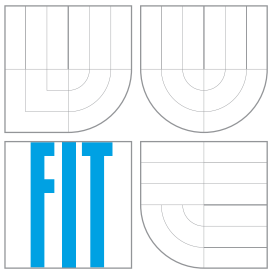
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYTVOŘENÍ SECURITY SERVICE PRO ÚČINNÝ BOJ PROTI INTERNETOVÝM ÚTOKŮM

IMPLEMENTATION OF SECURITY SERVICE FOR PREVENTING INTERNET ATTACKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAREL FAJKUS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL OČENÁŠEK, Ph.D.

BRNO 2016

Zadání bakalářské práce/18853/2015/xfajku05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Fajkus Karel**
Obor: Informační technologie
Téma: **Vytvoření Security Service pro účinný boj proti internetovým útokům**
Implement Security Service for Preventing Internet Attacks
Kategorie: Softwarové inženýrství

Pokyny:

1. Analýzujte NAT technologie a možnosti jednoznačného určení jednotlivých uživatelů.
2. Analýzujte možnosti propojení externích bezpečnostních mechanismů (například oosec) s DNS.
3. Analýzujte distribuce DNS black listu do všech systémů skrze DNS.
4. Navrhněte na základě předchozích analýz službu, která dokáže rozpoznat jednotlivé útočníky a ukončit s nimi spojení, případně dát do blacklistu.
5. Po konzultaci s vedoucím a externím konzultantem navržené řešení implementujte.
6. Implementované řešení důkladně otestujte, výsledky vyhodnoťte a nastíňte možnosti dalšího rozšíření.

Literatura:

- CHESWICK, William R, Steven M BELLOVIN a Avi RUBIN. *Firewalls and Internet security: repelling the wily hacker*. 2nd ed. Boston: Addison-Wesley, c2003, xx, 433 s. Addison-Wesley professional computing series. ISBN 02-016-3466-X.
- GARFINKEL, Simson a Gene SPAFFORD. *Practical UNIX and Internet security*. 2nd ed. Bonn: O'Reilly and Associates, c1996, xxix, 971 s. ISBN 15-659-2148-8.
- FILIP, Radek. *OFICIÁLNÍ WWW PREZENTACE OBCE JÍLOVICE BEZPEČNOSTNÍ ASPEKTY INTERNETU*. České Budějovice, 2003.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Očenášek Pavel, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cílem této práce je navrhnout a implementovat systém, který má zabránit uživatelům v přístupu do systémů na základě jejich aktivit. V dnešní době jsou kyber útoky běžnou věcí, a proto je proto potřeba mít své aplikace a systémy dobře chráněny. Díky použitým postupům by mělo dojít ke snížení počtu těchto útoků a zároveň zamezení situací, kdy kvůli zabránění přístupu na základě veřejné IP adresy útočníka, byli ovlivněni i běžní uživatelé, kteří tuto adresu měli také.

Abstract

The main purpose of this work is to design and implement a system, that would allow to ban users based on their actions. Currently, cyber attacks have become very common, which leads to a necessity to develop great application and system protections. This project offers a solution, that could decrease a number of cyber attacks, and also prevent banning common users because of the same public IP address as attacker.

Klíčová slova

IP adresa, DNS, OSSEC, Elasticsearch, Logstash, ElastAlert, Kibana, černá listina.

Keywords

IP address, DNS, OSSEC, Elasticsearch, Logstash, ElastAlert, Kibana, blacklist.

Citace

FAJKUS, Karel. *Vytvoření Security Service pro účinný boj proti internetovým útokům.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Očenášek Pavel.

Vytvoření Security Service pro účinný boj proti internetovým útokům

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Očenáška, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Karel Fajkus
13. května 2016

Poděkování

Chtěl bych poděkovat svému vedoucímu práce, Ing. Pavlu Očenáškovi, Ph.D., za pomoc a užitečné informace a Ing. Dominiku Wolfovi z firmy AVG při řešení technických problémů. Dále bych chtěl poděkovat mé rodině a přátelům za podporu při studiu.

Obsah

Kapitola 1

Úvod

V dnešní době, kdy má přístup k internetu skoro kdokoliv, vzrůstá počet kyberútoků[1]. Jako reakci na tuto situaci začínají firmy využívat různé formy obrany proti těmto útokům. Ve většině případů dochází k zablokování veřejné IP adresy, odkud přichází útok. Pokud ale používá tuto IP adresu více lidí, dochází k zabránění přístupu na webové stránky nevinným uživatelům a firma může přijít o potencionální příjmy. Uživatel se dá konkrétně rozpoznat podle veřejné a privátní IP adresy. Díky těmto informacím může dojít k zablokování konkrétní dvojice IP adres a ostatní uživatelé mohou i nadále navštěvovat webové stránky firmy.

Jednotlivé body této práce jsou shrnuty níže:

- Analýza získání veřejné a privátní IP adresy uživatele.
- Analýza využití externích bezpečnostních mechanismů.
- Analyzujte distribuce černé listiny uživatelů do všech systémů skrze DNS.
- Návrh bezpečnostní služby.
- Implementace.
- Testování.
- Závěr.

Práce je rozdělena do několika kapitol, které popisují kompletní proces vývoje od použitých technologií až po testování a výsledné zhodnocení práce. V kapitole 2, jsou detailně popsané důvody a způsoby použití jednotlivých nástrojů, programovacích jazyků a frameworků. V další kapitole 3 je návrh samotné služby. Tato kapitola obsahuje návrh aplikace pro komunikaci s DNS serverem a také návrh systému pro zpracování záznamů. Návrh obsahuje také diagramy. Kapitola 4 podrobně popisuje implementaci jednotlivých technologií. V neposlední řadě kapitola 5 obsahuje popis způsobu testování služby a jejích částí. V kapitole 6 je popsáno možné vylepšení této práce. V 7. kapitole se nachází celkové zhodnocení práce a výsledků testů.

[1] Nabourání do systému skrze internet za účelem získání dat či zapříčinění výpadku

Kapitola 2

Použité technologie

2.1 Úvod

IP adresa je 32 bitové číslo, skládající se ze čtyř oktetů[1] a každý oktet může nabývat hodnot 0-255. Tato hodnota je unikátní pro každé zařízení v síti. IP adresy se dělí na privátní a veřejné. Veřejná IP adresa je používána pro komunikaci mezi internetem a uživatelem. Pokud někdo zná tuto adresu, může se napojit na toto zařízení. Privátní IP adresy slouží výhradně k adresování zařízení v privátní síti. Tyto adresy nejsou mimo privátní síť dostupné. Aby bylo možné rozlišit jednotlivé typy adres, každý typ může nabývat určitého rozsahu hodnot.

Níže jsou vypsány rozsahy hodnot pro neveřejné IP adresy.

- 10.0.0.0 až 10.255.255.255,
- 172.16.0.0 až 172.31.255.255,
- 192.168.0.0 až 192.168.255.255,
- 100.64.0.0/10.

Jakmile přijde na server nějaký požadavek, jediný dostupný údaj je veřejná IP adresa, se kterou se daný uživatel připojil k internetu. Tuto adresu ale může sdílet více uživatelů a v tomto případě pak není jasné, kdo konkrétně požadavek zaslal. Zablokování veřejné IP adresy by poškodilo všechny uživatele, kteří sdílí stejnou veřejnou IP adresu, což je nežádoucí. Proto je potřeba zjistit, kdo konkrétně požadavek zaslal. Jednotlivé uživatele, kteří se připojují do systému přes veřejnou IP adresu lze rozlišit pomocí privátní IP adresy. Kombinací přihlašovacího jména a IP adres lze docílit přesného určení uživatele, který se pokouší komunikovat se systémy nebo webovými stránkami.

Většina domácích směrovačů používá typ `Network Address Translation` (dále jen NAT) `one-to-many`, což umožňuje zařízením v privátní síti přistupovat na internet přes jedinou veřejnou IP adresu. Díky tomu je uživatel více zabezpečen, jelikož do internetu je vystavena jen veřejná IP adresa. Také to řeší problém s nedostatkem IPv4 adres, kterých je i tak velmi málo „nepoužitých“. Existuje IPv6 adresa, která se skládá ze 128 bitů a nabízí 2^{128} unikátních kombinací. NAT však proces přechodu na IPv6 zpomalil díky využívání jediné veřejné IP adresy více uživateli.

Při komunikaci směrovače s internetem, směrovač příchozí a odchozí pakety upravuje podle směrovací tabulky, kde má uloženou jak privátní IP adresu konkrétního zařízení, tak

[1] 8 bitů.

i veřejnou a podle toho, zda je paket příchozí nebo odchozí, nahrazuje jeho IP adresu. Zasahuje pouze do hlaviček paketů, nikoliv do samotného obsahu paketu. Určitou nevýhodou je určení konkrétního zařízení/uživatele například z bezpečnostních důvodů. Za normálních okolností by nemělo být možné z internetu nebo obecně mimo privátní síť zjistit privátní adresu zařízení. Toto chování je ovšem v pořádku a díky tomu NAT chrání uživatele v privátní síti.

2.2 WebRTC

Naštěstí má většina prohlížečů integrováno technologii `WebRTC` [11], která slouží k video hovorům, běžným hovorům a sdílení souborů mezi dvěma prohlížeči. Tato technologie je volně dostupná od roku 2011.

Hlavními komponenty jsou:

- getUserMedia - Umožňuje prohlížeči přístup k mikrofonu a web kameře.
- RTCPeerConnection - Vytvoří spojení, díky kterému se můžou provádět audio a video hovory.
- RTCDataChannel - Umožňuje prohlížeči sdílet data `peer-to-peer`[2].
- getStats - Získání statistik z proběhlých spojení.

Pro běžné uživatele má tato technologie dvě hlavní výhody:

1. Jednoduchost - možnost provádění hovorů a sdílení bez přídavných pluginů a jiných zásahů do prohlížeče.
2. Bezpečnost - nabízí zatím nejvyšší ochranu pro online hovory.

Tato technologie je v dnešní době podporována v prohlížečích Mozilla Firefox, Chrome, Opera a v mobilních verzích těchto prohlížečů. Jsou zde ale také bezpečnostní slabiny, které umožňují přístup k citlivým údajům uživatelů, jako například veřejná a privátní adresa. Všechny tyto operace jsou přenášeny `Sip`[3] pakety, které směrovač neumí upravit. Aby byla možná komunikace mezi směrovači, které používají NAT technologii, vznikl protokol `STUN`, který obchází toto routování a dává k dispozici privátní adresu druhé straně při hovoru. Jelikož STUN nebyl dostačující, vznikl protokol `ICE`, který používá předchozí protokol jako nástroj. Přes ICE se zasílá žádost na STUN server, který pak následně vrací ICE odpověď, která obsahuje IP adresy, porty a další informace o síti, ve které se nachází uživatel. Tato hodnota se běžně předává druhému účástníkovi hovoru nebo jiných interakcí, které WebRTC podporuje. Jelikož ve skutečnosti neprobíhá žádný hovor, vrací server zpět tyto informace na prohlížeč, odkud pochází požadavek.

Tímto způsobem lze získat privátní IP adresy uživatelů, kteří používají výše zmíněné prohlížeče. Kombinace veřejné a privátní IP adresy umožňuje jednoznačně určit, z jakého zařízení v privátní síti byl proveden útok a zabránit mu v budoucím přístupu. Tato technologie má bohužel i nevýhody. Je přítomna jen v některých prohlížečích, navíc jde tato

[2] Označováno také jako klient-klient, kdy mezi sebou komunikují klienti a nikoliv přes server

[3] protokol pro signalizaci při online hovorech, her a dalších online multimediálních akcích. Více na `http://www.voip-info.org/wiki/view/Sip`.

technologie vypnout a pokud bude útočník útočit například z příkazové řádky, je tento postup nefunkční. Tuto metodu získávání privátních IP adres používal v minulosti například `New York Times`[4], aby odlišil uživatele od strojů, které zasílaly nevyžádanou poštu (spam).

2.3 OSSEC

V této podkapitole byly informace čerpány z [1] a [12].

Proces získávání privátní a veřejné IP adresy uživatele se provádí při každém vstupu na webovou stránku, ale to, zda se tato informace použije nebo nikoliv, závisí na zjištění, jestli se jedná o útok nebo jen uživatel zadal špatné přihlašovací jméno a heslo. `Open Source HIDS SECurity` (dále jen OSSEC) slouží ke zpracování záznamů produkovaných jednotlivými systémy. OSSEC architektura se dělí na agenty a centrální server. Agenti sbírají záznamy ze souborů, které jednotlivé systémy generují a zasílají je centrálnímu serveru na analýzu. Ten se na základě výsledku zpracování záznamu rozhodne, zda je potřeba vygenerovat varování nebo ne. OSSEC je volně dostupný nástroj, který svými možnostmi a nezávislostí na platformě konkuruje i těm nejlepším placeným řešením.

Základní fakta:

- centralizovaná služba pro analýzu logů od agentů,
- systém varování,
- mechanismy pro proaktivní odpovědi,
- monitorování souborových systémů,
- rychlý, vytvořen v jazyce C,
- soubor pravidel.

Tato služba funguje jako klient-server, kde klienti jsou agenti na jednotlivých serverech, kteří monitorují soubory se záznamy o aktivitě jednotlivých systémů. Tyto soubory se záznamy pak zasílají na server, který data ze souborů analyzuje a rozhodne, zda se jedná o útok nebo běžnou aktivitu. OSSEC nabízí také proaktivní ochranu, která v případě aktivace pravidla provede požadovanou akci. OSSEC nabízí několik naimplementovaných proaktivních skriptů, které jsou napsány v `bash`[5]. Například přidají příchozí IP adresu do firewallu[6], aby byla zablokována. Implementace například síťové komunikace se serverem by bylo v tomto jazyku složité a pozdější úpravy by nebyly triviální. Proto v této práci nebyla proaktivní ochrana využita.

Jednotlivá varování se dělí do 16 úrovní a podle toho se také odvíjí odpovídající reakce. Do úrovně 6 se většinou ignorují, vyšší jsou například zasílány mailem s kompletními informacemi o původu hrozby, času a dalších užitečných informací. Tyto akce spadají pod pasivní obranu, kdy je administrátor upozorněn na podezřelou aktivitu na systému, a pak je na něm, aby provedl příslušná protiopatření.

[4] Odkaz:`http://www.webrtc-solutions.com/topics/webrtc-solutions/articles/406851-webrtc-use-new-york-times-creates-privacy-hubbub.htm`.

[5] Slouží jako implicitní příkazový jazyk v operačních systémech založených na linuxovém jádře. Více na `https://www.gnu.org/software/bash/`.

[6] Firewall je software, který spravuje příchozí a odchozí síťovou komunikaci. Více na `http://windows.microsoft.com/cs-cz/windows/what-is-firewall#1TC=windows-7`

OSSEC může být implementován jako klient-server, kdy je jeden centrální server, který sbírá všechny záznamy a analyzuje je a provádí všechny rozhodnutí o následných akcích. Klient je agent, který sleduje daný server a zasílá záznamy centrálnímu serveru.

Další možností je OSSEC implementovat lokálně, kdy je vše na jedné stanici/serveru. Tento způsob implementace se hodí, pokud je potřeba spravovat jen jeden konkrétní systém. Výhoda je v lehké údržbě a správě celého systému a přizpůsobení. Nevýhodou je nekompatibilita s klient-server řešením a je pak potřeba odinstalovat tuto verzi a nainstalovat agenta.

2.4 Logstash

`Logstash` [7] je volně dostupný nástroj pro sběr, filtrování a přeposílání záznamů z různých zdrojů dalším nástrojům nebo do databází. Soubor obsahující záznamy má většinou příponu `.log` a obsahuje informace o činnostech daného systému. Tyto záznamy mohou mít různorodé formátování a není vždy jednoduché získat užitečná data. Nejvíce rozšířeným formátem je `syslog`[7]. Řešením různých formátů záznamů je právě Logstash, který všechny tyto formáty zpracuje a na výstup dodá jen ty informace, které považujeme za důležité.

Filtrování je prováděno pomocí `grok` [7], což je nástroj na zpracování záznamů. Tento nástroj podporuje i regulární výrazy a umožňuje vytvořit jakýkoliv filtr, který je zrovna potřeba. Je velmi lehké nadefinovat, které hodnoty jsou důležité, a které naopak mají být zahozeny.

Formát zápisu vypadá následovně: `%{SYNTAX:SEMANTIC}`, kde `SYNTAX` je určitý vzor, například vzor IP by odpovídal datům, které by obsahovali IP adresu. Při instalaci Logstash obsahuje kolem 120 vzorů, které se dají použít. V případě, že žádný není vyhovující, je možné vytvořit vlastní. `SEMANTIC` je název proměnné, ve které bude tato vyfiltrovaná hodnota uložena. Díky podpoře regulárních výrazů se dají filtrovat různorodé záznamy, kdy například každý záznam má přihlašovací jméno uživatele na jiné pozici v záznamu. Místo definování několika filtrů pro všechny možné kombinace, lze elegantně využít regulárních výrazů, které provedou správnou filtraci, pokud se kdekoliv v záznamu tato hodnota vyskytuje. Takto se zpracují záznamy ze všech příchozích zdrojů a jsou připraveny k přeposlání.

2.5 Elasticsearch

Data se následně přeposílají do `Elasticsearch` [4], což je databáze bez specifikovaného schématu, díky čemuž nepotřebuje definovat svoji strukturu, ale vytvoří si ji sama na základě příchozích dat. Umožňuje velmi rychlé vyhledávání, je dobře škálovatelná a nabízí `REST` (teorie v podkapitole 2.9) rozhraní, které usnadňuje komunikaci z jiných systémů. Také podporuje textové vyhledávání, které je velkým přínosem při vyhledávání konkrétních informací. Kromě těchto vlastností byl hlavním důvodem vybrání této databáze také fakt, že tato technologie se již používá ve firmě a integrace, správa a rozšiřitelnost budou jednoduché.

Díky REST rozhraní není potřeba vytvářet žádné konektory a stačí jen zaslat http požadavek s tělem ve formátu `JSON`[8]. Elasticsearch podporuje rozložení zátěže na více serverů,

[7] Formát záznamů aktivit systému, kdy na každém řádku je samostatný záznam.

[8] Odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Více na `http://www.json.org/json-cz.html`.

které pak tvoří celek nazvaný `cluster`[9]. Tato vlastnost je hojně využívána ve firmách, které mají mnoho systémů a záznamy čítají dohromady desítky gigabajtů. Mít všechna data uložená v jedné databázi by bylo neefektivní, a proto se využívá škálovatelnosti a rozdělení záznamů jednotlivých systémů mezi více databází. Tím dochází ke snížení celkové zátěže a zrychlení doby vyhledávání.

Další důležitou věcí je index. Index označuje kolekci dat, které mají společné vlastnosti. Díky indexu je vyhledávání, úprava nebo mazání dokumentů rychlejší. Uvnitř indexu lze rozlišovat typ dat. Díky tomu lze používat jeden index pro více typů dokumentů. Jelikož pod stejným indexem mohou být uloženy stovky záznamů, podporuje Elasticsearch také rozdělení indexů mezi více databází v rámci škálování.

2.6 Kibana

Kibana [6] běží nad Elasticsearch a slouží k vizualizaci dat. Díky tomu lze vytvářet různé náhledy, statistiky, grafy, nástěnky, apod. Aby mohla Kibana mít přístup k datům, je potřeba definovat index, podle kterého se bude vyhledávat v databázi. Těchto indexů může být více a pro každý může být vytvořena vlastní sada grafické reprezentace dat. Kibana je `open-source`[10] rozšíření Elasticsearch [8] a usnadňuje správu serverů a vytváření statistik. Vyhledávat lze podle jakéhokoliv parametru, který se v databázi vyskytuje. Pokud je podle konkrétního parametru vyhledáváno poprvé, vytvoří se index podobný z běžných relačních databází pro rychlejší vyhledávání.

název parametru	popis
server.port	port, na kterém bude Kibana dostupná
server.host	adresa, na které bude Kibana dostupná
server.maxPayloadBytes	maximální velikost příchozích dotazů
Elasticsearch.url	adresa Elasticsearch
kibana.index	index, pod kterým bude Kibana ukládat výsledky dotazů
Elasticsearch.username	přihlašovací jméno do Elasticsearch
Elasticsearch.password	heslo do Elasticsearch
server.ssl.cert	cesta k SSL certifikátu
server.ssl.key	cesta k SSL klíči

Tabulka 2.1: jednotlivá nastavení Kibany. Převzato z [6].

2.7 ElastAlert

`ElastAlert` [9] je další rozšíření pro Elasticsearch [4], ve kterém lze vytvořit pravidla, která v případě splnění definovaných podmínek generují varování. Tyto pravidla jsou napsána v `yaml`[11] formátu. Mezi hlavní rysy patří čitelnost nejen strojem, ale také i člověkem, neomezené úrovně zanoření, podpora pouze mezer a struktura je řešena předsazením.

Pravidla mají několik povinných parametrů, které se nacházejí v tabulce 2.2.

[9] Skupina spolupracujících serverů

[10] Software, který je volně dostupný včetně zdrojových kódů.

[11] Tento formát slouží pro serializaci strukturovaných dat. Více na `http://yaml.org`.

název parametru	popis
es_host	adresa serveru s databází
es_port	port serveru
index	index podle kterého se bude vyhledávat
name	název pravidla
type	typ pravidla
alert	druh varování

Tabulka 2.2: povinné parametry nastavení ElastAlert. Převzato z [9].

Parametr `type` určuje typ pravidla. ElastAlert má několik vzorových pravidel a pokud je potřeba využít pravidlo s jiným chování, je možnost vytvořit vlastní podle potřeby. V této práci nebylo potřeba vytvářet nový typ a byl použit již vytvořený `Frequency`. Tento typ vyžaduje další 2 povinné parametry, a to `num_events`, což je počet událostí, který je potřeba pro aktivaci tohoto pravidla, a `timeframe`, který určuje dobu, za kterou musí nastat počet specifikovaných událostí [9].

`alert` parametr specifikuje druh varování, které bude provedeno po aktivaci pravidla. V tomto projektu bylo naimplementováno nové, které provádí uložení záznamu do DNS zóny skrze java aplikaci. Všechna vygenerovaná varování jsou následně ukládány zpět do Elasticsearch a je pak možné v Kibaně pozorovat, kdy a kdo se pokoušel například o přihlášení se špatným heslem. Tyto informace se pak dají převést do grafu nebo jiných vizualizací dat.

2.8 DNS BlackList

`Domain Name System BlackList` (DNSBL) [3] je zóna v DNS serveru, která obsahuje záznamy s ip adresami, kterým má být zamezen přístup do systémů. Jakmile ElastAlert zjistí, že se uživatel pokouší přihlásit s nevalidním heslem vícekrát, je potřeba tohoto uživatele uložit do černé listiny. Černá listina obsahuje objekty, kterým má být odepřen přístup do systémů. Pokud tedy příchozí požadavek obsahuje prvek, který je součástí černé listiny, dojde k odepření přístupu. Zóny umožňují ukládat několik typů záznamů. IP adresy se ukládají do záznamu typu `A`[12] a `AAAA`[13] podle typu IP adresy. Velkou výhodou je implicitní kontrola, zda ukládaná hodnota splňuje formát daného typu záznamu. Textový záznam umožňuje uložit jakýkoliv řetězec, kromě speciálních znaků [10]. Další typy nebudou v projektu použity.

[12]Do tohoto typu záznamu se ukládá IP adresa IPv4.
[13]Do tohoto typu záznamu se ukládá IP adresa IPv6.

2.9 Spring Boot

Tento java framework se používá při tvorbě webových aplikací s využitím návrhového vzoru `Model View Controller` (dále MVC), který rozděluje aplikaci na 3 části:

- model,
- view,
- controller.

Díky tomuto návrhovému vzoru se dají provádět změny na jednotlivých částech s minimálním dopadem na ostatní části. View je většinou grafické rozhraní, se kterým pracuje uživatel. Pokud chce získat data, zašle požadavek na `controller`, který rozhodne, zda je tento požadavek autorizovaný a provede požadovanou operaci nad daty v modelu. Výsledek této operace je pak zaslán zpět na grafické rozhraní. Tento návrhový vzor v jazyce java řeší právě Spring Boot.

Spring Boot [5] je nadstavba Spring frameworku a jeho hlavním přínosem je konfigurace, která již neprobíhá formou xml souborů, ale přímo v kódu a také integrovaný server, na kterém aplikace běží. Není tedy potřeba exportovat projekt jako war[14] soubor, který by se nasazoval na server, ale stačí vytvořit java balík, který lze kdekoliv spustit a poběží jako server. Pro vytvoření java balíku používá Spring Boot nástroj `Maven`, který velmi zjednodušuje proces kompilace a vytváření java balíků.

Základní funkcí nástroje Maven je popsání aplikace pomocí `Project Object Model`. Tento soubor obsahuje všechny důležité informace o projektu. Například na jakých externích knihovnách je závislý, jaké testy se mají spustit předtím, než je vytvořen výsledný balík, jakého typu má být výsledný balík a podobně. Maven je modulární a pracuje na úrovni pluginů. Sám žádné pluginy neobsahuje a chová se jako správce, který jen volá nadefinované pluginy. Nemá žádné grafické rozhraní a díky tomu jsou plně kompatibilní všechny pluginy, které podporují standardní vstup.

V tabulce 2.3 je zobrazena struktura jednotlivých adresářů, které by měla každá aplikace využívající Maven dodržovat.

Kořenový adresář	v tomto adresáři se nachází pom soubor
src/main/java	zde jsou všechny java soubory
src/main/resources	zde jsou konfigurační soubory
src/test/java	obsahuje unit testy
src/test/resources	konfigurační soubory pro testy

Tabulka 2.3: Struktura Maven projektu. Převzato z [2].

V případě potřeby se dá toto nastavení změnit, ale je potřeba podle toho také upravit pom soubor. Pom soubor má formát XML[15] a obsahuje informace o všech externích knihovnách, potřebné pluginy pro vytvoření balíku apod. Pokud se projekt skládá z více podprojektů, každý obsahuje vlastní pom soubor, který může základní informace dědit z kořenového pom souboru. Díky dědičnosti může být vytvořen balík s velmi náročnými

[14] Formát souboru, který se používá při distribuci java webové aplikace. Více na `https://spring.io/understanding/WAR`.

[15] Obecný značkovací jazyk sloužící pro serializaci dat.

požadavky jediným příkazem, jelikož jednotlivé pom soubory na sebe navazují. Maven nabízí také provedení jen určitých fází procesu vytvoření výsledného balíku. Může se použít jen zkompilování zdrojových souborů bez spouštění testů či smazat vyrovnávací pamět, aby nedošlo k chybám v důsledku starých informací uložených v této paměti při kompilaci.

Díky těmto vlastnostem je i samotný vývoj takové aplikace rychlejší. Maven dokáže externí knihovny stáhnout, pokud je nalezne v hlavním nebo v explicitně definovaném repositáři. Tato vlastnost zaručuje přenositelnost takové aplikace a není potřeba složitě stahovat jednotlivé knihovny, ale jen spustit příkaz `mvn install`, a všechny knihovny se nainstalují sami.

Webové aplikace, které používají tento framework, implementují `Representational State Transfer` (dále jen REST) API[16] rozhraní, které se využívá při komunikaci s aplikací formou http požadavků. Obecně je REST rozhraní orientováno na zdroje místo volání procedur jako tomu je u `SOAP`[17] protokolu.

Každý zdroj má vlastní unikátní identifikátor (URI) a přes požadavek se správnou adresou je možné získat, upravit nebo smazat konkrétní zdroj. Pokud je potřeba zdroj vytvořit, upravit nebo smazat, slouží na to metoda POST. Tato metoda obsahuje kromě klasické hlavičky také tělo, které se pak předá aplikaci, a ta ho zpracuje. Metoda GET slouží ke čtení dat bez jakýchkoliv úprav. Podrobněji je spring-boot popsán v knize [5].

2.10 Knihovny

Pro komunikaci s aplikací budou využívány knihovny, které budou implementovány v jazycích `java`, `php` a `node.js`. Jakýkoliv systém napsaný v jednom z výše zmíněných jazyků bude tak moci během několika okamžiků komunikovat s aplikací bez nutnosti psaní vlastních metod či nutné znalosti samotného kódu aplikace.

[16] Rozhraní, které určuje jakým způsobem je možné komunikovat s aplikací.
[17] Protokol pro výměnu zpráv typu XML.

Kapitola 3

Návrh

Projekt se bude skládat ze dvou hlavních částí:

- Aplikace pro komunikaci s DNS.
- Zpracování záznamů a vygenerování varování.

3.1 Návrh aplikace

Aplikace bude implementována v jazyce java s frameworkem `spring-boot` (Více v sekci 2.9), díky kterému bude snadné vytvořit REST aplikaci s rozhraním, na které budou systémy pomocí vytvořených knihoven zasílat požadavky o přidání záznamu do DNS, smazání záznamu a kontrolu, zda se záznam nachází v dané zóně. Tento framework je jeden z nejrozšířenějších pro tvorbu webových aplikací. Je také důležité zajistit ochranu proti vnějšímu okolí, aby nemohl kdokoliv komunikovat s aplikací. Pro zabezpečení komunikace s aplikací bude použito asymetrické kryptografie.

To znamená, že na jedné straně bude privátní klíč, který bude používán pro podpisy komunikace, a na druhé straně bude veřejný klíč, kterým se tento podpis bude ověřovat. Velikost klíče je volitelná, ale některé klíče s menší velikostí již byly prolomeny, a proto nejsou považovány za bezpečné. V dnešní době je považováno za bezpečnou velikost 2048 bitů. Díky tomuto bezpečnostnímu opatření bude komunikace bezpečná, ale může mít dopad na rychlost komunikace mezi aplikacemi, jelikož podepsání a ověření podpisu je náročné na výpočetní výkon. Je proto nutné uvážit, zda v reálném provozu bude tento typ šifrování komunikace výhodný.

3.1.1 Návrh API

Aplikace bude podporovat několik operací, které budou namapovány na příslušné metody v řadiči. Aby byl příchozí požadavek platný, musí odpovídat některé z nadefinovaných cest ke zdrojům. Pokud tedy například bude chtít systém zkontrolovat konkrétního uživatele, musí zaslat požadavek typu GET[1]. Adresy pro kontrolu uživatelů budou obsahovat hodnoty IP adres a přihlašovacího jména uživatele, aby aplikace věděla, na jaká data se má dotazovat DNS serveru. Aplikace podporuje přidání všech kombinací přihlašovacího jména a IP adres do DNS zóny, jejich odebrání a kontrolu a také adresu pro získání tokenu[2] ke komunikaci

[1] Tento typ požadavku neobsahuje žádné tělo a slouží k získání dat ze serveru/aplikace

[2] Token slouží k ověření identity. Token obsahuje podpis od aplikace a další volitelné parametry

s aplikací. Jednotlivé hodnoty z adres se budou ukládat do parametrů, které se pak budou předávat konkrétním metodám pro práci s daty [5].

Všechny dostupné cesty jsou znázorněny v tabulce níže.

adresa	popis
server/v1/{login}/{veřejná IP}/{privátní IP}/checkUser	kontrola loginu a IP adres
server/v1/{login}/{veřejná IP}/checkPublicIP	kontrola loginu a veřejné IP
server/v1/{login}/checkLogin	kontrola loginu
server/v1/{login}/{veřejná IP}/privátní IP/banUser	přidání loginu a IP adres do DNS
server/v1/{login}/{veřejná IP}/banPublicip	přidání veřejné IP do DNS
server/v1/{login}/banLogin	přidání loginu do DNS
server/v1/{login}/{veřejná IP}/{privátní IP}/unbanUser	smazání všech záznamů loginu z DNS
server/v1/{login}/{veřejná IP}/unbanPrivateip	smazání privátní IP z DNS
server/v1/getToken	získání tokenu pro komunikaci

Tabulka 3.1: API java aplikace

3.2 Návrh zpracování záznamů

OSSEC poběží v rámci této práce lokálně, ale v praxi bude jeden server a několik agentů, kteří budou zasílat data na centrální server. Server bude obsahovat dekodér, který bude zpracovávat a filtrovat specifické informace a jejich hodnoty ukládat do proměnných. Toto zpracování bude probíhat pomocí regulárních výrazů a použití několika dekodérů. Dále bude vytvořeno pravidlo, které bude používat právě tyto dekodéry a pokud dekodér konkrétní informace nalezne, spustí se pravidlo a vygeneruje do záznamového souboru varování. V tomto souboru už nebude jen samotný záznam, který toto varování způsobil, ale také proměnné, ve kterých budou uloženy údaje o uživateli jako jeho přihlašovací jméno, veřejná a privátní IP adresa. Tento soubor se bude odesílat pomocí protokolu `UDP`[3] na další zpracování do Logstash, kde bude tento soubor analyzován a následně vyfiltrován pomocí grok. Jelikož už OSSEC předpřipravil důležité informace, grok je zpracuje a uloží do proměnných, které se budou pak ukládat do Elasticsearch. Nad Elasticsearch poběží grafické rozhraní Kibana, které bude vizualizovat uložená data na základě indexů. Tento návrh zpracování záznamů je popsán na stránce [8]. Jako doplňek Elasticsearch poběží ElastAlert, který bude jednou za čas kontrolovat data v databázi, a pokud dojde ke splnění podmínek pro spuštění pravidla, vygeneruje se varování, které zašle na java aplikaci požadavek o přidání uživatele do DNS zóny.

Zde mohou nastat 3 případy:

- login, veřejná IP adresa a privátní IP adresa,
- login a veřejná IP adresa,
- veřejná IP adresa.

V případě, že uživatel nebude používat Chrome, Firefox nebo Operu, tak bude k dispozici jen přihlašovací jméno uživatele a veřejná IP adresa a v některých případech bude

[3]UDP protokol je bezstavový a nezaručuje doručení, dochází tedy ke ztrátám během přenosu

dostupná jen veřejná IP adresa. Varování podle toho, kolik dostalo parametrů, zasílá konkrétní požadavek na java aplikaci.

Pokud při reálném použití bude síťový provoz příliš vysoký, může se tento návrh upravit tak, aby bylo více zón v DNS serveru, a tím došlo ke zrychlení komunikace mezi java aplikací a serverem.

Tento návrh má reálné využití pro větší společnosti, které nechtějí platit za velmi drahé komerční řešení. Jelikož jsou OSSEC, Logstash, Elasticsearch a ElastAlert `open-source`, jsou cenové náklady na tuto formu obrany velmi nízké a jedná se v podstatě jen o zařízení kvalitní infrastruktury. Výsledná struktura je zobrazena na obrázcích 3.2 a 3.1.

3.3 Licence

V této práci je použito několik nástrojů, které podléhají licencím. V této sekci budou jednotlivé licence stručně zmíněny a také budou všechny obsaženy na DVD v souboru `LICENCES`.

3.3.1 Nástroje

OSSEC podléhá licenci `GNU GPL Version 2`. Logstash podlé licenci Apache Version 2.0. Elasticsearch, Kibana a ElastAlert také podléhají licenci Apache Version 2.0.

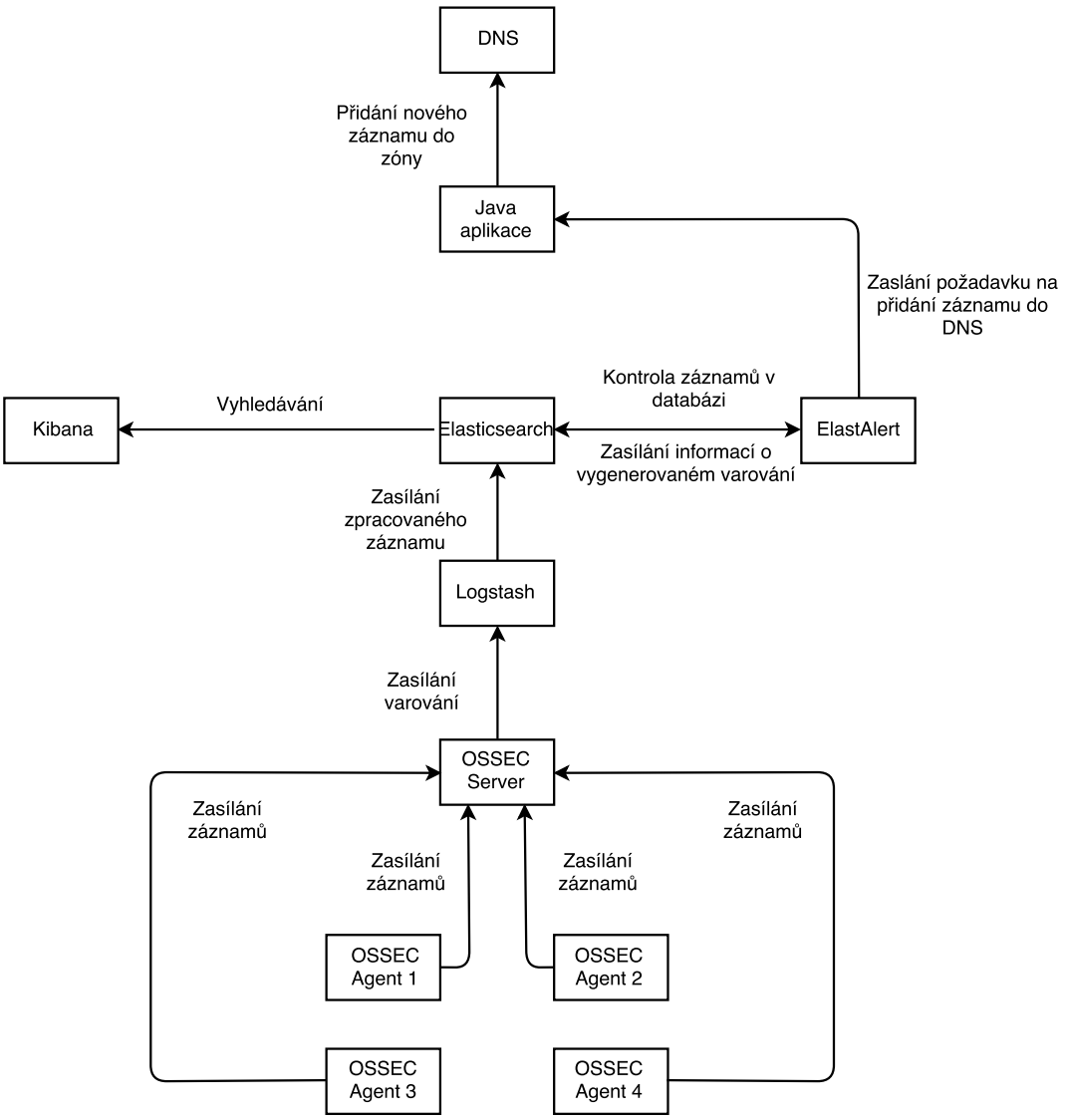
3.3.2 Knihovny

V této práci budou použity následující knihovny:

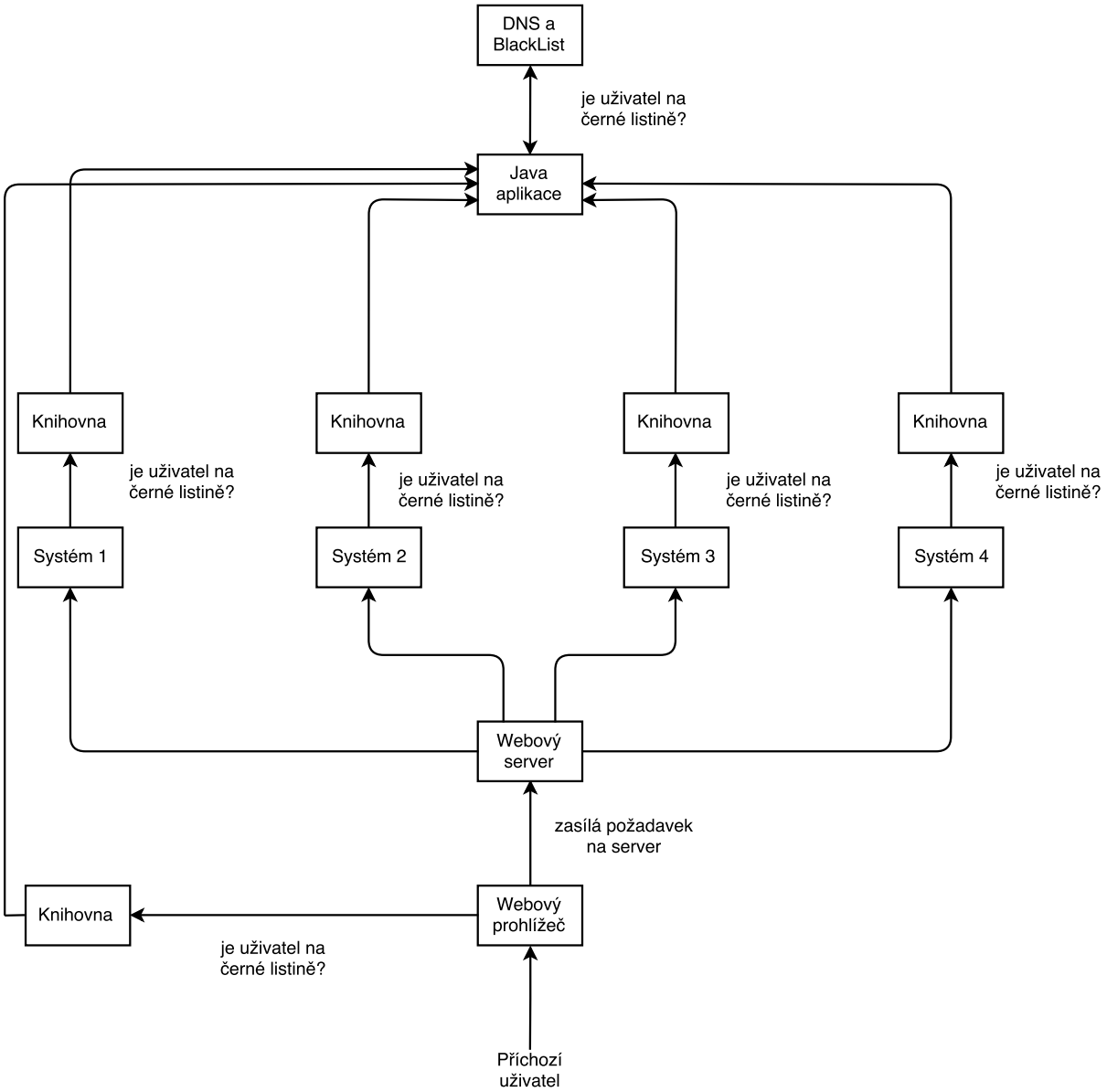
- dnslibrary,
- curl,
- requests,
- request.

`Dnslibrary` bude použita v java aplikaci pro komunikaci s DNS serverem. Díky této knihovně bude možné přidávat, mazat, upravovat a vyhledávat v DNS zónách. Tato knihovna podléhá licenci typu `Apache license`, Version 2.0.[4] `Curl` je knihovna pro programovací jazyk PHP, která umožňuje jednoduchou tvorbu http požadavků. Tato knihovna podléhá licenci, která se nachází na oficiálních stránkách[5]. `Requests` je knihovna pro skriptovací jazyk `python` a slouží také k jednoduché implementaci http požadavků. Knihovna podléhá také licenci Apache Version 2.0. `Request` je knihovna podobná té pro python, ale je pro jazyk `Node.JS`. Podléhá licenci Apache Version 2.0.

[4] Dostupné na `http://www.apache.org/licenses/LICENSE-2.0`
[5] Dostupné na `https://curl.haxx.se/docs/copyright.html`.



Obrázek 3.1: Návrh zpracování záznamů a uložení do DNS



Obrázek 3.2: Návrh systému pro kontrolu existence záznamů v DNS

Kapitola 4

Implementace

Implementační část je rozdělena na dvě části. První je implementace java aplikace a druhá nastavení systémů pro práci se záznamy a generování varování, které bude inicializovat komunikaci s java aplikací a předávat jí informace o uživatelích, kteří budou přidáni do černé listiny.

4.1 Java aplikace

Pro přístup do DNS serveru, provádění dotazů a přidávání nebo odebírání záznamů jsem zvolil jazyk java s frameworkem `Spring Boot` [5]. Cílem bylo naimplementovat aplikaci s webových rozhraním REST, aby bylo možné z jakéhokoliv systému, pokud bude mít oprávnění, komunikovat s touto aplikací, například i z příkazové řádky.

4.1.1 Controller

Hlavní částí aplikace je řadič (controller), který zpracovává požadavky a provádí následně potřebné akce nad daty (modelem). Tyto akce závisí na cestě ke zdrojům. Tyto cesty byly implementovány podle API 3.1. Předtím, než se určitý http dotaz dostane do řadiče, musí projít přes `Interceptor`, který umožňuje provést interakci s příchozím http dotazem dřív, než se dostane do řadiče. Tato vlastnost byla v projektu využita pro kontrolu hlavičky dotazu, zda obsahuje požadovaný podpis (signature). Tento podpis se kontroluje u dotazů, které směřují v řadiči na metody, které obsahují vytvořenou označovací(mark) anotaci. Podpis je tvořen pomocí asynchronního šifrování, kdy je vytvořena dvojice veřejný a privátní klíč. Podpis se vytváří pomocí privátního klíče a ověřování se provádí pomocí veřejného klíče. Tím je zajištěno, že aplikaci volá autorizovaný systém, který má správný podpis. Při prvním spuštění aplikace se kontroluje, zda existují tyto klíče a pokud ne, aplikace je sama vygeneruje a uloží. Velikost byla zvolena 2048 bitů, jelikož výpočetní síla dnešních počítačů je již schopna prolomit menší velikosti. Aby bylo možné získat podpis, existuje v řadiči metoda, která vytvoří podpis podle aktuálního privátního klíče a vrátí ho žadateli jako řetězec znaků. V rámci této práce není vyžadována žádná autorizace pro získání podpisu. V praxi ovšem bude prováděna kontrola příchozího dotazu, zda obsahuje podpis od důvěryhodného systému.

Pokud tedy dotaz obsahuje požadovaný podpis a jde rozšifrovat pomocí uloženého veřejného klíče, je tento podpis uložen do atributu požadavku a pak díky třídě `ArgumentResolver` uložen do parametru typu `Token`, která je označena anotací a dotaz pokračuje do řadiče. ArgumentResolver slouží k předání dat z příchozího http požadavku do parametru metody.

Aby Spring rozeznal, jaké data má uložit do parametru, volá při zpracování požadavku argumentResolver, kde zjistí, že do parametru typu Token se má předat atribut požadavku s názvem „token“. Aby tento resolver fungoval, je potřeba ještě v konfigurační třídě tento resolver přidat k listu resolverů, které aplikace zná. Kromě resolveru je ještě použita značkovací anotace, která říká, že do tohoto parametru je potřeba uložit podpis. V rámci této práce, je do tokenu uložen jen podpis, ale v praxi se do tokenu ukládají důležité informace o uživateli nebo systému. Pokud chce vnější systém nebo uživatel vědět, zda se konkrétní uživatel nachází v DNS, je potřeba zaslat požadavek se specifickou `URL` neboli řetězcem znaků o přesně dané posloupnosti, který určuje adresu serveru, protokol, jednotlivé zdroje a akci. Například `http://localhost:8081/v1/testlogin/15.25.15.20/15.6.4.8/checkAll` říká, že je použit protokol `http`, adresa serveru je `localhost`, který běží na portu `8081` a požaduje kontrolu, zda uživatel s tímto přihlašovacím jménem, veřejnou a privátní IP adresou se nachází v DNS zóně. Díky anotacím `@PathVariable` dokáže Spring vytáhnout tyto informace a uloží je do proměnných, které se pak předávají službě, která provádí potřebné dotazy nad DNS. Pokud provádění akcí v servise skončí vyjímkou, je odchycena třídou `ExceptionHandler`, která jakoukoliv neodchycenou vyjímku zpracuje a vrací zpět kód 500, což je Internal Server Error[1].

4.1.2 DNSService

Pro komunikaci s DNS serverem jsem zvolil knihovnu `dnsjava`, která umožňuje komunikaci s DNS. Použity byly třídy pro specifické typy záznamů, konkrétně pro záznam typu `A` a `TXT`. Předtím, než je možné získat tyto záznamy, je potřeba vytvořit novou instanci rozhraní `resolver`[2] a ještě třídu `lookup`, která zajišťuje samotný dotaz. Při přidávání záznamů do DNS se nejdříve aplikace dotazuje, zda daná zóna existuje. Pokud ne, vrací aplikace kód 404[3], který pak řadič zašle zpět. Pokud je daná zóna nalezena, vytvoří se nové záznamy, které se uloží do instance třídy `update`, která odpovídá příkazu `nsupdate`. Přidává se záznam typu A, k tomu také textový záznam, ve kterém je uložen čas vytvoření záznamu a další textový záznam, do kterého se uloží hodnota privátní IP adresy a čas vytvoření záznamu. Tyto hodnoty jsou odděleny pomocí „:“. Následně je přes resolver zaslána žádost o aktualizaci zóny a pak se vrací kód odpovědi DNS serveru. 200[4] v případě, kdy vše proběhlo v pořádku nebo 500 pokud nastala interní chyba při pokusu o aktualizaci zóny.

V druhém případě aplikace zasílá požadavek na kontrolu, zda se tento uživatel nachází v DNS zóně. Nejdříve se provede dotaz na DNS server, zda existuje záznam s konkrétním přihlašovacím jménem. Pokud ne, vrací se ta samá hodnota jako při předchozím případu. V případě, že je záznam nalezen, je potřeba zkontrolovat, aby IP adresy souhlasily s těmi, které přišly s požadavkem. Při této kontrole se také provádí ověření, kdy byl záznam vytvořen. K této hodnotě je připočtena další, na jak dlouho má být uživateli zabráněn přístup do systému. Pokud už tento čas uplynul, dochází k smazání buď všech záznamů vázaných na konkrétní přihlašovací jméno nebo jen záznamu s privátní IP adresou. Doba, po kterou uživatel nesmí mít přístup, se liší pro veřejnou a privátní adresu. Jelikož může několik uživatelů sdílet stejnou veřejnou adresu, musí být tento čas u veřejné IP adresy delší. Jakmile je kontrola u konce, vrací opět výsledek operace řadiči.

[1] Interní chyba na straně serveru/aplikace.

[2] Soubor dynamických knihoven a rutin, který má za úkol provést dotaz na DNS server a po obdržení odpovědi zaslat tuto informaci zpět klientovi. Více na `https://docs.oracle.com/cd/E19683-01/817-4843/ad1intro-12450/index.html`.

[3] Not Found. Požadovaná data nebyla nalezena

[4] Odpovídá OK

4.2 Login stránka

V rámci demonstrace projektu, byla vytvořena stránka na přihlášení uživatele v jazyce PHP, kde se kontroluje správnost hesla pouze v rámci imaginárního řetězce, jelikož tato stránka neběží nad žádnou konkrétní databází a heslo je pouze smyšlené a slouží k ukázce skutečného použití v praxi. Stránka se skládá ze dvou částí. První z nich je java script, který dokáže získat privátní a veřejnou IP adresu, pokud uživatel přistupuje na stránku přes prohlížeč Mozilla Firefox, Opera nebo Google Chrome. Tyto získané údaje se pak ukládají do `cookies`. Jakmile je odeslán formulář pro přihlášení, kontroluje se, zda se daný uživatel vyskytuje v DNS. Pokud ano, nedojde ani ke kontrole hesla a je mu zobrazena hláška „Banned“. V opačném případě se pokračuje ke kontrole hesla a pokud je správné, je uživatel přihlášen. Pokud zadá špatné heslo, uloží se do souboru záznam, který obsahuje přihlašovací jméno a IP adresy uživatele. Ke komunikaci s java aplikací se využívá knihovna `curl`.

4.3 Systém zpracování záznamů

Zpracování záznamů probíhá přes několik nástrojů zobrazených na obrázku 3.1. Každý nástroj záznam zpracuje a zasílá dalšímu nástroji. Výsledné generování varování a zaslání požadavku na java aplikaci provádí ElastAlert.

4.3.1 Získání IP adres uživatele

K získání IP adres byl použit převzatý zdrojový kód[5] v jazyce javascript. Nejdříve je nutné zajistit použití správného `RtcPeerConnection` podle toho, jaký prohlížeč uživatel používá. Poté je potřeba nastavit `RtpDataChannels` na hodnotu true. Tyto kanály jsou určeny pro přenos dat mezi prohlížeči přímo bez nutnosti mít server, na který by se oba prohlížeče dotazovali. Dále je třeba nastavit adresu STUN[6] serveru, na který se bude prohlížeč dotazovat. Pokud by tato adresa nebyla zadána, bylo by možné zjistit pouze privátní IP adresu uživatele. Poté se vytvoří nové RTC spojení a také nový data kanál. Poté se volá funkce `parseip`, která zpracovává `ICE Candidate`, který obsahuje informace o síti, v níž se uživatel nachází.

Z těchto informací je důležitá jen veřejná a privátní IP adresa, které jsou průběžně ukládány do pole. Jelikož je ICE Candidate více, některé IP adresy se mohou opakovat. Dochází tedy ke kontrole, zda tato IP adresa není již obsažena v poli. Celý tento proces spouští funkce `getUserips`, která asynchronně získá pole IP adres díky `callback` funkci. Poté se provádí iterace nad polem a podle rozsahu se určuje, zda se jedná o veřejnou nebo privátní IP adresu. Každá tato IP adresa je přiřazena do samostatné cookie, ke které pak přistupuje v php další funkce na zaslání dotazu na java aplikaci, zda tento uživatel existuje v DNS zóně. Proces získávání IP adres se provádí pokaždé, když uživatel přijde na stránku. Samotné zpracování těchto informací, vytvoření správné url a zaslání požadavku na java aplikaci, se provádí až po potvrzení formuláře pro přihlášení.

Formát vytvořené url vychází z API 3.1. V případě, že uživatel používá nepodporovaný prohlížeč, dojde k vytvoření záznamu pouze s přihlašovacím jménem a veřejnou IP adresou. K zasílání http požadavků o kontrolu existence záznamů se používá knihovna `curl`[7]. Jakmile

[5] Adresa git repositáře: `https://github.com/diafygi/webrtc-ips`.

[6] Server, který zajišťuje výměnu informací o sítích mezi prohlížeči.

[7] Dokumentace na adrese: `http://php.net/manual/en/book.curl.php`

je požadavek zpracován a java aplikace vrací kód odpovědi, dochází ke kontrole tohoto kódu a podle toho je uživatel buď přihlášen, nebo „zabanován“. Celý proces kontroly hesla a vypsání hlášek, zda je uživatel přihlášen nebo ne je pouze ilustrační. V praxi budou webové stránky již existovat a přidá se tam pouze knihovna na dotaz a java skript k získání IP adres.

4.3.2 OSSEC

OSSEC provádí prvotní kontrolu záznamů. Agenti zasílají soubory se záznamy jednotlivých systémů na centrální OSSEC server. Tento server pak tyto soubory prochází a pokud nalezne shodu, provede se zpracování pomocí dekodérů. Dekodér se spustí, pokud nějaký záznam splňuje určité předpoklady. V této práci je použit následující formát záznamů: `Timestamp POST 401 /login/auth login: hodnota publicIp: hodnota privateIp: hodnota`. Dekodér se pomocí funkce `prematch` spustí, pokud záznam obsahuje hodnotu `401 /login/auth`. Pokud se tedy uživatel pokoušel přihlásit se špatným heslem, server mu vrátil odpověď o neplatné autorizaci. Další dekodér dědí od předchozího a pomocí `regex offset="after_parent` pokračuje ve zpracování záznamu tam, kde minulý dekodér skončil. Po /login/auth následují informace o přihlašovacím jméně uživatele a jeho IP adresách. Aby bylo možné spustit dekodér i při chybějících adresách, byly vytvořeny další dekodéry se stejným jménem, ale každý zpracovává jinou informaci. Zpracování probíhá pomocí regulárních výrazů a následného uložení do proměnných, které OSSEC nabízí. Přihlašovací jméno je uložen do proměnné `user`, veřejná IP adresa do další proměnné `srcip` a privátní do `dstip`. Bohužel OSSEC nepodporuje vytvoření vlastních proměnných, a proto byly zvoleny nejvíce vhodné. Po tomto zpracování dojde k aktivaci obecného pravidla, které negeneruje varování, ale dojde k aktivaci dalšího pravidla, které provede vygenerování varování, pokud se ve zpracovaném záznamu nachází `/login/auth`. V praxi by těchto „pod-pravidel“ mohlo být více a každé by reagovalo na jiný typ neautorizovaného přístupu uživatele. OSSEC poté obsah souboru, do kterého se generují varování zasílá pomocí `UDP` protokolu dalšímu nástroji.

4.3.3 Logstash

Logstash slouží ke zpracování a přeposílání záznamů do databáze nebo do dalších nástrojů. Všechna nastavení se provádějí v jednom souboru, zde konkrétně v `logstash.conf`.

Tento soubor obsahuje 3 části:

- vstup,
- filtrování,
- výstup.

Vstup může být ze souboru, z konzole nebo z příchozí síťové komunikace. Jelikož OSSEC zasílá soubor pomocí UDP protokolu, Logstash naslouchá UDP protokolu s portem, který byl zadán u konfigurace OSSEC. Typ záznamů byl zvolen `syslog`. Jakmile je nastaven vstup dat, je potřeba provést filtrování.

Filtrování se provádí pomocí grok. Jelikož má grok předdefinované vzory, nebylo potřeba vytvářet nic nového a jen podle toho, zda daná část záznamu byla IP adresa, přihlašovací jméno nebo časové razítko, se uložila do správné proměnné. Ještě bylo potřeba přidat záznam, který obsahuje adresu serveru, ze kterého přišla data. Tato informace je užitečná pro statistiky nebo v Kibaně se dá podle ní filtrovat v případě, že by bylo více OSSEC serverů.

Několik informací, které se přidali při zasílání z OSSEC bylo odebráno pro jejich nevyužitelnost. Po filtrování jsou data předána na výstup, který je v tomto případě Elasticsearch databáze.

4.3.4 Elasticsearch

Nastavení Elasticsearch se provádí v souboru `Elasticsearch.yml`. Mezi hlavní možnosti patří:

název parametru	popis
cluster.name	název clusteru
node.name	název pro jednotlivé servery
path.data	cesta k adresáři pro ukládání dat
path.logs	cesta k adresáři pro ukládání záznamů

Tabulka 4.1: Parametry nastavení ElasticSearch. Převzato z [4].

Použito bylo jen nastavení jména clusteru, zbytek nastavení bylo ponecháno na výchozích hodnotách.

4.3.5 Kibana

Po nainstalování Kibany [8] bylo potřeba správně nastavit adresu databáze. Konfigurace se provádí v souboru `kibana.yml`, kde se nachází několik parametrů popsaných v tabulce 2.1. Nejprve se nastaví adresa a port Elasticsearch databáze. Po nastavení je možné se připojit v prohlížeči do grafického rozhrání Kibany a vytvořit `index pattern`, podle kterého Kibana identifikuje index v Elasticsearch a umožní tak provádět statistiky a vyhledávání. V této práci byl jako index využit `timestamp` jednotlivých záznamů. Druhý index byl použit z ElastAlert, který po vygenerování varování vrací výsledek do Elasticsearch.

4.3.6 ElastAlert

Při nastavování a integraci ElastAlert do Elasticsearch jsem vycházel z oficiální dokumentace a článku [9]. Hlavní nastavení se nachází v souboru `config.yaml`, ve kterém jsou parametry nastaveny následovně. Parametr `rules_folder` je ponechán na výchozím nastavení, parametr `run_every` je pro ilustraci nastaven na 5 minut. Další parametr `buffer_time` je nastaven na 3 minuty, jelikož běží vše lokálně a není důvod mít výsledky uloženy déle. Host a port jsou nastaveny stejně jako v nastavení Kibany. Přihlašovací jméno a heslo nejsou použity. Index, pod kterým ElastAlert ukládá výsledky do Elasticsearch je také ponechán na výchozí hodnotě, který je `ElastAlert_status`. Pod tímto indexem je pak možné v Kibaně sledovat počet varování, který byl vygenerován v určitém časovém úseku. `alert_time_limit` je také nastaveno na výchozí hodnotu. V praxi ovšem bude hodně záležet, jak velká tato hodnota bude. Příliš velká by měla za následek zahlcení linky v případě delšího výpadku sítě.

Dále je potřeba vytvořit samotné pravidlo. Pravidla se nastavují opět v souboru ve formátu `yaml` a obsahují nastavení popsané v tabulce 4.2. Nejprve se nastavuje jméno pravidla, které se poté bude zobrazovat v záznamech i v Kibaně. Typ pravidla je nastaven na `frequency`, který provede zaslání údajů java aplikaci a zažádá o jejich přidání do DNS zóny, pokud v určitém časovém úseku nastane „x“ událostí. Tyto události jsou varování

generovaná z OSSEC serveru. Aby bylo možné ukládat výsledky zpět do databáze, musí se vytvořit index, podle kterého se pak bude provádět vyhledávání v Kibaně. Nastavení `filter` určuje, co za hodnotu se musí v databázi objevit, aby bylo aktivováno pravidlo. Filtr je tedy nastaven na konkrétní identifikátor pravidla z OSSEC.

Další položkou je nastavení časového úseku, po jehož uplynutí bude ElastAlert opětovně kontaktovat java aplikaci s žádostí o přidání uživatele se shodným přihlašovacím jménem a IP adresami. Díky tomu bude zaručeno, že nedojde k opakovanému uložení té samé hodnoty do DNS zóny. Aby byla zaručena aktivace pravidla jen na unikátní hodnoty, ElastAlert si pamatuje, pro které kombinace přihlašovacího jména, veřejné a privátní IP adresy již provedl vygenerování varování. Pokud tedy uživatel se stejným přihlašovacím jménem a veřejnou IP adresou změní privátní IP adresu, bude uložena tato nová IP adresa do DNS zóny. Toto nastavení funguje i v případě, kdy není dostupná privátní IP adresa. V praxi si ElastAlert vytváří klíč z přihlašovacího jména, veřejné a privátní IP adresy ve tvaru `login,publicip,none` a tuto kombinaci si zapamatuje. Zbývá ještě nastavit, které hodnoty se mají předat programu, který provádí samotné varování a název tohoto programu pro jeho spuštění při aktivaci pravidla [9].

název parametru	popis
rules_folder	adresář s pravidly
run_every	jak často se má ElastAlert spouštět
buffer_time	doba, po kterou bude výsledek dotazu uložen
es_host	adresa Elasticsearch
es_port	port Elasticsearch
es_username	přihlašovací jméno do Elasticsearch
es_password	heslo do Elasticsearch
writeback_index	index, pod kterým budou výsledky uloženy v Elasticsearch
alert_time_limit	doba, po kterou se bude ElastAlert pokoušet provést neúspěšné varování

Tabulka 4.2: Parametry nastavení ElastAlert. Převzato z [9].

Ze začátku byl problém při implementaci programu pro volání aplikace, jelikož python standardně nepodporuje importování modulů z jiných složek. Nakonec je to vyřešeno přidáním modulu do stejného adresáře. Program samotný se skládá z jedné třídy, která dědí od třídy `alerter`. Tato třída musí implementovat metody `alert` a `get_info`. První metoda provádí samotné provádění volání na rozhraní java aplikace. Metoda dostane jako parametr slovník, ve kterém se nachází informace o uživateli a jeho IP adresách. Podle toho, kolik informací je k dispozici, se odešle buď požadavek na přidání samotného přihlašovacího jména, nebo v kombinaci s veřejnou a privátní adresou. V každém z těchto případů se liší výsledná url. Aby bylo možné komunikovat s java aplikací, musí žádost obsahovat v hlavičce správný podpis. Proto je zde konstanta, ve které je podpis uložen. Po odeslání žádosti java aplikace provede požadované akce a vrací http kód výsledku. Výsledek je zaznamenán do souboru a také zaslán zpět do databáze.

Kapitola 5

Testování

Testování probíhalo pouze na lokálním prostředí, protože ve firmě by vytvoření nových instancí a instalace celého systému zabralo příliš mnoho času.

5.1 Testování jednotlivých částí

Každá část výsledného systému byla testována nejprve samostatně.

5.1.1 OSSEC

OSSEC má vlastní nástroj pro kontrolu zpracování záznamů, kdy kontroluje dekodér, správné uložení důležitých informací do proměnných a následnou aktivaci pravidla a vygenerování varování. Tento test se spouští příkazem `varossecbinossec-logtest`. Poté očekává nástroj na vstupu jeden vzorový záznam, který se bude v praxi zpracovávat. Poté nástroj provede analýzu a výstup je rozdělen do tří fází. První fáze je testování pre-dekódování, kde se kontroluje správný formát záznamu. Po této fázi přichází samotné zpracování záznamu pomocí dekodérů. Pokud je dekodér nastaven správně, na výstupu se objeví název dekodéru a parametry, které obsahují požadované data. V tomto případě parametry `user`, `srcip` a `dstip`. Poslední fází je kontrola, zda dojde k aktivaci pravidla a vygeneruje se varování. Jakmile nastavení prošlo bez chyb testem, byla spuštěna samotná aplikace a bylo provedeno generování záznamů do sledovaného souboru a následná kontrola přítomnosti všech očekávaných varování ve výstupním souboru.

5.1.2 Logstash

Testování probíhalo manuálně. Nejdříve bylo potřeba zkontrolovat správnou konfiguraci a pak kontrolovat vyfiltrované záznamy v Kibaně.

5.1.3 Elasticsearch a Kibana

Testování databáze se provádělo v Kibaně, kde se kontrolovalo, zda se vytvoří index podle kterého má Kibana vyhledávat v databázi a poté zobrazovat výsledky uživateli.

5.1.4 ElastAlert

Zde bylo využito nástroje pro testování, které umožňuje nasimulovat chování programu při jeho zapnutí. Velkou výhodou je, že tento nástroj vypisuje i všechny vyjímky které nastaly

při běhu vytvořeného programu pro volání java aplikace. Díky tomu se povedlo odstranit několik zásadních chyb ještě předtím, než byl ElastAlert spuštěn. Pokud se povede bez chyb naimportovat modul a spustit, dojde k vyhledávání nad databází, a pokud se najde shoda s pravidlem, zašle se požadavek na java aplikaci. Díky tomu lze zjistit, zda vůbec je pravidlo správně nastaveno a zda probíhá správně komunikace na java aplikaci. Bylo testováno několik scénařů, které by mohly v praxi nastat. Například různé kombinace přihlašovacího jména, veřejné a privátní adresy a ověření, že se nevygeneruje varování dvakrát na stejnou kombinaci v nastaveném čase, kdy se nesmí stejná kombinace opakovat. Dále také pokud některý ze tří parametrů chybí, aby nedošlo k opakovanému volání programu.

5.1.5 Knihovny

Všechny knihovny byly také vyzkoušeny, zda jsou jejich implementace volání na aplikaci funkční. Vyzkoušeny byly všechny metody, které knihovny implementují, a ověřeny kódy odpovědí, aby se shodovaly s odpovědí aplikace.

5.1.6 Java aplikace

V java aplikaci byla testována funkcionalita podpisů a samotné ukládání, aktualizování a odebírání záznamů z DNS zóny. Podpisy se testovaly formou http žádostí na aplikaci, ale v hlavičce byl buď špatný nebo žádný token. Aplikace správně odepřela přístup a vrátila kód 401. Dále pokud ve složce chyběl veřejný a privátní klíč pro podepisování tokenů, aplikace je při startu vytvořila.

Důležité bylo také otestovat chování při chybových stavech. Aplikace správně reagovala na výjimky, které vznikaly během provádění požadovaných operací, a vracela kód odpovědi buď 401[1] v případě neplatného podpisu, 404 pokud zóna, nebo záznam neexistoval nebo 500 v ostatních případech. Test správného provedení operace nad DNS serverem probíhal následovně. Nejdříve byly otestovány všechny metody, které kontrolují, zda se požadovaná informace nachází v DNS zóně. Správně vracela odpověď `Not Found`, pokud zadaná doména v DNS neexistovala. S tímto také souvisí testovaní, zda textový záznam nebo záznam s IP adresou nemá být odstraněn, pokud už uplynula doba zákazu přístupu do systémů. Dalším cílem byl správný formát záznamů uložených v DNS zóně. Jako poslední byla testována doba, za kterou po upravení zóny aplikace nalezne tento záznam. Tato doba je dána nastavením parametrů v zóně. DNS server ukládá do vyrovnávací paměti výsledky dotazů od java aplikace a pokud se nenastaví správně čas pro smazání této paměti a načtení nových dat, může dojít k tomu, že uživatel bude sice přidán do černé listiny, ale java aplikace se při dotazu na existenci tohoto záznamu dozví od DNS, že tento záznam neexistuje, jelikož jí odpovídá na základě dat ve vyrovnávací paměti.

5.2 Testování celku

Testování probíhalo na všech systémech podle návrhu 3.1. Uživatel přijde na login stránku a pokouší se přihlásit s neplatným uživatelským heslem. Pokud nemá záznam v DNS zóně, zobrazí se na stránce hláška „Wrong Password“. Pokud se při dalším pokusu přihlásí se správným heslem, dostane hlášku `Logged` a tím scénář končí, je úspěšně přihlášen. V opačném případě dostane opět hlášku o špatném hesle. Takto může pokračovat ještě několikrát,

[1] Unauthorized.

dokud se na ElastAlert nesplní podmínka počtu neplatných přihlášení v daném čase. V takovém případě ElastAlert zašle java aplikaci žádost o zakázání přístupu tomuto uživateli na specifickou veřejnou a privátní IP adresu a dostane chybovou hlášku `Banned`. V takovém případě se nemůže přihlásit ani se správným heslem po dobu, která je nastavena v java aplikaci. Po uplynutí této doby se může opět pokusit přihlásit. Testováno bylo několik variant IP adres a přihlašovacích jmen. Java aplikace je také schopna rozpoznat nevalidní IP adresu a tuto žádost, která IP adresu obsahuje, nebude zpracovávat.

Testování ve větším měřítku se bude provádět až po nasazení do praxe. Nejdříve se začne s jedním systémem, kde se nainstaluje OSSEC agent a nakonfiguruje tak, aby odesílal záznamy na hlavní OSSEC server. Tyto záznamy budou ve formátu `syslog`, tedy každý záznam na jednom řádku. Server přijme tyto záznamy, provede jejich zpracování a pokud nějaký záznam bude odpovídat regulárnímu výrazu, dojde k aktivaci pravidla a uloží varování do souboru. Tento soubor je průběžně odesílán přes UDP protokol na Logstash, kde dochází k jeho filtraci. Nastavení filtru nebude potřeba měnit, jelikož důležité jsou v této fázi jen přihlašovací jméno a ip adresy uživatele. Poté se bude testovat v Kibaně, zda data prošla a správně se uložila do databáze. Bude se testovat vyhledávání podle určitého parametru, podle doby vzniku varování a nebo podle typu varování. Další částí budou měření rychlosti odezvy při přihlašování uživatele, kdy musí systém zaslat žádost na java aplikaci a pak počkat na její odpověď. U OSSEC agentů je toto riziko prodlev ještě větší, jelikož soubory se záznamy čítají desítky gigabajtů. Po této analýze se teprve rozhodne, zda toto řešení je opravdu vhodné i v praxi a ne jen na lokálním prostředí.

Kapitola 6

Rozšíření

V OSSEC by bylo možné lépe zpracovat dekodéry, aby například uměly reagovat na všechny záznamy, které by obsahovaly kód odpovědi 401, a pak by se vytvořily další, které by pokračovaly ve zpracování záznamu a ukládání důležitých informací do proměnných. Těchto informací by mohlo být více, například IP adresa serveru, ze kterého tento záznam pochází. Pravidla by se musela taktéž upravit, aby podle klíčových slov, která se objeví v záznamu, dokázali vygenerovat správné varování. Pokud by v záznamu kromě kódu odpovědi byl ještě důvod, mohla by pravidla reagovat na tento důvod a podle toho vygenerovat varování. Pokud by to bylo vážné riziko, OSSEC by posílal například email správci serveru.

Kibana nyní ukazuje jen základní statistiky jako počet varování ze serveru OSSEC a varování vygenerované pomocí ElastAlert. Kibana ale podporuje nejrůznější grafy, nástěnky apod. Díky tomu by se mohla všechna varování například vykreslovat v grafu nebo na nástěnce a bylo by možné sledovat v různém časovém období, kolik varování a z jakých IP adres proběhlo. Zajímavé by také bylo zobrazovat tyto grafy k určitému přihlašovacímu jménu nebo IP adrese. Díky tomu by se během pár okamžiků dalo zjistit, zda uživatel opravdu zapomněl své heslo nebo se snaží shodit server.

Java aplikace by mohla lépe používat klíče (certifikáty) a například ukládat některé informace do tokenu, který musí obsahovat hlavička http požadavku. Aby se dalo sledovat, kdo a kdy zavolal tuto aplikaci a jakou akci chtěl provést, mohla by aplikace ukládat záznamy do souboru a z tokenu by mohla dostat například název systému, který požádal o danou akci nebo IP adresu serveru, ze kterého přišel požadavek. Kromě tokenů by se mohla bezpečnost aplikace rozšířit ještě seznamem povolených IP adres a určitých rozsahů IP adres, ze kterých by bylo povolené komunikovat s aplikací. Aplikace by také mohla obsahovat vyrovnávací paměť, kde by si uchovávala nedávno přidané uživatele, aby se nemusela pokaždé dotazovat na DNS server. Tato vyrovnávací paměť by se vyčistila vždy, když by byl přidán nový záznam do DNS serveru. Ve většině případů se budou systémy dotazovat, zda je uživatel v DNS zóně. Vyrovnávací paměť by tento proces urychlila a snížila zatížení DNS serveru.

ElastAlert umožňuje vytvořit i vlastní typ pravidel nebo filtr, který dokáže ještě data upravit a vyfiltrovat, než se předají programu. Například zjištění přibližné lokality podle veřejné IP adresy. Tento údaj by se pak také ukládal do Elasticsearch a bylo by možné na základě tohoto parametru vyhledávat. Statistiky by pak byly přesnější a více informativní.

Kapitola 7

Závěr

Výsledkem této práce je fungující systém, který dokáže na základě souborů se záznamy vyfiltrovat informace o uživateli, jako je jeho přihlašovací jméno, veřejná a privátní IP adresa a přidat tyto hodnoty do DNS zóny, která plní funkci černé listiny. K vytvoření tohoto systému byly použity `open source` aplikace a je proto ideální pro firmy, které chtějí mít kvalitní řešení obrany proti útočníkům, ale přitom použít co nejméně finančních prostředků. V práci jsem se snažil používat technologie, které jsou již použity ve firmě, pro kterou je tato práce určena.

Nevýhodou je OSSEC, který potřebuje mít na koncových serverech klienty, kteří všechny záznamy zasílají na centrální server ke zpracování. V praxi, kde jsou soubory se záznamy obrovské, není ideální zasílat tyto informace po síti. Z toho vyplývá nárok na velmi dobrou infrastrukturu a potřeba omezit velikost záznamů. To by mohlo mít vliv na možnosti vysledování problému aplikací, pokud by byl omezen počet záznamů ukládaných do souborů.

Požadavky se podařilo splnit kromě testování, které bylo provedeno pouze na lokálním prostředí z důvodu nedostatku času ze strany firmy. Na lokálním prostředí nedochází k prodlevě při komunikacích jednotlivých systémů či vyhledávání nad databází a DNS serverem a proto nyní nelze jednoznačně určit, zda je toto řešení efektivní pro nasazení ve větším měřítku nebo nikoliv. Až při nasazení do praxe se mohou provést testy, které ukážou celkovou rychlost práce, přidávání záznamů do DNS zóny a rychlost zasílání záznamů z agentů na centrální server. Je zde také vyšší riziko poruchovosti, jelikož pokud přestane některý ze systémů fungovat, přestane fungovat všechno. Je také možné, že agenti nebudou stíhat posílat záznamy v reálném čase na server ke zpracování. V takovém případě nabízí ElastAlert možnost čekání na výsledky z Elasticsearch databáze, čímž by se tato komplikace dala vyřešit. Také Propagace příslušných klíčů pro komunikaci mezi jednotlivými systémy nebude triviální.

Literatura

[1] Andrew HAY, Daniel CID, Rory BRAY: *OSSEC Host-Based Intrusion Detection Guide*. Syngress, 2008, iSBN 9781597492409.

[2] Apache: Maven documentation. `https://maven.apache.org/index.html`, [online]. 2016 [cit. 2016-5-10].

[3] Bryan Costales, George JANSEN, Claus ABmann, Gregory Neil SHAPIRO: *Sendmail*. O'Reilly Media, 2007, iSBN 9780596510299.

[4] Clinton Gormley, Zachary Tong: *Elasticsearch: The Definitive Guide*. O'Reilly Media, 2015, iSBN 978-1-4493-5849-5.

[5] Craig WALLS: *Spring Boot in Action*. Manning publications, 2015, iSBN 9781617292545.

[6] Elasticsearch: Kibana documentation. `https://www.elastic.co/guide/en/kibana/current/index.html`, [online]. 2016 [cit. 2016-5-10].

[7] Elasticsearch: Logstash documentation. `https://www.elastic.co/guide/en/logstash/current/index.html`, [online]. 2016 [cit. 2016-5-5].

[8] Mitchell Anicas: How To Install Elasticsearch, Logstash, and Kibana (ELK Stack) on Ubuntu 14.04. `https://www.digitalocean.com/community/tutorials/how-to-install-elasticsearch-logstash-and-kibana-elk-stack-on-ubuntu-14-04`, [online]. 2015 [cit. 2016-5-5].

[9] Nicolas ZIN: ElastAlert: Alerting At Scale With Elasticsearch. `http://engineeringblog.yelp.com/2016/03/elastalert-part-two.html`, [online]. 2016 [cit. 2016-5-5].

[10] Ron AITCHISON: *Pro DNS and BIND 10*. Apress, 2011, iSBN 978-1-4302-3048-9.

[11] Salvatore LORETO, Simon Pietro ROMANO: *Real-Time Communication with WebRTC*. O'Reilly Media, 2014, iSBN 978-1-4493-7187-6.

[12] ZIN, N.: OSSEC howto, the quick and dirty way. `https://blog.savoirfairelinux.com/wp-content/uploads/2014/03/SFL-ED01-OSSec-the-quick-and-dirty-way-140326-01.pdf`, [online]. 2014 [cit. 2016-1-24].

Přílohy

Seznam příloh

Příloha A

Obsah CD

- Src,
- LICENSES,
- README.

Ve složce Src se nachází všechny zdrojové soubory a také obraz operačního systému ubuntu, kde jsou nainstalovány a nakonfigurovány všechny potřebné nástroje. V souboru LICENSES jsou zmíněny všechny licence, které se v práci vyskytují. README obsahuje návod na kompletní zprovoznění práce, příkladů spuštění a řešení chybových situací.