VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALÝZA APLIKAČNÍCH FIREWALLŮ SOCIÁLNÍCH SÍTÍ

ANALYSIS OF APPLICATION FIREWALLS IN ONLINE SOCIAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **RADIM ZÍTKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. FILIP JANUŠ,**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Zítka Radim**
Program: Informační technologie
Název: **Analýza aplikačních firewallů sociálních sítí**
Analysis of Application Firewalls in Online Social Networks
Kategorie: Bezpečnost

Zadání:

1. Seznamte se základy automatického získávání dat z webu (tzv. web scraping).
2. Prostudujte dostupné materiály na téma aplikační firewally, zaměřte se na detekci botů.
3. Identifikujte aktuálně používaná opatření proti technice web scraping.
4. Analyzujte identifikovaná opatření a diskutujte jejich vlastnosti.
5. Navrhněte a implementujte vylepšení vybraného bezpečnostního opatření.
6. Demonstrujte přínos vašeho řešení.

Literatura:

- JANUŠ, Filip. Pokročilá evaluace úrovně privátnosti v sociálních sítích. Brno, 2020. Master's Thesis. Brno University of Technology, Faculty of Information Technology. 2020-07-15. Supervised by Malinka Kamil. Available from: https://www.fit.vut.cz/study/thesis/18824/

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních čtyř bodů zadání.

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

Vedoucí práce: **Januš Filip, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Práce popisuje způsoby, jak lze přistupovat na sociální sítě pomocí automatického robota, smysl tohoto přístupu a důvody vedoucí k používání ochran před automatickými roboty. Cílem je analyzovat aktuálně používané ochrany před automatickými roboty nejznámějších sociálních sítí (Facebook, Twitter, LinkedIn a YouTube) a rozšířit tyto informace mezi ostatní vývojáře. Ti poté mohou využít těchto znalostí pro zlepšení ochrany svých webových stránek. Výstupem této bakalářské práce je popis aktuálně používaných ochran sociálních sítí a dále návrh ochrany, která pomáhá odhalit automatické roboty na základě podezřelého chování.

Abstract

The thesis describes ways to attend social networks using automatic robots, meaning of this approach and the reasons leading social networks to use protection against automated robots. The aim of this thesis is to analyze currently used protections against automatic robots of the most famous social networks (Facebook, Twitter, LinkedIn and Youtube). These informations are available for other developers, which may use these information to improve their protection of own websites. The output of this bachelor thesis is description of currently used social network protections and proposal of protection that reveals automatic robots based on multiple identical behavior.

Klíčová slova

sociální sítě, automatický přístup, webscraping, aplikační firewall, Selenium, analýza chování, User-Agent.

Keywords

social networks, automatic access, webscraping, application firewall, Selenium, behaviour analysis, User-Agent.

Citace

ZÍTKA, Radim. *Analýza aplikačních firewallů sociálních sítí.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Filip Januš,

Analýza aplikačních firewallů sociálních sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Filipa Januše. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Radim Zítka
10. května 2021

Poděkování

Děkuji panu Ing. Filipu Janušovi za vedení této bakalářské práce. Vděčím mu zejména za jeho zájem, poskytnuté materiály, kvalitu konzultací a všechny odpovědi na dotazy, které mě inspirovaly v tvorbě této práce. Dále děkuji Janu Švábíkovi za rady týkající se implementace této práce.

Obsah

Kapitola 1

Úvod

Automatický přístup na sociální sítě znamená přístup na webové stránky pomocí automatického bota, jehož chování na webové stránce je předem definováno (na rozdíl od chování lidské osoby). Automatický přístup slouží k získávání dat z webových serverů, mezi něž pochopitelně patří i sociální sítě. Existuje mnoho podnětů proč používat automatický přístup – může jít například o získání a následné zpracování určitých vyžádaných dat. Ta se mohou týkat například lidí nebo produktů, o kterých tvůrce robota potřebuje zjistit určitá data (chování, zájmy apod.), a následně je použít pro svoje konkrétní účely. Sociální sítě mají mnoho důvodů, proč bránit automatickým botům v přístupu na jejich stránky, mezi něž patří například šíření SPAMu. Sociální sítě by měly mít pokročilou ochranu proti automatickému přístupu a právě proto je vhodné tyto ochrany studovat právě na sociálních sítích.

Tato bakalářská práce se zabývá ochranou proti automatickému přístupu na web, proto je nezbytné pochopit základní principy webové komunikace, které jsou popsány v kapitole 2. Kapitola 3 je věnována ochranám na síťové a transportní vrstvě a dále popisuje útoky na tyto vrstvy. Způsoby, pomocí nichž lze implementovat automatický přístup, obsahuje kapitola 5. Hlavní cíl práce je detekce aktuálně používaných opatření, která používají sociální sítě pro rozpoznání automatického robota. Tyto metody jsou popsány v kapitole 4 a mohou sloužit ostatním vývojářům jako inspirace pro ochranu jejich webových serverů.

Implementační část práce (kapitola 7) popisuje navržený způsob ochrany proti automatickému přístupu. Řešení simuluje reálnou sociální síť, kde jednotliví uživatelé sdílejí svoje znalosti či myšlenky. Je implementována v jazyce NodeJS s využitím databáze MongoDB. Navržená ochrana spočívá v detekci identického chování uživatele v určitém časovém intervalu a dalších podezřelých a neobvyklých aktivit.

Cílem této práce je pomoci vývojářům v získávání vědomostí o použití ochran proti automatickým robotům, které mohou následně implementovat na svoje webové stránky. Pomocí funkční ochrany lze zachovat soukromí svých uživatelů a bezpečnost dat. Dalším důvodem k vylepšování ochrany je zabránění šíření nežádoucího spamu či psaní nevhodných komentářů. Současně používané techniky, které využívají automatičtí roboti, jsou velmi pokročilé a odhalit automatický přístup se stává těžkým úkolem.

Kapitola 2

Webová komunikace

V této kapitole jsou čtenáři objasněny některé základní pojmy týkající se webové komunikace. Tyto znalosti jsou nezbytné pro funkční nastavení automatického bota[1] tak, aby ochrana webového serveru (tzn. firewally nižších vrstev a aplikační firewall, kapitoly 3 a 4) neodhalila podezřelý provoz a naopak – aby bylo možné analyzovat podezřelý webový provoz z pohledu vlastníka serveru.

2.1 Protokol HTTP/HTTPS

Protokol je v informatice sada pravidel. HTTP protokol definuje pravidla, podle kterých probíhá datový tok mezi klientem a serverem. Pomocí HTTP protokolu lze navštěvovat různé webové stránky. HTTP spojení začíná vždy klientská strana komunikace. Nástroj, který komunikuje na klientské straně se serverem se nazývá `User-Agent`. `User-Agent` představuje zpravidla *webový prohlížeč* 2.2, ale může to být i jiný nástroj. Při návštěvě webové stránky vytvoří `User-Agent` HTTP dotaz. Poté tento dotaz pošle na dotazovaný server, který obsahuje webovou stránku, a následně obdrží odpověď obsahující data (příklad HTTP komunikace na obr. 2.1). S těmito daty může `User-Agent` naložit dle vlastního uvážení. Webový prohlížeč z těchto dat sestavuje webovou stránku tak, aby byla čitelná pro uživatele (klienta). V případě, že `User-Agent` není webový prohlížeč, může se jednat o automatický přístup na server.

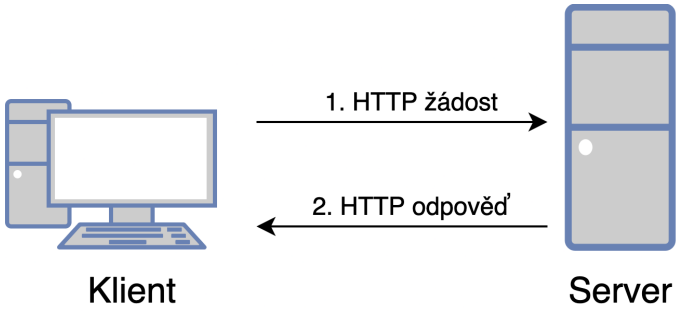
Klasické HTTP přenášelo data po síti nešifrovaně, takže si je mohl přečíst útočník, který v jakémkoliv bodě na lince odposlouchával přenos. Z tohoto důvodu vznikl protokol HTTPS, který zajišťuje šifrované spojení.

V rámci HTTP dotazu je nutné nějakým způsobem přenášet data od klienta na server. K tomu slouží metody GET a POST [26].

Metoda GET

Metoda GET využívá pro přenos dat URI požadovaného dokumentu. Pokud na webový server dojde požadavek ve tvaru: `GET /en-US/docs/Web/HTTP/ HTTP 1.1`, server ví, že tato koncová stanice žádá stránku odpovídající adrese `/en-US/docs/Web/HTTP/`. V tomto dotazu se však mohou nacházet další parametry, pokud se za touto cestou nachází znak otazníku. Ty obsahují nějaké další informace, které chce klient sdělit serveru. Jde například

[1] Pojmem bot je myšlen automat, který automaticky přistupuje na webové stránky.



Obrázek 2.1: Jednoduchá HTTP komunikace mezi dvěma uzly

o data z formuláře a má tvar: `?query=hledaný+výraz`. V tomto případě proměnná `query` obsahuje výraz „hledaný výraz“.

Metoda POST

Metoda POST slouží ke stejnému účelu jako metoda GET. Pomocí této metody může uživatel posílat objemnější množství dat na webový server. Odeslaná data mají stejný formát, avšak proti metodě GET se liší tím, že tato data posílá jako součást HTTP dotazu a nejsou tedy jednoduše viditelná v URI adrese.

HTTP hlavička

HTTP hlavička obsahuje metadata k odeslaným nebo přijatým datům. V dotazu HTTP hlavičky jsou informace jako je metoda, URL požadované stránky, IP adresa vzdáleného serveru, číslo portu, jazyk, čas vytvoření požadavku, informace o cookies, `User-Agent` (podrobněji níže) atd.

Automatický přístup často v HTTP hlavičce poskytuje pouze nejnutnější informace, čímž se stává jednodušeji odhalitelný. „Ruční“ úprava či vyplnění dalších informací v hlavičce HTTP dotazu může vést k úspěšnému překonání aplikačního firewallu.

User-Agent

Hlavička HTTP dotazu obsahuje několik informací, z nichž jedna je právě `User-Agent` (UA). Kvalitní webový prohlížeč dokáže poskytnout základní informace o svém systému pro účely optimalizace webové stránky. Typickým použitím je přehrávání videa ve vysoké kvalitě, které zvládnou přehrát pouze výkonnější koncové stanice. Tyto informace se přenáší prostřednictvím `User-Agent` hodnoty. `User-Agent` zahrnuje informace o systémových písmech, pluginech, verzi prohlížeče, CPU apod. Typický obsah UA má podobu: `Mozilla/5.0 (Macintosh; Intel Mac OS X 11_0_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36`.

2.2 Webový prohlížeč

Webový prohlížeč je aplikace, který na základě uživatelských příkazů komunikuje s určitým webovým serverem (viz protokol HTTP 2.1). Data, která získává touto komunikací, vykresluje jako webovou stránku podle přesně definovaných pravidel (viz 2.6). Webový prohlížeč

si ukládá data o navštívených stránkách, tzv. cookies 2.4. Při procházení jednotlivých stránek si může mimo cookies ukládat i obsah stránek do cache za účelem rychlejšího načítání v budoucnosti.

V moderních prohlížečích existují nástroje [2], pomocí nichž je možné jednoduše upravovat celou webovou stránku (HTML dokument), připojit se na konzoli a pracovat s JavaScriptovým obsahem apod. Tento nástroj také umožňuje sledovat síťový provoz dané webové stránky a poté jej analyzovat. Znalost získanou analýzou provozu lze využít pro naprogramování automatického bota, který poté může velmi dobře simulovat běžný uživatelský provoz [7].

Browser fingerprint

Browser fingerprinting, (volně přeloženo *„otisk prstu prohlížeče“*) je trackovací technika, která identifikuje jednotlivé uživatele na základě poskytnutých informací v HTTP hlavičce. Aby se mohla webová stránka zobrazit korektně, tak webový server žádá tyto informace (rozlišení obrazovky, operační systém, jazyková nastavení apod.) a pomocí nich ukládá Browser fingerprint daného uživatele. Pochopitelně může docházet ke shodám, kdy jeden uživatel má identický fingerprint jako jiný uživatel, ale to se spíše jedná o výjimku [24].

V tabulce 2.1 je znázorněn příklad otisku prstu prohlížeče. V pravém sloupci je uveden počet procent uživatelů, kteří mají stejnou hodnotu jako je uvedená v tabulce. Test byl proveden na stránce https://amiunique.org/fp. Hodnota, která se liší nejvíce, je hodnota `User-Agent`. V kombinaci s ostatními hodnotami a IP adresou lze s poměrně velkou pravděpodobností rozpoznat jednoho určitého uživatele.

Vlastnost	Hodnota	Poměr %
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.37	<0,01 %
Accept	text/html, application/xhtml+xml, application/xml; q=0.9,image/avif, image/webp,image/apng, */*;q=0.8,application/signed-exchange;v=b3;q=0.9	9,82 %
Content encoding	gzip, deflate, br	79,65 %
Content language	en-GB,en-US;q=0.9,en;q=0.8	1,77 %
Referer	https://www.google.com	0.32 %

Tabulka 2.1: Ukázka fingerprintu prohlížeče Google Chrome a procentuální poměr webových prohlížečů, které používají stejnou hodntou

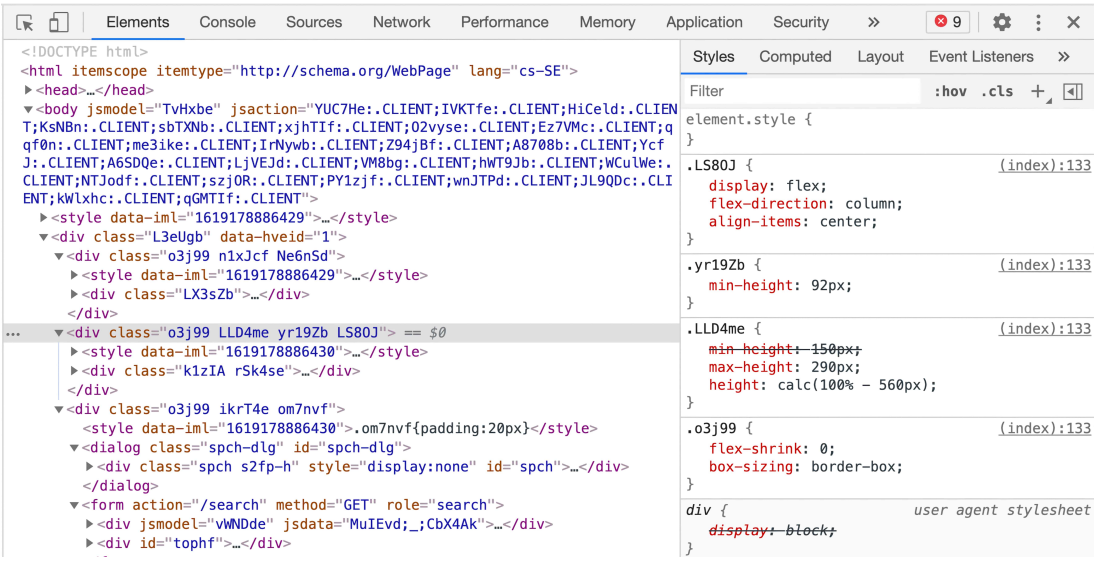
Chrome DevTools

Nástroj Chrome DevTools je sada nástrojů pro webové vývojáře, který umožňuje velmi rychle upravovat webovou stránku přímo v prohlížečích postavených na jádře Google Chrome. Těmito nástroji můžeme libovolně měnit vizuální vzhled stránky (uzly DOM stromu, CSS styly apod.), upravovat rozlišení displeje, zkoumat síťový provoz, vytížení paměti a vý-

[2] Pro prohlížeče pracující s jádrem Google Chrome se tento nástroj nazývá Chrome DevTools.

početní kapacity stanice, spravovat cookies aptd. Slouží k optimalizaci webových aplikací a hledání chyb.

Díky zobrazení zdrojového HTML dokumentu v Chrome DevTools lze identifikovat jednotlivé oblasti stránky, což lze využít například pro naprogramování automatických testů či sledovat změny HTML/CSS identifikátorů. Ukázka nástroje ChromeDevTools je zobrazena na obrázku 2.2. Nástroj Chrome DevTools se spouští v prohlížečích s jádrem Chromium *kliknutím pravého tlačítka myši → Inspect.*



Obrázek 2.2: Chrome DevTools

2.3 Anonymizační systém Tor

Síť Tor slouží k anonymizaci uživatelů, tedy slouží k ochraně jejich soukromí. Je udržována neziskouvou organizací *The Tor project* a veškerý zdrojový kód sítě Tor je open source. Nejvíce je využíván pro účely prohlížeče, jehož používáním získá uživatel téměř dokonalou anonymitu. Prohlížeč pro síť Tor se nazývá *TorBrowser* [25]. Může být využíván pro nelegální aktivity a umožňuje přístup do Dark Webu.

Princip sítě Tor funguje na principu tunelování na Tor síti, tzv. Onion routing. To zajišťuje, že veškerý provoz mezi klientem a serverem je šifrovaný (což je standard v dnešní době i bez použití Toru), nicméně jeho hlavní přednost spočívá v tom, že provoz je směrován přes více různých Tor serverů. Ty pak vystupují jako klient a zajistí původnímu klientovi takřka naprostou anonymitu. Pro účely této práce je nejdůležitější, že klient vystupuje pokaždé pod jinou adresou a velmi pravděpodobně z jiného místa (např. Londýn či New York). IP adresy, pod kterými klient vystupuje, jsou velmi často v gray listu[3], nicméně tyto adresy se neustále mění.

Rozpoznat Tor adresu není příliš složité. Jde například o nestandardní čísla portů, neexistující SSL vydavatel certifikátu, neexistující vydavatel certifikátu pro DNS apod. [9].

[3]Gray list je seznam IP adres, ze kterých byl dříve zaznamenán podezřelý provoz či jsou nějakým jiným způsobem podezřelé. Více v kapitole 3.4.

2.4 Cookies

Název cookies (přeloženo *„sušenka“)* byl odvozen z jeho původního významu. Při používání určitého webu se „pečou“ určitá data, která se ukládají právě do souborů cookies. Jsou to malé kusy dat, které obsahují data o proběhlých návštěvách různých webů. Usnadňují práci s webem, protože jednou nastavené údaje není nutné vyplňovat při každém přístupu znovu. Ukládají se do nich přihlašovací údaje (proto některé weby přihlašují uživatele automaticky), zvolený design stránky, obsah nákupního košíku apod. Soubory cookies jsou velmi důležité pro automatický přístup na webové servery. Pokud umí automatický robot správně pracovat s cookies, bude působit méně podezřele při dalších přístupech [19].

Vznik cookies

Pro automatický přístup je velmi důležité vědět, jak cookies vznikají. Při návštěvě stránky, která ještě nebyla uživatelem navštívena, resp. v souborech cookies neexistuje cookies pro danou stránku, není přítomen v HTTP žádosti atribut `cookies`. Server při vytváření odpovědi zahrne do HTTP odpovědi atribut (jeden či více) `Set-cookie` obsahující cookies, které se mají uložit do webového prohlížeče.

Doménová jména v cookies

Každá cookies musí obsahovat z podstaty věci informace o doméně, pro kterou dané cookies patří. Při návštěvě webové stránky hledá webový prohlížeč cookies patřící navštěvované doméně. Doménové jméno se nachází v poli `Domain`. Doméně *https://vutbr.cz/* bude odpovídat cookies, které obsahuje `Domain=vutbr.cz`. Avšak toto cookies bude uvažováno i v případě návštěvy *https://merlin.fit.vutbr.cz/*. Cookies často obsahuje atribut `Path`, který specifikuje místa na webovém serveru, při jejichž návštěvě se má toto cookies použít. Pro výše uvedenou doménu by se cookies s atributem `Path` nabývající hodnotu `/studuj` použilo pouze v případě návštěvy stránky *https://vutbr.cz/studuj/*.

2.5 Session

Data, se kterými uživatel pracuje na dané webové stránce, je poměrně nepohodlné a nebezpečné ukládat do uživatelského cookies (2.4) nebo je přenášet po síti od uživatele na server. Z tohoto důvodu se využívá HTTP session (překlad: *„relace“*). Klientské straně žádající přístup na server je přiřazen jednoznačný identifikátor (tzv. *JSESSIONID*), který je uložen v cookies a který jednoznačně identifikuje uživatelská data na serveru (čas přihlášení, počet přihlášení apod.). Při práci s daty uživatel a server pracují pouze s tímto identifikátorem. Uživatel se autentizuje zpravidla uživatelským jménem a heslem.

Tabulka s informacemi o sessions

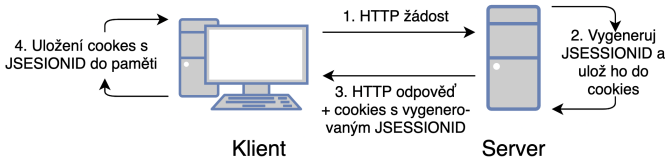
V tabulce 2.2 je ukázka, jak mohou být uložena data o sessions na straně serveru. Každý server může ukládat velké množství jakýchkoli informací, které potřebuje či může v budoucnu potřebovat.

JSESSIONID	username	data	lastAccess
A6CC39D2A106BAE4	radimzitka	78ha4d397ab	2020-12-02
76HE239B6D9CAA83	jansvabik	76e3a9cca40	2019-12-28

Tabulka 2.2: Uložená data o relacích na straně serveru

Vytvoření session

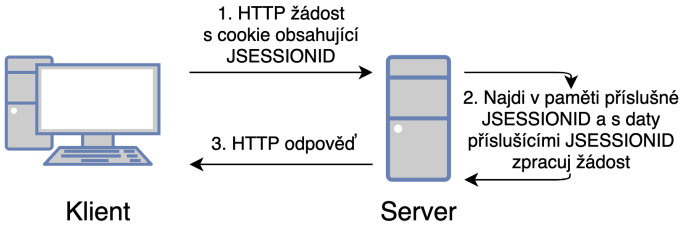
1. Klient vytvoří první HTTP žádost a pošle ji na webový server.
2. Webový server (resp. proces, který se stará o sessions) vytvoří HTTP session pro klienta a vygeneruje JSESSIONID, které uloží do cookies.
3. Odpověď včetně vytvořeného cookies je zaslána zpět klientovi a HTTP spojení je zrušeno.
4. Session cookie k webové stránce je uloženo ve webovém prohlížeči na klientské straně [32].



Obrázek 2.3: Průběh vygenerování JSESSIONID

Žádost o přístup na stránku s existujícím session ID

1. Klientský webový prohlížeč zažádá o přístup na stránku, přičemž do žádosti přidá cookies, které získal v předchozích spojeních.
2. Webový server obdrží žádost s existujícím session ID. Najde objekt odpovídající dané session v paměti serveru a zpracuje danou žádost s použitím dat uložených v nalezeném objektu.
3. Webový server vytvoří odpověď a zašle ji zpět na klientskou stranu [30].



Obrázek 2.4: Komunikace s již existujícím JSESSIONID

2.6 HTML

Značkovací jazyk HTML slouží pro popis webové stránky. HTML jazyk si klade za cíl, aby konečný vzhled popisované stránky nebyl ovlivněn parametry zařízení, na kterém je zobrazována. To znamená, že se webová stránka zobrazí ve velmi podobném designu jak na zařízení s malým displejem, tak na zařízení disponujícím velkým displejem. Jazyk vychází ze značkovacího jazyka SGML (viz RFC SGML), který je ale příliš komplikovaný pro využití v kódování webových stránek. Pro účely automatického přístupu je klíčová znalost jazyka HTML, jelikož program, který simuluje webový prohlížeč, stahuje určitou HTML stránku. Z ní poté extrahuje předem definovaná data nebo s ní jinak pracuje, ale veškerá tato práce se většinou řídí HTML tagy [31].

Popis jazyka

Jazyk HTML je množina tagů (značek), kde každý jeden tag může obsahovat atributy (vlastnosti), které nabývají různých hodnot. Každá HTML značka má jednoznačný účel. Tagy se dělí na párové a nepárové. Počet párových tagů v dokumentu musí být vždycky sudý. Typickým představitelem párové HTML značky je `<html>`, k němuž existuje dále v dokumentu stejný tag označený lomítkem, např. `</html>`. Nepárové tagy typicky nepokrývají žádnou oblast dokumentu. Zástupce nepárového tagu je znak nového řádku `<br>`.

Každý tag může mít určené svoje chování, které by mělo být definováno v příslušném dokumentu, který popisuje kaskádové styly stránky. Jde například o párový tag `<div>`, jehož atributem je identifikátor `id` a nabývá hodnoty `link-like-feed-f5x3`. Každý prvek s identifikátorem by měl být v HTML dokumentu jedinečný a díky tomu lze identifikovat akci (např. kliknutí), kterou uživatel provádí na stránce. S identifikátory HTML elementů pracuje aplikační firewall v implementační části, pomocí nichž rozeznává elementy, na které bylo uživatelem kliknuto.

2.7 JavaScript

JavaScript (JS) je nejznámější skriptovací jazyk pro webové stránky, který využívá mnoho dalších prostředí (Node.js, Adobe Acrobat). Společně s jazyky HTML a CSS tvoří jádro technologie WWW [33]. Javascript provádí kód ve webovém prohlížeči (tedy na straně uživatele).

S použitím jazyka JS je možno prakticky ihned reagovat na vstup uživatele, umožňuje tedy měnit HTML prvky (popř. je vytvářet), pracovat s jejich obsahem či designem. JS umožňuje reagovat na pohyb myši, kliknutí na grafický prvek nebo v reálném čase změnit hodnotu čísla či textu. Dále umožňuje práci s webovým prohlížečem jako je pohyb v historii a otevírání či zavírání oken. Kód JavaScriptu se vykonává v okamžiku, kdy na něj prohlížeč při procházení stránky narazí [28].

V JavaScriptu je napsáno poměrně velké množství desktopových aplikací pomocí frameworku Electron. Mezi tyto aplikace patří např. vývojové prostředí Visual Studio Code, chatovací programy Slack či Discord a Skype [17].

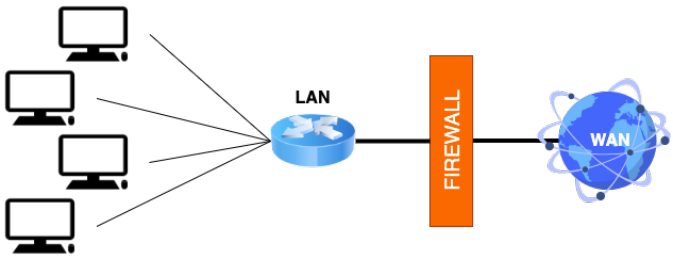
Kapitola 3

Firewally nižších vrstev

Kapitola se věnuje ochraně sítí na nižších vrstvách než je aplikační. Popisuje ochrany používané na těchto vrstvách a dále se zabývá různými útoky na nižší vrstvy. Každému serveru nebo stanici v síti hrozí jisté nebezpečí, že budou cílem útoku z vnější sítě. Důvodů útočit na nějaké zařízení může být mnoho (krádež citlivých dat či hesel nebo odstávka sítě) a je třeba jej chránit. Mnoho firem investuje nemalé částky pro ochranu svých dat a zařízení, protože v případě úspěšného útoku by následky mohly mít fatální dopad. Následující kapitola 4 věnující se aplikačními firewallu využívá získané znalosti z této kapitoly a úzce na tuto kapitolu navazuje.

3.1 Firewall

Brána firewall je zařízení či soustava zařízení, která určitým způsobem kontroluje komunikaci mezi zařízením v interní sítí a zařízením ve vnější síti za účelem ochrany vnitřní sítě. Historie firewallu sahá až do roku 1987, kdy se instituce pracující s citlivými informacemi začaly starat o důvěrnost svých dat. Hlavní myšlenka byla postavit jakousi bránu mezi zdánlivě bezpečnou interní síť a externími sítěmi. Dnešní firewally jsou často implementovány na lokální stanici v podobě antivirového programu. V rozsáhlejších (např. korporátních) LAN sítích představují firewall samostatná síťová zařízení, přes která teče veškerá komunikace s externími sítěmi. Role firewallu v síti je znázorněna na obrázku 3.1.



Obrázek 3.1: Schéma fungování firewallu

S postupem času se firewall vyvíjel a jeho druhy lze různě rozdělit v závislosti na vlastnostech každého z nich. V této práci jsou rozděleny [1] do tří velkých skupin:

- Paketové filtry
- Stavové filtry

- Aplikační brány[1]

3.2 Paketové filtry

Jde o nejjednodušší filtr síťového provozu. Filtruje pakety na úrovni síťové a transportní vrstvy a bývá velmi často implementován přímo na routeru nebo na serverech a v mnoha případech ho lze najít i na koncových stanicích. Paketový filtr je velmi rychlý a jednoduše implementovatelný, ale poskytuje pouze nízkou míru zabezpečení. Tento typ ochrany sítě staticky rozhoduje o propuštění paketu pouze na základě informací uvedených správcem sítě a neřídí se pořadím či významem dat. Paketový filtr se řídí sadou pravidel, mezi které patří zejména:

- Zdrojová a cílová IP adresa
- Zdrojové a cílové číslo portu

Všechny pakety, které nevyhovují zadaným kriteriím, jsou zahozeny. Filtr může pracovat i s dalšími parametry uvedené v hlavičce síťové či transportní vrstvy, avšak to už záleží na konkrétní implementaci.

3.3 Stavové filtry

Jde o pokročilejší technologii ochrany sítě, než poskytuje paketový filtr. Pracuje na úrovni transportní vrstvy, tedy s protokoly TCP a UDP. Pro každé jedno spojení si udržuje veškeré informace (zdrojovou a cílovou IP adresu, čísla portů), které vyplňuje při zakládání spojení.

Každý paket, který směřuje ven nebo dovnitř sítě, je zkoumán tímto filtrem. Podle informací v hlavičce transportní vrstvy hledá stavový filtr v tabulce existujících spojení záznam příslušný tomuto paketu. Podle aktuálních informací obsažených v hlavičce je nalezený záznam upraven a dále filtr rozhoduje, zda je nebo není paket zahozen.

3.4 IP listy

IP listy pracují podle vzoru paketového filtru, ale jsou rozšířeny na několik kategorií a díky tomu fungují inteligentněji než prostý paketový filtr. Každá IP adresa má svoje zařazení v IP listu, které se může v průběhu času měnit (na rozdíl od klasického paketového filtru, kde je filtrace IP adres nastavena staticky). Na základě zařazení určité IP adresy se dále odvíjí její oprávnění přistupovat k webové službě. S tímto typem ochrany pracuje také velmi často aplikační firewall. Existují 3 IP listy, mezi něž můžeme zařadit kteroukoli adresu:

- White list
- Gray list
- Black list

[1] Protože se tato kapitola věnuje pouze ochranám na síťové a transportní vrstvě, je ochrana na úrovni aplikační vrstvy popsána v kapitole 4, kde je tato problematika popsána podrobněji.

White listy

White IP list udržuje seznam IP adres, ze kterých je zaznamenáván pouze legitimní provoz. Při obdržení dotazu je velmi pravděpodobné, že se nejedná o útok. V případě, že server zaznamenává náznaky, které mohou jen vzdáleně připomínat podezřelou komunikaci, může tuto IP adresu zařadit do gray listu. Každá adresa, která není vedena v nějakém IP listu, je považována jako legitimní a je zařazena do white IP listu.

Gray listy

IP adresy v gray listu jsou vedeny jako podezřelé, ale provoz z nich není za každou cenu zablokován jako tomu bylo v případě IP adres v black listu. Pokud server přijímá dotaz z adresy, která náleží gray listu, server velmi dobře kontroluje aktivitu tohoto uživatele. V případě, že zjistí nějakou podezřelou aktivitu (např. vícenásobné zadání špatného hesla), může po uživateli chtít ověření třetí strany, např. CAPTCHA (viz 4.1). Adresy, které používá síť Tor (viz 2.3), se velmi často nacházejí právě v gray nebo black listech.

Black listy

Pokud server zaznamená dotaz z IP adresy, která se nachází na blacklistu, komunikaci blokuje a nepustí ji dál [11]. Jde o adresy nebo rozsahy adres, ze kterých byl dříve zaznamenán velmi podezřelý provoz, například odhalená phishingová aktivita nebo šíření SPAMu. Pokud útočník ví, že je jeho adresa evidována v black IP listu, existuje několik způsobů, jak to obejít. Nejčastějším způsobem je změna IP adresy (např. pomocí VPN[2]).

Velmi účinným nástrojem je podvržení IP adresy, tzv. IP spoofing, kdy si útočník uměle upravuje IP hlavičku v paketu. Existuje mnohem více způsobů, jak lze tento filtr obejít a o kterých je zmínka ve zdrojové literatuře tohoto odstavce.

3.5 Útoky a hrozby

Každá webová služba, která je provozována na Internetu, může být terčem kybernetického útoku. S čím citlivějšími daty daná služba pracuje, tím vzrůstá riziko útoku. Protože sociální sítě pracují zejména s citlivými daty svých uživatelů, útoky na tyto sítě jsou velmi časté a občas úspěšné. K obraně proti těmto útokům slouží zejména firewall.

DoS útok

Denial of Service je útok na webový server, kde se útočník snaží dostat server do nepoužitelného stavu pro normální uživatele. Jejím cílem je tedy zastavit cizí webovou službu běžící na napadeném serveru, aby její zákazníci či uživatelé *nemohli* webovou službu dál normálně používat. Hlavní podstata DoS útoku je vyčerpání jakékoliv kapacity serveru (např. procesor, RAM nebo šířka pásma internetového připojení). Útok lze provádět např. obrovským počtem ICMP žádostí (klasický `ping`) od útočníka či vytvářením falešných TCP spojení.

DDoS útok

Distributed Denial of Service je ve své podstatě stejný jako útok DoS. Rozdíl je v počtu stanic útočníka. V DoS útoku útočí pouze jedna stanice a v útoku DDoS útočí 2 a více

[2] O technologii VPN zde.

stanic. Jde tedy o řízený útok z více směrů ve stejný (domluvený) čas. Útoky DDoS mohou fungovat na stejném principu jako DoS útoky, ale mohou být sofistikovanější a složitější na odhalení [12].

Man-in-the-middle

Každá stanice připojená k síťovému provozu posílá či přijímá data, která je v případě špatného zabezpečení síťových zařízení možné odposlouchávat [4]. Funguje na principu odposlouchávání komunikace na lince mezi odesílatelem a příjemcem. Man-in-the-middle není útok, který by se aktivně snažil poškodit nějakou službu či server, ale poskytuje cizí osobě data, která odposlechla na určité síťové lince. V dnešní době jsou data velmi často šifrována, proto je tento útok velmi často neefektivní, ale v případě nešifrované komunikace (např. protokolem HTTP) jde o velmi účinný útok.

Kapitola 4

Aplikační firewall

Firewall na aplikační úrovni by měla být nejpokročilejší ochrana stanic v síti. Princip klasických firewallů je v tom, že zkoumají každý paket procházející do nebo ze sítě a firewall buď paket zahodí nebo ho propustí (více viz 3). Ochrana na aplikační úrovni spočívá v kontrole jednotlivých procesů a práce se soubory. Pokud se tedy útočníkovi podaří získat vstup na server a získat soubory obsahující citlivá data, je úkolem aplikačního firewallu poznat snahu pracovat s těmito soubory a včas zasáhnout [15].

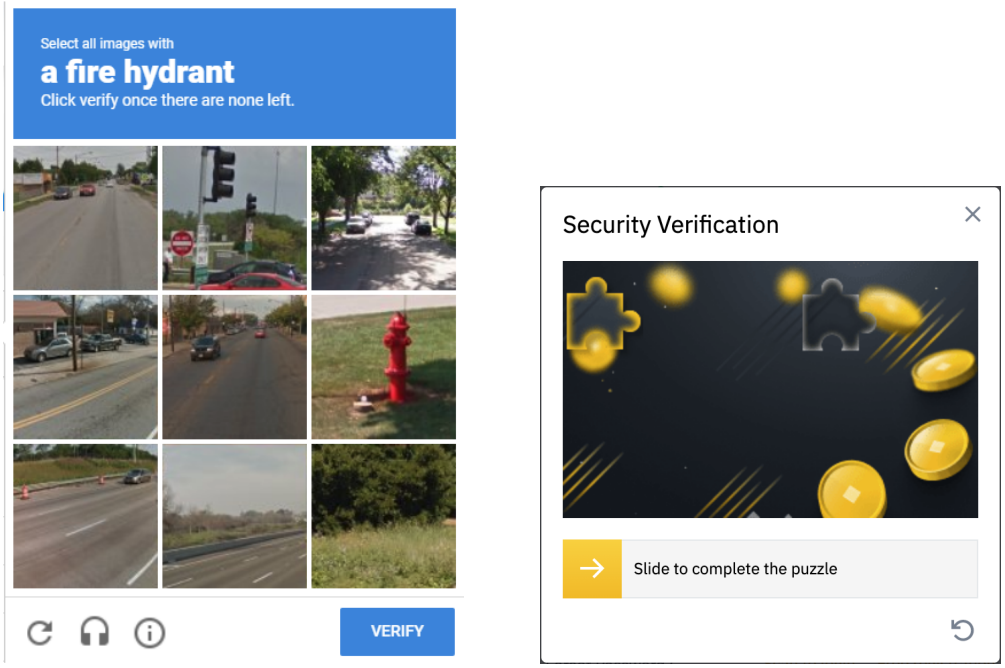
Mezi činnosti aplikačního firewallu patří rozpoznávání automatického přístupu na web, tedy rozpoznávat tzv. *webscraping* (viz 5.2). Každá analýza, kterou klasický nebo aplikační firewall provádí, zabírá určitý čas a tím zpomaluje odezvu k uživateli, což může vést k nižšímu uživatelskému pohodlí. Další věc je cena aplikačního firewallu, protože udržet trend se stávajícími druhy útoků není jednoduché. Aby provozovatel udržel svou webovou službu bezpečnou a zároveň uživatelsky přívětivou, může ho to stát mnoho finančních prostředků. V mnoha případech provozovatel serveru netuší, že je jeho stránka navštěvována automatickými boty a tuto skutečnost se dovídá třeba až v okamžiku, když zjišťuje příčiny snížené návštěvnosti jeho stránek.

4.1 Ověření lidské inteligence

Účinným nástrojem pro rozpoznání lidské inteligence jsou Turingovy testy [6]. Vychází z principu, že stroje nedokážou myslet, díky čemuž dokáže detekovat lidskou inteligenci. Pokud uživatel úspěšně splní daný Turingův test, čímž ověří lidskou inteligenci, může být úspěšně vpuštěn na webovou stránku. V opačném případě musí test buď opakovat nebo opustit stránku. V těchto případech bývá jeho IP adresa zařazena do IP gray listů (viz 3.4). Mezi velmi rozšířený způsob ověřování patří tzv. CAPTCHA.

CAPTCHA

CAPTCHA (Completly Automated Public Turing test to tell Computers and Humans Apart) je mechanismus, který ověřuje, zda je příchozí uživatel člověk nebo automatický stroj. Funguje na základě rozpoznávání, se kterým nemá problém většina lidí, avšak pro automatického bota činí zpravidla nepřekonatelnou překážku. CAPTCHA může mít různé úrovně obtížnosti, které se mohou stát obtížně překonatelné i pro člověka. I když tento mechanismus funguje poměrně spolehlivě na obranu proti botům, na uživatele dobrý dojem neudělá a může způsobit odchod návštěvníka ze stránky. Ukázka mechanismu CAPTCHA je znázorněna na obrázku 4.1.



Obrázek 4.1: Mechanismus CAPTCHA používaný pro rozeznání lidského faktoru.

4.2 Hrozby na aplikační vrstvě

Aplikační vrstva je nejcitlivější vrstva ze tří důvodů: je nejblíže uživateli, úspěšný útok má největší dopad ze všech dalších vrstev a má největší škálu možných útoků. Všechny níže uvedené útoky by měly být odhaleny ochranami vyšších vrstev, tedy například aplikačním firewallem.

Phisning

Cílem phishingu je získání citlivých dat ostatních uživatelů. Velmi častý způsob, jak lze tyto informace získat, je podvodný email nebo falešný účet na sociálních sítích. Tyto účty nebo emaily vypadají většinou velmi důvěryhodně a vedou uživatele k zadání citlivých dat (např. jméno a heslo do internetového bankovnictví), po jejichž vyplnění servírují citlivá data útočníkovi přesně podle jeho požadavků [4]. Sociální sítě se snaží bránit těmto útokům, ale mnohdy je veškerá snaha neúspěšná. Phishing zpravidla probíhá automaticky, proto je vhodné tyto boty odhalit a zakázat jim přístup na sociální síť.

Útoky DDoS na aplikační vrstvu

Podobně jako klasické útoky DDoS (viz 3.5) mají za cíl dostat webovou službu do stavu, kdy není schopna obsloužit normální uživatele. Oproti klasickému útoku DDoS však útočí na aplikační vrstvu, z nich nejznámější je tzv. HTTP floods attack („HTTP záplava“). To je útok, kdy se několik distribuovaných a synchronizovaných stanic dotazuje HTTP dotazem GET nebo POST, ale server nestíhá odbavit takové množství požadavků a donutí službu (či server) spadnout [5]. Aplikační firewall by měl být schopen tento typ útoku odhalit.

SQL injection

SQL injection útok může odhalovat mnoho citlivých informací uložených v SQL databázi (např. přihlašovací údaje, hesla, telefonní čísla atd.). SQL Injection se pokouší získat neoprávněný vstup do databáze vložením SQL kódu do formuláře. V případě, že se útočníkovi podaří získat strukturu databáze, může jednoduše manipulovat s daty uloženými v databázi [3]. Data získaná pomocí formuláře by tedy měla být zkontrolována (např. aplikačním firewallem), zda neobsahují škodlivý (spustitelný) kód a předejít útokům tohoto typu.

Cross-Site Scripting

Cross-Site Scripting (XSS) je typ útoku, kdy je škodlivý kód injektován na neškodný a důvěryhodný web tak, že se útočník chová jako normální uživatel, avšak ukládá do webové databáze škodlivý skript. K útoku dochází, když útočník získá kontrolu nad HTML obsahem, který je doručován uživateli webové aplikace, protože může do odesílané HTML stránky přidat vlastní JavaScriptový kód, který dříve uložil do databáze. Útočník může využít uživatelův účet k získání citlivých informací, k šíření spamu apod. [14, 22]. Ochrana webového serveru musí zabránit snaze nepovolených uživatelů o jakoukoli manipulaci se soubory na serveru.

Broken Authentication and Session Management

Příčinou tohoto útoku je chybná autentizace a správa relace, protože může dojít k odcizení uživatelských účtu. Obvyklá příčina je špatné schéma správy relací a autentizace uživatelů (např. nevhodný způsob obnovy hesla) [22]. Automatičtí roboti mohou tento nedostatek webových stránek využívat např. k získání přístupu do cizího účtu.

Sensitive Data Exposure

Veškerá citlivá data, která se ukládají do webové databáze, by měla být důkladně zašifrována. Pokud tomu tak není a útočník ovládne databázi, získá přístup k citlivým datům, která jsou v databázi uložena. Tento útok může mít vážné dopady, protože dokáže získat citlivé údaje a zneužít jich. Je tedy důležité udržovat data na serveru zašifrovaná a přístup k nim jistit aplikačním firewallem [29].

4.3 Analýza provozu

Aplikační firewall používá k odhalení automatických botů zpravidla více technik. Techniky spadající pod analýzu provozu lze definovat tím, že nepracují s HTML kódem požadované stránky, ale snaží se identifikovat hosta a na základě jeho chování určit, zda se jedná o podezřelý provoz či nikoliv. O každému hostovi, který je většinou definován IP adresou, může aplikační firewall ukládat několik informací týkajících se jeho chování, na jejichž základě později rozpoznává automatický přístup. Metody pracující na základě analýzy provozu jsou popsány v následujících odstavcích.

Kontrola hodnoty `User-Agent`

Aplikační firewall velmi často pracuje s hodnotami v HTTP hlavičce. Jednodušší boti neposkytují hodnotu `User-Agent` (viz 2.1) a aplikační firewall je může na základě absence

těchto informací zařadit do gray IP listu (viz 3.4) či rovnou zakázat provoz. Tedy uživatel, resp. bot, který neposkytuje tyto informace a aplikační firewall to odhalí, zpravidla řeší úlohu CAPTCHA 4.1 nebo jiný Turingův test[1] pro přístup do webové služby.

Analýza četnosti návštěv

Analýza četnosti návštěv je *střednědobá technika*[2], která slouží k rozpoznávání automatického přístupu na základě analýzy časů přihlašování uživatele. Pokud se *časové intervaly* mezi jednotlivými přihlášeními jednoho uživatele liší velmi málo či vůbec, lze z toho usoudit, že uživatel je automatický bot. Podobně lze analyzovat *denní časy* přihlašování uživatele – pokud se uživatel přihlašuje pouze ve stejné nebo velmi podobné denní časy, může jít o automatický přístup.

Aby tato ochrana odhalovala pouze automatické boty a neblokovala provoz normálním uživatelům, je nutné analyzovat časy uživatelských přihlašování *po určitou dobu* (čím déle, tím lépe, proto střednědobá technika), než bude uživateli např. odebrán účet. Může se totiž stát, že se i normální uživatel bude přihlašovat ve velmi podobné denní časy, ale jistě se někdy přihlásí dříve nebo později (normální uživatel se nedokáže přihlásit každý den v naprosto stejný čas).

Analýza doby strávené na webu

Doba strávená uživatelem na webu bývá odlišná od doby strávené na webu v minulých návštěvách. Jestliže aplikační firewall zjistí, že se doba strávená na stránce pohybuje stále ve stejných intervalech, může se jednat o podezřelou aktivitu a vést k zařazení IP adresy do gray IP listu. Kvalitní informace o průměrném čase stráveném na webové stránce lze získat dlouhodobým sledováním a jsou odlišné v závislosti na poskytované webové službě.

Analýza chování

Analýza chování je složitá disciplína, která vyplývá ze strojového učení [27]. Aplikační firewall zkoumá, zda je chování uživatele (nebo automatického bota) podezřelé nebo ne. Mezi podezřelou aktivitu se řadí např. prodleva mezi jednotlivými stisky klávesnice, velmi častá návštěva jednoho konkrétního místa na webu, počet kliknutí na určitý objekt na stránce nebo stejná doba strávená na webu (viz 4.3).

Honeypots/honeynets

Velké firmy, jako je Amazon či CloudFare, používají k zachycení botů sítě honeynets (překlad: *„medové sítě“*) po celém světě, které poskytují aktualizované informace o botech aplikačním firewallům ve svých sítích. Typická stránka, která slouží jako honeypot (překlad: *„hrnec medu“*), je stránka obsahující velké množství zdánlivě užitečných informací, čímž se stává lákavější pro automatické boty, kteří tuto stránku navštíví za účelem získání velkého množství kvalitních dat.

[1] Turingovy testy ověřují, zda má systém navštěvující webovou stránku lidskou inteligenci nebo ne.
[2] Střednědobá technika znamená, že její zavedení vyžaduje nějaký časový rámec, například 2 týdny.

4.4 Úprava HTML dokumentu

Většina automatických botů, kteří se snaží o získání dat z webové stránky, má přesně danou pozici a dokonale zná HTML (viz 2.6) strukturu navštívené stránky. Na základě této znalosti dokážou automatičtí boti získávat požadovaná data. Při změně této struktury (např. hlubší zanoření HTML elementů) může dojít k neschopnosti automatického robota získat požadovaná data.

Změna identifikátorů HTML tagů

Jedna z možností, jak se bránit automatickému přístupu, je *změnit identifikátory tagů v HTML dokumentu*. Tato ochrana vychází z předpokladu, že si bot stáhne webovou stránku a poté z ní extrahuje data, která se nacházejí na předem určených místech. Tato místa jsou označena unikátními jmény, díky nimž dokáže bot najít požadovaná data. V případě, jsou tato jména změněna na náhodnou sekvenci písmen či čísel[3], bot nedokáže najít požadovaná data a ztrácí zájem získávat data z tohoto serveru. Tento způsob ochrany zpravidla nemá na normálního uživatele zásadní dopad.

Další způsob, jak udělat HTML dokument hůře čitelný pro automatického bota, je změna pozic jednotlivých částí (změnit design webové stránky), ve kterých se mohou vyskytovat potenciálně užitečná data. Znamená to, že každá stránka pravidelně mění svůj vzhled. Tento přístup má však jisté nevýhody, zejména jde o uživatelské pohodlí. Každá změna, která se na vzhledu stránky projeví, může mít negativní dopad na uživatele a může vést k ztrátě jeho zájmu o publikovaný obsah.

Website cloaking

Website cloaking (přeloženo jako *„maskování webových stránek“*) je chování webových stránek, které se snaží doručovat jiný obsah webovým botům než běžným prohlížečům. Mezi cloaking patří např. poskytnutí pouze čisté HTML stránky webovým robotům, ve které není zahrnuta žádná grafika. Tato technika bývá často používána pro internetové vyhledávače kvůli snazšímu vyhledání klíčových slov. Website cloaking může ale být použit jako obrana proti automatickému přístupu například tak, že data na této stránce budou upravena či nebudou na stránce obsaženy vůbec.

Informace v obrázku

Veškerý text, který je prezentován na webové stránce, lze převést na obrázek a na stránce změnit text na odpovídající obrázky. Bot navštěvující stránku musí tedy pomocí OCR[4] rozpoznat uvedený text a převést ho na textovou podobu. Tento způsob ochrany je poměrně účinný proti jednodušším botům, avšak pokročilejší boti zejména s umělou inteligencí si s obrázkem dokážou relativně snadno poradit. Velká nevýhoda textu v obrázku je výrazně vyšší uživatelské nepohodlí, protože webový prohlížeč uživatele stahuje více dat (načítání stránky je pomalejší). Dalším problémem, který uživatel mnohdy pocítí, je nemožnost pracovat s funkcemi jako je kopírování textu nebo hledání textu na stránce. To opět může vést k odchodu a nespokojenosti potenciálního klienta.

[3] Provozovatel má systém, který mu udržuje původní smysluplná jména značek, přičemž automaticky generuje HTML dokument s náhodnými jmény značek a který pravidelně aktualizuje na webovém serveru.

[4] Optical Character Recognition, více o OCR lze najít na Wikipedii.

Z výše uvedeného je zřejmé, že neexistuje dokonalá ochrana a je poměrně náročné bránit se automatickým botům. Nejlepší ochrana je tedy zkoušet nové způsoby a kombinovat je s těmi osvědčenými s cílem, že uživatel nic nepozná zejména na rychlosti odezvy a designu uživatelského rozhraní.

Kapitola 5

Způsoby automatického přístupu

Jak název napovídá, jde o naprogramovaný automatický přístup na webové stránky, z něhož má prospěch ve většině případů provozovatel bota, avšak v některých případech může prosperovat i provozovatel webového serveru. Bot je tedy program, který dle daných podmínek pracuje s obsahem na webu ve snaze simulovat normálního uživatele. Jinými slovy, bota si lze představit jako člověka, který pracuje s webovou stránkou (pracuje s daty na dané stránce) podle určitých pravidel nebo naučených postupů. Pokud webový server chce umožnit přístup botům, tak poskytuje *speciální rozhraní API*, přes něž mohou tyto automaty komunikovat. V opačném případě je nutné stránky navštěvovat jiným způsobem – *webscrapingem*.

Automatický přístup na web (tedy i sociální sítě) lze tedy realizovat více způsoby. Některé metody jsou provozovány se souhlasem provozovatele a jejich použití je v souladu se zájmu dané stránky. Tento způsob se nazývá *API* a věnuje se mu kapitola 5.1, nicméně cílem této práce není popsat tuto metodu. Na druhé straně proti API existuje metoda, jak získávat data z webových stránek bez použití API, nazývaná *webscraping*. Websraping nabízí oproti API větší škálu využití a proto bývá využíván pro účely, které mohou být nežádoucí pro danou sociální síť (provoz přes API lze kontrolovat a popř. zablokovat). Velká nevýhoda API je totiž omezené spektrum možností, které se s API dají provádět – provozovatel sociální sítě nemá pochopitelně zájem o to, aby se na jeho síti šířil SPAM či jiné podobné věci. Tato práce se zabývá zejména přístupem na webovou stránku pomocí webscrapingu.

5.1 API

Application Programming Interface (API) je obecný název pro rozhraní programovatelných aplikací. Pomocí API se dokážeme připojit k lokálnímu či vzdálenému programu nebo službě a využívat jeho funkcí, získávat nebo ukládat data či s ním jinak komunikovat podle předem daných pravidel. Na obrázku 5.1 je schéma komunikace API: klient se táže API serveru Bittrex na kryptoměnové trhy, které je možné obchodovat, a server mu je ve formátu JSON posílá zpět. API se dělí podle jeho použití na několik druhů. Tím je například API operačních systémů pro komunikaci operačního systému a služby běžící na dané stanici, JavaScript API atd. [20].

Obrázek 5.1: Získávání dat pomocí API

Third-party API

Velké firmy, jako jsou Facebook, Google či PayPal, poskytují ostatním vývojářům jiných webových stránek svoje rozhraní API. To slouží k využívání dat, která firma poskytující API vlastní. Uživatel využívá Thirt-party API například při vytváření účtu na webové stránce, kdy tato webová stránka požádá o data o uživateli Facebook, a ten následně poskytne data (např. jméno a profilovou fotku) o uživateli (pochopitelně se souhlasem daného uživatele) [21]. To výrazně zjednoduší průběh vytváření nového účtu, protože uživatel vytvoří účet doslova jedním kliknutím. Další přihlašování probíhají také jednodušeji, protože se uživatel může přihlásit například pomocí účtu na Facebooku a nemusí si pamatovat žádné další heslo.

5.2 Webscraping

Jde o aktivitu, která se snaží dostat data z webových stránek[1] pomocí *automatického programu*. Cílem webscrapingu je nahradit co nejdůvěryhodněji webový prohlížeč, simulovat uživatelskou interakci a překonat ochranu proti automatickému přístupu (pokud nějaká existuje). Data, která jsou tímto způsobem získána, může sám webscrapingový program nebo jeho majitel dále upravovat či s nimi nějak dále pracovat. Tento způsob získávání dat probíhá relativně často bez vědomí vlastníka webové stránky, ze které jsou data dolována. Webové servery se snaží tomuto přístupu bránit, k čemuž mají mnoho důvodů, nicméně je nutno dodat, že ochrany proti automatickému přístupu bývají neefektivní vůči vynaloženým nákladům. Pokud ochrana webového serveru odhalí automatický přístup, má několik možností, jak se zachovat – tyto možnosti jsou například:

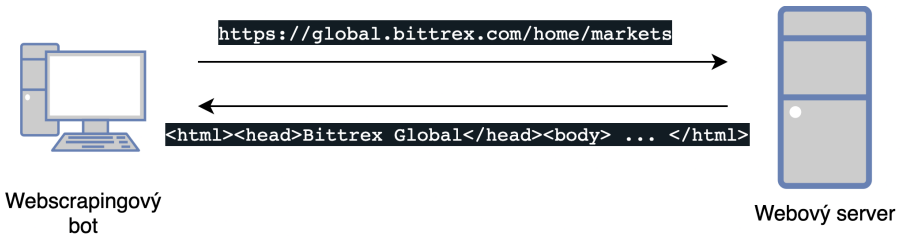
- **Nezasahovat** – nechat automatického bota pokračovat ve své činnosti.
- **Obfuskovat**[2] **obsah** – poskytnout automatickému botovi stránku s upravenou strukturou *zdrojového kódu* a tím mu znesnadnit čtení obsahu.
- **Zakázat provoz** – zablokovat provoz z IP adresy či webového prohlížeče nebo zablokovat uživatelský účet (pokud existuje).

Jedním z důvodů, proč se bránit webscrapingu, je vytíženost serveru. Automatický server totiž může provádět takové operace, že se webová stránka stane pro ostatní uživatele obtížněji použitelná (např. pomalejší odezva). Dalším důvodem, proč se bránit webscrapingu,

[1] Ačkoli se tato práce zabývá automatickým přístupem na sociální sítě, je v této kapitole diskutováno obecně o automatickém přístupu na webové stránky (sociální sítě jsou podmnožina webových stránek).

[2] *Obfuskace* je proces, při kterém se upraví zdrojový kód tak, aby se stal (téměř) nečitelným, ale jeho funkčnost zůstane zachována. Používá se například jako ochrana vlastnictví autora [23].

je ochrana dat, která jsou na stránce obsažena. Některá data jsou sice veřejně dostupná, ale je nežádoucí, aby byla získávána automaticky. Struktura získávání dat webscrapingem je znázorněna na obrázku 5.2.



Obrázek 5.2: Průběh získání dat pomocí webscrapingu

Využití webscrapingu

Když existují rozhraní API, přes která lze bez problémů a se souhlasem druhé strany komunikovat, proč tedy existuje metoda webscraping? Po přehlédnutí poškození druhé strany a jakýchkoli jiných druhů hackerského útoku, má velké využití v situacích, kdy je zkrátka člověk samotný moc pomalý. Jde například o situaci, kdy se očekává vydání lístků na koncert, o které bude pravděpodobně obrovský zájem. Uživatel, který o tyto lístky velmi stojí, využije webscrapingu (ačkoli je to poněkud nespravedlivé k ostatním) a bot mu tyto lístky nakoupí přesně podle jeho požadavků.

Webscraping na sociálních sítích

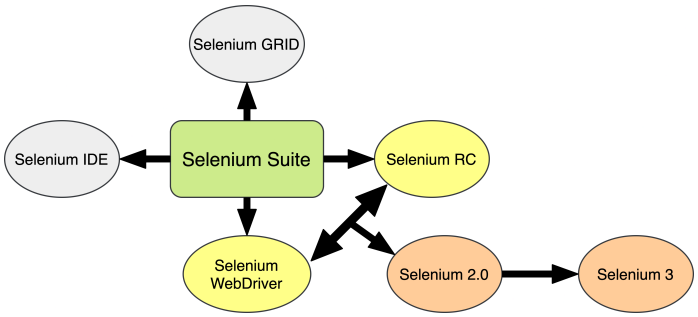
Každá sociální síť si chce vytvořit či udržet dobrou pověst a jejím hlavním cílem je, aby byl uživateli předkládán zajímavý obsah a při trávení času na sociální síti se uživatel cítil pohodlně. V případě sociálních sítí zpravidla není zásadní problém vytíženost serveru, kterou by automatický přístup mohl negativně ovlivnit. Hlavní důvod, proč se tyto sítě brání automatickému přístupu je, že by byl uživatelům servírován nezajímavý nebo podvodný obsah. Na to by uživatelé mohli reagovat odchodem ze stránky, což by vedlo k menšímu počtu zobrazení reklam. To by citelně snížilo zisk sociální sítě z reklamy a menší důvěru uživatelů v tuto síť. Dalším důležitým důvodem je ochrana citlivých uživatelských dat nebo nastavení, jejichž soukromí může být narušeno. Z tohoto důvodu se sociální sítě usilovně snaží zabránit automatickému přístupu na jejich stránky. Ochrana proti automatickému přístupu sociálních sítí by měla být z výše uvedených důvodů pokročilá a díky tomu je vhodné studovat aplikační firewally právě na sociálních sítích.

5.3 Selenium

Selenium je sada nástrojů publikovaných ve formě otevřeného kódu (*opensource*) sloužících pro automatizaci webových prohlížečů a testování webových aplikací. Dokáže věrně simulovat webový prohlížeč Google Chrome (Selenium WebDriver, více 5.3) a kvůli tomu je často využíván právě pro účely webscrapingu. Selenium používá velké množství firem, mezi něž patří např. MIT, Google, Fitbit a další [34]. Sada nástrojů Selenium je napsána v jazyce Java, čímž umožňuje použití na mnoha platformách. Tato sada nástrojů je velmi flexibilní z hlediska použitého programovacího jazyka, protože podporuje jazyky jako Perl, Python, Ruby, Java, C#, PHP a další [34].

Selenium je kompatibilní se všemi nejpoužívanějšími prohlížeči, čímž se stává atraktivnější pro testování webových aplikací. Je totiž důležité, aby tyto aplikace běžely správně na všech používaných webových prohlížečích. Sada nástrojů je open-source, je tedy zdarma pro všechny uživatele.

Selenium Suite se dělí na 4 projekty: *Selenium IDE*, *Selenium Remote Control*, *Selenium WebDriver* a *Selenium Grid* [13]. Schéma rozdělení Selenia je uvedena na obrázku 5.3. Každý projekt z této sady je vhodný pro něco jiného a nabízí jiné možnosti využití pro testování webových prohlížečů nebo testování webových aplikací.

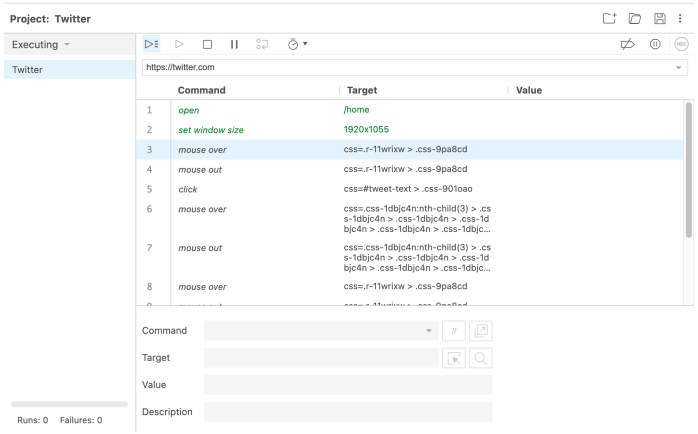


Obrázek 5.3: Architektura Selenium včetně tvorby Selenium 2.0 spojením Selenium RC a Selenium WebDriver

Selenium IDE

Selenium IDE (*Selenium Integrated Development Environment*) je doplněk do webového prohlížeče (dříve pouze Firefox, dnes už i Google Chrome [2]). Pomocí něj můžeme vytvářet automatické testy, které fungují na základě nahrávání interakcí uživatele s webovou stránkou. Tento záznam lze pomocí Selenium IDE zpětně přehrát. Instalace Selenium IDE je velmi jednoduchá: v prohlížeči Google Chrome (nebo v prohlížečích pracujících s jádrem Google Chrome) lze aplikaci nainstalovat přes Obchod Chrome; v prohlížeči Firefox přes Firefox Add-ons.

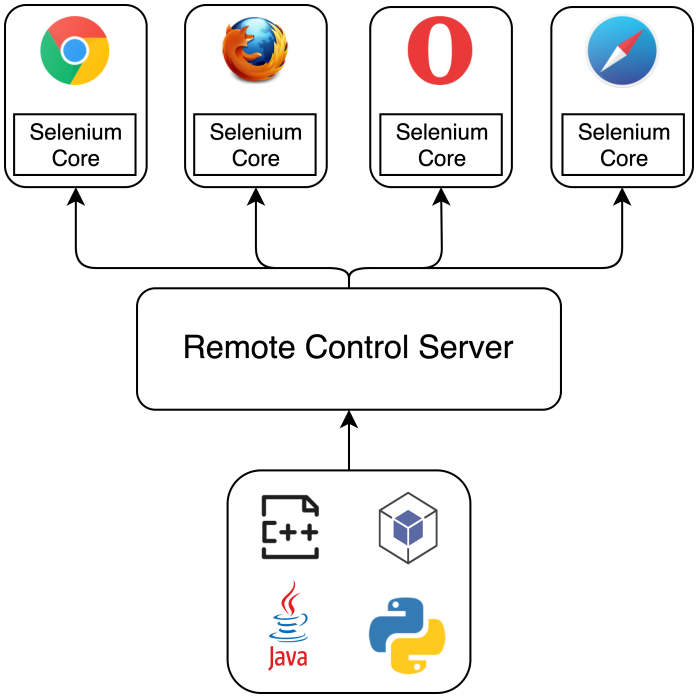
Na obrázku 5.4 je ukázka aplikace Selenium IDE, která otevře hlavní stránku sociální sítě Twitter.



Obrázek 5.4: Ukázka průběhu simulace přístupu na webovou stránku programem Selenium IDE

Selenium Remote Control

Selenium RC je testovací nástroj klient-server. Umožňuje interpretovat automatické testy webových aplikací v libovolném programovacím jazyce podporovaném Selenium Suite na jakékoli webové stránce nad protokolem HTTP(s) využívající JavaScript. Zahrnuje také HTTP Proxy server[3], který dává důvěru prohlížeči, že testovaná webová aplikace pochází z domény poskytované proxy serverem. Nevýhodou Selenia RC je, že musí existovat server, který přijímá HTTP(S) požadavky od klienta (tento server může běžet na stejném i vzdáleném fyzickém zařízení) [16, 18]. Architektura Selenium RC je zobrazena na obrázku 5.5.



Obrázek 5.5: Selenium RC architektura

Selenium WebDriver

Selenium WebDriver je nástupce nástroje Selenium RC a zároveň nejdůležitějším nástrojem sady Selenium Suite. Oproti Seleniu RC nabízí jednodušší a stručnější programovací rozhraní, která řeší nedostatky a omezení předchozí verze. Poskytuje programovací rozhraní pro vytváření a provádění testovacích skriptů. Ty jsou psány za účelem identifikace jednotlivých prvků ve webové aplikaci a dále jsou testovacími skripty vyhodnocovány akce, které testy provádí s těmito prvky [18]. Cílem je větší podpora dynamických webových stránek, v nichž se jednotlivé prvky často mění [16].

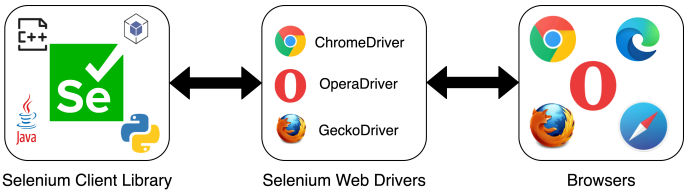
WebDriver funguje mnohem rychleji ve srovnání s RC, protože WebDriver nepotřebuje spuštěný samostatný server, který provádí testy. Protože WebDriver přímo volá metody různých prohlížečů, je nutné, aby existoval pro každý prohlížeč samostatný ovladač (Mozilla Firefox Driver, Google Chrome Driver, Opera Driver apod.). V aktuálně používané verzi

[3]Proxy server je neviditelný prostředník mezi klientem a serverem a může například filtrovat nebo optimalizovat provoz.

Selenium 3 jsou nástroje WebDriver a RC spojeny do jednoho nástroje WebDriver, do kterého je přidána další funkcionalita [16].

Na obrázku 5.6 je znázorněna architektura nástroje Selenium WebDriver. Skládá se ze tří komponent (zdroj [10]):

- *Selenium WebDriver* obsahující klientské knihovny pro vytváření a provádění testovacích skriptů.
- *Ovladače Selenium WebDrivers* specifické pro prohlížeč, které předávají požadavky z WebDriveru.
- *Prohlížeč Browsers*, který přijímá požadavky od ovladačů a následně je zpracovává.



Obrázek 5.6: Selenium WebDriver architektura

Selenium Grid

Nástroj Selenium Grid je nástroj, který umožňuje provádění testů v distribuovaném prostředí, tedy na různých strojích s různými prohlížeči současně. Skládá se z jednoho zařízení, která má roli *Hub* – testovací server, a libovolného počtu zařízení, která mají roli *Node* – testovací uzel [18]. Výsledkem testování je tedy zkouška webové aplikace na různých počítačích s různými prohlížeči a odlišnými operačními systémy, čímž se výrazně urychluje doba testování webové aplikace.

Selenium Grid není příliš odlišný od ostatních nástrojů v sadě Selenium. Je to velmi podobný nástroj jako Selenium WebDrive rozšířený o možnost testovat aplikace paralelně v odlišných prostředích.

Kapitola 6

Testování aplikačních firewallů sociálních sítí

Sociální sítě mají pravděpodobně implementovány jedny z nejpokročilejších ochran proti automatickým botům, a proto je vhodné jejich studium. Sociální sítě totiž nemají z jistých důvodů uvedených v předchozích kapitolách zájem o přístup automatických botů, nicméně na druhou stranu jsou právě sociální sítě jejich častým cílem a sociální sítě se jim musí bránit. Znalost principu těchto aplikačních firewallů může být užitečnou inspirací pro návrh nebo vylepšení ochran vlastních webových stránek či jiných veřejných služeb, které mohou být obětí automatických botů.

Kapitola začíná ukázkou chování skutečných uživatelů na sociálních sítích a dále popisuje výsledky testování aktuálně používaných ochran proti automatickému přístupu. Poté popisuje možnosti přístupu na sociální sítě přes aplikační rozhraní API a jeho srovnání s webscrapingem. Testované sociální sítě jsou:

- Facebook (včetně verze mBasic)
- Twitter
- LinkedIn
- YouTube

6.1 Skutečné chování uživatelů na sociálních sítích

Každý uživatel navštěvující určitou sociální síť má obvykle ve zvyku provádět jednotlivé akce postupně ve stejném pořadí. V případech, kdy uživatel navštíví sociální síť kvůli nějakému účelu, např. poslání zprávy či vyhledání informace[1], většinou svou posloupnost prvních akcí obejde. Testy popsané v tabulkách č. 6.1, 6.2 a 6.3 popisují sekvence typického chování na sociálních sítích 3 osob. Tyto testy byly prováděny kvůli implementaci opatření navrhovaném v této práci spočívající v totožném chování po každém přihlášení. Testování chování osoby je prováděno pouze na platformách a sociálních sítích, které daná osoba pravidelně navštěvuje. Většina časů v tabulce je uvedena v *sekundách* a časy jsou *přibližné* (někdy může daná akce trvat kratší dobu, někdy delší dobu). Časy znamenají dobu uplynulou od spuštění sociální sítě, nikoli od poslední akce.

[1] Mnoho uživatelů totiž navštěvuje sociální síť bezúčelně např. kvůli závislosti.

Osoba 1					
Platforma	**1. akce**	**∼ čas**	**2. akce**	**∼ čas**	**Poznámka**
Facebook mobil	Prohlédne si první 3 příspěvky	15 s	Přečte si upozornění (pokud nějaká jsou)	25 s	
Facebook desktop	Zkontroluje upozornění a přečte si ho	8 s	Kontrola zpráv a odepsání svým přátelům	35 s	Facebook na desktopu téměř nenavštěvuje
YouTube mobil	Zkontroluje upozornění	7 s	Podívá se na nějaké video podle upozornění	75 s	
YouTube desktop	Začne vyhledávat video s požadovaným obsahem	12 s	Prohlíží nalezená videa	20 s	Na desktopu chodí vyhledávat videa pouze pokud potřebuje najít určitý obsah
Instagram mobil	Prohlédne si přibližně 10 prvních stories	50 s	Přečte si zprávy, popř. na ně odepíše	70 s	

Tabulka 6.1: Typické chování na sociální síti první testované osoby

Osoba 2					
Platforma	**1. akce**	**∼ čas**	**2. akce**	**∼ čas**	**Poznámka**
Facebook mobil	Najde příspěvek, který zaujme	18 s	Zareaguje na tento příspěvek a přečte si komentáře	30 s	Reklamy přeskakuje
YouTube mobil	Podívá se na doporučená videa	7 s	Video, které zaujme, si uloží do seznamu "Později"	15 s	Videa neoznačuje *To se mi líbí* ani nezanechává komentář

Tabulka 6.2: Typické chování na sociální síti druhé testované osoby

Osoba 3					
Platforma	**1. akce**	**∼ čas**	**2. akce**	**∼ čas**	**Poznámka**
Facebook mobil	Najde příspěvek, který zaujme	15 s	Zareaguje na tento příspěvek a přečte si komentáře	30 s	Reklamy přeskakuje
Facebook desktop	Přečte si upozornění	10 s	Prohlédne si prvních cca 7 příspěvků	25 s	Zareaguje (nikoli okomentuje) příspěvek, který ho zaujme
YouTube mobil	Zkontroluje upozornění	7 s	Podívá se na nějaké video podle upozornění	120 s	
YouTube desktop	Doporučená videa (2-3), která zaujmou, otevře v nové záložce	13 s	Postupně shlédne tato videa	600 s	U doporučených videí velmi často reaguje *To se mi líbí*
Twitter desktop	Prohlíží zeď, dokud nenarazí na sekci "Koho sledovat"	18 s	Začne sledovat uživatele, jejichž obsah je přínosný	17 s	Nečte upozornění

Tabulka 6.3: Typické chování na sociální síti třetí testované osoby

6.2 Popis testování reálných sociálních sítí

Každá vybraná sociální síť byla otestována 5 testy. První dva jsou *automatické*, třetí je *poloautomatický* a zbývající 2 jsou prováděny *manuálně*. Automatické testy byly provedeny nástrojem Selenium WebDriver (viz 5.3) s využitím programovacího jazyka Python. Ostatní testy byly provedeny pomocí nástrojů uvedených v seznamu níže. Každá sociální síť byla otestována v tomto pořadí:

- *Automatický test stejného chování* (automaticky pomocí Selenium WebDriver)
- *Automatický test hromadného přístupu* (automaticky pomocí Selenium WebDriver)
- *Test ověřování* `User-Agent` *hodnoty* (změna hodnoty `User-Agent` v hlavičce pomocí Selenium WebDriver)
- *Přístup z prohlížeče Tor* (manuálně pomocí prohlížeče Tor)
- *Test kódování identifikátorů HTML elementů* (manuálně pomocí prohlížeče Google Chrome)

Výsledky testů jsou uvedeny v jednotlivých sekcích zvlášť pro každou testovanou sociální síť. V tabulkách je navíc uvedeno, zda sociální síť testuje lidskou inteligenci pomocí CAPTCHA (viz 4.1), protože se jedná o jednoduchý a efektivní způsob, jak zabránit vstup automatickým botům.

Automatický test stejného chování

První automatický test zjišťuje, zda aplikační firewally sociálních sítí kontrolují podezřele velký počet přihlášení se do jednoho uživatelského účtu za krátký časový rámec. Dále odhaluje, zda aplikační firewally zkoumají uživatelské chování či nikoli. Jinými slovy test ověřuje, zda sociální sítě odhalují více přihlášení za sebou, po kterém se uživatel chová identicky. Aktivita, která proběhne vícekrát za sebou a dělá naprosto samou věc, se totiž jeví jako podezřelá.

Selenium WebDriver po spuštění otevřel nové okno prohlížeče Google Chrome a načetl přihlašovací stránku dané sociální sítě. Vyplnil uživatelské jméno a heslo a přihlásil se do sociální sítě. Na hlavní stránce se posunul o 1000 pixelů dolů, chvíli setrval a vrátil se nahoru – tento pohyb na stránce se prováděl pouze kvůli větší důvěryhodnosti přístupu. Test kliknul na tlačítko *To se mi líbí* u prvního příspěvku a 5 sekund poté zavřel prohlížeč. Po zavření okna prohlížeče vyčkal test 30 sekund a celý test opakoval znovu. Celý popsaný cyklus proběhl celkem 50×.

Automatický test hromadného přístupu

Druhý test zkoušel (opět s použitím Selenium WebDriver), stejně jako první test, přihlášení a kliknutí u prvního příspěvku na *To se mi líbí*, nicméně tyto testy nebyly spouštěny postupně, ale v jeden čas. Byl tedy spuštěn skript, který spustil 10[2] testů najednou, tedy podobný průběh jako při DoS útoku (viz 3.5). Účel analýzy byl ověřit, zda sociální sítě dokáží tuto aktivitu odhalit, popř. jak se jí brání.

[2]Více testů najednou nebylo možné spustit z důvodu výkonu testovací stanice.

Test ověřování `User-Agent` hodnoty

Tento test ověřoval, zda sociální sítě kontrolují správnost hodnoty `User-Agent`. Využívá skriptu z prvního testu, ale je změněna HTML hlavička a prohlížeč je spuštěn pouze jednou, nikoli 50×. Cíl testu byl stejný jako v předchozích dvou testech, tedy kliknout na *To se mi líbí*, ale žádost měla upravené hodnotu v HTTP hlavičcce `User-Agent` na hodnotu `False_user_agent`. Příklad korektní hodnoty je popsán v části 2.1. Ačkoli se jedná o poměrně jednoduchý test opatření, které lze jednoduše obejít, tak dokáže odfiltrovat alespoň triviální pokusy o automatický přístup.

Přístup z prohlížeče Tor

Anonymizační systém Tor a prohlížeč Torbrowser je popsán v sekci 2.3. Jde tedy o test přístupu z různých IP adres, které jsou v mnoha případech zařazeny do *Black listů* nebo *Gray listů* a aplikační firewall tím dokáže jednoduše odhalit podezřelou aktivitu. Každou IP adresu lze alespoň přibližně lokalizovat, tedy pokud se jeden uživatel přihlašuje na sociální síť z různých oblastí během krátkého času, aplikační firewall přístup zpravidla vždy zablokuje. Princip analýzy přístupu pomocí prohlížeče Tor spočívá jen v přihlášení se na sociální síť.

Test kódování identifikátorů HTML elementů

Každá načítaná stránka ve webovém prohlížeči se načítá pomocí HTML (viz 2.6). Webscrapingové automaty často pracují právě s HTML strukturou a jednotlivé elementy poznávají podle jedinečných HTML identifikátorů k získání dat, které tento HTML element obsahuje. V testu se ověřuje, zda webová stránka používá *intuitivní* názvy HTML identifikátorů (např. `heading`) nebo zda používá nějaké *kódování* (např. `0h4jek3iS`). Získání informací o tom, zda sociální síť používá nějaké kódování HTML identifikátorů, bylo provedeno ručně pomocí nástroje Google Developers (více zde 2.2).

6.3 Facebook

Nejpopulárnější sociální síť Facebook[3] je platforma, která byla původně plánovaná jako studentská školní síť. Postupem času se stávala známější a v dnešní době ji zná takřka každý. Umožňuje sdílet pocity uživatele či fotky, dopisovat si s přáteli a další podobné sociální aktivity. Facebook je velmi silný nástroj pro šíření různých aktivit či názorů, kvůli čemuž je využíván či zneužíván mnoha populisty a vlivnými lidmi.

Při analýze klasického Facebooku nebyly nalezeny žádné ochrany proti automatickému přístupu. Při analýze facebookové verze mBasic[4] se přihlašovací stránka v pořádku načetla, nicméně při procesu přihlášení (po stisknutí klávesy Enter) Facebook vrátí neznámou chybu. Byl vyzkoušen ruční přístup na mBasic po načtení vyhledávače Google WebDriver (Selenium skript spustil pouze prohlížeč), nicméně po vyplnění jména a hesla mBasic vrátil taktéž neznámou chybu. Pomohlo znovu načíst přihlašovací stránku a po vyplnění přihlašovacích údajů a stisknutí Enter bylo přihlášení úspěšně vykonáno. Jinými slovy bylo nutné přihlašovací stránku pro Facebook mBasic načíst dvakrát, aby přihlášení mohlo proběhnout úspěšně. Facebook používá jako identifikátory řetězce, které zdánlivě nedávají žádný

[3] https://www.facebook.com
[4] https://mbasic.facebook.com

smysl, ale je na uvážení čtenáře, zda je to kvůli ochraně proti webscrapingu či je toto kódování využíváno pro účely frontendu (designu stránky). Výsledky analýzy jsou uvedeny tabulce 6.4.

Automatický test stejného chování	Ochrana NEDETEKOVÁNA
Automatický test hromadného přístupu	Ochrana NEDETEKOVÁNA
Test hodnoty `User-Agent`	Ochrana NEDETEKOVÁNA
Přístup z Tor browser	Při druhém a všech dalších přihlášeních vyžaduje ověření identity pomocí emailu a změnu hesla
Kódování HTML identifikátorů	Neintuitivní identifikátory, které se mění přibližně jednou za 2–3 měsíce (verze mBasic využívá pro jména identifikátorů dvouznakové řetězce, které se mění s každým přihlášením)
CAPTCHA/UI	V části Facebook Developers vyžaduje splnění testu CAPTCHA

Tabulka 6.4: Výsledky analýzy aplikačního firewallu Facebooku

Facebook API

API pro Facebook se nazývá *Graph API*[5]. Slouží k nahrávání fotek, přidávání příspěvků, komentování příspěvků, vytvoření příběhu apod. Každý objekt (např. fotka) má jednoznačné ID, pomocí kterého lze objekt identifikovat. Aby bylo možné využívat služeb Graph API, je nutné mít aktivovaný Access Token pro přístup (zpravidla má platnost 3 měsíce). Správa tokenů je v části Facebook Developers. V tabulce 6.5 jsou přehledně uvedeny vlastnosti obou způsobů automatického přístupu.

Webscraping	**Facebook Graph API**
Jednodušší	Umožňuje spravovat reklamy
Nevyžaduje token	Nepokládá robota za osobu – nebude mu vnucovat reklamy
Není prováděna aktualizace tokenu	Publikování příspěvku do více skupin naráz
Nevědomost Facebooku o vyžadovaném obsahu	Jednodušší kontrola skupin (1000 skupin je složité spravovat jinak než přes API)
	Eliminace rizika zablokování účtu při rozpoznání aut. robota
	Automatická upozornění na určitou aktivitu
	Zdarma

Tabulka 6.5: Rozdíly mezi webscrapingem Facebooku a Facebookem Graph API

[5] https://developers.facebook.com/docs/graph-api/overview/

6.4 Twitter

Twitter[6] je sociální síť, kde uživatelé sdílí svoje příspěvky a čtou příspěvky jiných uživatelů. Obsah na uživatelské zdi je tvořen lidmi, které daný uživatel sleduje a předměty zájmu, tzv. *hashtagy*[7]. Twitter je velmi vlivná sociální síť, díky čemuž mohou známí lidé ovlivnit např. vývoj finančních burz či politické názory.

Jako jediná z analyzovaných sociálních sítí má *placené API* a z tohoto důvodu se snaží co nejvíce zabránit jinému nežádoucímu automatickému přístupu. Zajímavý je postup ověřování při použití prohlížeče TorBrowser – složitost ověření se při přihlašování během krátké doby stupňuje. Twitter navíc rozlišuje, zda se uživatel přihlašuje pomocí emailové adresy nebo uživatelského jména. Twitter podobně jako Facebook používá kódované názvy HTML identifikátorů (pravděpodobně ze stejných důvodů). Při každém přihlášení z prohlížeče Torbrowser je odesláno bezpečnostní upozornění na uvedenou emailovou adresu. Výsledky analýzy jsou uvedeny v tabulce 6.6.

Automatický test stejného chování	Ochrana NEDETEKOVÁNA
Automatický test hromadného přístupu	Provoz zablokuje, pokud je k přihlášení použita emailová adresa. V případě zadání uživatelského jména přihlášení proběhne v pořádku.
Test hodnoty `User-Agent`	Ochrana DETEKOVÁNA – bez validní hodnoty `User-Agent` zablokuje provoz
Přístup z Tor browser	Složitost ověření se po každém přihlášení zvětšuje (při přihlašování bylo vždy použito správné uživatelské jméno a heslo): 1. přihlášení: Bez ověření. 2. přihlášení: Ověření telefonního čísla (SMS) a změna hesla. 3. a další přihlášení: Ověření emailové adresy, CAPTCHA (5 – 7 opakování), zadání hesla a ověření telefonního čísla.
Kódování HTML identifikátorů	Neintuitivní identifikátory jako v případě Facebooku
CAPTCHA/UI	Po prvním přihlášení pomocí Selenia ChromeDriver, další přihlášení proběhla v pořádku.

Tabulka 6.6: Výsledky analýzy aplikačního firewallu Twitteru

[6] https://www.twitter.com

[7] Hashtag je slovo či fráze popisující libovolný zájem, která se vyznačuje znakem # na začátku. Typickým příkladem je například *#bitcoin*.

Twitter API

Aplikační programovatelné rozhraní Twitteru je na rozdíl od ostatních API sociálních sítí placené[8]. Existují různé plány – ve verzi zdarma může být za 1 měsíc maximálně 250 žádostí (málo pro normální používání). Pro využívání služeb API je nutné mít schválený účet pro vývojáře. Twitter API je rozděleno na několik částí a každá z nich má odlišný účel:

- *Twitter API* (vyhledávání Tweetů, komunikace s ostatními uživateli, vyhledávání trendů apod.)
- *Twitter Ads API* (vytváření a správa reklam na Twitteru)
- *Twitter for websites* (umožňuje vkládat obsah z určitého místa na Twitteru na webovou stránku)
- *Twitter Developer* (vyvíjení nového prostředí vývojářskou komunitou Twitteru, testování nových produktů a funkcí API)

Nejvíce bývá využíváno služeb klasického *Twitteru API*, který taktéž bývá nejvíce obcházen webscrapingem. V tabulce 6.7 jsou znázorněny vlastnosti obou způsobů automatického přístupu.

Webscraping	Twitter API
Neomezený počet žádostí	Omezený počet žádostí
Zdarma	Placený
Dokáže získat všechny dostupné informace	Nezávislá na časté změně HTML kódu Twitteru
Riziko zablokování účtu z důvodu rozpoznání robota	Neposkytuje počet sledujících a sledovaných uživatele [8]
Rychlejší a flexibilnější [8]	Automatická upozornění na určitou aktivitu

Tabulka 6.7: Rozdíly mezi webscrapingem Twitteru a Twitter API

6.5 LinkedIn

LinkedIn[9] je největší profesní sociální síť, která slouží svým uživatelům k vyhledání svého zaměstnavatele či partnera nebo naopak k hledání nových zaměstnanců do pracovního poměru. Uživatele na LinkedInu sdílí svoje zkušenosti s dosavadní praxí, vzděláním, zájmy či mimoškolních aktivit. Lze říci, že LinkedIn je svým zaměřením podobný Facebooku, nicméně LinkedIn slouží ke sdílení svých profesních zkušeností a nikoli pro sdílení osobních záležitostí.

Analýza ukazuje, že LinkedIn nemá příliš pokročilé ochrany na úrovni aplikačního firewallu. Jedním z důvodů, proč se LinkedIn příliš nebrání automatickému přístupu může být ten, že neobsahuje tolik „užitečných“ informací, takže přestává být užitečným cílem pro automatické roboty. Aplikační firewall LinkedInu však pozná, zda se uživatel přihlašuje přes Torbrowser a v tomto případě vyžaduje ověření pomocí emailové schránky. Výsledky analýzy jsou uvedeny v tabulce 6.8.

[8] https://developer.twitter.com/en/pricing/search-30day
[9] https://www.linkedin.com

Automatický test stejného chování	Ochrana NEDETEKOVÁNA
Automatický test hromadného přístupu	Ochrana NEDETEKOVÁNA
Test hodnoty `User-Agent`	Ochrana NEDETEKOVÁNA
Přístup z Tor browser	Při každém (prvním a dalším) přihlašování posílá verifikační email s kódem, který je nutné zadat do prohlížeče
Kódování HTML identifikátorů	Ochrana NEDETEKOVÁNA
CAPTCHA/UI	Při vytváření účtu je nutné vyřešit jednoduchý úkol

Tabulka 6.8: Výsledky analýzy aplikačního firewallu LinkedInu

LinkedIn API

LinkedIn API[10] je založeno na REST[11] API a využívá protokolu OAuth 2.0. Veškeré zabezpečení týkající se autentizace a autorizace LinkedIn API je zajišťováno OAuth 2.0.

Webscraping	Twitter API
Jednodušší	Jednodušší sdílení obsahu (pro firmy a pro jednotlivé členy)
	Využití externích aplikací pro práci se svým obsahem na LinkedIn
	Je nutné mít vygenerovaný token[12]
	OAuth 2.0 autorizace[13]

Tabulka 6.9: Rozdíly mezi webscrapingem LinkedInu a LinkedIn API

6.6 YouTube

Nejpopulárnější síť na sdílení videa YouTube je sociální síť, kde jednotliví uživatelé mohou sledovat videa jiných uživatelů, reagovat na ně a sdílet svoje videa. Odhaduje se, že každou minutu je nahráno na YouTube asi 300 hodin videa[14]. Video server YouTube je součástí společnosti Google a pro přihlášení do YouTube se využívá právě účtu Google. Příjmy YouTube spočívají v příjmech z reklamy, jejíž cílování je zajištěno daty získanými společností Google.

Zájem bojovat proti webscrapingu je nejen kvůli omezování ostatních uživatelů v kvalitě videa (YouTube není schopen zajistit video ve vysoké kvalitě pro příliš mnoho uživatelů), ale kvůli šíření spamu, tedy psaní nevhodných komentářů či ovlivňování algoritmů vybírajících doporučená videa. Automatický robot totiž může uměle zvednout popularitu videa, čímž dosáhne toho, že uživateli budou nabízena videa, o které nemá zájem. Výsledky analýzy jsou uvedeny v tabulce 6.10.

[10] https://www.linkedin.com/developers

[11] REST (REpresentational State Transfer) je podmnožina protokolu HTTP splňující předem daný okruh pravidel.

[14] https://merchdope.com/youtube-stats/

Automatický test stejného chování	Ochrana NEDETEKOVÁNA
Automatický test hromadného přístupu	Ochrana NEDETEKOVÁNA
Test hodnoty `User-Agent`	Ochrana NEDETEKOVÁNA
Přístup z Tor browser	Při každém (prvním i dalším) přihlašování požaduje ověření CAPTCHA a ověření telefonního čísla
Kódování HTML identifikátorů	Ochrana NEDETEKOVÁNA
CAPTCHA/UI	Ochrana NEDETEKOVÁNA

Tabulka 6.10: Výsledky analýzy aplikačního firewallu sítě YouTube

YouTube API

S YouTube API lze nahrávat videa, reagovat na komentáře, spravovat svůj účet upravovat svoje playlisty apod. ze svého libovolné aplikace. Pro přístup je nutné mít účet Google a autorizovanou aplikaci, která bude služeb API využívat. V tabulce 6.11 jsou přehledně vyznačeny vlastnosti YouTube API a webscrapingu.

Webscraping	Twitter API
Jednodušší	Nutné mít Google účet
Není nutné mít Google účet	Nabízí téměř všechny dostupné akce
YouTube si neukládá data o vyhledávání na server (pokud není použit Google účet), pouze do cookies	YouTube má přehled o aktivitě API
	Zdarma
	Téměř neomezený počet žádostí

Tabulka 6.11: Rozdíly mezi webscrapingem LinkedInu a LinkedIn API

Kapitola 7

Návrh řešení a implementace

Kapitola na začátku popisuje použitý programovací jazyk a technologie, které byly použity pro implementační část této práce. Hlavní část kapitoly se věnuje *návrhu funkcí pro detekci botů aplikačním firewallem*. Detekce botů spočívá v posloupnosti jednotlivých testů uživatele prováděných aplikačním firewallem. Řešení vychází z analýzy reálných sociálních sítí, protože jsou navrhovány ochrany, které na těchto sociálních sítích nejsou implementovány vůbec, či pouze okrajově. Pokud je nějaký test vyhodnocen pozitivně, je *uživatel nebo jeho webový prohlížeč* zablokován na náhodnou dobu[1]. Jednotlivé testy jsou podrobněji popsány v sekci 7.3 a jsou prováděny v tomto pořadí:

- *Počet neúspěšných přihlášení* – zda nedochází v posledních 30 sekundách k více než 3 pokusům o přihlášení se se špatným heslem.
- *Počet úspěšných přihlášení* – zda se uživatel nepřihlásil v posledních 10 sekundách více než 5× do svého účtu.
- *Kontrola hodnoty* `User-Agent` – zda má uživatelský `User-Agent` v HTTP hlavičce korektní tvar.
- *Chování uživatele na sociální síti* – zda se uživatelské chování neopakuje až příliš podezřele za posledních 10 dnů.

Dále kapitola obsahuje postup testování implementovaných funkcí aplikačního firewallu a použité nástroje pro testování. Poslední část je věnována dalšímu možnému vylepšení, které mohou zlepšit detekci botů aplikačním firewallem.

7.1 Použité technologie

Pro vývoj aplikačního firewallu byl vybrán jazyk *Node.js*, který se často používá pro tvorbu webových aplikací. Node.js umožňuje jednoduchou implementaci potřebných vlastností jako je sledování chování uživatele, sledování síťového provozu a propojení s databází. Pro správu dat je využit databázový systém *MongoDB*. Mezi největší výhody tohoto databázového systému patří jeho rychlost, která je pro účely aplikačního firewallu nezbytná.

[1] Rozmezí náhodných časových intervalů je možné nastavit v souboru `config.js`.

Výhody jazyka Node.js

Node.js je jazyk založený na programovacím jazyce JavaScript (více zde 2.7) a jeho využití spočívá zejména pro implementaci backendu (funkčnosti) webových serverů. Oproti klasickému JavaScriptu se však liší v tom, že kód je vykonáván právě na straně serveru, nikoliv na straně klienta. Největší výhoda Node.js je provádění asynchronních operací, která se projeví zejména v rychlosti komunikace s databází. Protože sociální sítě mají počet uživatelů v řádech desítek až stovek milionů, je klíčové, aby byla tato vlastnost splněná. Node.js je vysoce škálovatelný, což je pro účely aplikačního firewallu taktéž velmi žádoucí.

Nevýhody jazyka Node.js

Mezi nevýhody jazyka Node.js patří právě asynchronní provádění některých operací, což může způsobit nekonzistence v databázi, což je třeba ošetřit za cenu méně přehledného kódu. Další nevýhoda je nutnost vytvořit prostředí (tzn. správně nastavit webový server) pro běh kódu Node.js a toto prostředí udržovat dobře zabezpečené. Výkon webového serveru je samozřejmě třeba přizpůsobovat počtu uživatelů sociální sítě, čímž úměrně tomu rostou provozní náklady.

Databáze MongoDB

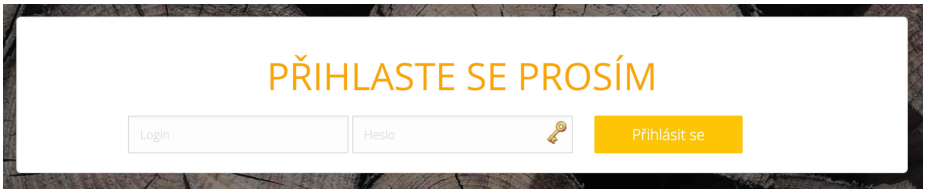
MongoDB je databáze typu NoSQL, která je tvořena tzv. *kolekcemi* (nikoli *tabulkami*) a obsahuje *dokumenty* (nikoli *řádky*). Dokument v databázi je řetězec ve formátu JSON, což je formát, který nativně používá JavaScript, takže není třeba data z databáze jakkoli upravovat. MongoDB umožňuje načtení odpovídajících dat jedním příkazem, která se následně zpracovávají v Node.js.

7.2 Simulovaná sociální síť

Síť, která představuje reálnou sociální síť, se jmenuje *Socnet*. Protože v této práci nejde o implementaci funkční sociální sítě, ale o způsob implementace aplikačního firewallu, chybí zde mnoho prvků, které by v reálné sociální síti neměly chybět (např. vytváření uživatelů). Byl použit opensource design dostupný z webu https://usebootstrap.com/theme/garo-estate-master.

Na úvodní stránce této sítě se uživatel může přihlásit pomocí svého jména a hesla. Referenční uživatelské jméno je '`user`' a heslo je '`user`'. Po přihlášení uživatel vidí dva příspěvky, které jsou staticky vytvořeny[2]. Uživatel může číst příspěvky ostatních uživatelů, klikat na *To se mi líbí* na libovolném příspěvku či napsat text do pole pro komentáře. Snímána jsou pouze uživatelská kliknutí na stránce pro účely aplikačního firewallu, takže zanechaný komentář či kliknutí na *To se mi líbí* jsou při každém přihlášení resetovány. Po dokončení práce se může uživatel odhlásit a tím se dostat zpět na přihlašovací stránku. Přihlašovací stránka sítě Socnet je zobrazena na obrázku 7.1.

[2]Na reálné sociální síti by algoritmus uživateli zobrazil nejvhodnější příspěvek.



Obrázek 7.1: Přihlašovací stránka sítě Socnet

7.3 Návrh detekce botů aplikačním firewallem

Tato sekce obsahuje podrobný popis jednotlivých opatření, které jsou implementovány v rámci aplikačního firewallu Socnetu. Návrh řešení spočívá zejména v odhalování *stejného uživatelského chování*. Tento postup byl navrhnut z důvodu, že žádná sociální síť neodhalila naprosto identické chování 50× za sebou v jedné hodině, z čehož plyne, že žádné síti nepřipadalo podezřelé 50 přihlášení jednoho uživatele během jedné hodiny. Ostatní navrhované testy také vychází z analýzy aplikačních firewallů reálných sociálních sítí, protože většina firewallů sociálních sítí *neodhaluje* takový provoz, který je velmi podezřelý a téměř jistě není v souladu s normálním lidským chováním.

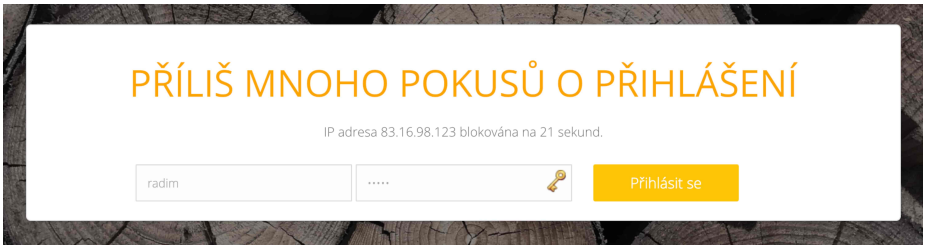
Každé opatření je implementováno samostatně a pokud je vyhodnoceno jako platné, nevyhodnocují se žádná další opatření. Schéma prováděných testů je znázorněno v diagramu 7.2.

Sociální síť Socnet funguje jako klasická sociální síť a normální uživatel by neměl jakkoli poznat, že je jeho aktivita snímána z důvodu ochrany před automatickým přístupem. Kliknutím na tlačítko 'Přihlásit se' se spustí přihlašovací proces, tedy proběhne kontrola, zda-li není uživatelský provoz podezřelý, a pokud není, vytvoří se nová uživatelská session (session viz 2.5). Seznam kontrol, který probíhají při přihlašování, je uveden v následujících podsekcích a probíhá v tomto pořadí. Kontroly pracují s logy uživatelů, které se při každé uživatelské akci ukládají do databáze. Všechny časy, které představují intervaly, jsou hodnoty proměnných a dají se nastavit v souboru `config.js`. V reálném prostředí by kontroly nebyly tak přísné, ale jejich správné nastavení vyžaduje dlouhodobější časovou analýzu.

Počet neúspěšných přihlášení

Jde o ochranu, která chrání uživatele před prolomením hesla *útokem hrubou silou* (viz 3.5). Po každém pokusu o přihlášení se špatným heslem je vytvořen log, jenž je uložen do databáze s hodnotou `"action":"BadPassword"`. Pokud se uživatel snaží přihlásit do svého účtu více než 3× za posledních 30 sekund se špatným heslem, je jeho IP adresa v kombinaci s `User-Agent` zablokována na náhodný čas v intervalu <15, 35> sekund (v reálné sociální síti by pravděpodobně byla tato doba delší a uživateli by byl zaslán email). Využívá se kombinace `User-Agent` a IP adresy proto, aby zabránilo blokování ostatních uživatelů používajících stejnou IP adresu – uživatelský `User-Agent` v lozích musí být shodný s aktuální hodnotou `User-Agent` (jinými slovy se musí jednat o přihlášení ze stejného prohlížeče). Příklad upozornění uživatele o zablokování IP adresy je vyobrazeno na obrázku 7.3.

Pokud se uživatel pokusí přihlásit ze stejného prohlížeče znovu v tomto intervalu, tak se tento čas do nejbližšího přihlášení *nezmění* ani *není vytvořen žádný log*.

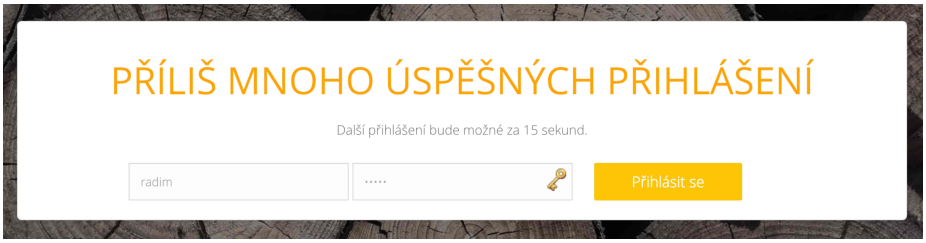


Obrázek 7.3: Příliš mnoho pokusů o přihlášení se špatným heslem

Počet úspěšných přihlášení

Jde o ochranu, která zabraňuje více přihlášením v intervalu 10 sekund na jeden uživatelský účet. Hlavní účel přihlašování se v krátký časový interval (výchozí časový interval je 10 sekund) na jeden uživatelský účet je odstavit síť DoS útokem. Jeden reálný uživatel není schopen přihlásit se v rámci krátkého časového intervalu do svého účtu více než 5×, proto je jeho účet zablokován. Každé úspěšné přihlášení do sítě Socnet je zaznamenáno vytvořením logu s hodnotou `"action":"login"`. Při přihlašování jsou načteny tyto záznamy mladší než 10 sekund. Pokud je počet těchto logů pro daného uživatele větší než 5, je uživatelský účet zablokován na náhodnou dobu v intervalu <10, 30> sekund.

V případě, že se uživatel pokusí přihlásit v tomto intervalu znovu, doba k nejbližšímu přihlášení se *aktualizuje* na náhodnou dobu v intervalu <10, 30> sekund a *není vytvořen žádný log*. Nový čas se generuje z důvodu aplikace co největší náhodnosti, která je pro automatického robota obtížněji překonatelná. Upozornění o zablokovaném účtu je zobrazeno na obrázku 7.4.



Obrázek 7.4: Příliš mnoho úspěšných přihlášení v krátkém časovém okamžiku

Kontrola hodnoty `User-Agent`

Hlavička `User-Agent` obsahuje informace (více viz 2.1) o uživatelské stanici. Jestliže není tato hodnota korektní, pravděpodobně se jedná o podezřelý provoz, protože velmi jednoduchý bot tuto hodnotu neuvádí vůbec nebo má takovou hodnotu, kterou server neakceptuje. Aplikační firewall sítě Socnet kontroluje pomocí regulárního výrazu, zda uživatelský `User-Agent` odpovídá standardnímu tvaru pro hlavičku `User-Agent` a zda neobsahuje nepovolené znaky.

Pokud aplikační firewall odhalí neplatný `User-Agent`, *nepovolí* uživateli přihlásit se z tohoto prohlížeče. Varovná hláška je vyobrazena na obrázku 7.5.

Obrázek 7.5: Upozornění Socnetu na nesprávnou hodnotu `User-Agent`

Chování uživatele na sociální síti

Analýza uživatelského chování je nejpodstatnější a nejsložitější ochrana sítě Socnet. Snaží se rozpoznávat identické chování uživatele od chvíle, kdy se přihlásí do okamžiku zrušení session, tzn. do odhlášení nebo expirace session. Funguje na principu zaznamenávání aktivity do databáze pomocí logů. Aplikační firewall tedy zaznamenává do databáze okamžik přihlášení uživatele a dále všechna jeho kliknutí. Log vzniklý kliknutím obsahuje hodnotu `"action":"click"` a dále čas kliknutí a jméno elementu, na který uživatel kliknul. V okamžiku kliknutí na tlačítko 'Odhlásit se' se zaznamenává poslední log s hodnotou `"action":"logout"`. Pokud automaticky expiruje session, nezaznamenává se žádný log.

Každé toto chování je vyhodnoceno jako vzorec chování. Vzorec chování se tedy skládá z minimálně jedné akce s hodnotou `"action":"login"`. Další akce odpovídají jednotlivým kliknutím na stránce a poslední log zpravidla obsahuje hodnotu `"action":"logout"`. Vzorec chování je uložen ve formátu JSON, jehož příklad je uveden zde 7.6.

```
{
    "action": ["login", "click", "click", "click", "logout"],
    "element": ["null", "like-post1", "comments1", "like-post2", "logout"],
    "seconds_after_login": [0, 2, 5, 7, 15],
    "last_login": "2021-04-24T17:18:31.932+00:00",
    "userId": "607718ec6a5f38c89e030614",
    "count": 3,
}
```

Obrázek 7.6: Vzorec chování ve formátu JSON

Při každém přihlášení se vyhodnocuje *chování odpovídající poslední session* následujícím způsobem:

1. Z databáze se načtou všechny logy odpovídající *poslední session* a z nich:
   - Je vytvořeno pole, do kterého jsou přidány názvy jednotlivých akcí, například `[login, click, logout]`.
   - Je vytvořeno pole, do kterého jsou přidány časy akcí v sekundách od přihlášení (zaokrouhleno na celé sekundy) včetně samotné akce přihlášení, například `[0, 2, 9]`.
   - Je vytvořeno pole, do kterého se ukládají názvy HTML elementů, na které bylo kliknuto (pro akci `login` se ukládá hodnota `null`), například `[null, like-post1, logout]`.

- Poté se vytvoří příslušný vzorec chování s těmito hodnotami.

2. V databázi se hledá identický vzorec (příklad vzorce chování zde 7.6) chování *ne starší 10 dnů* a pokud je nalezen, přičte se 1 k počtu opakování vzorců tohoto chování, v opačném případě se uloží nově vytvořený vzorec a hodnota `count` je nastavena na 1. Se vzorcem se dále pracuje v bodě 3.

3. Pokud je počet opakování vzorce `count` větší než povolený počet[3] (závislost počtu akcí na povoleném počtu opakování je uvedena v tabulce 7.1), znamená to, že uživatelské chování se za posledních 10 dnů příliš často opakuje, a uživatelský účet je zablokován na náhodný čas v intervalu <10, 30> sekund. V opačném případě je uživatel přihlášen.

4. Pokud byl uživatel zablokován (nebyl přihlášen), odečte se od počtu opakování `count` tohoto vzoru 1, aby byla při příštím přihlášení splněna podmínka maximálního počtu opakování vzorce. Uživatel poté musí počkat vygenerovaný náhodný čas, aby se mohl přihlásit – varovná hláška je zobrazena na obrázku 7.7.

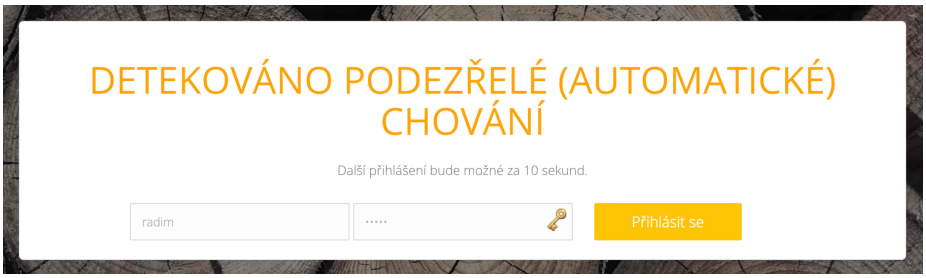
5. Až náhodný čas blokace uplyne, uživateli bude umožněno se přihlásit, přičemž znovu budou provedeny všechny testy uživatele včetně tohoto testu.

Pokud se uživatel v době své blokace pokusí přihlásit znovu, doba k jeho nejbližšímu přihlášení se aktualizuje na náhodnou dobu v intervalu <10, 30> sekund (podobně jako v případě vícenásobného úspěšného přihlášení v krátkém intervalu).

Všechny hodnoty v tabulce 7.1 jsou pouze orientační a je vhodné je optimalizovat tak, aby byl *správně odhalen* automatický přístup, ale také tak, aby *nebyl zablokován* normální uživatel.

Počet akcí na stránce	**Počet povolených opakování**
2	10
3	7
4	5
5	4
6	3
7 a více	2

Tabulka 7.1: Počet povolených opakování vzorce vůči počtu akcí



Obrázek 7.7: Upozornění uživatele o zachyceném podezřelém provozu

[3]S narůstajícím počtem akcí je méně pravděpodobné, že se vzorec chování bude opakovat a naopak – pokud vzorec obsahuje 2 akce, je více pravděpodobné, že se vzorec zopakuje, proto je počet dovolených opakování vzorce závislý na počtu akcí.

7.4 Testování implementovaných funkcí aplikačního firewallu

Implementované funkce aplikačního firewallu byly otestovány nástrojem Selenium WebDriver. Testy byly implementovány v jazyce Python. Hromadné testy byly spouštěny skriptem v jazyce Bash. To znamená, že každá ochrana proti automatickému přístupu byla odzkoušena samostatným testem. Testy, které byly provedeny, jsou popsány v podsekcích níže.

Test počtu neúspěšných přihlášení

Test ověřuje další funkci aplikačního firewallu popsanou v sekci 7.3. Ochrana sítě Socnetu by měla odhalit případy, kdy se uživatel snaží 3× nebo více *přihlásit se špatným heslem* z jednoho webového prohlížeče (tzn. z jedné IP adresy a prohlížeče s daným `User-Agent`). Skript funguje stejně jako v předchozím testu, ale do pole pro heslo vyplňuje *špatné* heslo. Probíhá více testů pro zjištění správné funkce:

1. Méně než 3 neúspěšná přihlášení během 30 sekund z jednoho prohlížeče – firewall *nesmí* zablokovat provoz z webového prohlížeče.
2. Více nebo rovno 3 přihlášením během 30 sekund z jednoho prohlížeče – firewall *musí* zablokovat uživatelský webový prohlížeč.
3. V době blokace jiného prohlížeče uživatel nesmí poznat, že se někdo pokusil uhádnout jeho heslo – firewall *nesmí* uživateli zakázat přihlášení se z ostatních stanic v době blokace útočícího prohlížeče[4]
4. Žádnému uživateli nesmí být umožněno přihlásit se ze zablokovaného prohlížeče.

Pro první část se skript v jednom okamžiku spouštěl 2× a ihned poté se uživatel zkoušel přihlásit se *správným heslem* ve stejném prohlížeči, což se mu podařilo. V druhé části bylo spuštěno 5 oken, ve kterých se uživatel snažil přihlásit se špatným heslem – ve dvou oknech byl uživatel upozorněn na špatné heslo a ve dalších byl upozorněn, že jeho IP adresa byla zablokována. Třetí část probíhala těsně po skončení druhé části, protože uživateli musí být umožněno se přihlásit z jiného webového prohlížeče – uživatel se úspěšně přihlásil. Čtvrtá část testu probíhala ihned po provedení třetího testu a výsledek byl, že ze zablokovaného prohlížeče nebylo umožněno přihlásit se jakémukoli uživateli. Všechny čtyři části tohoto testu byly vyhodnoceny jako *úspěšné*.

Test počtu úspěšných přihlášení

Test ověřuje, zda dokáže aplikační firewall odhalit více než 5 úspěšných přihlášení jednoho uživatele za posledních 10 sekund. Smysl tohoto opatření je popsán v sekci 7.3. Skript byl naprogramován tak, aby spustil okno prohlížeče a úspěšně se pokusil přihlásit do sociální sítě. Test se skládá ze 2 částí, aby mohl být správně otestován:

1. Méně než 6 přihlášení během 10 sekund – firewall *nesmí* odhalit podezřelý provoz a ve všech 5 oknech nechat uživatele úspěšně se přihlásit.
2. Více než 5 přihlášení během 10 sekund – firewall *musí* odhalit podezřelý provoz, tzn. v 5 oknech nechat uživatele se úspěšně přihlásit a v šestém a ve všech dalších oknech nesmí dovolit uživateli se přihlásit.

[4] Klasická sociální síť by však zaslala uživateli email o zablokování účtu a pravděpodobně vyžadovala změnu hesla.

Tento test probíhal tak, že se v jednom okamžiku 5× (pro první část) či 6× (pro druhou část) spustil Python skript (tzn. otevřelo se okno Google Chrome) a uživatel se pokusil přihlásit. V obou částech se aplikační firewall choval přesně podle očekávání: dovolil prvních 5 přihlášení a šestá a další přihlášení byla blokována. Poté bylo vyzkoušeno 10 pokusů o přihlášení ve stejný čas: uživatel se přihlásil úspěšně pouze 5× a ostatní pokusy byly zamítnuty. Aplikační firewall *úspěšně prošel* tímto testem.

Testování nesprávné hodnoty `User-Agent`

Hodnota `User-Agent` prohlížeče, který spouští Selenium, byla nastavena na hodnotu `False_user_agent`. Testované opatření je popsáno v sekci 7.3. Při použití nevalidní hodnoty `User-Agent` byl uživatel přesměrován na stránku s upozorněním přesně podle předpokladu. Aplikační firewall sítě Socnet tento útok *úspěšně odhalil*.

Test chování uživatele na sociální síti

Tento test zkouší, zda aplikační firewall dokáže odhalit identické chování uživatele na sociální síti tak, jak je popsáno v sekci 7.3. Uživatelské chování je dáno posloupností těchto kroků:

1. Uživatel se úspěšně přihlásí
2. Klikne na *To se mi líbí* u prvního příspěvku
3. Zobrazí si komentáře u druhého příspěvku
4. Odhlásí se

Tyto kroky představují *vzorec chování*, který se bude opakovat. Selenium *postupně* spouští okna prohlížeče 8× za sebou (nikoli v jeden okamžik jako v předchozích testech) a v každém okně se chová identicky dle naprogramovaného chování. Testuje se, zda aplikační firewall zablokuje uživatele při šestém[5] procesu přihlášení. Sedmé přihlášení by mělo proběhnout v pořádku, ale osmé by mělo být opět zablokováno. Aplikační firewall se zachoval přesně podle předpokladu, test byl tedy vyhodnocen jako *úspěšný*.

7.5 Další vývoj a možná vylepšení

Navrhovaný aplikační firewall nebere v potaz aktuální denní čas či pořadí dne v týdnu. Automatický robot může být však naprogramován tak, že bude přistupovat na sociální síť každé pondělí v 13:00 a bude provádět určité akce. Jiné akce bude provádět ve středu v 07:15 apod.

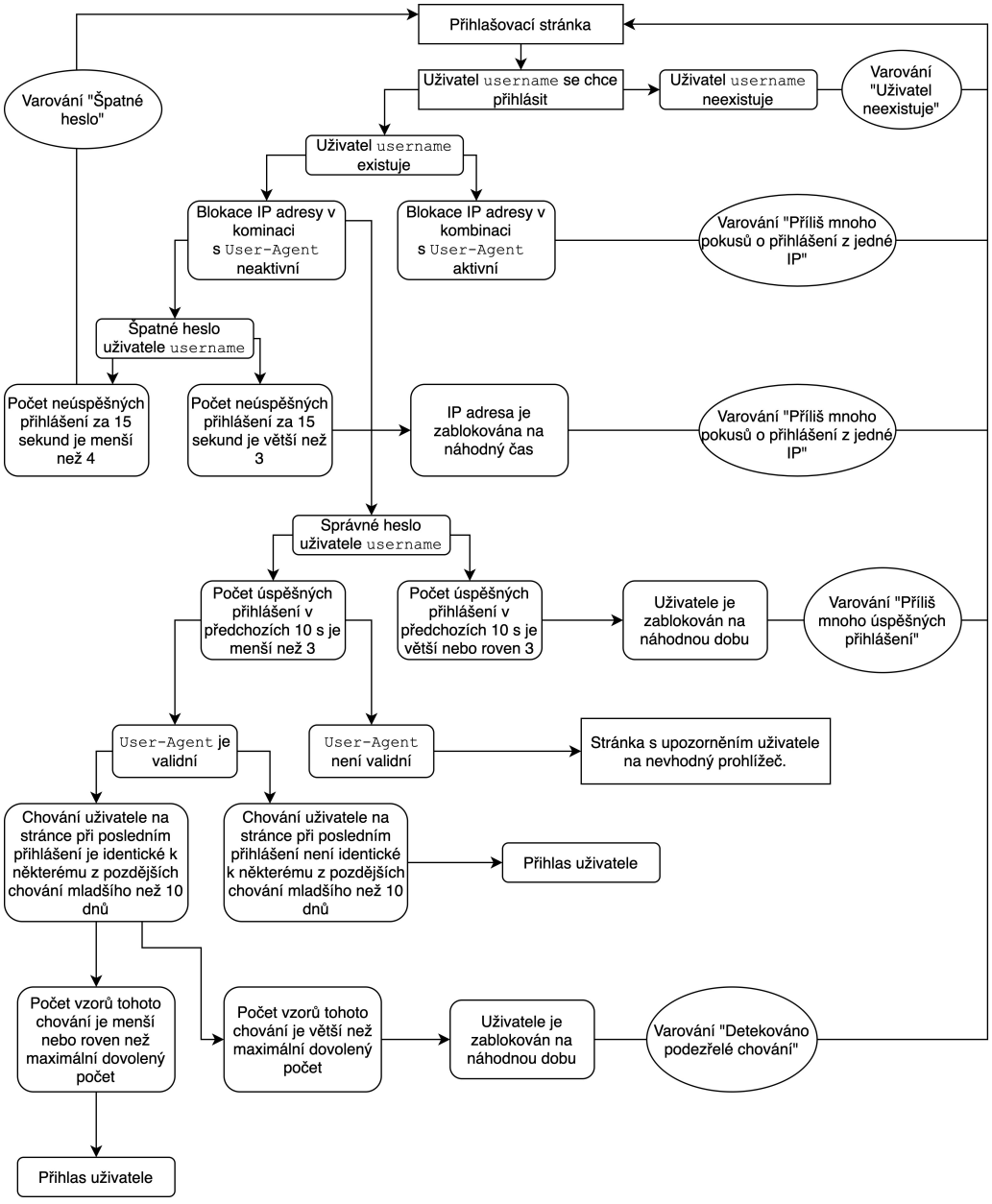
Vylepšení, které by pravděpodobně zvýšilo účinnost aplikačního firewallu, je analýza chování v závislosti na *dni v týdnu*. To znamená, že v neděli by se aplikační firewall choval jinak než v úterý. Tím je myšleno, že by bylo analyzováno chování pro *všechna úterý v posledních 3 měsících* a pokud by počet identických chování (hodnota `count`) překročil určitý počet (tabulka s povoleným počtem opakujících se chování 7.1 se může libovolně upravovat), provoz by se zakázal.

[5] Firewall by měl zablokovat uživatele právě po 5 přihlášeních podle tabulky 7.1.

Další věc, jež může zvýšit účinnost ochrany před automatickým přístupem, je *analýza chování v závislosti na denním času*. V případě, že se uživatel přihlásí v 14:04, jsou analyzována všechna chování, do kterých se uživatel přihlásil v době od 14:00 do 14:10 například v posledních 100 dnech. Samozřejmě je vhodné dobře optimalizovat tabulku s povoleným počtem opakujících se chování 7.1.

Poslední navrhované vylepšení je *zaokrouhlení časů od přihlášení* provedených akcí. Pokročilí roboti totiž mohou provádět stejné akce na sociální síti za jiný čas. Pokud je tedy pole s časy vypadá například takto: `[0, 3, 5, 7, 11]`, může být díky zaokrouhlení vyhodnoceno jako ekvivalentní s tímto polem: `[0, 3, 6, 8, 10]`. Rozsah zaokrouhlení lze nastavit téměř libovolně, avšak příliš vysoký rozsah nebude fungovat a může vyhodnotit normálního uživatele jako robota.

Výše uvedená vylepšení lze různě kombinovat a optimalizovat. Normální uživatel by však neměl zaznamenat jakýkoliv náznak kontroly před automatickým přístupem. Je tedy třeba tyto ochrany nastavovat s citem a je nutné získané výsledky často analyzovat. Jestliže aplikační firewall vyhodnotí 50 % přístupů jako automatické, je pravděpodobně firewall nastaven špatně.



Obrázek 7.2: Pořadí kontrol prováděných aplikačním firewallem při přihlašování uživatele

Kapitola 8

Závěr

Práce popisuje způsoby, jak se mohou webové stránky bránit automatickému přístupu. Byly uvedeny základy webové komunikace, které jsou nutné pro pochopení problematiky automatického přístupu na web. Síťový firewall byl rozdělen na dvě části – *firewally nižších vrstev* a *aplikační firewall*. Firewally nižších vrstev jsou popsány pouze okrajově, avšak součástí této kapitoly jsou i možné útoky na tyto vrstvy. Aplikačnímu firewallu je věnována celá kapitola, která obsahuje způsoby, jak se mohou webové servery bránit automatickým robotům.

Další část práce se věnuje metodám, jakými lze automatický přístup implementovat, včetně popisu výhod a nevýhod každé metody. Mnoho webových serverů má tzv. *rozhraní API*, přes něž mohou ostatní stanice komunikovat automaticky. Druhým způsobem je *web-craping*, což je snaha simulovat přístup člověka pomocí automatického robota například pro získání dat, která jsou pomocí API nedostupná.

Hlavní cíl práce byla analýza aplikačních firewallů sociálních sítí. V této práci jsou tyto firewally analyzovány pomocí nástroje Selenium WebDriver, což je nástroj, který se často používá pro účely webscrapingu. Byly analyzovány sítě Facebook, Twitter, LinkedIn a YouTube. Práce obsahuje tabulky se skutečným chováním uživatelů na sociální síti po jejich přihlášení, což jsou informace, které byly využity při implementaci navrhovaného aplikačního firewallu.

Implementační část, tedy vytvořená sociální síť, se snaží rozlišit robota od člověka. Návrhy implementovaných ochran před automatickými roboty vychází z analýzy aplikačních firewallů reálných sociálních sítí. Pokud robot přistupuje na webovou stránku, bývá jeho chování předem naprogramováno, takže se zpravidla opakuje. Toho využívá navrhnutá detekce botů, která zjišťuje opakované chování, pomocí čehož odhaluje automatický provoz. Byly implementovány i další způsoby, jak odhalit automatický přístup. V případě, že aplikační firewall vyhodnotí uživatelský provoz jako automatický, zablokuje uživatele či jeho webový prohlížeč na náhodnou dobu. Všechna navrhovaná opatření se úspěšně podařilo implemenovat a následně otestovat.

Účelem navrhovaného aplikačního firewallu bylo vylepšit stávající ochrany proti automatickým botům, což může vést k omezení šíření nežádoucího spamu a k lepšímu uživatelskému zážitku na sociální síti.

Literatura

[1] Amy L.DeCarlo a Robert G. Ferrell. *The 5 different types of firewalls explained* [online]. Leden 2021 [cit. 2021-01-18]. Dostupné z: https://searchsecurity.techtarget.com/feature/The-five-different-types-of-firewalls.

[2] Applitools BLOG. *16 reasons why to use Selenium IDE in 2019 (and 2 why not)* [online]. 2019 [cit. 2020-12-16]. Dostupné z: https://applitools.com/blog/why-selenium-ide-2019/.

[3] Beáta Bahureková. *Technika SQL injection - její metody a způsoby ochrany* [online]. Červen 2020 [cit. 2020-12-17]. Dostupné z: https://securityboulevard.com/2020/06/eye-on-the-end-user-application-layer-security/.

[4] Dan Kresa. *Jaké jsou nejčastější typy kybernetických útoků?* [online]. Březen 2018 [cit. 2021-01-08]. Dostupné z: https://www.kybez.cz/clanky/detail?urltitle=jake-jsou-nejcastejsi-typy-kybernetickych-utoku-.

[5] Gary Stevens. *Pravidla pro bibliografické citace* [online]. FIT VUT v Brně, březen 2019 [cit. 2020-10-02]. Dostupné z: https://www.fit.vut.cz/study/theses/citations/.

[6] George D. Montanez. *The Turing Test and Other Design Detection Methodologies* [online]. Květen 2016 [cit. 2021-01-18]. Dostupné z: https://www.cs.cmu.edu/~gmontane/pdfs/montanez-2016-detecting.pdf.

[7] Google Developers. *Chrome DevTools* [online]. Červenec 2020 [cit. 2020-12-10]. Dostupné z: https://developers.google.com/web/tools/chrome-devtools/.

[8] Irvin Dongo, Fabiola Martínez a další. *Web Scraping versus Twitter API: A Comparison for a Credibility Analysis* [online]. Prosinec 2020 [cit. 2021-03-05]. Dostupné z: http://www.iiwas.org/conferences/iiwas2020/proceedings/iiwas-papers/p263-dongo.pdf.

[9] John Paul. *Detecting TOR Communication from the Organization* [online]. Leden 2018 [cit. 2021-03-03]. Dostupné z: https://www.linkedin.com/pulse/detecting-tor-communication-from-organization-gift-john-paul.

[10] Katalon Studio. *A Deep Dive into Selenium, Its Alternative Solution for 2020 and Beyond* [online]. 2020 [cit. 2020-12-16]. Dostupné z: https://medium.com/katalon-studio/a-deep-dive-into-selenium-its-alternative-solution-for-2020-and-beyond-e49bcd428fa8.

[11] Kolektiv autorů. *IP Blacklist* [online]. Leden 2019 [cit. 2020-12-20]. Dostupné z: https://www.imperva.com/learn/application-security/ip-blacklist/.

[12] Kolektiv autorů. *What Is a DDoS Attack?* [online]. Únor 2019 [cit. 2021-01-12]. Dostupné z: https://www.cisco.com/c/en/us/products/security/what-is-a-ddos-attack.html.

[13] Kolektiv autorů. *The Selenium Browser Automation Project* [online]. 2020 [cit. 2020-12-16]. Dostupné z: https://www.selenium.dev/documentation/en/.

[14] Kolektiv autorů. *Web Parameter Tampering* [online]. 2020 [cit. 2020-12-17]. Dostupné z: https://owasp.org/www-community/attacks/Web_Parameter_Tampering.

[15] Kolektiv autorů. *What is an Application Firewall?* [online]. 2020 [cit. 2021-01-15]. Dostupné z: https://www.f5.com/services/resources/glossary/application-firewall.

[16] Kolektiv autorů JavaTPoint. *Selenium Tool Suite* [online]. 2018 [cit. 2020-12-16]. Dostupné z: https://www.javatpoint.com/selenium-tool-suite.

[17] Macil 1.0.1. *Discord client in Electron* [online]. Prosinec 2016 [cit. 2020-12-18]. Dostupné z: https://github.com/Macil/discord-tron.

[18] Martin Bednář. *Automatické testování projektu JavaScript Restrictor* [online]. 2019 [cit. 2020-12-16]. Dostupné z: https://dspace.vutbr.cz/handle/11012/195009.

[19] MDN contributors. *Overview of HTTP* [online]. Září 2019 [cit. 2020-10-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[20] MDN Contributors. *Introduction to web APIs* [online]. Září 2020 [cit. 2020-12-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction.

[21] MDN Contributors. *Third-part API* [online]. Září 2020 [cit. 2020-12-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Third_party_APIs.

[22] Michal Dobeš. *Nástroj pro penetrační testování webových aplikací* [online]. Brno: vutbr.cz, červen 2015 [cit. 2020-12-17]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace?zp_id=88410.

[23] Miroslav Čermák. *Obfuskace a základní obfuskační techniky* [online]. Květen 2016 [cit. 2021-50-07]. Dostupné z: https://www.cleverandsmart.cz/obfuskace-a-zakladni-obfuskacni-techniky/.

[24] Nick Briz. *This is Your Digital Fingerprint* [online]. 2018 [cit. 2020-12-17]. Dostupné z: https://blog.mozilla.org/internetcitizen/2018/07/26/this-is-your-digital-fingerprint/.

[25] Petr Medek. *Porovnání a využití nástrojů simulujících síť Tor* [online]. Květen 2020 [cit. 2021-03-04]. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/191529/final-thesis.pdf?sequence=3&isAllowed=y.

[26] R. Fielding a další. *Hypertext Transfer Protocol* [online]. Červenec 1999 [cit. 2020-10-28]. Dostupné z: https://tools.ietf.org/html/rfc2616.

[27] Rabih Haidar, Shady Elbassuoni. *Website Navigation Behavior Analysis for Bot Detection* [online]. Říjen 2017 [cit. 2021-01-08]. Dostupné z: https://ieeexplore.ieee.org/document/8259764.

[28] Radek Burget. *JavaScript a jQuery* [private]. Brno: vutbr.cz, listopad 2020 [cit. 2020-12-18]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIIS-IT%2Flectures%2Fp09b_JavaScript.pdf&cid=13985.

[29] Sayeed Z Sajal. *Interactive sensitive data exposure detection through static analysis* [online]. researchgate.net, květen 2017 [cit. 2020-12-17]. Dostupné z: https://www.researchgate.net/publication/320174430_Interactive_sensitive_data_exposure_detection_through_static_analysis.

[30] Petr Štěpánek. *Method and system for separating HTTP session* [online]. 2013 [cit. 2020-12-15]. Dostupné z: https://www.researchgate.net/publication/302648786_Method_and_system_for_separating_HTTP_session.

[31] T. Berners-Lee. *Hypertext Markup Language - 2.0* [online]. Listopad 1995 [cit. 2020-11-09]. Dostupné z: https://tools.ietf.org/html/rfc1866.

[32] TmaxSoft Co., Ltd.. *Session Tracking* [online]. Březen 2017 [cit. 2020-12-15]. Dostupné z: https://technet.tmaxsoft.com/upload/download/online/jeus/pver-20160610-000001/session/chapter_session_tracking.html.

[33] Victor R. L. Shen, Chin-Shan Wei a další. *Javascript Malware Detection Using A High-Level Fuzzy Petri Net* [online]. Brno: ieeexplore.ieee.org, červenec 2018 [cit. 2020-12-18]. Dostupné z: https://ieeexplore.ieee.org/document/8527036.

[34] Vojtěch Bastl. *Automatizace webového prohlížeče* [online]. 2019 [cit. 2020-12-16]. Dostupné z: https://dspace.vutbr.cz/handle/11012/180342.

Příloha A

Obsah přiloženého paměťového média

/app	Zdrojové soubory v jazyce Node.JS včetně samostatného README, který obsahuje popis spuštění, a soubor `config.js` pro nastavení časových intervalů používaných v práci.
/selenium_tests	Veškeré soubory pro analýzu reálných sociálních sítí (kapitola 6) a skripty provádějící testy funkcí implementovaného aplikačního firewallu (kapitola 7).
/text	Zdrojové soubory LaTeX pro sestavení textu práce.
text.pdf	Text práce (odevzdávaná verze).
README.md	Popis řešené problematiky a jednotlivých částí práce.