX, I. *NGINX ngx_http_geoip_module Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_geoip_module.html.

[62] NGINX, I. *NGINX ngx_http_geo_module Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_geo_module.html.

[63] NGINX, I. *NGINX ngx_http_map_module Documentation*. 2024 [cit. 2023-04-28]. Dostupné z: https://nginx.org/en/docs/http/ngx_http_map_module.html.

[64] Nginx Documentation. *Configuring Proxy Response Headers*. 2024 [cit. 2023-12-26]. Dostupné z: https://docs.nginx.com/nginx-management-suite/acm/how-to/policies/proxy-response-headers/.

[65] Nidecki, T. A. *Hardening NGINX*. 2020 [cit. 2023-04-22]. Dostupné z: https://www.acunetix.com/blog/web-security-zone/hardening-nginx/.

[66] OWASP. *OWASP Secure Configuration Guide - Nginx*. 2017 [cit. 2023-04-22]. Dostupné z: https://wiki.owasp.org/index.php/SCG_WS_nginx.

[67] OWASP. *Session Hijacking Attack*. 2023 [cit. 2023-12-21]. Dostupné z: https://owasp.org/www-community/attacks/Session_hijacking_attack.

[68] Park, J., Iwai, K., Tanaka, H. a Kurokawa, T. Analysis of slow read DoS attack and countermeasures on web servers. *International Journal of Cyber-Security and Digital Forensics*. The Society of Digital Information and Wireless Communications (SDIWC). 2015, sv. 4, č. 2, s. 339–353.

[69] Pauli, J. *The Basics of Web Hacking: Tools and Techniques to Attack the Web*. Elsevier Science, 2013. ISBN 9780124166592. Dostupné z: https://books.google.cz/books?id=qFIJaYv5bI4C.

[70] Project lighttpd. *Mod_proxy - Lighttpd Wiki*. 2020 [cit. 2023-12-20]. Dostupné z: https://redmine.lighttpd.net/projects/lighttpd/wiki/Mod_proxy.

[71] Project, T. Z. *ZAP Documentation*. 2024 [cit. 2023-12-22]. Dostupné z: https://www.zaproxy.org/docs/.

[72] Redscan. *Types of Pen Testing: White Box, Black Box and Everything In Between*. 2023 [cit. 2023-12-22]. Dostupné z: https://www.redscan.com/news/types-of-pen-testing-white-box-black-box-and-everything-in-between/.

[73] Ristic, I. *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck, 2014. Computers / Security. ISBN 9781907117046. Dostupné z: https://books.google.cz/books?id=fQOLBAAAQBAJ.

[74] Soni, R. *Nginx: From Beginner to Pro*. Apress, 2016. ISBN 9781484216569. Dostupné z: https://books.google.cz/books?id=uQvpDAAAQBAJ.

[75] Stack Overflow. *Setting X-XSS-Protection with Nginx*. 2019 [cit. 2023-12-26]. Dostupné z: https://stackoverflow.com/a/57802070.

[76] Sullivan, B. a Liu, V. *Web Application Security, A Beginner's Guide*. McGraw Hill LLC, 2011. Beginner's Guide. ISBN 9780071776127. Dostupné z: https://books.google.cz/books?id=uQkLr5ZnSmUC.

[77] Sullo. *Nikto - Web Server Scanner*. 2023 [cit. 2023-12-24]. Dostupné z: https://github.com/sullo/nikto/wiki.

[78] Technologies, L. *LiteSpeed Documentation*. 2023 [cit. 2023-12-20]. Dostupné z: https://docs.litespeedtech.com.

[79] Tips, M. L. *Best Security Practices for Nginx Web Server*. 2021 [cit. 2023-04-22]. Dostupné z: https://mylinuxtips.info/linuxtipstutorials/best-security-practices-for-nginx-webserver/.

[80] Tripathi, N., Hubballi, N. a Singh, Y. How secure are web servers? An empirical study of slow HTTP DoS attacks and detection. In: IEEE. *2016 11th International Conference on Availability, Reliability and Security (ARES)*. 2016, s. 454–463.

[81] w3af. *W3af - Web Application Attack and Audit Framework*. 2014 [cit. 2023-12-22]. Dostupné z: https://w3af.org.

[82] Webdock. *Setting Cache-Control Headers for Common Content Types in NGINX and Apache*. 2022 [cit. 2023-04-01]. Dostupné z: https://webdock.io/en/docs/webdock-control-panel/optimizing-performance/setting-cache-control-headers-common-content-types-nginx-and-apache.

[83] Weidman, G. *Penetration Testing: A Hands-On Introduction to Hacking*. No Starch Press, 2014. ISBN 9781593275648. Dostupné z: https://books.google.cz/books?id=T_LlAwAAQBAJ.

Příloha A

Obsah paměťového média

V této příloze je popsán obsah odevzdaného paměťového média.

```
/
├── attacks/
│   ├── __init__.py ..... Tvorba vláken, soketů a probe připojení pro útoky
│   ├── crlf.py ..... Kontrola CRLF injekce
│   ├── dos.py ..... Útoky UDP, SYN, ICMP Flood
│   ├── slow_header.py ..... Slow HEADER denial of service útok
│   ├── slow_post.py ..... Slow POST denial of service útok
│   └── slow_read.py ..... Slow READ denial of service útok
├── doc ..... Zdrojové soubory pro LaTeX
├── monitoring/
│   ├── generate_graph.py ..... Generování grafu z csv souboru
│   ├── monitor_regular.py ..... Monitorování Flood útoků a interpretaci grafů
│   ├── monitor_slow.py ..... Monitorování Slow DoS útoků a interpretaci grafů
│   └── server_monitor.py ... Spustitelný skript pro monitorování mimo hlavní nástroj
├── reports/
│   ├── scanner.txt ..... Výstup logování - txt
│   ├── scanner_csv.csv ..... Výstup logování - csv
│   └── scanner_data.json ..... Výstup logování - json
├── results/ ..... Výsledky jednotlivých experimentů
├── utils/
│   ├── attack_utils.py ..... Zahájení útoku a monitorování dostupnosti služby
│   ├── http_utils.py ..... Tvorba HTTP hlaviček a zkouška dostupnosti serveru
│   ├── logger.py ..... Logování
│   ├── nginx_port_scan.py ..... Skenování portů
│   └── user_agent.txt ... Textový soubor obsahující User-Agent pro HTTP požadavek
├── config.yaml ..... Konfigurační soubor
├── configuration.py ..... Zpracování konfigurace
├── nginx_scanner.py ..... Spustitelný program
├── README.md ..... Popis a spuštění nástroje
├── requirements.txt ..... Knihovny potřebné pro spuštění nástroje
├── testing.xlsx ..... Přehled provedených experimentů
└── xwagne12_thesis.pdf ..... Text bakalářské práce
```

Nástroj lze spustit následujícím způsobem: `python3 nginx_scanner.py` Chování nástroje je možné měnit pomocí argumentů, které jsou zmíněny v kapitole 5 nebo pomocí konfiguračního souboru `config.yaml`

Příloha B

Technická specifikace Nginx

Tato část popisuje doporučení pro konfiguraci serveru Nginx. Zaměřuje se na nastavení důležité pro obranu proti možným útokům. Tato technická specifikace je tvořena pro potřeby společnosti Kyndryl, aby ulehčila specialistům nastavení reverzního proxy serveru Nginx. Na závěr je ukázka finální konfigurace.

B.1 Obecná doporučení

V této sekci jsou probírány obecná doporučení pro konfiguraci serveru Nginx.

B.1.1 Zkratky v direktivách

Následující zkratky pro velikost lze zadat v kontextu hodnoty direktiv [17]:

- **k** nebo **K**: Kilobajty
- **m** nebo **M**: Megabajty
- **g** nebo **G**: Gigabajty

Další zkratky pro specifikaci času, které lze zadat v kontextu hodnoty direktiv [17]:

- **ms**: milisekundy
- **s**: sekundy
- **m**: minuty
- **h**: hodiny
- **d**: dny
- **w**: týdny
- **M**: měsíc (30 dnů)
- **y**: rok (365 dnů)

B.1.2 Uzavírání spojení pomocí časového limitu

Správné časové limity jsou klíčové pro řízení připojení a prevenci zahlcení serveru nečinnými spojeními, která škodí výkonu a můžou vyčerpat zdroje serveru. Nginx umožňuje uzavření spojení, která zapisují data neobvykle dlouho, což může být indikátorem pokusu udržet spojení otevřená co nejdéle. Správné nastavení následujících direktiv zabraňuje zatížení serveru pomalými klienty a udržuje výkon serveru [17, 48]:

- `client_body_timeout <time>` (Výchozí hodnota: 60s): Určuje časový limit pro čtení těla požadavku klienta. Časový limit je nastaven pouze pro dobu mezi dvěma po sobě jdoucími operacemi čtení požadavku, nikoli pro přenos celého těla požadavku. Pokud klient během nastaveného limitu nic neodešle, požadavek je ukončen s chybou 408.
- `client_header_timeout <time>` (Výchozí hodnota: 60s): Určuje časový limit pro čtení hlavičky požadavku klienta. Pokud klient během této doby neodešle celou hlavičku, požadavek je ukončen s chybou 408.
- `keepalive_timeout <timeout> [header_timeout]` (Výchozí hodnota: 75s): Určuje časový limit, po který zůstane klientské spojení otevřené na straně serveru. Druhý (volitelný) parametr nastavuje hodnotu v hlavičce „Keep-Alive: timeout=time“. Příliš vysoké hodnoty direktivy `keepalive_timeout` vedou k ztrátě paměti, protože spojení zůstanou otevřená i bez provozu.
- `send_timeout <time>` (Výchozí hodnota: 60s): Určuje časový limit pro přenos odpovědi klientovi. Časový limit je nastavený pouze mezi dvěmi po sobě jdoucími operacemi zápisu, nikoliv pro přenos celé odpovědi. Spojení se uzavře pokud klient během časového limitu nic neobrží.

Konfigurace

Následující konfigurace byla úspěšně ověřena proti `Slow Denial of Service` útokům: `slow header` a `slow post`. Na testovacím serveru bylo vytvořeno až 15 000 spojení, která simulovala tyto útoky. Hodnota 5 sekund pro direktivy `client_header_timeout` a `client_body_timeout` uspěla v obraně proti útokům `slow header` a `slow post`.

```
http {
    client_body_timeout 5s;
    client_header_timeout 5s;
    keepalive_timeout 5s 5s;
    send_timeout 10s;
}
```

Výpis B.1: Konfigurace limitů

B.1.3 Omezení připojení

Nginx umožňuje omezit počet připojení podle klíče, jako je IP adresa, což chrání server proti `DoS` a `DDoS` útokům. Bez omezení může jeden klíč otevřít libovolné množství připojení. Omezení připojení by mělo být přizpůsobeno potřebám provozu. Spojení je započítáno do limitu v případě, že celá hlavička požadavku byla načtena a požadavek je zpracovaný serverem. V případě `slow header DoS` útoků je toto nastavení neúčinné kvůli

započítání spojení až po načtení celé hlavičky. Následující direktivy slouží k nastavení modulu `ngx_http_limit_conn_module`: [3, 11, 12]

- `limit_conn_zone <key> zone=<name:size>` (Výchozí hodnota: -): Nastaví sdílenou paměťovou zónu pro uložení požadavků pro daný klíč.
- `limit_conn <zone> <number>` (Výchozí hodnota: -): Určuje počet připojení, které klient může vytvořit pro daný klíč.
- `limit_conn_status <code>` (Výchozí hodnota: -): Nastaví stavový kód, který se vrací jako odpověď na odmítnuté požadavky. Vrací stavový kód v případě, že počet připojení dosáhne limitní hodnoty nastavené v direktivě `limit_conn`.

Omezení připojení bylo testováno proti `Slow Denial of Service` útokům: `slow post` a `slow read`. Na testovacím serveru bylo vytvořeno až 15 000 spojení, která simulovala tyto útoky. Úspěšně byla ověřena hodnota 10-50 připojení, které jedna IP adresa může vytvořit.

Direktiv `limit_conn` a `limit_conn_zone` může být několik. Následující konfigurace omezí počet připojení k serveru pro jednu IP adresu klienta a zároveň celkový počet připojení k serveru [12]:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr_limit:10m;
    limit_conn_zone $server_name zone=server_limit:10m;
    limit_conn_status 429;
    server {
        limit_conn addr_limit 10;
        limit_conn server_limit 2000;
    }
}
```

Výpis B.2: Konfigurace omezení připojení pro všechny IP adresy a celý server

V případě, že direktiva `limit_conn` je použita v kontextu `server` a zároveň `location` a IP adresa splňuje požadavky obou limitů, tak je využita více specifická direktiva v kontextu `location` [11, 12]. Následující konfigurace slouží jako ukázka precedence výběru:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr_limit:10m;
    limit_conn_status 429;
    server {
        limit_conn addr_limit 10;
        location /private {
            # Used limit on URI /private
            limit_conn addr_limit 50;
        }
    }
}
```

Výpis B.3: Ukázka precedence výběru limitu

V případě, že mnoho uživatelů pracuje ze stejné sítě a sdílí stejnou IP adresu (časté ve firemních sítích), tak lze využít direktivy `geo` a `map`. Direktiva `geo` určuje IP adresy, které nemají být limitovány. Povolené IP adresy získají hodnotu 0 a všechny ostatní hodnotu

1. Následně je v direktivě `map` namapována proměnná `$whitelist` na `$limit`. Pokud IP adresa není specifikována v direktivě `geo`, tudíž má hodnotu 1, tak je do proměnné `$limit` namapována IP adresa (`$binary_remote_addr`), jinak je nastavena na prázdný řetězec. Pokud proměnná `$limit` je prázdný řetězec, tak není aplikován žádný limit. V případě potřeby omezit i IP adresy v seznamu povolených IP adres je možné přidat `$binary_remote_addr zone=conn_limit_wl`. Tímto je vytvořen limit pro všechny IP adresy, ale Nginx aplikuje nejvíce restriktivní direktivu, které požadavky byly splněny. To znamená, že nelimitované adresy v proměnné `$whitelist` využijí `conn_limit_wl`, protože splňují požadavky pouze této direktivy a ostatní splňují požadavky obou direktiv, ale přísnější direktiva `conn_limit` je použita. Následující konfigurace omezí připojení na základě listu adres: [11, 62, 12, 51, 63]:

```
http {
    geo $whitelist {
        default 1;
        10.0.0.1/32 0;
        192.168.1.0/24 0;
    }
    map $whitelist $limit {
        0 "";
        1 $binary_remote_addr;
    }
    limit_conn_zone $limit zone=conn_limit:10m;
    limit_conn_zone $binary_remote_addr zone=conn_limit_wl:10m;
    server {
        location / {
            limit_conn conn_limit 10;
            limit_conn conn_limit_wl 100;
        }
    }
}
```

Výpis B.4: Konfigurace omezení připojení s výjimkou určitých IP adres na základě listu adres

B.1.4 Omezení šířky pásma

Nginx poskytuje omezení šířky pásma stahování pomocí následujících direktiv [11, 48]:

- `limit_rate_after <size>`(Výchozí hodnota: 0 [B]): Určuje, že připojení by nemělo být omezeno direktivou `limit_rate`, dokud není přeneseno určité množství dat.
- `limit_rate <rate>`(Výchozí hodnota: 0 [B]): Umožňuje omezit přenosovou rychlost připojení klienta po překročení hodnoty direktivy `limit_rate_after`. Omezení je nastaveno na jedno připojení tudíž, pokud klient otevře dvě připojení, bude rychlost dvakrát vyšší než zadaný limit. Této situaci je možné předejít použitím direktiv `limit_conn_zone` a `limit_conn`.

Konfigurace

```
http {
    limit_conn_zone $limit zone=conn_limit:10m;
    limit_conn_status 429;
    limit_rate_after 1m;
    limit_rate 250k;
    server {
        location /download/ {
            limit_conn conn_limit 10;
        }
    }
}
```

Výpis B.5: Konfigurace pro omezení šířky pásma.

B.1.5 Omezení provozu

Nginx umožňuje omezit rychlost zpracování požadavků podle klíče, jako je například IP adresa, což poskytuje ochranu proti různým brute-force útokům a pokusům o `Denial of Service` útok. Jedná se o důležitý mechanismus, který umožňuje efektivně řídit tok požadavků na server a zabránit nadměrnému zatížení serveru. Následující direktivy slouží k nastavení modulu `ngx_http_limit_req_module` [11, 17, 42]:

- `limit_req_zone <key> zone=<name:size> rate=<rate>` (Výchozí hodnota: -): Určuje klíč, který má být omezen, a sdílenou paměťovou zónu a její velikost pro ukládání požadavků (ukládá počet přebytečných požadavků). Poslední parametr určuje počet požadavků za sekundu (r/s) nebo za minutu (m/s) před zavedním limitu.
- `limit_req zone=<name> [burst=number] [nodelay|delay=number]` (Výchozí hodnota: -): Určuje použitou zónu sdílené paměti a maximálně velikost `burst` požadavků. Pokud počet požadavků překročí rychlost zpracování požadavků nastavenou pro zónu, jejich zpracování se zpozdí tak, aby byly požadavky zpracovávány definovanou rychlostí. Nadměrné požadavky jsou odkládány dokud jejich počet nepřekročí maximální velikost `burst`, v takovém případě je požadavek ukončen s chybou.
  - `burst` (Výchozí hodnota: 0): Určuje maximální počet požadavků, který může překročit definovanou rychlost.
- `limit_req_status <code>` (Výchozí hodnota: -): Nastaví stavový kód, který se vrací jako odpověď na odmítnuté požadavky. Vrací stavový kód v případě, že počet požadavků dosáhne limitní hodnoty.

Direktiv `limit_req` a `limit_req_zone` může být několik. Následující konfigurace nastaví velikost sdílené paměti na 10MB a omezí rychlost zpracování požadavků přicházejících z jedné IP adresy a zároveň rychlost zpracování požadavků serverem. V případě, že požadavky překročí stanovenou hranici, jsou ukonečny se statusem 429 [11, 17, 42]:

```
http {
    limit_req_zone $binary_remote_addr zone=addr_limit:10m rate=1r/s;
    limit_req_zone $server_name zone=server_limit:10m rate=10r/s;
    limit_req_status 429;
    server {
```

```
        limit_req zone=addr_limit burst=5 nodelay;
        limit_req zone=server_limit burst=10;
    }
}
```

Výpis B.6: Konfigurace pro omezení provozu.

Pro případ, že klient potřebuje vytvořit více požadavků najednou a následně omezí svou aktivitu, slouží argument `burst`. Klient bude mít možnost překročit nastavenou hranici a jeho požadavky nebudou odmítnuty. Požadavky, které překročí nastavenou hranici, jsou zpožděny tak, aby byly zpracovány s definovanou rychlostí v direktivě `limit_req_zone`.

Argument `burst` je důležitý pro případ, že uživatel pošle dva požadavky v rozmezí stanoveném limitem. Například v následující konfiguraci je stanoveno deset požadavků za sekundu, což znamená jeden požadavek za 100ms. Pokud uživatel pošle dva požadavky v rozmezí 100ms a není použitý argument `burst`, tak bude navrácen chybový kód. Pokud je použitý argument `burst` a z jedné IP adresy přijde najednou 21 požadavků, tak Nginx předá serveru první požadavek a zbylých 20 dá do fronty. Poté každých 100 ms předá požadavek z fronty a vrací chybový kód pouze v případě, že fronta přesáhne počet požadavků stanovený velikostí argumentem `burst`, což by v následující konfiguraci bylo 21 požadavků ve frontě [11, 17, 42].

```
http {
    limit_req_zone $binary_remote_addr zone=req_zone:10m rate=10r/s;
    limit_req_status 429;
    server {
        location / {
            limit_req zone=req_zone burst=20;
        }
    }
}
```

Výpis B.7: Konfigurace pro omezení provozu s argumentem burst.

Chování argumentu `burst` lze změnit pomocí argumentů `delay` a `nodelay`. Bez argumentů `delay` a `nodelay` se může projevit zpomalení aplikace. Pokud bereme do úvahy předchozí konfiguraci, tak poslední paket ve frontě čeká na předání 2 sekundy. Tato doba by mohla způsobit, že paket již není relevantní pro klienta v době předání odpovědi.

Argument `nodelay` klientovi umožňuje spotřebovat hodnotu `burst` najednou, ale přebytečné požadavky jsou odmítnuty, dokud neuplyne dostatek času k uspokojení rychlostního limitu. Argument `nodelay` zajišťuje to, že požadavky jsou předány okamžitě, pokud je pro ně ve frontě volné místo. V případě následující konfigurace by to znamenalo, že pokud přijde najedou 21 požadavků a jsou volná všechny místa fronty, tak Nginx předá všech 21 požadavků a označí 20 míst ve frontě jako obsazené. Následně každých 100 ms uvolní jedno místo ve frontě. V případě, že současně přijde více jak 21 požadavků, tak přebytečné ihned odmítne s chybovým kódem.

Argument `delay` definuje, kolik požadavků může být provedeno ihned bez útlumu. V následující konfiguraci může klient vytvořit devět požadavků ihned bez zpoždění, další tři budou zpožděny a jakýkoliv další požadavek během následujících čtyř sekund bude odmítnut [11, 17, 42, 51].

```
http {
    limit_req_zone $binary_remote_addr zone=req_zone:10m rate=10r/s;
```

```
    limit_req_zone $binary_remote_addr zone=req_zone_se:10m rate=3r/s;
    limit_req_status 429;
    server {
        location /home/ {
            limit_req zone=req_zone_se burst=12 delay=9;
        }
        location /search/ {
            limit_req zone=req_zone burst=20 nodelay;
        }
    }
}
```

Výpis B.8: Konfigurace pro omezení provozu pomocí argumentů delay a nodelay.

V případě, že mnoho uživatelů pracuje ze stejné sítě sdílí stejnou IP adresu (časté ve firemních sítích). Pro tento případ lze využít direktiv `geo` a `map`. V direktivě `geo` se přiřadí do `$whitelist` hodnota 0 pro povolené IP adresy a hodnota 1 pro všechny ostatní. Následně je v direktivě `map` namapována proměnná `$whitelist` na `$limit_key`. Pokud IP adresa není specifikována v direktivě `geo`, tudíž má hodnotu 1, tak je namapována proměnna $limit_key na IP adresu (`$binary_remote_addr`), jinak je nastavena na prázdný řetězec. Pokud proměnná $limit_key je prázdný řětězec, tak není aplikován žádný limit na provoz. V případě potřeby omezit i povolené IP adresy je možné přidat `$binary_remote_addr zone=req_zone_wl`. Tímto je vytvořen limit pro všechny IP adresy, ale Nginx aplikuje nejvíce restriktivní direktivu, které požadavky byly splněny. To znamená, že nelimitované adresy v promměnné `$whitelist` využijí `req_zone_wl`, protože splňují pouze požadavky této direktivy a ostatní splňují požadavky obou direktiv, ale přísnější direktiva `req_zone` je použita [51, 62, 63].

Konfigurace

```
http {
    geo $whitelist {
        default 1;
        10.0.0.0/8 0;
        192.168.0.0/24 0;
    }
    map $whitelist $limit_key {
        0 "";
        1 $binary_remote_addr;
    }
    limit_req_zone $limit_key zone=req_zone:10m rate=4r/s;
    limit_req_zone $binary_remote_addr zone=req_zone_wl:10m rate=40r/s;
    limit_req_status 429;
    server {
        limit_req zone=req_zone burst=12 nodelay;
        limit_req zone=req_zone_wl burst=80 nodelay;
    }
}
```

Výpis B.9: Konfigurace pro omezení provozu pouze určitým adresám.

B.1.6 Zpracování požadavků klientů

Bezpečné a efektivní zpracování požadavků klientů je dosaženo omezením zdrojů a limitů, což brání zahlcení serveru velkými požadavky. Špatně nastavené direktivy mohou vést k útokům přetečení vyrovnávací paměti a následnému úspěšnému provedení útoku `Denial of Service`. Správné hodnoty direktiv závisí na dostupné paměti, CPU a očekávaném provozu.

K přetečení vyrovnávací paměti dochází, když se program pokusí zapsat příliš mnoho dat do bloku vyrovnávací paměti s pevnou délkou a překročí hranici této paměti. Toto může útočníkovi usnadnit využít systemové prostředky systému. Důležité direktivy pro efektivní a bezpečné zpracování požadavků klientů [17, 48, 66, 79]:

- `client_body_buffer_size <size>`(Výchozí hodnota: 8k (32-bit) | 16k (64-bit): Určuje maximální velikost vyrovnávací paměti, kterou Nginx používá k ukládání těla příchozího požadavku klienta. Pokud tělo požadavku tuto velikost překročí, začne Nginx přebytečná data zapisovat do dočasného souboru. Tato direktiva pomáhá zabránit zahlcení serveru velkými těly požadavků.

- `client_header_buffer_size <size>`(Výchozí hodnota: 1k): Určuje maximální velikost vyrovnávací paměti, kterou Nginx používá k ukládání hlaviček požadavků klienta. V případě velmi velké hlavičky požadavku, která se do této vyrovnávací paměti nevejde, Nginx alokuje větší vyrovnávací paměť, která je konfigurována direktivou `large_client_header_buffers`

- `client_max_body_size <size>`(Výchozí hodnota: 1m): Tato direktiva určuje maximální povolenou velikost těla požadavku klienta. Pokud tělo požadavku tuto velikost překročí, Nginx vrátí chybu `413 Request Entity Too Large`. Tato direktiva pomáhá zabránit `Denial of Service` útoku v případě, že je klientům umožněno nahrávat soubory prostřednictvím protokolu HTTP.

- `large_client_header_buffers <number> <size>`(Výchozí hodnota: 4 8k): Určuje maximální počet a velikost vyrovnávacích pamětí používaných pro výjimečně velké hlavičky požadavků klientů. V případě, že řádek URI požadavku je větší než velikost jednoho bufferu Nginx vrací chybu `414 Request-URI Too Large` a pokud jiný řádek hlavičky přesahuje velikost bufferu, Nginx vrací chybu `400 Bad request`. Tato direktiva pomáhá zmírnit problém v případě, že výchozí vyrovnávací paměť `client_header_buffer_size` neměla dostatečnou velikost pro uložení hlavičky požadavku klienta.

Konfigurace

Následující konfigurace je vytvořena na základě doporučení OWASP proti útokům přetečení vyrovnávací paměti [66].

```
http {
    server {
        location / {
            client_body_buffer_size 100k;
            client_header_buffer_size 1k;
            client_max_body_size 100k;
            large_client_header_buffers 2 1k;
        }
```

```
    }
}
```

Výpis B.10: Konfigurace pro zpracování požadavků klientů

B.1.7 catch-all blok serveru

Parametr `default_server` určuje výchozí blok `server`, který zpracovává požadavky, pokud požadovanému názvu serveru neodpovídá jiný blok `server`. Tato direktiva je užitečná v situaci, kdy existuje více bloků `server`, z nichž každý zpracovává požadavky pro různé domény.

Nginx používá hlavičku `Host` v požadavku k rozpoznání, který blok `server` použít, přiřazuje hlavičku `Host` k direktivě `server_name`. Pokud není nalezen odpovídající blok `server`, tak je požadavek přesměrován na blok `server` s parametrem `default_server`. Pokud žádná direktiva `listen` neobsahuje parametr `default_server`, první server s dvojicí `adresa:port` bude výchozím serverem.

Je důležité odmítat požadavky bez hlavičky `Host` a s neplatnými doménovými jmény. K tomu slouží tzv. `catch-all server` blok. Tento blok odmítá požadavky bez hlavičky `Host` a hlavičky s neplatnými doménovými jmény, což chrání před chybami v konfiguraci(například směrování na nesprávny backend). [17, 48, 52]

Konfigurace

V následující konfiguraci jsou odmítnuty požadavky bez hlavičky `Host` pomocí parametru `""` a hlavičky s neplatnými doménovými jmény pomocí parametru `_`. Pokud požadavky dorazí do tohoto bloku, server vrátí `444`, což uzavře spojení, aniž by klientovi odeslal jakoukoli odpověď.

```
http {
    server {
        listen <ip_adress>:<port> default_server;
        server_name _ "";
        return 444;
    }
}
```

Výpis B.11: Konfigurace pro požadavky s nedefinovanými názvy serveru

B.1.8 Caching senzitivních dat

Stránky obsahující citlivé informace by měly obsahovat zabezpečení `Cache-Control`, aby se zabránilo ukládání obsahu do mezipaměti. Pro tento účel lze použít direktivy: `Cache-Control: no-cache, no-store` a `expires: 0`. Direktiva `Cache-Control` nabízí následující možnosti [21, 82]:

- `Cache-Control: public`: Odpovědi mohou být uloženy do jakékoli cache, i když nejsou určeny pro ukládání. Klasicky nelze uložit například požadavky s `Authorization` hlavičkou, ale tato direktiva uložení povolí.
- `Cache-Control: private`: Odpovědi mohou být uloženy do cache prohlíže. Proxy nemůže požadavek uložit do cache.

- `Cache-Control: no-cache`: Prohlížeč může uložit odpověď do cache, ale musí odeslat požadavek na ověření serveru.
- `Cache-Control: no-store`: Odpovědi nejsou nikdy uloženy do cache.

Konfigurace

```
http {
    server {
        location /api {
          expires 0;
          add_header Cache-Control "no-cache, no-store";
        }
        location ~* \.(js|css|jpg|jpeg|png|gif|js|css|ico|swf)$ {
            expires 1y;
            add_header Cache-Control "public, no-transform";
        }
        location ~* \.(html)$ {
            add_header Cache-Control "no-cache";
        }
    }
}
```

Výpis B.12: Catching senzitivních dat

B.1.9 Omezení povolených metod HTTP

Většina webových serverů vyžaduje ke správné funkci pouze GET, POST a HEAD metody. Omezením dalších metod jako PUT, DELETE, TRACE a OPTIONS lze snížit dosah útoku a zmírnit potenciální zranitelnosti [48, 65].

- `OPTIONS`: Metoda OPTIONS by mohla útočníkovi poskytnout zdroj dalších informací o serveru, které mohou usnadnit další útoky.
- `TRACE`: Metoda TRACE může útočníkovi umožnit `cross-site-tracing` útok, který může usnadnit odchycení ID relace jiného uživatele. Útočník také může do HTTP hlaviček vložit škodlivé skripty a sledovat je zpětně prostřednictvím požadavků TRACE, což vede k `cross-site scripting` (XSS) zranitelnosti.
- `DELETE`: Metoda DELETE může útočnikovi umožnit odstranit kritické soubory nebo adresáře serveru.
- `PUT`: Metoda PUT může útočnikovi umožnit nahrávat škodlivé soubory na server, což může vést k vzdálenému spuštění kódu.

Konfigurace

Následující konfigurací Nginx omezí všechny metody v bloku `location` kromě povolených `GET`, `HEAD` a `POST`.

```
http {
    server {
        location /api {
            limit_except GET HEAD POST {
                allow 192.168.1.0/24;
                deny all;
            }
        }
    }
}
```

Výpis B.13: Omezení povolených metod HTTP

B.1.10 Direktiva `worker_processes` a `worker_connections`

Direktiva `worker_processes` určuje počet pracovních procesů, které se mají spustit. Optimální hodnota závisí mimo jiné na počtu jader procesoru a zátěži serveru. Výchozím bodem je nastavit hodnotu na `auto` nebo na počet jader procesoru (hodnota `auto` se snaží o automatickou detekci počtu jader). Hodnotu je vhodné zvýšit pokud pracovní procesy provádí hodně diskových I/O operací a jsou skrze počet pracovních procesů blokovány na této operaci.

Direktiva `worker_connections` určuje maximální počet současných připojení, která může pracovní proces otevřít. Výchozí hodnota je 512 a vhodné nastavení závisí na velikosti serveru a provozu serveru. Direktiva `worker_connections` zahrnuje všechna spojení, nejen spojení s klienty, ale i spojení s proxy servery, upstream spojení a listen sokety. [47, 40]

Nginx je schopen zpracovat `worker_connections * worker_processes` současných připojení. Počet připojení je omezen maximálním počtem otevřených souborů `rlimit_nofile`, který lze změnit pomocí direktivy `worker_rlimit_nofile` (Výchozí hodnota: `none`). V případě hodnoty `none`, Nginx nastavuje tuto hodnotu ze systémových limitů. Počáteční bezpečná hodnota pro `worker_rlimit_nofile` a systémové limity je `worker_connections * 2`.

Direktiva `worker_rlimit_nofile` slouží k změně maximálního počtu otevřených souborů, které pracovní procesy Nginx mohou zpracovávat, což je nastaveno měkkým systémovým limitem `ulimit -Sn`, ale stále bude omezena trvdým systémovým limitem `ulimit -Hn` [15, 47].

Nastavení a použití této hodnoty může omezovat i SELinux. Pro změnu systémových limitů a `worker_rlimit_nofile` Nginx je nutné povolit `httpd_setrlimit` v nastavení SELinux.

B.1.11 Neprivilegovaný uživatel

Programy by měly mít minimální oprávnění potřebná k jejich činnosti. V případě, že útočník nalezne způsob, jak zneužít zranitelnost v Nginx, běh jako neprivilegovaný uživatel omezí rozsah škod.

Princip nejmenších oprávnění říká to, že entita by měla mít pouze tolik oprávnění, kolik je nezbytné k dosažení cílů v rámci daného systému. Podle tohoto principu běží pouze hlavní proces Nginx jako `root` [13].

Konfigurace

```
# nginx.conf:
user nginx;

# Nastaveni vlastnika a skupiny pro korenovy adresar:
chown -R root:root /etc/nginx
```

Výpis B.14: Konfigurace oprávnění Nginx

B.1.12 Skrytí verze a podpisu serveru Nginx

Zveřejnění verze Nginx může být riskantní, protože může sloužit jako výchozí bod pro útočníky hledající zranitelnosti. Skrytí verze pomocí direktivy `server_tokens` a skrytí používaného softwaru s direktivou `more_clear_headers` může snížit riziko cílených útoků a ztížit sběr informací o serveru. Nginx v základní konfiguraci sdílí verzi v hlavičce HTTP odpovědi a chybových stránkách a software v `Server` hlaviččce odpovědi. Pro využití direktivy `more_clear_headers` je nutné nainstalovat modul `headers_more` [17, 57].

Konfigurace

```
http {
    server {
        server_tokens off;
        more_clear_headers 'Server:';
    }
}
```

Výpis B.15: Konfigurace pro skrytí verze a podpisu Nginx

B.1.13 Bezpečnostní hlavičky

Direktiva `add_header` v Nginx slouží k přidání nebo úpravě hlaviček HTTP odpovědí, které Nginx odesílá klientům při úspěšném požadavku (pro odeslaní i při neúspěšném požadavku je nutné nastavit parametr `always`).

Content-Security-Policy

Content Security Policy (CSP) pomáhá detekovat a zmírňovat určité typy útoků, jako je Cross-Site Scripting (XSS) nebo data injection útoků. Nastavením `Content-Security-Policy` lze specifikovat, které typy prostředků (skripty, obrázky, atd.) jsou povoleny a ze kterých zdrojů je lze načíst. K nastavení této vlastnosti by se mělo přistupovat velmi individuálně. Slepé nasazení hlavičky CSP může rozbít webovou aplikaci. [73].

Mezi důležité nastavení direktivy `Content-Security-Policy` patří [20, 32]:

- `default-src`: Určuje platné zdroje pro: `child-src`, `connect-src`, `font-src`, `img-src`, `media-src`, `object-src`, `script-src` a `style-src`.
- `script-src`: Určuje platné zdroje pro JavaScript.
- `connect-src`: Omezuje adresy URL, které lze načíst pomocí rozhraní skriptů.

- `img-src`: Určuje platné zdroje obrázků.
- `style-src`: Určuje platné zdroje pro soubory stylů

Důležité hodnoty direktiv [20, 32]:

- `*`: Umožňuje načíst jakýkoliv obsah pro daný typ prostředku.
- `none`: Zabraňuje direktivě načíst jakýkoliv obsah.
- `self`: Umožňuje načíst pouze obsah, který pochází ze zdroje stejného jako je aplikace.

Konfigurace

Následující konfigurace povoluje načítání obrázků, skriptů, AJAXu a CSS ze stejného zdroje a nepovoluje načítání jiných zdrojů.

```
add_header Content-Security-Policy "default-src 'none'; script-src 'self';
connect-src 'self'; img-src 'self'; style-src 'self';" always;
```

Výpis B.16: Konfigurace pro Content-Security-Policy

Referrer-Policy

Tato hlavička ovládá, jaké informace související s adresou URL se posílají na server při načítání externích zdrojů. Použití hodnoty `no-referrer` určuje, že se s požadavkem nemají odesílat žádné informace o zdroji.

Důležité hodnoty direktivy `Referrer-Policy` [33]:

- `no-referrer`: Hlavička `Refer` bude vynechána a požadavek neobsahuje žádné informace o zdroji.
- `no-referrer-when-downgrade`: Pošle origin, cestu a řetězec dotazu pro požadavky, kde protokol zůstane na stejné úrovni nebo se zlepší na vyšší úroveň. Pro požadavky zaslány na slabší protokol hlavičku nezašle.
- `origin`: V hlavičce požadavku zašle origin: požadavek na stránku `https://example.com/page.html` pošle pouze `https://example.com/`
- `origin-when-cross-origin`: Pokud je požadavek poslán na stejný protokol pošle se origin, cesta a řetězec dotazu. Zašle pouze origin pro požadavky zaslány na slabší protokol.
- `same-origin`: Pošle `origin`, cestu a řetězec dotazu pouze pro požadavky ze stejného zdroje. Hlavičku nezašle pro požadavky z jiných zdrojů
- `strict-origin`: Zašle `origin`, pokud protokol požadavku zůstane stejný. Hlavičku nezašle pro požadavky zaslány na slabší protokol.
- `strict-origin-when-cross-origin`: Pošle `origin`, cestu a řetězec dotazu pouze pro požadavky ze stejného zdroje. Pro požadavky z jiných zdrojů zašle `origin`, pokud protokol zůstane na stejné úrovni. Nepošle hlavičku, pokud se úroveň protokolu zhorší.

Konfigurace

```
add_header Referrer-Policy "no-referrer";
```

Výpis B.17: Konfigurace pro Referrer-Policy

X-Frame-Options

Tato hlavička řídí, zda je povoleno vložit stránku pomocí rámců (`frame` nebo `iframe`). Pomáhá chránit proti útokům typu clickjacking. [36].

- `deny`: Zakazuje vložení zdroje.
- `sameorigin`: Umožňuje vložení zdroje na stejný zdroj.

Konfigurace

```
add_header X-Frame-Options "SAMEORIGIN" always;
```

Výpis B.18: Konfigurace pro X-Frame-Options

X-XSS-Protection

Hlavička navržená ke zmírnění útoků XSS ve webových prohlížečích. Tato funkce je v moderních prohlížečích povolena ve výchozím nastavení. Úkolem této hlavičky je tedy znovu povolit tento filtr, pokud byl vypnut uživatelem. Tato funkce může chránit uživatele starších webových prohlížečů, které ještě nepodporují CSP. V některých případech naopak může tato ochrana XSS vytvořit zranitelost XSS na jinak bezpečných webových stránkách. Ve finalé je proti XSS zranitelosti lepší použít hlavičku CSP a posílat hodnotu 0 přes tuto chybnou funkci [37, 75].

X-Content-Type-Options

Tato hlavička zabraňuje prohlížeči provádět MIME-type sniffing tím, že říká, že typy MIME v hlavičkách `Content-Type` jsou nakonfigurovány záměrně a neměly by se měnit. [35, 64]

Konfigurace

```
add_header X-Content-Type-Options "nosniff" always;
```

Výpis B.19: Konfigurace pro X-Content-Type-Options

Feature-Policy

Tato hlavička poskytuje kontrolu nad tím, ke kterým funkcím prohlížeče a API lze přistupovat nebo je používat v rámci webu a jeho vložených zdrojů. Chrání web před třetími stranami, které chtějí použít API, která mají dopad na zabezpečení a soukromí. Primárním účelem je zlepšit bezpečnost a soukromí tím, že selektivně povolí nebo zakáže konkrétní funkce prohlížeče.

Hlavička `Feature-Policy` lze nastavit na tři parametry [34]:

- `*`: Povolí vlastnost ze všech zdrojů.
- `none`: Slouží pro vypnutí vlastnosti.
- `self`: Povolí požadavky ze stejného zdroje jako je aplikace.

Konfigurace

```
add_header Feature-Policy "geolocation 'none'; midi 'none';
notifications 'none'; push 'none'; sync-xhr 'none'; microphone 'none';
camera 'none'; magnetometer 'none'; gyroscope 'none'; speaker 'none';
vibrate 'none'; fullscreen 'none'; payment 'none'; usb 'none';";
```

Výpis B.20: Konfigurace pro Feature-Policy

B.1.14 Omezení přístupu

Nginx umožňuje omezit přístup k citlivým zdrojům na serveru pomocí různých metod, včetně [11]:

`deny` a `allow`

Direktivy `deny` a `allow` umožňují omezit přístup k určitým lokacím na serveru na základě IP adresy, rozsahu adres nebo podsítě. Tato pravidla jsou kontrolována postupně, a pokud požadavek pochází z adresy jiné než povolené je vracen kód odpovědi 403. V případě velkého počtu pravidel je lepší použít metodu `Seznam pro řízení přístupu (ACL)`, který využívá modulu `ngx_http_geo_module` [58].

Konfigurace

```
location /private {
    deny 192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny all;
}
```

Výpis B.21: Omezení přístupu pomocí direktiv `deny` a `allow`

Základní HTTP ověření

Tato metoda umožňuje omezit přístup ke zdrojům pomocí uživatelského jména a hesla před udělením přístupu za pomoci protokolu `HTTP Basic Authentication`. Pomocí direktivy `auth_basic` lze povolit ověření přístupu zadáním řetězce, který se zobrazí při příchodu neověřeného uživatele, které je ve výchozí konfiguraci vypnuto.

Direktiva `auth_basic_user_file` určuje soubor, který obsahuje uživatelská jména a hesla ve formátu `name1:password1`, kde heslo je zašifrováno nebo zahashované. [59].

Konfigurace

```
location /private {
    auth_basic "Restricted Area";
    auth_basic_user_file /etc/nginx/.passwd;
}
```

Výpis B.22: Omezení přístupu pomocí základního HTTP ověření

Splnění více bezpečnostních metod

Pokud je potřeba kombinovat více metod pro ověření, lze využít direktivu `satisfy`, která umožňuje povolit přístup pouze pokud jsou splněny všechny požadavky hodnotou `all` nebo pokud je splněn alespoň jeden hodnotou `any`. Pomocí následující konfigurace uživatel, který posílá požadavek na daný blok `location` musí splnit alespoň jednu z bezpečnostních metod. Musí pocházet z bloku adres `192.168.1.0/24` nebo uživatel musí zadat uživatelské jméno a heslo [48, 54].

Konfigurace

```
location /private {
    satisfy any;
    allow 192.168.1.0/24;
    deny all;

    auth_basic "Restricted Area";
    auth_basic_user_file /etc/nginx/.passwd;
}
```

Výpis B.23: Omezení přístupu za pomoci více metod přístupu

Seznam pro řízení přístupu (ACL)

Modul `ngx_http_geo_module` umožňuje vytvářet seznamy pro řízení přístupu na základě IP adresy. Tím lze přesněji kontrolovat, kdo má přístup k určitým lokalitám.

Následující konfigurace povolí přístup pouze pokud proměnná `$allowed_ip` má hodnotu 1, tudíž povoluje přístup pouze adresám z bloku adres `192.168.1.0/24` a ostatním vrací kód 403 [62].

Konfigurace

```
http {
    geo $allowed_ip {
        default 0;
        192.168.1.0/24 1;
    }
    server {
        listen 80;
        server_name example.com;
```

```
        location /restricted {
            if ($allowed_ip = 0) {
                return 403;
            }
        }
    }
}
}
```

Výpis B.24: Omezení přístupu pomocí seznamu

Omezení přistupu na základě polohy

Modul `nginx-module-geoip` umožňuje omezit přístup na základě země či regionu uživatele. Tento modul je nutné nainstalovat a povolit v Nginx konfiguraci. Pomocí proměnné `$geoip_country_code` lze specifikovat dvoupísemný kód země (například RU, US, CA).

Následující konfigurace nastaví hodnotu proměnné `$country_access` na 1 pokud IP adresa klienta pochází z US nebo Kanady a na hodnotu 0 pro ostatní země. Tímto povolí přístup pouze IP adresám, které pochází z US nebo Kanady a ostatním vrací kód 403 [56, 61].

Konfigurace

```
load_module "/etc/nginx/modules/ngx_http_geoip_module.so";
    http {
        map $geoip_country_code $country_access {
            "US" 1;
            "CA" 1;
            default 0;
        }
        server {
            if ($country_access = '0') {
                return 403;
            }
    }
}
```

Výpis B.25: Omezení přístupu na základě polohy

Ověření na základě dílčího požadavku

Modul `http_auth_request_module` ověří klienta během zpracovaní požadavku pomocí dílčího požadavku na externí ověřovací službu. Pokud dílčí požadavek vrátí kód odpovědi 2xx, je přístup povolen, pokud vrátí kód 401 nebo 403, je přístup odepřen. Tímto je možné implementovat různá schémata ověřovaní, jako například vícefaktorové ověřování. Modul `http_auth_request_module` je nutné před použitím povolit pomocí `./configure -with-http_auth_request_module`.

Direktiva `auth_request` povolí autorizaci na základě dílčího požadavku a nastaví URI pro odeslání požadavku. Volitelně lze použít direktivu `auth_request_set`, která po dokončení dílčího požadavku nastaví proměnné na hodnoty z dílčího požadavku. V případě, že

ověřovací služba nevyžaduje tělo požadavku, lze tělo požadavku upustit pomocí direktivy `proxy_pass_request_body`, čímž se sníží doba zpracování požadavku. Pokud je tělo požadavku vypuštěno musí být hlavička `Content-Length` nastavena na prázdný řetězec pomocí direktivy `proxy_set_header`. V případě, že služba vyžaduje URI, ke kterému klient přistupuje skrze požadavek, lze ho předa pomocí direktivy `proxy_set_header` pomocí vlastní hlavičky `X-Original-URI`. Direktiva `internal` určuje, že blok `location` lze použít pouze pro interní požadavky [55, 60].

Konfigurace

```
location /private/ {
    auth_request /auth;
    auth_request_set $auth_status $upstream_status;
}
location = /auth {
    internal;
    proxy_pass http://auth-server;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
    proxy_set_header X-Original-URI $request_uri;
}
```

Výpis B.26: Omezení přístupu na základě dílčího požadavku

B.2 SSL/TLS

Tato sekce se zabývá konfigurací pro SSL/TLS. Probírá přesměrování požadavků na HTTPS, hlavičku `Strict-Transport-Security`, udržování relací SSL, certifikáty, klíče a šifrovací sady.

B.2.1 Přesměrování na HTTPS

Hlavním cílem TLS je zabezpečit data přenášená mezi klientem a serverem, zabraňuje odposlouchávání a zajišťuje soukromí citlivých informací. Také se zajistí, že data, která jsou přenášena, nebyla nijak změněna. Jakýkoliv pokus o manipulaci s přenášenými daty bude detekován. TLS využívá digitální certifikáty k ověření identity serveru a brání útokům jako je man-in-the-middle.

Konfigurace poslouchá na portu 80, jakožto defaultní server pro IPv4, IPv6 a hostname. Pomocí `return` je vráceno 301 permanentní přesměrování na server HTTPS na stejném hostiteli ($host) a URI požadavku ($request_uri). Veškerý provoz je přesměrován na HTTPS. V případě, že je potřeba komunikaci přesměrovat jen v určitých lokacích, kde se nachází senzitivní informace, je možné použít blok `location` [11].

Konfigurace

```
http {
    server {
        listen 80 default_server;
```

```
        listen [::]:80 default_server;
        location /login {
            return 301 https://$host$request_uri;
        }
    }
    server {
        listen 443 ssl;
        listen [::]:443 ssl;
        ssl_certificate /etc/nginx/ssl/example.crt;
        ssl_certificate_key /etc/nginx/ssl/example.key;
    }
}
```

Výpis B.27: Konfigurace pro přesměrování komunikace

B.2.2 HTTP Strict transport security

Pomocí hlavičky `Strict-Transport-Security` lze prohlížeče informovat, aby vždy používaly HTTPS. Tato konfigurace je dobrá i z hlediska výkonu, protože prohlížeč provádí přesměrování z HTTP na HTTPS na straně klienta, aniž by použil síť serveru. Prohlížeč si informaci o `Strict-Transport-Security` hlavičce udržuje po definovanou dobu. Tato konfigurace ochraňuje aplikaci proti man-in-the-middle útokům a protokol downgrade útokům.

Hlavička `Strict-Transport-Security` musí být nastavena uvnitř bloku `http` s příkazem `listen ssl`, aby se zabránilo posílání hlaviček na HTTP stránky. Je vhodné ji kombinovat s přesměrováním na HTTPS, které je popsané v B.2.1.

Ideálně by měla být použita volba `includeSubdomains`, což poskytuje robustnější zabezpečení pro hlavní hostname i pro subdomény. To však vyžaduje nasazení SSL na všech subdoménách. Problém využití bez `includeSubdomains` je, že man-in-the-middle útočník může vytvořit libovolné subdomény a používat je pro vložení cookies do aplikace a nebo může dojít k úniku informací. [11].

Konfigurace

```
http {
    server {
        listen 443 ssl;
        listen [::]:443 ssl;
        #(31536000 seconds)
        add_header Strict-Transport-Security
        "max-age=31536000; includeSubdomains" always;
    }
}
```

Výpis B.28: Konfigurace strict transport security

B.2.3 Udržování relací SSL

Parametry `ssl_session_cache` a `ssl_session_timeout` umožňují pracovním procesům Nginx udržovat informace o relacích po určitou dobu, což snižuje počet handshaků a zvyšuje

výkon aplikace. Tyto pracovní procesy sdílejí tuto cache mezi sebou uvnitř jedné instance, což umožňuje v případě následující konfigurace ukládat informace o relacích po dobu 4 hodin s dostupnou pamětí o velikosti 40 MB. Do jednoho megabajtu cache paměti lze uložit přibližně 4000 relací.

```
http {
    server {
        ssl_session_timeout 4h;
        ssl_session_cache shared:SSL:40m; # about 40000 sessions
        ssl_session_tickets off;
        ssl_buffer_size 4k;
    }
}
```

Výpis B.29: Konfigurace pro udržování relací SSL

Alternativou k direktivě `ssl_session_cache` je direktiva `ssl_session_tickets`, která klientovi předá informace o připojení, a tím minimalizovat opakované vyjednávání. Direktiva `ssl_buffer_size` nastavuje velikost vyrovnávací paměti, která je používaná pro odesílání dat.

Důležité direktivy modulu `ngx_http_ssl_module` [11, 45]:

- `ssl_session_cache <off>|<none>|[builtin[:size]] [shared:name:size]` (Výchozí hodnota: -): Nastavuje typy a velikosti použitých cache pamětí, které ukládájí parametry relace. Lze nastavit následující typy této direktivy:
  - `off`: Určuje, že použití cache pamětí je zákazáno. Nginx klientovi řekne, že nemůže znovu využít relaci.
  - `none`: Určuje, že cache paměti nejsou použity na ukládání parametrů relací, ale klientovi je řečeno, že relace mohou být znovu využity.
  - `buildin`: Určuje, že je použita cache paměť zabudovaná v OpenSSL. Mezipaměť využívá pouze jeden pracovní proces Nginx a velikost se udává v počtu relací. Pokud velikost není zadána, tak má tento typ hodnotu 20 480. Použití vestavěné cache paměti může způsobit fragmentaci paměti.
  - `shared`: Tento typ určuje, že cache paměť je sdílena mezi všemi pracovními procesy Nginx. Velikost cache paměti se udává v bajtech a do jednoho megabajtu paměti lze uložit přibližně 4 000 relací.
- `ssl_session_timeout <time>` (Výchozí hodnota: 5m): Určuje dobu, po kterou může klient znovu použít parametry relace.
- `ssl_buffer_size <size>` (Výchozí hodnota: 16k): Určuje velikost vyrovnávací paměti používané pro odesílání dat.

B.2.4 Certifikáty

Pro SSL certifikáty se obvykle používají asymetrické algoritmy, jako je `RSA` a `ECC`. Doporučená délka klíče pro `RSA` je 2 048 bitů a pro `ECC` 256 bitů. I když jsou oba algoritmy považovány za stejně odolné, `ECC` je rychlejší díky menší velikosti klíče. Generování klíčů `RSA` trvá déle a vyžaduje více výpočetního výkonu. Ve většině případů je větší délka klíče `RSA` a `ECC` považována za plýtvání CPU [73].

- `ssl_certificate <file>` (Výchozí hodnota: -): Tato direktiva určuje použitý certifikát pro server ve formátu PEM.
- `ssl_certificate_key` (Výchozí hodnota: -): Tato direktiva určuje tajný klíč ve formátu PEM pro server.

Konfigurace

```
http {
    server {
        # Retezec RSA certifikatu
        ssl_certificate /etc/nginx/example_cert.crt;
        # RSA sifrovaci klic
        ssl_certificate_key /etc/nginx/example_key.pem;

        # Certifikat elipticke krivky
        ssl_certificate $ecdsa_cert;
        # Klic elipticke krivky
        ssl_certificate_key data:$ecdsa_key;
    }
}
```

Výpis B.30: Konfigurace pro certifikáty

B.2.5 Šifrovací sada a protokoly

Nginx vyjednává nejvyšší standard šifrování mezi klientem a serverem. V následující konfiguraci Nginx akceptuje nejnovější protokoly verze `TLSv1.2` a `TLSv1.3`, které jsou bez známých bezpečnostních problémů. Udržování aktuálního seznamu povolených šifer a protokolů je klíčové pro zajištění bezpečnosti.

Seznam povolených šifer je nastaven na ty, které jsou aktuálně doporučované pro univerzální servery s různými klienty. Tento seznam se častěji mění než ostatní parametry, a proto je důležité udržovat jej aktuální. Doporučená konfigurace pro dnešek může být zítra zastaralá.

Pro kontrolu podporovaných šifer v OpenSSL na serveru lze použít následující příkazy:

```
openssl ciphers -s -v
openssl ciphers -s -v ECDHE
openssl ciphers -s -v DHE
```

Výpis B.31: Kontrola podporovaných šifer

Bez pečlivého výběru šifrovací sady hrozí riziko vyjednávání s méně bezpečnou sadou, která může být kompromitována. Pro maximální zabezpečení je nezbytné používat pouze silné a nezranitelné šifrovací sady, přičemž na pořadí seznamu záleží. Šifry `ECDHE` jsou obvykle rychlejší než `DHE`, proto by měly být upřednostňovány [10, 31].

- `ssl_protocols <protocol>` (Výchozí hodnota: `TLSv1 TLSv1.1 TLSv1.2 TLSv1.3`): Povolí zadané protokoly.
- `ssl_prefer_server_ciphers <on|off>` (Výchozí hodnota: -): Informuje klienta, že upřednostňujeme šifry serveru před šiframi klienta.

- `ssl_ciphers <ciphers>` (Výchozí hodnota: HIGH:!aNULL:!MD5): Určuje povolené šifry.

Konfigurace

```
http {
    server {
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_prefer_server_ciphers on;
        ssl_ciphers
        ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:
        ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:
        ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:
        DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:
        DHE-RSA-CHACHA20-POLY1305;
    }
}
```

Výpis B.32: Konfigurace pro šifrovací sadu a protokoly

B.2.6 Silná výměna klíčů

Pro SSL handshake se využívá klíč Diffie-Hellman, který umožňuje získání sdíleného tajného klíče mezi uživateli. Výměna klíčů probíhá přes veřejnou síť, což znamená, že všechny zprávy odeslané mezi dvěma uživateli mohou být zachyceny a přečteny jakýmkoli jiným uživatelem.

Nginx deleguje veškerou práci ohledně klíčové výměny na OpenSSL, který ve výchozím nastavení používá klíč o velikosti 1 024 bitů. Je nutné zajistit silnější klíč, aby nedošlo k snížení úrovně bezpečnosti. Použití slabého klíče by mohlo umožnit útočníkovi provést Logjam útok, který by mu dovolil provádět downgrade TLS v `man-in-the-middle` situaci.

Pokud je používán pouze protokol `TLSv1.3` nebo pouze šifry ECDHE/ECDH, pak není potřeba direktiva `ssl_dhparam`. Pokud jsou používány šifry DHE, je nutné použít direktivu `ssl_dhparam` s parametry DH (min. 2 048 bitů). Ve výchozím nastavení nejsou žádné parametry nastaveny, což znamená, že šifry DHE nebudou použity [2, 45, 73].

- `ssl_cdhparam <file>` (Výchozí hodnota: -): Tato direktiva určuje soubor s parametry DH pro šifry DHE.

Konfigurace

```
# Vygenerovani parametru DH:
openssl dhparam -out /etc/nginx/ssl/dhparam.pem 2048

# nginx.conf:
ssl_dhparam /etc/nginx/dhparam.pem;
```

Výpis B.33: Konfigurace pro výměnu klíčů

B.2.7 Upstream šifrování

Pokud je nutné šifrovat provoz mezi Nginx a upstream službou a nastavit pravidla vyjednávání kvůli dodržování předpisů společnosti nebo je upstream služba mimo zabezpečenou síť. Nastavené direktivy zajišťují ověření platnosti certifikátu upstream služby a stanovují hloubku tohoto ověření [11, 44].

- `proxy_ssl_verify <on|off>` (Výchozí hodnota: off): Určuje zda je povolené ověření certifikátu proxy serveru.
- `proxy_ssl_verify_depth <number>` (Výchozí hodnota: 1): Nastaví hloubku ověření certifikátů proxy serveru.

Konfigurace

```
location / {
    proxy_pass https://upstream.example.com;
    proxy_ssl_verify on;
    proxy_ssl_verify_depth 2;
    proxy_ssl_protocols TLSv1.2 TLSv1.3;
}
```

Výpis B.34: Konfigurace pro upstream šifrování

B.2.8 OSCP stapling

OCSP (Online Certificate Status Protocol) stapling je technika v systému Nginx, která snižuje dobu vyjednávání SSL tím, že umožňuje klientům snadněji obnovit relaci k serveru SSL/TLS. Při běžných transakcích OCSP klient kontaktuje certifikační autoritu, aby zkontroloval stav certifikátu serveru. V případě vysokého provozu na serveru tato kontrola může způsobit vysokou zátěž pro server. OCSP stapling je navržen tak, že server pravidelně získává OCSP záznamy od certifikační autority a přikládá je ke komunikaci s klientem. Záznam OSCP je serverem ukládán do mezipaměti, aby se omezila komunikace se certifikační autoritou [17] .

Direktivy pro OSCP stapling [45, 48]:

- `ssl_stapling <on|off>` (Výchozí hodnota: off): Povoluje OCSP stapling.
- `ssl_stapling_verify <on|off>` (Výchozí hodnota: off): Povoluje ověřování OSCP odpovědí serverem. Pro správnou funkčnost ověřování OSCP odpovědí je nutné mít všechny certifikáty serveru nastaveny jako důvěryhodné pomocí direktivy `ssl_trusted_certificate`.
- `ssl_trusted_certificate <file>` (Výchozí hodnota: -): Nastavuje cestu k důvěryhodnému certifikátu (.pem).
- `resolver <adress> [valid=time] [ipv4=on|off] [ipv6=on|off] [status_zone=zone]` (Výchozí hodnota: -): Nastavuje jeden nebo více serverů, které mají být použity k překladu názvů upstream serverů na IP adresy. Volitelný parametr `valid` přepíše TTL záznam domény. Ve výchozím nastavení Nginx využije pro překlad výchozí DNS server.

- `resolver_timeout <time>` (Výchozí hodnota: 30s): Nastavuje časový limit pro překlad názvu.
- `ssl_trusted_certificate <file>` (Výchozí hodnota: -): Volitelná direktiva, která určuje cestu k záznamu OSCP. Tento záznam nahrazuje dotaz na respondér OSCP, který je uvedný v certifikátu serveru.
- `ssl_stapling_responder <url>` (Výchozí hodnota: -): Volitelná direktiva, která přepisuje URL respondéru OSCP.

Konfigurace

```
http {
    server {
        ssl_stapling on;
        ssl_stapling_verify on;
        ssl_trusted_certificate /path/to/root_CA_cert_plus_intermediates;
        resolver 1.1.1.1 valid=300s;
        resolver_timeout 5s;
    }
}
```

Výpis B.35: Konfigurace pro OSCP stapling

B.2.9 Obrana proti BEAST útoku

BEAST útok cílí na šifrování `cipher block chaining (CBC)` v protokolech SSL/TLS a využívá zranitelnosti v TLS 1.0 a starších verzích.

Zabránění útoku BEAST lze dosáhnout nastavením verze protokolu TLS na verzi 1.2 nebo vyšší a vyloučení protokolu TLS verze 1.0. Šifry, které byly zranitelné v TLS verze 1.2 a vyšší, byly zrušeny, takže není nutné používat `ssl_prefer_server_ciphers`. To umožňuje klientovi vybrat si preferovanou metodu šifrování na základě jeho možností hardwaru.

Pokud server podporuje starší verze TLS, které jsou zranitelné proti útoku BEAST, je nezbytné použí `ssl_prefer_server_ciphers`. Bez tohoto nastavení by útočník mohl donutit připojení k použití slabých šifer, což by umožnilo dešifrování připojení. [5, 45]

Konfigurace

```
http {
    server{
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_prefer_server_ciphers off;
    }
}
```

Výpis B.36: Konfigurace proti BEAST útoku

B.3 Výkon

V této sekci jsou probírány konfigurace pro optimalizaci výkonu.

B.3.1 `gzip` komprese

Kompresí odpovědí Nginx výrazně snižuje jejich velikost, což zlepšuje dobu přenosu obsahu a snižuje režii serveru. Povolením direktivy `gzip` lze ušetřit šířku pásma a zlepšit dobu načítání stránek pro klienty s pomalým připojením. Direktiva `gzip` by měla být povolena pouze pro relevantní typy obsahu jako je text, CSS nebo JavaScript. Je nutné se vyhnout nadměrnému využívání komprese, která může vést k vyššímu využití procesoru a kompresi obsahu, který již prošel kompresí, například JPEG [41, 49].

- `gzip <on|off>` (Výchozí hodnota: off): Slouží k povolení komprese `gzip`
- `gzip_types` (Výchozí hodnota: text/html): Ve výchozí konfiguraci Nginx komprimuje pouze odpovědi typu MIME `text/html`. Nastavením direktivy `gzip_types` lze určit další typy MIME, které mají projít kompresí.
- `gzip_vary <on|off>` (Výchozí hodnota: off): Povoluje vkládání hlavičky `Vary: Accept-Encoding` v případě, že odpověď byla komprimována pomocí `gzip`. Tímto Nginx ukládá do mezipaměti zazipovanou i běžnou verzi zdroje. Tato direktiva zvyšuje účinnost mezipaměti a celkový výkon.
- `gzip_min_length <length>` (Výchozí hodnota: 20B): Určuje minimální délku odpovědi pro spuštění komprese. Tímto se lze vyhnout komprimaci malých souborů, které z komprese nemají téměř žádný prospěch.
- `gzip_comp_level <level>` (Výchozí hodnota: 1): Určuje úroveň komprese (1-9). Určuje, jak moc jsou data komprimována na této stupnici, kde 9 je nejvíce komprimované. Ideální hodnota této direktivy je 4 nebo 6. Vyšší hodnoty jsou zbytečné a pouze plýtvají čas procesoru [7].
- `gzip_proxied` (Výchozí hodnota: off): Určuje podmínky, za kterých má být u proxy požadavků povolena komprese `gzip`. Tuto direktivu lze nastavit na následující hodnoty:
  - `off`: Zakáže kompresi pro všechny proxy požadavky.
  - `expired`: Povolí kompresi pro odpovědi, jejichž platnost vypršela podle hlavičky `Expires`.
  - `no-cache`: Povolí kompresi v případě, že hlavička odpovědi obsahuje `Cache-Control: no-cache`.
  - `no-store`: Povolí kompresi v případě, že hlavička odpovědi obsahuje `Cache-Control: no-store`.
  - `private`: Povolí kompresi v případě, že hlavička odpovědi obsahuje `Cache-Control: private`.
  - `no_last_modified`: Zakáže kompresi v případě, že hlavička odpovědi obsahuje `Last-Modified`.
  - `no_etag`: Zakáže kompresi v případě, že hlavička odpovědi obsahuje `ETag`.
  - `auth`: Povolí kompresi v případě, že hlavička odpovědi obsahuje `Authorization`, tudíž vyžaduje ověření.
  - `any`: Povolí kompresi pro všechny proxy požadavky.

- `gzip_disable`: Zakáže kompresi `gzip` pro určité řetězce User-Agent, které odpovídají zadanému regulárnímu výrazu. Tato direktiva je užitečná v případě, kdy je komprese nežádoucí nebo může způsobit problémy.
- `gzip_static`: Tato direktiva slouží k statické kompresi. Umožňuje odesíla předkomprimované soubory s příponou `.gz`. (Pro tuto direktivu je nutno povolit modul pomocí parametru `-with-http_gzip_static_module`.)

Konfigurace

```
http {
    server {
        location / {
            gzip on;
            gzip_types text/plain text/xml text/css application/javascript
            application/x-javascript application/xml application/json;
            gzip_vary on;
            gzip_min_length 1000;
            gzip_comp_level 4;
            gzip_proxied expired no-cache -no-store private auth;
            gzip_disable "MSIE [1-6]\." "image/jpeg" "image/png"
            "audio/mpeg";
        }
        location ^~ /assets/ {
          gzip_static on;
        }
    }
}
```

Výpis B.37: `gzip` komprese

B.4 Reverzní proxy

Tato sekce se zabývá konfigurací pro reverzní proxy. Probírá předávání požadavků, proxy hlavičky, časové limity a vyrovnávací paměť.

B.4.1 Předávání požadavků

Direktiva `proxy_pass` umožňuje předání požadavků HTTP a HTTPS serverům. Nastavuje se protokol a adresa proxy serveru (doména nebo IP adresa), a volitelně URI pro mapování umístění. V případě, že location blok obsahuje specifikaci umístění a direktiva `proxy_pass` je definována s URI, část URI požadavku nahradí část URI uvedenou v direktivě `proxy_pass`. Pokud direktiva `proxy_pass` neobsahuje URI, server použije URI přímo od klienta [44].

Lokace	proxy_pass	URI požadavku	Výsledné URI
/welcome/	http://localhost:8080/login/	/welcome/app	/login/app
/welcome/	http://localhost:8080/login	/welcome/app	/loginapp
/welcome	http://localhost:8080/login/	/welcome/app	/login//app
/welcome	http://localhost:8080/login	/welcome/app	/login/app

Tabulka B.1: Ukázka přesměrování požadavků

Pokud je potřeba všechny požadavky nebo požadavky pod určitou lokací přesměrovat na specifické URI, lze použít direktivu `rewrite`. Tato direktiva umožňuje manipulaci s adresami URL pomocí regulárních výrazů a umožňuje přesměrování a přepisování URI. Například všechny požadavky lze přesměrovat na specifickou lokaci v bloku `location /`. [50]

Konfigurace

```
http {
    server {
        location / {
            rewrite ^/$ http://localhost:8080/welcome/login redirect;
            rewrite ^/someuri$ http://localhost:8080/welcome/login redirect;
            proxy_pass http://localhost:8080;
        }
        location /app/ {
            proxy_pass http://www.example.com/login/;
        }
        location /welcome/ {
            proxy_pass http://localhost:8080/login/;
        }
        location /login/ {
            proxy_pass http://localhost:8080;
        }
    }
}
```

Výpis B.38: Konfigurace proxy_pass

B.4.2 Proxy hlavičky

Direktiva `proxy_set_header` umožňuje nastavit HTTP hlavičky, které se předávají na backendové servery. Pokud je hodnota hlavičky prázdná, není tato hlavička předána proxy serveru. Tato direktiva umožňuje úpravu nebo přidání polí do hlavičky požadavku, který se předává na backendový server [44].

Běžné případy použití `proxy_set_header` [44, 17, 27]:

- `proxy_set_header Host $host`: Výchozí hodnota hlavičky `Host` v předaném požadavku obvykle obsahuje hostname proxy serveru. Toto nastavení zajistí, že hlavička `Host` z požadavku klienta je předána backendovému serveru beze změny, což je důležité pro správnou identifikaci domény požadované uživatelem.

- `proxy_set_header X-Real-IP $remote_addr`: Tato hlavička předává skutečnou IP adresu klienta proxy serveru.
- `proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for`: Pokud klient na své straně využívá proxy server nebo vyvažovač zátěže, tato hlavička zajišťuje předání původní IP adresy klienta a IP adresy komunikujícího soketu backend serveru.
- `proxy_set_header X-Forwarded-Proto $scheme`: Tato hlavička identifikuje použitý protokol (HTTP nebo HTTPS) pro připojení klienta k proxy serveru nebo vyvažovači zátěže, čímž informuje backend server, že připojení bylo zabezpečeno.
- `proxy_set_header X-Forwarded-Host $host`: Tato hlavička předává původního hostitele, kterého klient požadoval.
- `proxy_set_header X-Forwarded-Port $server_port`: Tato hlavička předává původní port, který klient požadoval.

Konfigurace

```
location / {
    proxy_pass http://localhost:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Port $server_port;
}
```

Výpis B.39: Konfigurace proxy hlaviček

B.4.3 Proxy timeouts

Následující časové limity určují, jak dlouho bude webový server Nginx čekat na dokončení různých operací při komunikaci s backend servery. Tyto operace zahrnují návázaní spojení, odeslání požadavků a přijímání odpovědí. Správně nastavené limity mohou optimalizovat dobu odezvy a zabránit zbytečným prodlevám. Nevhodně nastavené limity mohou vést k problém, jako je vyčerpání prostředků nebo zvýšená latence. Základní direktivy pro proxy timeout [44, 17]:

- `proxy_connect_timeout`(Výchozí hodnota: 60 sekund): Určuje časový limit pro navázání spojení s backend serverem. Nastavení hodnoty tohoto parametru záleží na průměrném zatížení, ale hodnota nemůže přesáhnout 75 sekund.
- `proxy_send_timeout`(Výchozí hodnota: 60 sekund): Určuje časový limit pro přenos požadavku na backend server. Tento časový limit je definován mezi dvěma operacemi zápisu během komunikace.
- `proxy_read_timeout`(Výchozí hodnota: 60 sekund): Určuje časový limit pro načtení požadavku z backend serveru. Tento časový limit je definován mezi dvěma operacemi čtení během komunikace.

B.4.4 Proxy vyrovnávací paměť

Ve výchozím nastavení Nginx používá vyrovnávací paměť k dočasnému ukládání odpovědí z backend serverů do vyrovnávací paměti. Odpověď je uložena do interní vyrovnávací pamětí a klientovi je předána až po obdržení celé odpovědi. Ukládání do vyrovnávací paměti pomáha optimalizovat výkon u pomalých klientů, kteří proxy server mohou zpomalovat, pokud je jim odpověď předávána synchronně. Nginx odpověď ukládá po dobu, kterou klienti potřebují k jejímu stažení. Pokud je vyrovnávací paměť vypnuta, tak je odpověď předána klientovi synchronně, zatímco ji přijímá z backend serveru. Vypnutá vyrovnávací paměť může být žádoucí pro rychlé klienty, kteří potřebují přijímat odpověď co nejrychleji. Správné nastavení následujících direktiv může výrazně zvýšit výkon proxy serveru [44, 11]:

- `proxy_buffering <on|off>`(Výchozí hodnota: on): Určuje, zda má Nginx ukládat odpovědi z backend serveru do vyrovnávací paměti.
- `proxy_buffer_size <size>`(Výchozí hodnota: 4k|8k): Určuje maximální velikost vyrovnávací paměti použité pro první část odpovědi přijaté z backend serveru.
- `proxy_buffers <number size>`(Výchozí hodnota: 8 4k|8k): Určuje počet vyrovnávacích pamětí a jejich velikost. Tyto vyrovnávací paměti jsou použité pro jedno připojení ke čtení odpovědi z backend serveru.
- `proxy_busy_buffers_size <size>`(Výchozí hodnota: 8k|16k): Omezuje celkovou velikost všech dočasných vyrovnávacích pamětí, která může být obsazena odesíláním odpovědi klientovi, zatímco odpoveď ještě není zcela přečtena. Zbytek velikosti vyrovnávacích paměti může mezitím být využit ke čtení odpovědi. Ve výchozím nastavení je velikost této direktivy dvojnásobek velikosti direktivy `proxy_buffers` nebo `proxy_buffer_size`.

Konfigurace

```
http {
    server {
        proxy_buffering on;
        proxy_buffer_size 3k;
        proxy_buffers 4 16k;
        proxy_busy_buffers_size 48k;
    }
}
```

Výpis B.40: Konfigurace vyrovnávací paměti proxy

Pomocí dostupné paměti pro Nginx server a velikosti direktivy `proxy_buffers` a direktivy `proxy_busy_buffers_size` lze vypočítat přibližný počet připojení, který reverzní proxy bude schopný obsloužit na základě vyrovnávací paměti:

Čtyři buffery o velikosti 8 KB je 32 768 bajtů ($8 \times 4 \times 1\,024$) pro jedno aktivní připojení. V případě, že Nginx má k dispozici 1 000 MB paměti získáme hodnotu 1 048 576 000 ($1\,000 \times 1\,024 \times 1\,024$).

Po vydělení těchto čísel nám vyjde 32 000 aktivních připojení. ($\frac{1\,048\,576\,000}{32\,768} = 32\,000$) [3].

B.5 Load Balancing

Tato sekce se zabývá konfigurací vyvažování zátěže. Probírá pasivní health checks, omezení počtu připojení a metody vyvažování zátěže.

B.5.1 Pasivní health checks

Monitorování stavu je důležité u všech metod vyrovnávání zátěže. Pasivní kontroly sledují probíhající připojení, která jsou neúspěšná nebo časově omezená. Pokud se připojení nepodaří obnovit, tak Nginx označí server jako nedostupný a dočasně omezí provoz pro tento server.

Tato funkce je ve výchozím nastavení povolena, ale následující parametry umožňují upravit její chování [17, 46]:

- `fail_timeout` (Výchozí hodnota: 10s): Nastavuje dobu, v které musí dojít k určitému počtu neúspěšných pokusů o připojení, aby byl server označen jako nedostupný. Tato doba určuje i dobu, po kterou je server označen jako nedostupný.
- `max_fails` (Výchozí hodnota: 1): Nastavuje počet neúspěšných pokusů o připojení, které musí nastat během doby nastavené pomocí parametru `fail_timeout`, aby byl server označen jako nedostupný.

Konfigurace

```
upstream backend {
   server example1.com max_fails=3 fail_timeout=5s;
   server example2.com max_fails=3 fail_timeout=5s;
}
```

Výpis B.41: Konfigurace pro pasivní health checks

B.5.2 Omezení počtu připojení pomocí `max_conns`

Nginx nabízí možnost omezit počet aktivních připojení na backend server pomocí argumentu `max_conns`. Výchozí hodnotou argumentu `max_conns` je 0, což znamená, že není nastaven žádný limit. Direktiva `queue` omezuje počet požadavků, které jsou zařazeny do fronty, když server dosáhne limitu připojení. Časový limit, který je určen parametrem `timeout` určuje, jak dlouho má být požadavek ve frontě ponechán. [17, 46]

Konfigurace

```
upstream backend {
   server 192.168.237.130:80 max_conns=200;
   server 192.168.237.131:80 max_conns=200;
   queue 10 timeout=30s;
}
```

Výpis B.42: Konfigurace pro omezení připojení pomocí parametru max_conns.

B.5.3 Metody vyvažování zátěže

Nginx poskytuje následující metody vyvažování zátěže [46, 74]:

Round Robin

Tato metoda je používaná ve výchozím nastavení. Požadavky jsou distribuovány rovnoměrně mezi servery, přičemž se berou v úvahu váhy serverů.

```
upstream backend {
   server example1.com;
   server example2.com;
}
```

Výpis B.43: Konfigurace metody Round Robin

Least Connections

Požadavek je odeslán na server s nejmenším počtem aktivních spojení, opět s ohledem na váhy serveru.

```
upstream backend {
    least_conn;
    server example1.com;
    server example2.com;
}
```

Výpis B.44: Konfigurace metody Least Connections

IP Hash

Server, na který je požadavek odeslán, je určen z IP adresy klienta. V tomto případě se pro výpočet hodnoty hash použijí buď první tři oktety IPv4 adresy, nebo celá adresa IPv6. Metoda zaručuje, že požadavky ze stejné adresy se dostanou na stejný server, pokud je dostupný.

```
upstream backend {
    ip_hash;
    server example1.com;
    server example2.com;
}
```

Výpis B.45: Konfigurace metody IP Hash

Generic Hash

Server, na který je požadavek odeslán, je určen z uživatelem definovaného klíče, kterým může být textový řetězec, proměnná nebo jejich kombinace.

```
upstream backend {
    hash $request_uri consistent;
    server example1.com;
    server example2.com;
```

```
}
```

Výpis B.46: Konfigurace metody Generic Hash

Random

Každý požadavek bude předán na náhodně vybraný server. Pokud je zadán parametr `two`, tak Nginx nejprve náhodně vybere dva servery s ohledem na váhu serveru a poté z těchto dvou serverů vybere jeden pomocí jedné z následujících metod:

- `least_conn`: Server s nejmenším počtem aktivních připojení.
- `least_time=header` (Nginx Plus): Server s nejkratší průměrnou dobou přijetí hlavičky odpovědi.
- `least_time=last_byte` (Nginx Plus): Server s nejkratší průměrnou dobou pro obdržení kompletní odpovědi.

```
upstream backend {
    random two least_time=last_byte;
    server example1.com;
    server example2.com;
    server example3.com;
    server example4.com;
}
```

Výpis B.47: Konfigurace metody Random

Least Time

Pro každý požadavek vybere server s nejnižší průměrnou latencí a nejnižším počtem aktivních připojení. Tato metoda je dostupná pouze s NGINX Plus

```
upstream backend {
    least_time header;
    server example1.com;
    server example2.com;
}
```

Výpis B.48: Konfigurace metody Least Time

B.5.4 Parametr `down`

Pokud je nutné dočasně vyřadit server z rotace vyrovnávání zátěže, lze jej označit parametrem `down`. Požadavky, které měl tento server zpracovat, se automaticky odešlou na další server ve skupině [17].

Konfigurace

```
upstream backend {
    server example1.com;
    server example2.com down;
}
```

Výpis B.49: Konfigurace s použitím parametru down

B.5.5 Váhy serverů

Ve výchozím nastavení rozdělení požadavků mezi servery ve skupině funguje na základě jejich váhy metodou `Round Robin`. Parametr `backup` slouží k určení náhradního serveru. Server označený tímto parametrem nepřijímá požadavky dokud nejsou ostatní servery nedostupné [17, 46].

Konfigurace

V následující konfiguraci je z každých šesti požadavků pět odesláno na server `example1.com` a jeden na server `example2.com`.

```
upstream backend {
    server example1.com weight=5;
    server example2.com;
    server 192.0.0.1 backup;
}
```

Výpis B.50: Konfigurace s použitím parametru down