ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Detekce pomalých útoků hrubou silou
Student:	Libor Šlechta
Vedoucí:	Mgr. Jan Starý, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

1. Vytvořte aplikaci, která na stroji s IP konektivitou detekuje "slow brute" útoky, popsané například v http://bsdly.blogspot.cz/2013/10/the-hail-mary-cloud-and-lessons-learned.html. Využijte existující logy jako materiál, sbírejte v reálném provozu i své vlastní.
2. Aplikaci napište v jazyce C/C++ s použitím standardních síťových knihoven (libpcap, libdnet). Podporujte čtení záznamu (tcpdump, pflog, ssh log), případně i živý provoz.
3. Dbejte na čistotu, korektnost, rozšiřitelnost a přenositelnost kódu, přinejmenším mezi systémy UNIXového typu.
4. Vytvořte uživatelskou dokumentaci, nejlépe standardní manpage.
5. Aplikaci řádně otestujte a metody testování zdokumentujte.
6. Proveďte základní vyhodnocení úspěšnosti detektoru.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 17. prosince 2015

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

Bakalářská práce

Detekce pomalých útoků hrubou silou

Libor Šlechta

Vedoucí práce: Mgr. Jan Starý, Ph.D.

16. května 2016

Poděkování

Rád bych poděkoval především vedoucímu práce Mgr. Janu Starému, Ph.D. za pravidelné konzultace, které mi velmi pomohly při tvorbě bakalářské práce. Poděkoval bych rád také mé rodině za veškerou jejich podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl 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, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. května 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2016 Libor Šlechta. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Šlechta, Libor. *Detekce pomalých útoků hrubou silou*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá vytvořením aplikace pro detekci skrytých útoků na SSH servery hrubou silou. Výsledná aplikace se skládá ze sady skriptů napsaných v jazyce Python, které jsou spolu schopny spolupracovat. Získané záznamy jsou skladovány pomocí SQLite3 databáze. Vytvořené řešení je schopno v simulovaném provozu detekovat více než polovinu skrytých útočníků. Aplikace je vhodná pro další výzkum zvolené metody detekce a může sloužit jako základ pro volně dostupný detektor skrytých útoků. V závěru práce jsou pak navrženy další kroky nezbytné pro reálné nasazení takového detektoru.

Klíčová slova detektor útoků, útoky hrubou silou, slovníkové útoky, skryté útoky, distribuované útoky, SSH server, počítačová bezpečnost, CUSUM

Abstract

This thesis implements a detector of stealthy brute force attacks on SSH servers. The application consists of a set of cooperating Python scripts. Obtained records are being stored in an SQLite3 database. This program is able to find more than half of attackers in simulated datasets. The application may be useful in further research of the method and can also serve as foundation to a real-world detector. In the conclusion topics for further research are proposed.

Keywords attack detector, brute force attacks, dictionary attacks, stealthy attacks, distributed attacks, SSH server, computer security, CUSUM

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Používání internetových služeb se v dnešní společnosti stalo již naprosto běžnou součástí našich každodenních životů. Bohužel přes veškeré snahy není jejich bezpečnost dokonalá a existuje velké množství uživatelů internetu, kteří se snaží této skutečnosti zneužít.

SSH služba umožňuje vzdálené přihlášení k počítači, a proto jsou na ní velmi často vedeny útoky. Mnoho z těchto útoků je známo již dlouhou dobu, a proto byla vytvořena řada protiopatření. Vývoj bezpečnostních prvků samozřejmě zaznamenali i útočníci a začali útoky modifikovat, aby se ochranám vyhnuli.

Jedním ze známých útoků na SSH službu je útok hrubou silou, při němž se útočník snaží získat přístup ke vzdálenému počítači hádáním přihlašovacích údajů. Modifikaci útoku, která se snaží vyhnout konvenčním obranným mechanismům, označujeme jako skryté útoky hrubou silou.

Detekování skrytých útoku hrubou silou jsem si zvolil za téma své práce, protože v současné době neexistuje žádné volně dostupné řešení, které by bylo schopno tyto útoky odhalit.

V první části práce jsem se zaměřil na představení útoků hrubou silou a jejich známých variant. Ve druhé části práce jsem představil metody detekce těchto útoků a dostupná známá řešení. Třetí část práce jsem věnoval popisu svého řešení a implementačním detailům. V poslední části práce jsem popsal shromážděná testovací data a uvedl výsledky testů.

V praktické části práce vycházím z metody představené v článku Detecting Stealthy, Distributed SSH Brute-Forcing [1] s cílem vytvořit dostupný prototyp pro potřeby dalšího výzkumu a vytvoření volně dostupného detektoru skrytých útoků hrubou silou.

Kapitola 1

Útoky hrubou silou na autentifikační služby

Kapitola čerpá především ze zdrojů [2, 3].

1.1 Princip útoků hrubou silou

Většina autentifikačních služeb na internetu je založena na takzvaném sdíleném tajemství. Jedná se o znalost klíčové fráze, sdílené mezi uživatelem a autentifikační službou. Při pokusu o přístup k systému se ověřuje znalost tohoto sdíleného tajemství. Při znalosti hesla dochází k ověření identity, při neznalosti klíčové fráze je přístup odepřen. Přestože je tato technika ověřování identity jedna z prvních vymyšlených metod, je stále jednou z nejčastěji využívaných.

1.1.1 Útoky hrubou silou v kontextu ostatních druhů útoků

Útoky, které se snaží zmíněný způsob autentifikace prolomit, můžeme rozdělit do dvou kategorií.

Do první kategorie spadají útoky, kdy se útočník snaží heslo zcizit uživateli. Mezi takovéto techniky patří například odposlouchávaní komunikace. Varianta tohoto útoku nazývaná *man in the middle* je úspěšná dokonce i při použití šifrované síťové komunikace. Dále spadá do této kategorie například falšování identity, kdy se útočník vydává za systém, ke kterému se snaží získat přístup a pokouší se tak z uživatelů vylákat přihlašovací údaje. V neposlední řadě sem spadá i škodlivý software schopný monitorovat aktivitu uživatelovy klávesnice, nazývaný *keylogger*.

Druhou kategorií jsou pak útoky, při kterých se útočník snaží získat heslo od autentifikační služby. Do této kategorie spadají veškeré útoky, které využívají neopravených slabin v operačních systémech. Zároveň sem spadají tak-

zvané útoky hrubou silou. Při tomto typu útoků se útočník snaží uživatelské heslo uhodnout. Často při tomto pokusu o získání přístupu do systému útočník nezná ani uživatelské jméno a musí tak hádat i tento údaj.

1.1.2 Předpoklady úspěšnosti útoků hrubou silou

Důležitým pojmem pro pochopení předpokladů úspěšnosti útoků hrubou silou je pojem entropie. Entropie neboli míra neurčitosti, se ve vztahu k heslům definuje jako míra nepředvídatelnosti hesla. Entropie se obvykle uvádí v počtu náhodně zvolených bitů, které odpovídají užitému heslu.

Zde musíme brát v úvahu fakt, který nahrává útočníkům: hesla nejsou obvykle generována automaticky, jsou vymýšlena lidmi. Lidské chování je do značné míry předvídatelné. Nejčastější požadavek na heslo je, aby bylo zapamatovatelné. Ovšem to, že se uživatelé snaží volit zapamatovatelná hesla, vede k tomu, že heslem často bývá slovníková fráze.

Předchozí tvrzení se opírá zejména o analýzu portálu wpengine [4], který se zaměřil na analýzu dvou databází uniklých hesel, z nichž první obsahovala hesla uniklá z portálu Google v roce 2014. Druhá datová sada pak obsahovala záznamy zveřejněné bezpečnostním analytikem Markem Burnettem, který po dobu několika let shromažďoval veřejně uniklé přihlašovací údaje po celém webu. Výsledky této analýzy především potvrdily obecně známý fakt, že se uživatelská hesla překrývají, tedy že jedno heslo střeží více rozdílných účtů. Dále je zde uvedeno, že uživatelé často užívají slovníkové fráze, a když už je modifikují, nejčastěji použijí první velké písmeno a na konec fráze připojí číslo.

1.1.3 Motivace útočníků

Motivace, kterou mají útočníci k prolomení autentifikace, se liší dle služby, na kterou útočí. Obecně však vždy jde buď o snahu zcizit důvěrné informace, převzít nad systémem kontrolu nebo zneužít systém pro nekalé praktiky.

Kroky, které útočníci činí po úspěšně provedeném útoku na službu SSH, popisuje Daniel Cid [5]. Ten za pomoci síťové pasti, jejíž princip popíšu v 2.2.2, zachytil aktivitu útočníků poté, co dostali přístup k systému. Ti zde nejdříve změnili heslo. Poté se pokusili získat přístup k administrátorským právům. Posledním krokem byla instalace škodlivého softwaru umožňující jim vzdáleně ovládat systém.

1.2 Klasifikace útoků hrubou silou

1.2.1 Off-line útoky

V případě, že útočník hádá hesla v rámci své počítačové infrastruktury a s napadeným systémem interaguje jen minimálně, označujeme útok za off-line variantu útoku hrubou silou.

Pro provedení tohoto typu útoku potřebuje nejprve útočník získat uživatelská jména a hash otisky odpovídajících hesel. Tyto otisky se používají k bezpečnějšímu uchování přihlašovacích údajů a například v systémech Unixového typu jsou zpravidla uchovávany v systémovém souboru `/etc/shadow`. Tento soubor je přístupný pouze uživatelům s administrátorským oprávněním, přesto však může dojít k úniku těchto informací, zejména díky využití neopravených chyb v operačním systému.

Převod hash otisku na původní heslo by měl být z definice hashovací funkce nemožný. Přesto však existují techniky, které se o takovýto převod pokoušejí.

1.2.1.1 Předpočítávání výsledků

V případě této metody si útočník opatří řetězce a jim odpovídají hash otisky, které uloží do vhodné datové struktury. Následně vyhledává, jestli tento seznam obsahuje některý z hledaných otisků.

Výhoda tohoto seznamu spočívá v rychlém ověření, zda seznam obsahuje hledaný záznam. Navíc lze jednou takto vytvořený seznam použít pro stejnou hashovací funkci opakovaně. Nevýhodou pak jsou paměťové nároky na uchování těchto záznamů, kdy je nutné ukládat jak celý původní řetězec, tak kompletní výsledný hash otisk. V praxi tak útočníci používají tento přístup především k provedení slovníkového útoku, při kterém se pokoušejí zjistit, zda uživatel nepoužil některé z často používaných hesel.

1.2.1.2 Rainbow table

Rainbow table přináší kompromis mezi paměťovými nároky a výpočetní složitostí na znovunalezení předpočítaných záznamů. Vzniká tvořením takzvaných řetězů. Články řetězu jsou vytvářeny za pomocí opakovaného používání hashovací funkce a redukčních funkcí. Redukční funkce jsou takové funkce, které dokáží hash převést do podoby, která je blízká původnímu textu. Výsledná tabulka pak obsahuje dvojice původního textu a posledního spočítaného hashe v daném řetězu.

1.2.2 Online útoky

Pokud útočník k získání přihlašovacích údajů potřebuje s napadeným systémem komunikovat s vysokou intenzitou, označujeme tento typ útoků jako takzvaný on-line útok hrubou silou.

Při tomto typu útoků útočník nevyužívá chyb ve funkčnosti systému, ale naopak jeho zamýšlené funkcionality: po zadání správných přihlašovacích údajů je vpuštěn do systému bez ohledu na svou minulou aktivitu. Za předpokladu, že není na systému nainstalována žádná ochrana před tímto typem útoku, je útočníkovi opakovaně umožněno zkoušet všechny možné kombinace přihlašovacích údajů, dokud neuspěje.

1.2.2.1 Dělení dle frekcence útoků

Pokud se podíváme na útoky hrubou silou z hlediska četnosti pokusů o přihlášení, logicky nám vyplyne, že větší frekvence je lepší při provádění útoku. Vzhledem k tomu, že je v informatickém světě o tomto typu útoku již značné povědomí, snaží se většina správců svůj systém před touto hrozbou ochránit.

Na základě aktivity jednoho vzdáleného hosta pak můžeme útoky rozdělit na útoky s vysokou, nebo s nízkou frekvencí.

- High rate - Jedná se o nejprimitivnější variantu útoku hrubou silou a její detekce je triviální. Při zanedbání ochrany před touto hrozbou je ovšem tento způsob nejnebezpečnější, protože má předpoklad dosáhnout úspěchu v nejkratší době.
- Low rate - Při tomto způsobu útoku se útočníci pokouší splynout s legitimními uživateli. Snaží se napodobit jejich chování a zamaskovat tak svůj útok. Obecně lze do této kategorie řadit všechny útoky, které nelze objevit pomocí klasických detektorů útoků s vysokou frekvencí.

1.2.2.2 Dělení dle zdrojů útočníka

Při tomto pohledu na útoky hrubou silou zjišťujeme, jakou infrastrukturu využívá útočník k útoku.

- Jednoduchý útok – Útočník má k dispozici pouze jeden stroj, kterým se snaží kompromitovat cílený systém.
- Distribuovaný útok – Varianta útoku, která vznikla jako reakce na ochranné prvky vzešlé v boji s útoky hrubou silou. Útočníci při ní rozdělují úlohu mezi více strojů. Tyto stroje jsou obvykle součástí takzvaného *botnetu*, což je skupina již dříve kompromitovaných strojů ovládaná útočníkem.

Kapitola 2

Metody ochrany před útoky hrubou silou

Kapitola čerpá především ze zdrojů [3, 6].

Obecné zásady pro prevenci úspěšných útoků hrubou silou existují jak pro uživatele, tak administrátory služeb. Uživatelům je zejména doporučováno, aby volili svá hesla zodpovědně, tedy aby se pokoušeli vytvářet hesla, která nejsou útočníkem snadno uhodnutelná. Správci systému by pak měli dbát především na správnou konfiguraci služby a pravidelnou aktualizaci systému. Také by měli zvážit, zda vůbec chtějí uživatelům povolit přihlašování pomocí hesel. Jak jsem již zmínil v sekci 1.1.2, přihlašování pomocí hesel sebou přináší bezpečnostní riziko ve formě útoků hrubou silou. Varianta založená na přihlašování pomocí veřejného a privátního klíče je oproti tomuto útoku podstatně odolnější.

2.1 Prevence off-line útoků

Detekce off-line útoků hrubou silou je z pohledu administrátora napadené služby velice komplikovaná záležitost vzhledem k tomu, že se většina útoku odehrává na straně útočníka.

Ochranou před tímto konkrétním typem útoku pak je především správná konfigurace systému. Zejména je důležitý výběr bezpečné hashovací funkce, která slouží k uložení otisku hesla. V dnešní době existuje několik druhů hashovacích algoritmů, některé z nich však v současnosti již nejsou nadále považovány za bezpečné.[1]

Další metoda, která útočníkům může ztížit prolomení hesel, se označuje jako kryptografické solení. Jedná se o přidání náhodného řetězce za heslo ještě

[1] Aktualizovaný a přehledný souhrn existujících kryptografických, hashovacích funkcí lze dohledat například na http://valerieaurora.org/hash.html

před převáděním hesla na otisk. Tento náhodný řetězec je pak uložen zároveň s otiskem hesla. Použití kryptografické soli neztěžuje útočníkovi odhalení jednoho hesla, ale zabraňuje opakovanému použití vypočítaných kombinací.

2.2 Detekce on-line útoků

Pro detekování on-line útoků hrubou silou existuje několik možných přístupů. V následující části tyto přístupy obecně představím. Ukázky konkrétních příkladů těchto opatření pak následují v kapitole 3.

2.2.1 Detekce na straně serveru

2.2.1.1 Analýza přístupových záznamů

Všechny služby, které vyžadují autentizaci, mají možnost evidovat nějakým způsobem aktivity uživatelů. Obvykle se jedná o takzvané přístupové záznamy, které obsahují údaje o tom, kdo se pokusil přihlásit, kdy tento pokus uskutečnil a zda uspěl. Tyto údaje lze pak analyzovat a rozhodnout zda se jedná o útok.

2.2.2 Síťové pasti

Síťová past (*honeypot*) je dle své definice zařízení, na které se útočí. V rámci sítě lze provozovat i zařízení, která se pouze tváří jako legitimní služba, avšak nemají krom detekce útoků žádné jiné využití. Kdokoliv se pokouší přihlásit k takovému zařízení, je automaticky prohlášen za útočníka, protože toto zařízení není určené pro legitimní uživatele.

2.2.3 Detekce na základě analýzy síťové komunikace

Dalším způsobem, jak odhalovat útoky hrubou silou, je zkoumání síťové komunikace. Zde se pak na základě známých signatur útoků snažíme zjistit, zda na nás vzdálený host útočí.

2.2.3.1 Analýza procházejících paketů

Pokud se při hledání síťových hrozeb zaměříme na obsah síťové komunikace, mluvíme o takzvané hluboké inspekci síťové komunikace. Tento princip je potenciálně velmi úspěšný, protože umožňuje vyhodnocovat veškeré informace, které vzdálený host poskytuje a lze se tak lépe rozhodovat, zda se se jedná o útočníka. Tento přístup však nelze obvykle použít, protože většina dnešní komunikace je již šifrována.

2.2.3.2 Analýza síťového toku

Do této kategorie spadají metody, které se snaží analyzovat informace potřebné k přenosu dat mezi vzdálenými systémy. Z pohledu referenčního modelu ISO/OSI se jedná o informace ze síťové a transportní vrstvy.

Kapitola 3

Současný stav řešení problematiky útoků hrubou silou

3.1 Existující programy

Metoda slovníkových útoků je v současnosti mezi provozovateli síťových služeb dobře známá, a existuje již řada programů, které se snaží takové útoky detekovat. V rámci průzkumu dostupných zdrojů na internetu se mi ale krom síťových pastí nepodařilo nalézt žádné řešení, které by bylo schopné detekovat i distribuované slovníkové útoky.

3.1.1 Analyzátory přístupových záznamů

3.1.1.1 Sshguard

Sshguard je volně dostupný nástroj implementovaný v jazyce C, který vznikl původně pro ochranu SSH služby. Postupně však byl rozšířen a v současné době je schopen monitorovat i další služby. Jako příklad může sloužit podpora VSFTPD, což je implementace FTP serveru pro unixové stroje. Plný výčet podporovaných služeb je dostupný na domovské stránce projektu.

3.1.1.2 Fail2ban

Fail2ban je volně dostupný nástroj funkčností podobný programu Sshguard. Hlavním rozdílem mezi těmito programy je především rozsah podporovaných služeb. Více informací o funkčnosti programu lze opět dohledat na jeho domovské stránce. Fail2ban je implementován v jazyce Python.

3.1.1.3 DenyHosts

DenyHosts je principiálně velmi podobný programu Fail2ban. Opět se jedná o volně dostupný nástroj, který je také implementován v jazyce Python. Oproti

konkurenci však přináší navíc volitelný takzvaný synchronizační režim, který umožňuje uživatelům kooperovat a proaktivně blokovat adresy, které byly detekovány jako hrozba na ostatních strojích využívajících DenyHost v synchronizačním režimu. Avšak tento režim se jeví jako potenciálně velmi nebezpečný, protože lze pomocí něj dosáhnout upírání služeb na synchornizovaných serverech legitimním uživatelům.

3.1.2 Síťové pasti

Síťová past může být buď skutečná serverová instance, která provozuje SSH službu pouze za účelem přilákání útočníka, nebo emulátor SSH služby.

3.1.2.1 Kippo

Jedním z příkladů emulátorů SSH služby je program Kippo. Jedná se o program napsaný v jazyce Python, který emuluje virtuální instanci SSH serveru. Po úspěšném přihlášení od útočníka ukládá veškerou jeho aktivitu, čímž pomáhá monitorovat chování útočníků. Protože se nejedná o skutečný systém, nehrozí, že by útočník mohl infiltrovaný stroj zneužít. Naopak lze útočníkovi záměrně povolit přístup nastavením snadné kombinace přihlašovacích údajů, aby se zmapovala jeho aktivita po úspěšně provedeném útoku.

3.1.3 Analyzátory síťových toků

V této kategorii existují dvě hlavní skupiny programů. Firewally a systémy detekce/prevence narušení. Mezi těmito skupinami jsou dle definice určité rozdíly, avšak díky vývoji v oblasti obou skupin se v současnosti rozdíly smývají a obě skupiny programů mají vesměs stejné funkce. Mezi tyto funkce patří především monitorování síťové komunikace v reálném čase a hledání známých příznaků útoků.

Schopnost odhalit síťové hrozby spočívá především ve správně definované sadě bezpečnostních pravidel, která popisují signatury známých útoků. Programy tak nejsou schopny odhalit doposud neznámé síťové útoky, ani modifikace již známých hrozeb.

Nejznámějšími zástupci firewallů na Unixových strojích jsou programy Iptables a Packet Filter. V oblasti komplexních detektorů pak stojí za zmínku především program Snort, který je volně dostupný a má širokou komunitu, která sdílí bezpečnostní pravidla a pomáhá tak ostatním uživatelům s nastavením bezpečnostní politiky.

3.2 Současný stav výzkumu

3.2.1 Analyzátory přístupových záznamů

3.2.1.1 Vizuální analýza přístupových záznamů

Malecot a kolektiv [7] navrhli detekovat slovníkové útoky pomocí vizuální analýzy logů. Využívají k tomu mapování adres neúspěšných hostů pomocí quad stromů. Přednost této metody je především ve schopnosti odhalit útočníky se stejným segmentem síťové adresy vzhledem k tomu, že ve výsledném grafu budou mapovány do podobných částí grafu a pro analytika je pak snadnější vypozorovat souvislosti mezi jejich aktivitami.

3.2.1.2 Statistická analýza přístupových záznamů

Javed a Paxson [1] v rámci svého výzkumu distribuovaných slovníkových útoků navrhli a otestovali obecný návrh detektoru založeného na detekování statisticky významné odchylky od standardního provozu.

Tuto metodu jsem zvolil pro realizaci detektoru, její podrobný popis je obsahem kapitoly 4.2.

3.2.2 Analyzátory síťových toků

3.2.2.1 Detekování na základě slučování podobných toků

Vykopal [6] ve své práci zveřejnil detektor, který slučuje toky na základě jejich vlastností. Následně označí na základě předpokladu, že standardní síťová komunikace má náhodný charakter, velké shluky za podezřelé. Ve své práci se zabývá i problematikou vyloučení planých poplachů a detekcí užití NATu.

3.2.2.2 Detekování anomálií

Drašar [8] přistoupil k detekování distribuovaných slovníkových útoků za pomoci hledání anomálií v síťovém provozu. Svůj postup otestoval pouze na simulovaných síťových útocích a ve své práci se nezabývá rozlišením planých poplachů.

Kapitola 4

Statistická analýza přístupových záznamů

4.1 Užívané pojmy

4.1.1 Náhodná veličina

Náhodná veličina je funkce, která přiřazuje číselné hodnoty výsledkům náhodného experimentu. Její správné zavedení je pro následné matematické zpracování užitečné, zejména pokud výsledkem experimentu není číslo. Pokud hodnoty náhodné veličiny spadají do nějaké nanejvýše spočetné množiny, nazýváme ji *diskrétní*. [9]

4.1.2 Střední hodnota

Střední hodnota náhodné veličiny udává rovnovážný bod pravděpodobností. Díky tomu jsou výsledky pokusů očekávány poblíž této hodnoty. Střední hodnotu lze vypočítat jako vážený průměr všech možných hodnot náhodné veličiny. [10]

4.1.3 CUSUM

CUSUM (*kumulativní suma*) je technika *regulačních diagramů* rozšířená především v průmyslové praxi, kde se využívá k monitorování výrobních procesů. Tato technika je založena na sekvenčním pozorování procesu, u nějž známe charakteristiky za běžného provozu. Pomocí průběžného výpočtu kumulativní sumy lze pak při správné parametrizaci detektoru odhalit odchylku od očekávaného rozdělení.

Algoritmus kumuluje odchylku zvlášť v záporném a kladném směru. Ve výsledném řešení detekuji odchylku pouze v kladném směru, a proto budu veškeré následující příklady uvádět pouze pro tuto variantu.

Výpočet kumulativní sumy C pro n-té pozorování náhodné veličiny X se řídí vzorcem (4.1), kde μ udává střední hodnotu pozorované veličiny a k *referenční hodnotu*. Počáteční hodnota kumulativní sumy je 0. [11]

$$C_n = max(0, X_n - (\mu + k) + C_{n-1}) \tag{4.1}$$

Referenční hodnotu lze zvolit několika způsoby. Autoři článku [1] nastavují referenční hodnotu na polovinu detekované odchylky.

Přesáhne-li kumulativní suma *mezní hodnotu*, došlo k posunu v pozorované veličině. Volba mezní hodnoty se odráží na průměrné době běhu (*average run length*) za normálního provozu (*in control ARL*) a průměrné době běhu potřebné k detekování odchylky (*out of control ARL*). *In control ARL* udává průměrný počet pozorování, který předchází falešnému poplachu a *out of control ARL* odpovídá počtu pozorování potřebných k detekování odchylky.

Výpočet průměrné doby běhu pro oba případy lze provést pomocí *Markovových řetězců*. Průměrnou dobu běhu algoritmu za normálního provozu lze vypočíst pomocí rovnice (4.2), kde R je přechodová pravděpodobnostní matice bez posledního sloupce a řádku, I je jednotková matice a 1 je matice s jedním sloupcem samých jedniček. Výsledkem výpočtu je vektor obsahující průměrné doby běhu pro různé startovní hodnoty kumulativní sumy. Na prvním řádku se nachází průměrná doba běhu algoritmu startujícího s hodnotou kumulativní sumy 0. Tento výsledek lze efektivně vypočíst pomocí Cramerova pravidla.[12]

$$incontrolARL = (I - R)^{-1} * 1 \tag{4.2}$$

4.2 Obecný návrh detektoru

Následující část textu popisuje obecný návrh detektoru jak jej představili ve svém článku Mobin a Javed [1].

Detektor je založený na statistické analýze přístupových záznamů. V průběhu své činnosti především monitoruje vývoj trendu, v jakém uživatelé chybují při přihlašování.

Na rozdíl od většiny ostatních metod se při této metodě nepozorují fixní časové úseky, ale úseky definované jako výskyt fixního počtu pokusů o přihlášení. Díky tomu budou mít jednotlivé úseky stejnou váhu, protože nejsou ovlivněny výkyvy v četnosti přihlašování uživatelů.

Monitorovanou náhodnou veličinou je pak indikátor selhání, který definujeme jako počet neúspěšných přihlášení během jednoho úseku.

Samotná detekce útoků pak probíhá ve třech krocích. Vyloučení útoků s vysokou frekvencí, analýza provozu a určení útočníků.

4.2.1 Vyloučení útoků s vysokou frekvencí

Detektor je schopen odhalit i útoky s vysokou frekvencí. Nicméně je především navržen tak, aby byl schopen detekovat již malou změnu v indikátoru selhání, což je důležité pro odhalení skrytého útoku v běžném provozu. Útoky s vysokou frekvencí se před detekováním skrytých útoků vyloučí, aby nedocházelo k zbytečně častému hlášení neobvyklého provozu.

Přístupů, kterými lze o hostovi prohlásit, že útočí s vysokou frekvencí, existuje mnoho. Autoři detektoru se rozhodli označovat hosty, kteří se během jedné hodiny pokusí přihlásit bez úspěchu více než dvacetkrát v řadě. Tuto definici útočníků s vysokou frekvencí zvolili na základě analýzy záznamů shromážděných v Lawrence Berkeley National Laboratory za dobu sedmi let.

4.2.2 Analýza provozu

Analýza provozu je částí detektoru, která má na starosti rozhodování o tom, zda se v provozu objevil útok na SSH službu. Činí tak na základě CUSUM regulačního diagramu. Nejdříve vždy označí fixní počet pokusů o přihlášení za událost (*event*). Následně vyhodnocuje, zda došlo k významnému nárůstu v pozorovaném indikátoru selhání. V případě odhalení takového posunu označí tuto událost a několik předcházejících za vymykající se kontrole (*out of control*). Počet předcházejících událostí, které detektor označí, udává průměrná doba běhu detektoru.

4.2.3 Určení útočníků

Finální fáze se zabývá vyloučením běžných uživatelů z událostí, které byly označeny při analýze provozu. Ověřování se provádí pro každou kombinaci hosta a uživatelského jména zvlášť, především kvůli technologii NAT, která umožňuje více zařízením komunikovat z jedné veřejné adresy.

Prvně je třeba rozlišit uživatele, kteří se pouze spletli při psaní hesla ke svému uživatelskému účtu, čehož se dosahuje pomocí zkoumání historie příslušného hosta. Za validní uživatele jsou pak označeni ti, kteří se již v minulosti byli schopni k účtu přihlásit: pokud se host dokázal k účtu přihlásit již dříve, nemá důvod na tento účet nyní útočit. Tímto způsobem sice můžeme vyloučit i útočníky, kteří k účtu získali přístup před tím, než způsobili detekovatelnou změnu v provozu, avšak tato situace je při striktnější parametrizaci analyzátoru provozu velmi nepravděpodobná.

Druhým případem pak jsou překlepy v uživatelských jménech, které autoři metody navrhují odhalovat za pomoci historie hosta. Konkrétně pomocí zjišťování zda se host byl schopen přihlásit k účtu, jehož název je podobný zaznamenanému pokusu. Podobnost je doporučeno zjišťovat pomocí editační vzdálenosti jedna mezi názvy.

Kapitola 5

Návrh řešení

Po důsledné analýze problematiky slovníkových útoků jsem se zaměřil na návrh aplikace, která by byla schopná detekovat útoky hrubou silou, především distribuovanou variantu tohoto útoku popsanou v sekci 1.2.2.2.

Z prací zabývajících se detekcí distribuovaných útoků, které jsem představil v sekci 3.2, jsem zvolil metodu zabývající se statistickou analýzou přístupových záznamů popsanou v kapitole 4.2.

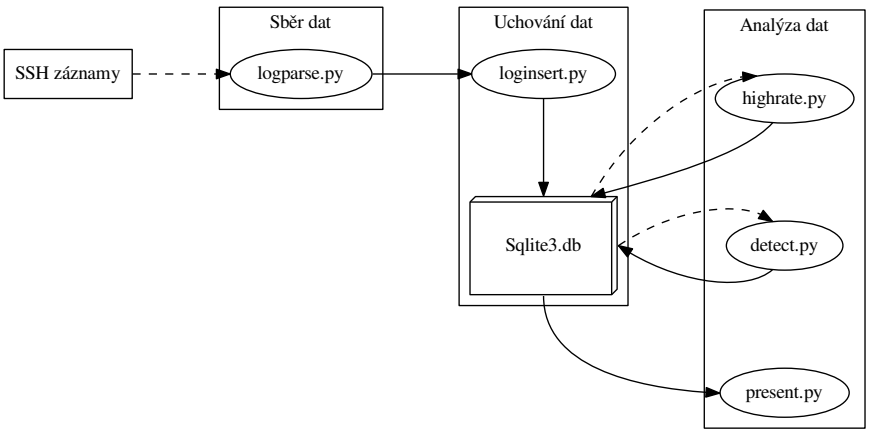
Tuto metodu jsem zvolil především z následujících důvodů:

- schopnost odhalit úspěšný distribuovaný útok – Tuto vlastnost považuji u detektoru za naprosto klíčovou. V případě pouhého identifikování osamělých útočníků lze zavést určitá protiopatření, jako například zamezení přístupu těmto vzdáleným hostům ke službě. Pokud není detekce útočníků bezchybná, může tím dojít k zablokování legitimních uživatelů. Pokud jsme ale schopni detekovat, že se k uživatelskému účtu přihlásil host podezřelý z účasti na distribuovaném útoku, můžeme se pokusit jeho totožnost ověřit například za pomoci vynuceného resetování hesla pomocí emailové schránky spjaté s účtem.

- přímočarý postup k identifikaci chybných poplachů – Metoda bohužel není plně automatizovaná a finální rozhodnutí o potenciální hrozbě vzdáleného hosta je ponecháno na provozovateli programu. Přesto je tato metoda na základě historie vzdálených hostů schopna vyloučit vysoké procento planých poplachů.

- není dostupný prototyp – V současnosti neexistuje žádná dostupná implementace tohoto postupu.[2] Pro další výzkum této metody považuji existenci dostupného prototypu za nezbytnou.

[2] Autoři této metody sice při vytváření vědeckého článku implementovali prototyp. V průběhu emailové konverzace však uvedli, že detektor není v publikovatelném stavu.



Obrázek 5.1: Návrh aplikace

5.1 Struktura projektu

Kvůli rozsahu projektu jsem se rozhodl rozdělit práci do následujících úloh:

- Sběr dat – Formát záznamů o přihlášení k SSH službě není jednotný, liší se jak mezi jednotlivými systémy, tak mezi jednotlivými stroji. Z tohoto důvodu jsem se rozhodl věnovat jednu část projektu získávání potřebných informací ze systémových logů.

- Skladování dat – Pro správnou funkčnost detektoru je nezbytné archivovat historii záznamů. Logickou volbou se zdálo být využití databázového stroje, protože je pro tyto účely nejvhodnějším řešením a usnadňuje i konečnou analýzu dat.

- Analýza dat – Poslední a nejdůležitější část projektu se zabývá odhalováním útoků v zaznamenaných datech.

Při návrhu aplikace jsem se inspiroval jednou z hlavních myšlenek unixových systémů: programy by měly být spíše jednoúčelové nástroje,[3] které jsou schopné spolu navzájem spolupracovat.

Návrh mé aplikace ilustruje obrázek 5.1.

[3] *Do one thing well.*

5.2 Použité technologie

5.2.1 Python

Vzhledem k prvotní myšlence na vytvoření detektoru, který bude založen na analyzování paketů procházející síťové komunikace, se jevilo jako logické zvolit k implementaci aplikace jazyk C, především kvůli standardní síťové knihovně `libpcap`. Na základě analýzy problematiky jsem však dospěl k názoru, že tento přístup se nejeví jako vhodný, protože SSH je zabezpečený protokol.

Protože tím odpadly největší přínosy, pro které bylo původně zamýšleno aplikaci implementovat v jazyce C, rozhodl jsem se zvolit jiný jazyk. Nakonec jsem zvolil jazyk Python verze 3, který se mi v minulosti osvědčil při provádění textových transformací.

5.2.2 GeoLite2

Pro získání maximálního množství informací o vzdáleném klientovi, jsem se rozhodl použít Geolite2. Jedná se o bezplatnou variantu projektu společnosti MaxMind, který shromažďuje informace o lokaci IP adres. Tyto informace jsou relevantní především v posledním kroku detekce, protože je založena na lidské analýze výstupu a tyto informace mohou pomoci při rozhodování o legitimitě klienta. Obzvlášť přínosné pak tyto informace mohou být v případě serveru, jež není obvykle navštěvován z mnoha různých zemí.

5.2.3 SQLite3

Pro uchování dat jsem zvolil databázi SQLite3 z důvodu snadného nasazení a použití. Jedná se o minimalistickou databázi, která nevyžaduje žádnou konfiguraci a nepotřebuje ani samostatný obslužný proces. Umožňuje přístup k databázi pomocí SQL a od verze Pythonu 2.4 je databázové API součástí standardní knihovny.

5.2.4 SciPy

S ohledem na numerické výpočty, které je potřeba provádět pro správnou parametrizaci detekčního skriptu, jsem se rozhodl použít matematické knihovny pro Python SciPy a NumPy. Tyto knihovny jsou vhodné jak pro statistické výpočty ARL, tak pro generování testovacích dat.

Kapitola 6

Implementace

Aplikaci tvoří sada skriptů v jazyce Python, které implementují jednotlivé podúlohy, tak jak jsem je popsal v návrhu. Každý skript doprovází standardní manuálová stránka.

Vytvořené skripty lze při jejich spouštění parametrizovat pomocí přepínačů a pozičních argumentů na příkazové řádce. K těmto účelům jsem využil modul `argparse`, jenž je součástí standardní knihovny a umožňuje snadné zpracování zadaných argumentů.

Pro potřeby práce se SQLite3 databází jsem vytvořil modul `database.py`, který slouží k ověřování, zda uživatelem specifikovaná databáze existuje, zda k ní má skript přístup a zda obsahuje všechny nezbytné položky pro správnou funkčnost aplikace.

6.1 Sběr dat

6.1.1 logparse.py

Účelem skriptu je transformace relevantních záznamů do jednotného formátu. Za podstatné považuji záznamy, které obsahují informace o přihlašování, jako je čas pokusu, uživatelské jméno, vzdálená adresa a informace, zda byl pokus úspěšný. Při rozhodování o způsobu, jakým ve vstupu rozeznávat podstatné záznamy, jsem volil mezi užitím regulárních výrazů a vytvořením lexikálního analyzátoru. Nejprve jsem se pokusil použít existující lexikální analyzátor, který je součástí programu Sshguard. Tento analyzátor je schopný monitorovat více systémových logů v reálném čase a rozeznávat širokou škálu formátů. Kvůli časové náročnosti modifikace analyzátoru jsem pro implementaci prototypu zvolil regulární výrazy. Užití regulárních výrazů mi značně urychlilo vývoj funkčního prototypu.

Systémové logy obvykle neobsahují informace o roku, v němž byly pořízeny, proto lze skript parametrizovat pomocí přepínače `-y`, kterým uživatel specifikuje rok, v kterém byl pořízen první relevantní záznam ve vstupu. V případě,

že skript zaznamená podezřelé datum, varuje uživatele pomocí standardního chybového výstupu.

Skript je schopen číst data jak ze souboru, tak ze standardního vstupu. Tyto možnosti lze specifikovat jako argumenty na příkazové řádce. Pokud je skriptu předáno více argumentů, pokusí se přečíst a zpracovat data ze všech udaných zdrojů, přičemž jejich pořadí považuje za chronologické.

6.2 Skladování dat

6.2.1 Struktura dat

Údaje o pokusech o přihlášení udržuje jednoduchá databáze, jejíž schéma znázorňuje obrázek 6.1.

6.2.2 loginsert.py

Skript `loginsert.py` slouží k ukládání záznamů do databáze. Skript je schopen číst data buď ze souboru, nebo ze standardního vstupu. Záznamy, které neodpovídají jednotnému formátu popsanému v dokumentaci, jsou ignorovány.

Pro specifikování databáze existují dva přepínače, které se vzájemně vylučují, avšak jeden z nich musí být zvolen. Pomocí `-c` lze vytvořit prázdnou databázi, přepínač `-d` pak specifikuje použití již existující databáze.

Skript může také pracovat v režimu, kdy na standardní chybový výstup udává informace o počtu zpracovaných záznamů. Tento režim lze aktivovat pomocí přepínače `-v`.

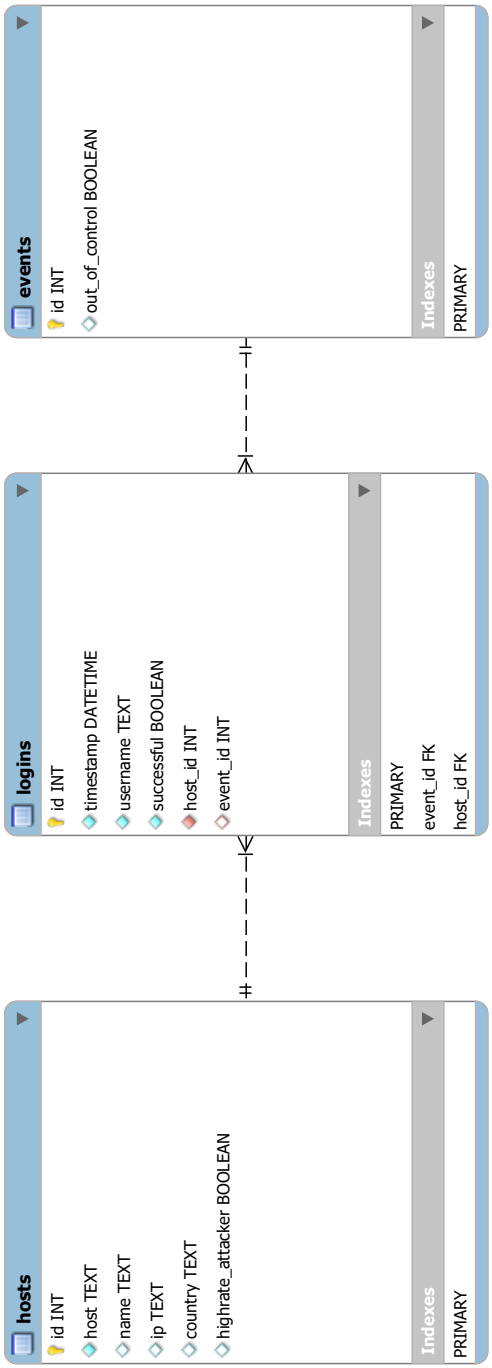
Při opakovaném běhu skriptu nad stejnými daty dochází k duplikování záznamů o přihlášení. Toto je zamýšlené chování, protože jsem se rozhodl ponechat uživateli plnou volnost při vkládání záznamů. Záznamy o vzdálených hostech duplikovány nejsou, pokud host již jednou v databázi existuje, jsou mu nové záznamy o aktivitě přiděleny.

6.3 Analýza dat

Nejdůležitější částí projektu pak je samotná analýza dat získaných pomocí předchozích kroků. Analýza se skládá ze tří skriptů, které uvádím v pořadí, ve kterém je zamýšleno jejich použití.

6.3.1 highrate.py

Pro potřeby detekce hostů, kteří útočí s vysokou frekvencí, jsem se rozhodl použít postup, který jsem popsal v 4.2.1. Mezní počet neúspěšných přihlášení lze nastavit pomocí přepínače `-t` (`--treshold`).



Obrázek 6.1: Entity-relationship diagram databáze

Samotnou implementaci tohoto přístupu jsem provedl pomocí fronty, jež je průběžně naplňována do doby, dokud jsou první a poslední záznam vzdáleny maximálně v rozmezí jedné hodiny. Pokud je vzdálenost větší než jedna hodina, je ze začátku fronty odebrán jeden záznam. V případě nalezení úspěšného pokusu o přihlášení je fronta vyprázdněna. Algoritmus takto pokračuje, dokud nejsou přečteny všechny záznamy hosta. Nastane-li situace, že délka fronty v průběhu běhu algoritmu překročí uživatelem definovanou mezní hodnotu (`-t`), je host prohlášen za útočníka. Jedná se o optimální algoritmus pro tuto úlohu, který pracuje asymptoticky v lineárním čase. Neformálním důkazem tohoto tvrzení je, že pro rozhodnutí zda host útočil, stačí každý jeho záznam o přihlášení zpracovat pouze jednou.

Při označení útočníka je tato informace zaznamenaná v databázi. Skript také kontroluje, zda se host v dostupné historii byl schopen přihlásit k systému. Pokud ano, informuje o této skutečnosti uživatele pomocí standardního chybového výstupu. Je pak na posouzení uživatele, zda nedošlo k úspěšnému útoku na jeho systém.

Standardně skript nevypisuje krom potenciálních bezpečnostních rizik žádné informace. Uživatel však může zapnout pomocí přepínače `-v` režim, při kterém bude skript uvádět na standardní chybový výstup informace o detekovaných útočnících.

6.3.2 detect.py

Tento skript v zadané databázi označuje podezřelé události.

V průběhu činnosti skriptu dochází k vytváření událostí. Skript určuje, zda byla událost v normě, pomocí algoritmu CUSUM, jehož všechny části lze parametrizovat pomocí příslušných přepínačů. Události a informace o nich jsou při každém spuštění nejdříve smazány a to především pro usnadnění testování parametrizace skriptu.

Přiřazování záznamů k událostem jsem vyřešil pro každou událost v jediném SQL dotazu. Stejně tak označení předchozích událostí při nalezení útoku se děje pomocí jednoho SQL dotazu. Dosáhl jsem toho pomocí užití datových struktur seznam a fronta, které slouží k průběžnému skladování potřebných identifikátorů.

Standardně skript neposkytuje žádný výstup, pouze aktualizuje příslušné položky v databázi. Lze však aktivovat režim pomocí přepínače `-v`, při němž skript poskytuje na standardní chybový výstup informace o tom, které události označil za podezřelé.

6.3.2.1 arl.py

Tento pomocný skript slouží k usnadnění parametrizace CUSUM algoritmu. V současné době je skript schopen vypočíst průměrnou dobu běhu a očekávanou střední hodnotu pro diskrétní binomiální rozdělení. Při drobném zásahu

do kódu lze však změnit jeho funkčnost a tyto hodnoty spočíst pro libovolné diskrétní rozdělení náhodné veličiny.

Výpočet průměrné doby běhu jsem vyřešil pomocí Markovových řetězců. Pro zadané parametry je vytvořena přechodová pravděpodobnostní matice a spočtena první položka výsledného vektoru. Protože nepotřebuji znát celý výsledný vektor (ten obsahuje i hodnoty pro kumultativní sumy s jiným počátkem než nulou), a protože rovnice splňuje podmínky pro užití Cramerova pravidla, bylo toto pravidlo využito pro efektivní řešení výpočtu průměrné doby běhu detektoru.

6.3.3 present.py

Vytvořené události z předchozích kroků je ještě nutné zpracovat. Je třeba označit útočníky a dbát na to, aby pokud možno nedocházelo k označování legitimních uživatelů.

Skript `present.py` tento problém řeší tím, že postupně projde všechny události, které byly označeny jako podezřelé, a pro každý neúspěšný pokus o přihlášení zjistí, zda se v minulosti host byl schopen k účtu přihlásit. Pokud ano, pak je host ve vztahu k tomuto účtu prohlášen za jeho oprávněného vlastníka. Výsledky tohoto dotazování jsou průběžně ukládány do dvoudimenzionálního asociativního pole. Z jedné položky tohoto pole lze pak zjistit identifikátor uživatele, účet, ke kterému se přihlašoval během podezřelé události, a zda byl označen za útočníka.

Posledním krokem skriptu je ověření, zda během zaznamenané historie některý z útočníků při útoku uspěl.

Skript standardně vypisuje pouze varování v případě detekovaní úspěšného útoku. Pomocí přepínače `-v` lze však aktivovat režim, při kterém skript poskytuje informace o všech hostech, kteří útočili, a vypíše uživatelská jména, která zkoušeli během označených událostí.

Kapitola 7

Testování aplikace

7.1 Datové sady

Schopnost aplikace odhalit skryté útoky jsem testoval na datových sadách, které obsahovaly útoky zachycené v reálném provozu a simulovaný provoz legitimních uživatelů.

7.1.1 Získání záznamů reálných útoků

Pro testy aplikace jsem shromáždil tři datové sady, které pocházejí z SSH serverů s velmi nízkým počtem uživatelů. (Konkrétně se jedná o SSH logy ze serverů `biblio.stare.cz` a `slechlib.fit.cvut.cz`.) Pro potřeby testování detektoru jsou datové sady s minimálním provozem nevhodné, protože příliš zjednodušují úlohu. Lze však z těchto datových sad snadno vyloučit legitimní klienty a vytvořit tak záznamy reálných útoků. Statistiky získaných útoků uvádí tabulka 7.1. Výrazný pokles útoků s vysokou frekvencí po roce 2012 je zapříčiněný nasazením firewallu, který odhaleným útočníkům zamezil přístup k serveru.

Analýzou útoků, které nebyly odhaleny za pomoci firewallu ani mého highrate skriptu, jsem zjistil, že na sledované servery bylo skrytě útočeno téměř

Tabulka 7.1: Statistiky zachycených útoků

Název datové sady	**Biblio12**	**Biblio15**	**Slechlib16**
Od	30. 6. 2012	2. 7. 2015	17. 3. 2016
Do	5. 9. 2012	25. 12. 2015	9. 5. 2016
Highrate útočníků	17	48	15
Highrate útoků	10644	1861	245
Lowrate útočníků	137	281	85
Lowrate útoků	900	1418	894

každý den. Ukázku vývoje takových útoků znázorňuje graf 7.1.

Po bližším zkoumání útoků jsem objevil dvě rozdílné strategie, které útočníci využili k maskování svých pokusů. První strategie spočívala v rozložení útoků na co nejvíce strojů a z jedné adresy tak nikdy nepřišlo více než pár požadavků.

```
2015-12-13 00:53:37 Failed password for support from 67.14.209.71
2015-12-13 00:54:25 Failed password for support from 72.28.235.45
2015-12-13 00:54:27 Failed password for support from 81.8.187.170
2015-12-13 00:54:29 Failed password for support from 71.245.83.22
2015-12-13 00:54:37 Failed password for support from 138.36.62.164
2015-12-13 00:54:40 Failed password for support from 108.46.20.66
2015-12-13 00:55:03 Failed password for support from 191.5.194.46
2015-12-13 00:55:09 Failed password for support from 186.64.92.58
2015-12-13 00:55:14 Failed password for support from 168.167.132.203
2015-12-13 00:55:18 Failed password for support from 208.123.154.16
```

Druhá metoda byla založena na odhalení nastavení firewallu, kdy útočník obětoval několik strojů, než objevil frekvenci při níž firewall již nezasáhl.

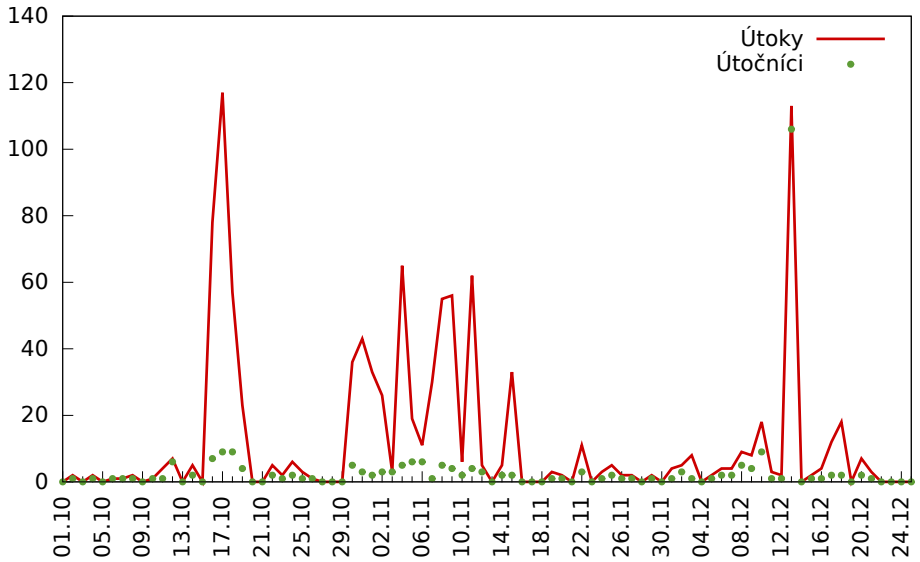
```
Highrate útočníci zablokováni firewallem:
222.186.56.138
222.186.56.143
222.186.56.150
222.186.56.160
222.186.56.168
222.186.56.175
222.186.56.42
222.186.56.43
222.186.56.44
222.186.56.45
222.186.56.48

Neodhalení útočníci v datové sadě biblio15:
222.186.56.92 - 418 pokusů
222.186.56.117 - 416 pokusů
222.186.56.97 - 90 pokusů
```

7.1.2 Simulace provozu

Pro vytváření simulací provozu jsem napsal skript, jehož výstup je shodný s požadovaným formátem vstupu skriptu `loginsert.py`, a proto je snadné vygenerovaná data vložit do databáze.

7.1.2.1 simulatetrafic.py

Skript generuje v určeném časovém rozmezí záznamy o přihlášení od daného počtu uživatelů. Rozestup mezi pokusy je fixní a velikost je závislá na po-



Obrázek 7.1: Graf zaznamenaných útoků

Tabulka 7.2: Průměrná doba běhu detektoru udávaná v počtu událostí

Mezní hodnota	**Doba běhu při normálním provozu**	**Doba běhu při útoku**
1	11,02	1,2
5	159,92	2,25
10	6572,89	3,93
15	26953,37	5,6
20	11028849,01	7,27

žadovaném počtu pokusů za den. Přihlášení selže s pravděpodobností danou přepínačem `-p`.

7.2 Parametrizace skriptů

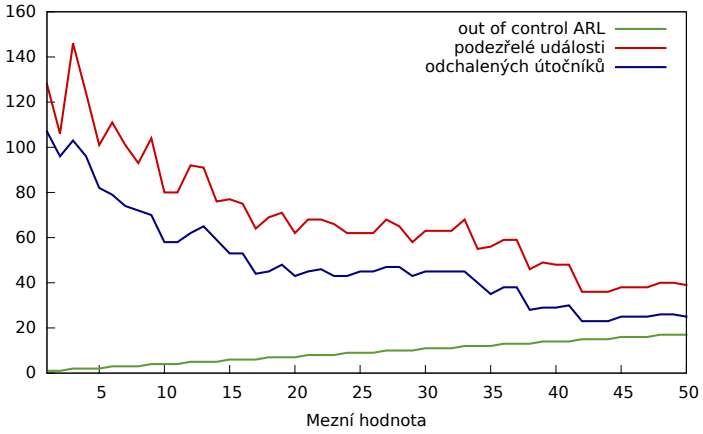
Parametry jsem určil na základě prostudování kapitoly, která se zabývá parametrizací detektoru, v článku [1].

Při vytváření simulací jsem generoval 500 pokusů o přihlášení za den s pravděpodobností chybného pokusu `-p 0,07`.

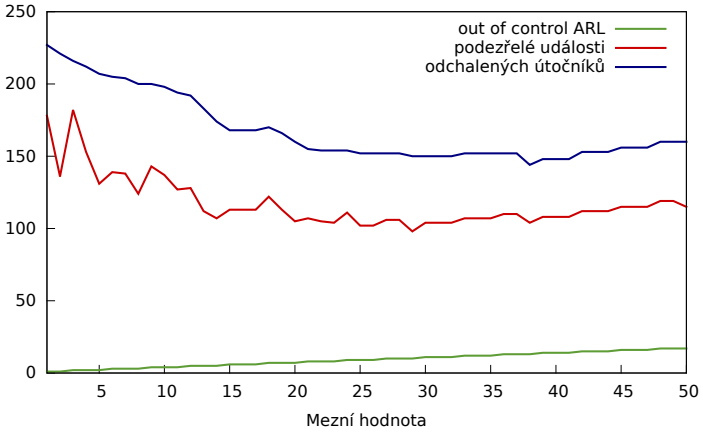
Detektor jsem nakonfiguroval tak, aby každých 100 pokusů o přihlášení vytvořil novou událost. Výpočtem jsem určil očekávanou hodnotu počtu chybných pokusů v události na 7. Odchylku od střední hodnoty, která má být detekována, jsem nastavil na 6. Průměrnou dobu běhu detektoru jsem pak pro určené parametry vypočetl pro různé mezní hodnoty a výsledky těchto výpočtů uvádím v tabulce 7.2.

7.3 Výsledky testů

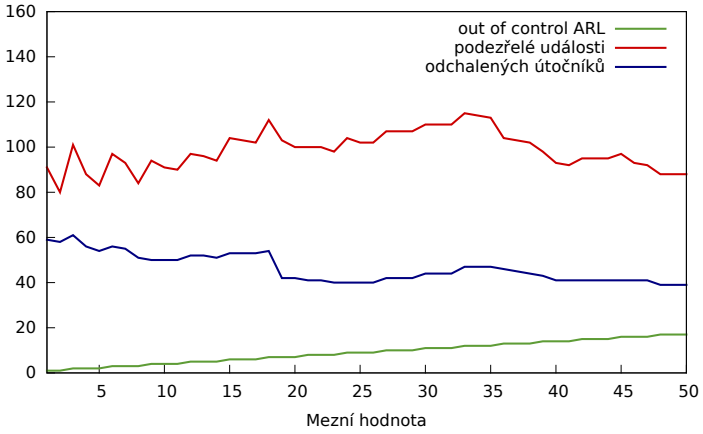
Testy jsem spustil pro několik různých nastavení mezní hodnoty. Počet odhalených skrytých útočníků, počet podezřelých událostí a vypočítanou průměrnou dobu běhu při útoku uvádím v grafech na obrázku 7.2. Z výsledků je patrné, že detektor byl schopen odhalit vysoké procento útočníků. Nejlepších výsledků detektor dosáhl očekávaně při nejnižší mezní hodnotě. Neprojevil se zde však vliv doby běhu při normálním provozu, protože datové sady obsahovaly relativně malý počet záznamů o přihlášení. Výsledky testování především ověřují, že aplikace je schopná detekovat skryté útoky, ale nejsou přínosné pro zjištění obecné parametrizace detektoru. Myslím si, že dalším krokem ve vývoji detektoru by mělo být určení parametrů pro reálný provoz. Autoři článku již s parametrizací pro reálný provoz přišli, netvrdí však, že je tato parametrizace použitelná obecně. Detektor nastavili tak, aby byl schopný odhalit skryté útoky ve veškerém provozu v Lawrence Berkeley National Laboratory, který je složen z provozu na více než dvou tisících SSH serverů.



(a) Datová sada: Biblio12



(b) Datová sada: Biblio15



(c) Datová sada: Slechlib16

Obrázek 7.2: Výsledky testů

Závěr

Hlavními cíli práce bylo vytvoření aplikace pro detekci skrytých útoků hrubou silou, vytvoření uživatelské dokumentace, provedení testování funkčnosti a základní vyhodnocení úspěšnosti. Všechny tyto cíle se podařilo splnit. Výsledná aplikace je schopna sbírat a uchovávat záznamy o přihlášení ze systémových logů a pomocí statistické analýzy odhalit skryté útočníky. Testování proběhlo na datech obsahujících mnou vytvořené simulace provozu legitimních uživatelů a útoky zaznamenané v reálném provozu.

Z důvodu časové náročnosti zůstaly některé části detektoru pouze rozpracované. Aplikace je schopna získávat data pouze z jednoho formátu logů. Nekompletní je část zabývající se eliminací falešných poplachů. Více pozornosti by si také zasloužila prezentace odhalených útočníků, která by znázorňovala jednotlivé distribuované útoky.

Pro nasazení detektoru v praxi je nezbytné určit, jaké parametry jsou pro běh detektoru nejvhodnější. Je pravděpodobné, že tyto paramatry nelze určit paušálně — nutným vstupem takového průzkumu jsou reálné dlouhodobé logy SSH provozu s netriviálním počtem legitimních uživatelů.

Původním záměrem bylo implementovat prototyp aplikace v jazyce C, protože tento jazyk nejvíce vyhovoval zamýšlenému řešení. Po důsledné analýze problematiky jsem objevil vhodnější řešení, které však již výhody jazyka C neupotřebovalo. Se svolením vedoucího práce jsem se rozhodl aplikaci implementovat v jazyce Python verze 3. Tato volba umožňuje snadnější přenositelnost aplikace mezi různými operačními systémy a také výrazně usnadnila vytvoření funkčního prototypu.

Literatura

[1] JAVED, M.; PAXSON, V.: Detecting stealthy, distributed SSH bruteforcing. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM, 2013, s. 85–96.

[2] MENEZES, A. J.; OORSCHOT, P. C.; VANSTONE, S. A.: *Handbook of applied cryptography*. CRC press, 1996.

[3] ŠEMBERA, R.: *Detekce slovníkových útoků na autentizační služby*. Diplomová práce, Masarykova univerzita, 2010.

[4] *Unmasked: What 10 million passwords reveal about the people who choose them* [online]. [cit. 2016-04-29]. Dostupné z: `http://wpengine.com/unmasked/`

[5] CID, D.: *SSH Brute Force – The 10 Year Old Attack That Still Persists* [online]. [cit. 2016-04-29]. Dostupné z: `https://blog.sucuri.net/2013/07/ssh-brute-force-the-10-year-old-attack-that-still-persists.html`

[6] VYKOPAL, J.: *Flow-based Brute-force Attack Detection in Large and High-speed Networks*. Dizertační práce, Masarykova univerzita, Fakulta informatiky, 2013.

[7] MALÉCOT, E. L.; HORI, Y.; SAKURAI, K.; aj.: Tracking Distributed SSH BruteForce Attacks. In *3rd International Joint Workshop on Information Security and Its Applications*, 2008, s. 1–8.

[8] DRAŠAR, M.: *Behavioral Detection of Distributed Dictionary Attacks*. Dizertační práce, Masarykova univerzita, Fakulta informatiky, 2015.

[9] BLAŽEK, R.: BI-PST - Přednáška č.3 - Náhodné veličiny I. 2014, [cit. 2016-05-03].

[10] BLAŽEK, R.: BI-PST - Přednáška č.4 - Náhodné veličiny II. 2014, [cit. 2016-05-03].

[11] KOSHTI, V. V.: Cumulative Sum Control Chart. *International Journal of Physics and Mathematical Sciences*, ročník 1, 2011: s. 28–32.

[12] BROOK, D.; EVANS, D.: An approach to the probability distribution of CUSUM run length. *Biometrika*, ročník 59, č. 3, 1972: s. 539–549.

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

FTP File Transfer Protocol

NAT Network Address Translation

SSH Secure Shell

VSFTPD Very Secure File Transfer Protocol Daemon

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `impl`
  - `src` .... zdrojové kódy skriptů, instalační instrukce, manuálové stránky
- `thesis` ..... text práce
  - `src` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF