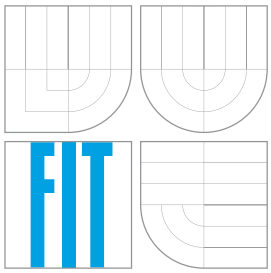
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

AUTOMATIZOVANÉ TESTOVÁNÍ ZRANITELNOSTÍ WEBOVÝCH PROHLÍŽEČŮ

AUTOMATED TESTING OF WEB BROWSER VULNERABILITIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JÁN JUSKO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VOJTĚCH FRÖML

BRNO 2016

Zadání bakalářské práce/18735/2015/xjusko00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Jusko Ján**
Obor: Informační technologie
Téma: **Automatizované testování zranitelností webových prohlížečů**
Automated Testing of Web Browser Vulnerabilities
Kategorie: Bezpečnost

Pokyny:

1. Seznamte se s nástroji pro automatizované testování aplikací (Robot Framework, Selenium).
2. Seznamte se se základními technikami útoků na webové prohlížeče v prostředí operačního systému Windows.
3. Navrhněte vhodné útoky implementovatelné pomocí zvolených nástrojů, které prověří efektivitu bezpečnostních aplikací.
4. Implementujte navržené útoky.
5. Zhodnoťte nebezpečnost Vámi implementovaných útoků v reálném světě a ověřte reakce na ně bezpečnostními aplikacemi.

Literatura:

- H. Percival. *Test-Driven Development with Python.* O'Reilly Media, 2014
- M. Russinovich, D. Solomon, A. Ionescu. *Windows Internals (6th edition).* Microsoft Press, 2012
- W. Alcorn, C. Frichot, M. Orru. *The Browser Hacker's Handbook.* Wiley, 2014

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

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

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

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

Vedoucí: **Fröml Vojtěch, Ing.**, UIFS FIT VUT
Konzultant: Fröml Vojtěch, Ing., UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cieľom tejto práce je zoznámiť sa so základnými technikami útokov na webové prehliadače a následne navrhnúť a implementovať automatizované testovanie bezpečnostných aplikácií spúšťaním nelegálnej aktivity. Pomocou nástroja Robot Framework bola pre tri najrozšírenejšie webové prehliadače vytvorená sada 31 automatizovaných testov, ktoré simulujú škodlivú činnosť v prehliadači. Následne bola analyzovaná efektivita a reaktivita bezpečnostných aplikácií. Zistili sme, že používaním vhodného anti-exploit systému je možné zabrániť až 30% útokom. Prínosom tejto práce je samotná testovacia sada, ktorá bude integrovaná do databáze testov spoločnosti TrustPort a využívaná pri vývoji bezpečnostných systémov.

Abstract

Aim of the thesis is to study basic techniques of attacking a web browser and afterwards to propose and implement automated testing of security applications by running malicious activity. By means of Robot Framework was created a set of 31 automated tests which simulate malicious activity inside a three most popular web browsers. After that we analysed effectiveness and reactivity of security applications. We found out that by using suitable anti–exploit software it is possible to prevent 30% of attacks. The asset of this thesis is the set of tests which will be integrated into the database of tests at TrustPort company and will be used at development of their security software.

Kľúčové slová

bezpečnosť, web, automatizované testovanie, Robot Framework, FireBreath Framework, plug-in, Windows

Keywords

security, web, automated testing, Robot Framework, FireBreath Framework, plug-in, Windows

Citácia

JUSKO, Ján. *Automatizované testování zranitelností webových prohlížečů*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Fröml Vojtěch.

Automatizované testování zranitelností webových prohlížečů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vojtěcha Frömla. Dalšie informácie mi poskytol pán Marek Střihavka zo spoločnosti TrustPort. Uviedol som všetky literárne prameňe a publikácie, z ktorých som čerpal.

.....
Ján Jusko
17. mája 2016

Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce Ing. Vojtěchovi Frömlovi za odborné vedenie, užitočné rady, pripomienky a trpezlivosť. Ďalej ďakujem Marekovi Střihavkovi a celému tímu v spoločnosti TrustPort a.s. za odbornú spoluprácu pri testovaní.

Obsah

Kapitola 1

Úvod

Webové prehliadače, programy slúžiace na sprístupnenie Internetu, sú v dnešnej dobe bez pochýb najviac využívané aplikácie na osobných počítačoch pripojených k sieti. V prenesenom slova význame by sa webové prehliadače dali nazvať oknom do internetu, cez ktoré je užívateľ schopný vykonávať rôzne aktivity, počnúc bežnou e-mailovou komunikáciou, prehľadávanie webu, online nákupy až po správu svojho biznisu alebo diaľkové ovládanie inteligentných domov. Webový prehliadač slúži ako interpret, ktorý aplikuje kontext a vzhľad prijatým informáciám. Prehliadač má k všetkým prijatým dátam prístup v podobe bežného textu, a to aj napriek použitiu šifrovania komunikácie na nižšej vrstve modelu TCP/IP. Preto sa webové prehliadače stávajú zraniteľným článkom a zároveň veľmi obľúbeným terčom útočníkov snažiacich sa o zneužitie, krádež alebo narušenie užívateľových dát. Zaistiť bezpečnosť a úplnú odolnosť prehliadača voči útočníkom je plne v réžii jeho vývojárov. Zranitelnosť prehliadača sa však mnohonásobne zvyšuje pri používaní rozšírení alebo plug-inov vývíjaných tretími stranami. Rozšírenie v prehliadači je modul, ktorý umožnuje pridať na funkcionalite alebo akokoľvek upraviť či spríjemniť zážitok z prehliadania webu. Jedným z najpoužívanejších rozšírení je AdBlock [35], ktorý blokuje zobrazovanie častí webových stránok, hlavne reklám. Napriek klesajúcemu trendu sú rozšírenia a pluginy stále veľmi rozšírené, len prehliadač Mozzilla registruje vyše 500 miliónov [17] aktívne používaných rozšírení. V tejto práci bude vysvetlený dôvod úpadku používania zásuvných modulov - a to koniec podpory NPAPI rozhrania, ktoré poskytuje pluginom prehnané privilégia v operačnom systéme. Na odhalenie a následnú ochranu pred škodlivými aplikáciami slúžia antivírové programy, ktoré pracujú najmä na princípe pravidelného skenovania súborov na disku a hľadania podobnosti s už odhalenými vírusmi uloženými v databáze. Takáto podobnosť môže nastať buď pri zhode častí kódu alebo parametra checksum [28] oboch súborov.

Cieľom tejto práce je vytvoriť sadu automatizovaných testov, ktoré budú simulovať škodlivú činnosť spúštanú z pluginu webového prehliadača a následne analyzovať efektivitu bezpečnostných programov. Táto práca je vedená v spolupráci s TrustPort, firmou vyvíjajúcou antivir a anti-exploit systémy. Práca sa zameriava práve na analýzu účinnosti ich produktu. Pred samotnou implementáciou bolo potrebné sa zoznámiť s fundamentálnymi pojmami a princípmi použitými v práci. V kapitole 2 je popísané ako fungujú webové prehliadače. Následne kapitola 3 sa zaoberá typmi najbežnejších útokov, ktoré ohrozujú bezpečnosť prehliadačov. Druhá polovica technickej správy sa venuje predstaveniu použitých technológií pri implementácii v kapitole 4, podrobnému popisu implementácie v kapitole 5 a vyhodnotením získaných výsledkov testovania v kapitole 6.

Kapitola 2

Architektúra prehliadačov

Táto práca sa snaží čo najlepšie navrhnúť a implementovať testy, ktoré overia zraniteľnosť najpoužívanejších webových prehliadačov. Nevyhnutnou prerekvizitou tejto úlohy je pochopiť fundamentálne základy fungovania webového prehliadača a jeho prvkov, tak ako ich definoval patent [30] v roku 1995. Tak isto je podstatné rozumieť špecifikám rozdielných prehliadačov a rozoznať rozdiely medzi nimi. V tejto kapitole si po načrtnutí typických prvkov prehliadača priblížime aj trojicu tých najznámejších, menovite: Mozilla Firefox verzie 6, Internet Explorer 9 a Google Chrome verzie 23. Ku každému prehliadaču je vypracovaná schéma s prirovnaním k referenčnej architektúre červenou prerušovanou čiarou.

2.1 Referenčná architektúra webového prehliadača

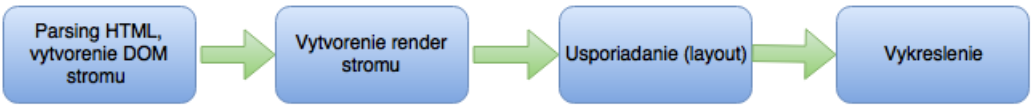
Pojem referenčná architekúra prehliadača zachytáva základnú charakteristiku jeho hlavných programových modulov a vzťahov medzi nimi. Spája charakteristické črty systému do jednej množiny, pomáha pri porozumení systému a mala by byť oporou a šablónou pre vývojárov pri vývoji nového prehliadača takéhoto typu.
Webový prehliadač slúži na sprístupnenie a zobrazovanie obsahu na sieti WWWW (World Wide Web). WWW je sieť internetových serverov, kde každý zdroj je jednoznačne identifikovateľný pomocou URI (Uniform Resource Identifier). Medzi takéto zdroje patria dokumenty, obrázky, videá atd. Dokumenty sú typicky zapísané v jazyku HTML (HyperText Markup Language viď kapitola 4.4), ktorý podporuje vkladanie odkazov na dalšie zdroje alebo dokumenty. Pomocou HTTP protokolu (HyperText Transfer Protocol) webový prehliadač získava tieto zdroje a užívateľovi ich zobrazuje na obrazovku v prívetivej forme. Prehliadač poskytuje aj iné dôležité funkcie ako napríklad ukladanie histórie prehliadania a správu HTTP cookies.
Z pohľadu softwarovej architektúry sa typický prehliadač skladá z ôsmich základných modulov. [12, 13] Pre lepšie pochopenie sú vzťahy znázornené na obrázku 2.2.

Užívateľské rozhranie *(User Interface)* je časť prehliadača viditeľná pre bežného užívateľa. Tvorí vrstvu medzi užívateľom a modulom browser engine. Užívateľovi sprístupňuje mnoho funkcií ktoré zaobstarávaju už ostatné časti prehliadača. Napríklad správu záložiek a sťahovaní, prehliadanie vo viacerých kartách zároveň alebo ukladanie hesiel. Základné prvky každého užívateľského rozhrania sú address bar, tlačidlá back, forward, refresh a home. Pre pokročilých užívateľov a vývojárov poskytujú štandardné prehliadače aj konzolu, ktorá slúži

na interakciu s webovou stránkou pomocou JavaScript príkazov alebo logovanie chybových hlášok.

Browser engine Modul browser engine vytvára rozhranie medzi užívateľským rozhraním a modulom rendering engine. Posúva tým abstrakciu funkcií rendering enginu na vyššiu úroveň. Je zodpovedný za načítanie URI a pomocou dotazov na rendering engine poskytuje základné operácie prehliadania ako back, forward, reload.

Rendering engine Subsystém rendering engine, najdôlejžitejšia časť prehliadača, vytvára vizuálnu reprezentáciu načítaných zdrojov. Jeho súčasťou je vstavaný HTML parser a spolu so samostatným XML (eXtensible Markup Language) parserom, ktorý dotazuje, je schopný zobrazovať HTML a XML dokumenty. Ďalej dokáže zobrazovať obrázky a štylyzovať stránku v prípade definovaného CSS (Cascade Styling Sheet) súboru. Zobrazovanie neštandardných zdrojov zabezpečujú rôzne zásuvné moduly, napríklad Adobe Reader pre PDF dokumenty. Proces renderovania stránky ilustrovaný na obrázku 2.1 začína dotazovaním obsahu zdrojov na sieťový modul, prenos prebieha zvyčajne v chunkoch o veľkosti 8kB. Nasleduje parsing HTML dokumentu, preklad elementov na uzly DOM (Document Object Model) stromu a vytvorenie samotného stromu. Po sparsovaní prípadných CSS predpisov umiestnených či už v samostatnom súbore alebo vnútri tagov *<style>* sa vytvorí takzvaný render strom ktorý pozostáva z obdĺžnikov rôznych rozmerov a definovaných farieb ktoré sa majú zobraziť na obrazovku. Render strom následne prechádza procesom preusporiadania (layout process) ktorý jednoznačne určí súradnice pre každý uzol stromu a v poslednej etape sa tieto uzly postupne prechádzajú a vykresľujú v module display backend.



Obr. 2.1: Postup renderingu.

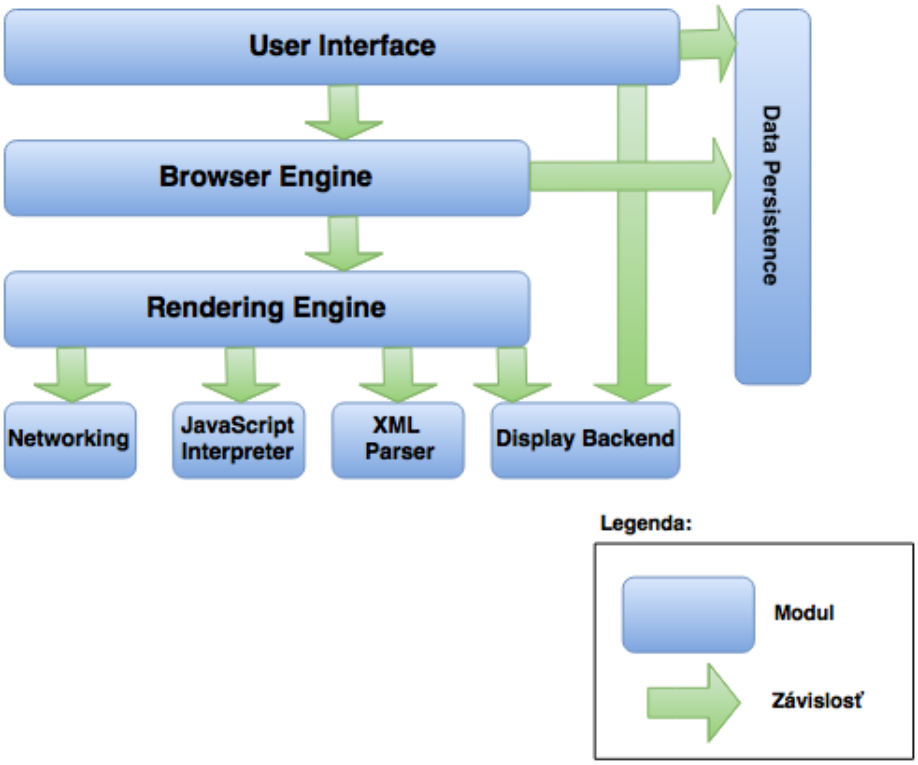
Networking modul Sieťový modul implementuje protokoly na prenos súborov z internetu. Základné protokoly, ktoré podporujú prehliadače sú HTTP, HTTPS a FTP (File Transfer Protocol). Môže takisto ukladať aj malé množstvo cache dát, kde sa ukladajú nedávno načítané zdroje.

JavaScript interpret V prípade, že HTML parser narazí v kóde na tag *<script>*, riadenie sa okamžite predá JavaScript interpretu. JavaScript jazyk slúži na tvorbu interaktívnych webových stránok, preto sa zvyčajne jeho kód namapuje na udalosti ako napríklad kliknutie alebo hover myšou. Pri zaznamenaní udalosti sa spustí príslušný kód ktorý ju obsluhuje. Z bezpečnostných dôvodov sa istá množina JavaScript príkazov (pop-up window, regulárne výrazy atd.) zakazuje už v module browser engine.

XML parser Stará sa o parsovanie prípadných súborov v značkovacom jazyku XML. V novších architektúrach už nezvykne figurovať ako samostatný modul, ale ako súčasť rendering engine.

Display backend Display backend vykresluje základné vektorové grafické prvky a text, z ktorých sa vytvára obraz. Je úzko spätý s operačným systémom.

Data storage Úložisko dát. Prehliadač potrebuje pri svojej činnosti ukladať na disk rôzne dáta spojené s aktuálnym prehliadaním [2] (cache, cookies) alebo perzistentné dáta (záložky, bezpečnostné certifikáty, nastavenia atd.). Tento modul sa stará o ukladanie a načítavanie týchto dát.



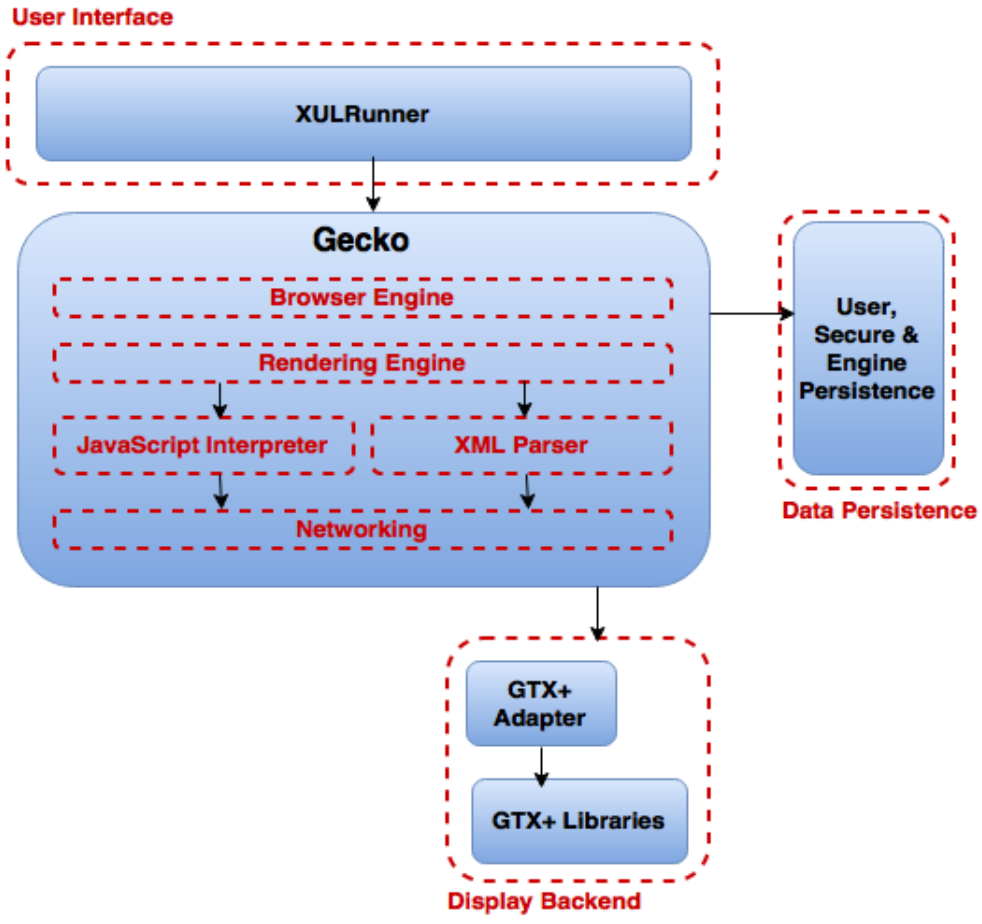
Obr. 2.2: Referenčná architektúra prehliadača

2.2 Mozilla Firefox 6

Mozilla Firefox 6 [18] (ďalej len Firefox) je volný a zároveň open-source webový prehliadač vyvíjaný spoločnosťou Mozilla Foundation s históriou siahajúcou až do roku 2003, kedy bola vydaná prvá verzia s označením Firefox 0.5. Už vtedy sa jednalo o plnohodnotný webový prehliadač podporujúci všetky základné funkcie ba aj niečo navyše. Pre účely tejto práce si priblížime konceptuálnu architektúru verzie 6 z roku 2011. [5] Schéma 2.3 naznačuje, že hlavnú rolu v prehliadači Firefox zohráva Gecko, ktorý dostáva od viacerých zdrojov prívlastok browser engine, táto terminológia je však nesprávna. Jedná sa o jadro prehliadača zapuzdrujúce browser engine a rendering engine, sieťový modul, JavaScript interpret a XML parser. Príznačný pre prehliadač je jeho multiplatformný prístup k užívateľskému rozhraniu, ktoré je definované v jazyku XUL (XML User Interface Language). Nasleduje popis jednotlivých modulov.

Jazyk XUL / XULRunner XML User Interface Language [19] je značkovací jazyk spoločnosti Mozilla používaný na tvorbu multiplatformných užívateľských prostredí. V podstate sa jedná o XML jazyk podporujúci HTML elementy a JavaScript. Konštrukcie tohto jazyka sú mapované na knižnice špecifické operačnému systému pomocou vrstvy adaptérov. Prostredie, v ktorom bežia programy v XUL jazyku je XULrunner, ten priamo sprístupňuje všetky Gecko submoduly pre aplikácie tretích strán. To prinášalo mnoho bezpečnostných rizík a preto od Júla 2015 sa upustilo od podpory a používania prostredia XULrunner.

Gecko Ako bolo spomenuté vyššie, Gecko zohráva dôležitú úlohu, kedže okrem Display backend a uložiska dát zapuzdruje všetky moduly popísané v referenčnej architektúre. Na jadre Gecko teda závisí parsing, rendering, layout a takisto sieťové operácie. Z hľadiska bezpečnosti je najzaujímavejší sieťový submodul Necko ktorý využíva Network Security Services čo predstavuje množinu knižníc na podporu zabezpečenej komunikácie klient–server pomocou štandardov SSL (Secure Sockets Layer), TLS (Transport Layer Security), PCKS (Public Key Cryptography Standard) atd.



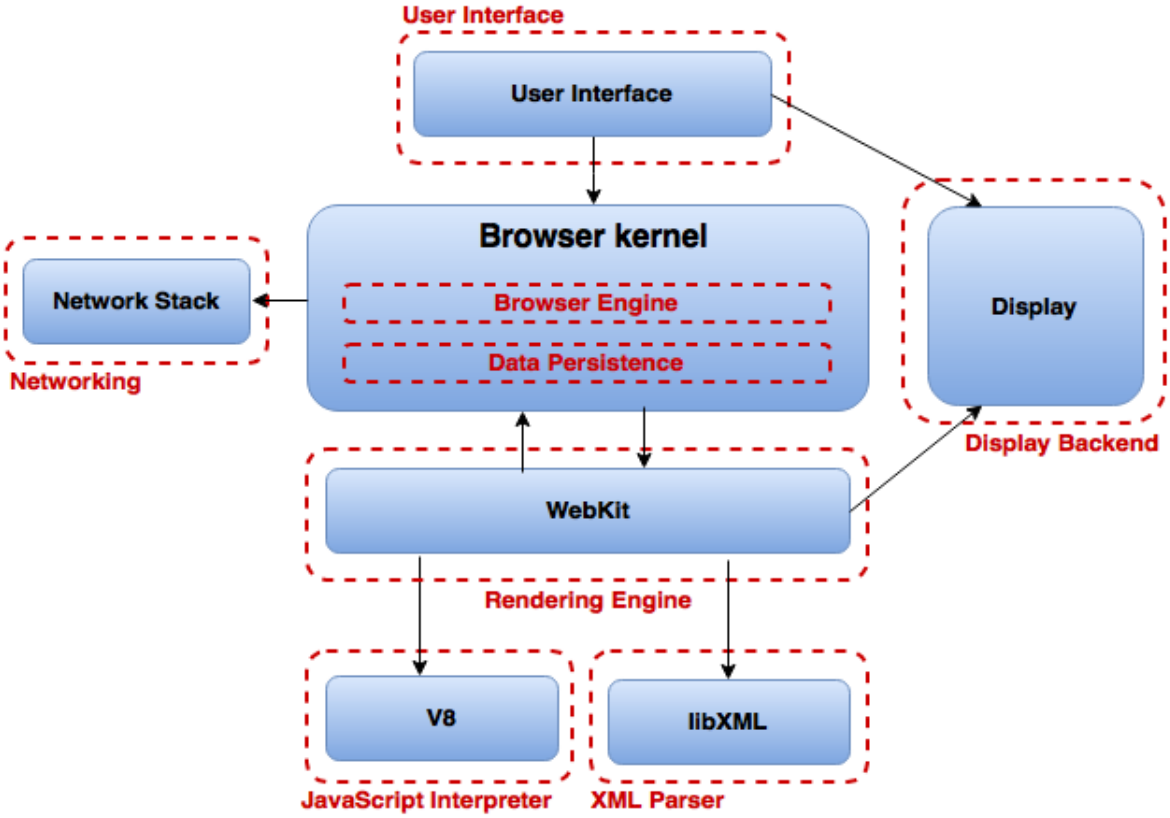
Obr. 2.3: Referenčná architektúra prehliadača Firefox

2.3 Chrome v23

Chrome je freeware webový prehliadač vyvíjaný technologickým gigantom Google. [11] Verejnosti bol predstavený v roku 2008 a odvtedy sa vyznačuje svojim jednoduchým užívateľským rozhraním ktoré čím ďalej priťahuje viac a viac užívateľov. Posledné prieskumy ukazujú, že až 60 percent užívateľov používa práve prehliadač Chrome. Chrome je postavený na multi-procesovej architektúre, znamená to že každá otvorená karta beží ako samostatný proces nezávisle na prehliadači, čo umožnuje obslúžiť aj náročné webové aplikácie. Takisto to prispieva na stabilite celého prehliadača, kedže zlyhanie jedného procesu neovplyvní chod ostatných procesov, ani samotného prehliadača. Pri vývoji prehliadača Chrome sa kladie veľký dôraz na bezpečnosť a prevenciu pred útokmi. V snahe minimalizovať zraniteľnosti sa architektúra Chrome vo veľkej miere odlišuje od tej tradičnej monolitickej, kde browser engine a rendering engine beží ako jeden proces. Ak v takomto prípade útočník využije nejakú dieru zvyčajne v rendering engine, je tak schopný získať privilégia celého prehliadača, teda napríklad spúšťať svoj škodlivý kód. Architektúra Chrome sa vyznačuje oddelením Rendering engine od Browsing engine do samostatných procesov, pričom zraniteľnejší Rendering engine, ktorý je v priamom kontakte s webovými zdrojmi, je umiestnený do Sandbox prostredia.[6] Sandbox zakazuje procesu systémové volania, ktoré môžu viesť k získaniu kontroly napríklad nad súborovým systémom. Rendering engine ale k svojmu fungovaniu niektoré systémové služby nevyhnutne potrebuje, avšak pristupuje k nim sprostredkovane, cez API rozhranie implementované Browser enginom. Vzťah Browser enginu a Rendering enginu je znázornený na 2.4. Do roku 2015 podporoval prehliadač Chrome aj zásuvné moduly (plug-in), ktoré boli spustené ako samostatný proces s plnými užívateľskými právami. Z bezpečnostných dôvodov bola podpora NPAPI rozhrania, implementujúce systémové služby plug-inom, zrušená. Náhradou sú rozšírenia, ktoré nemajú také vysoké privilégia v operačnom systéme, lebo bežia v sandbox prostredí rendering enginu. Nasleduje popis dvoch hlavných modulov Chrome architektúry, ich priority a služby.

Browser kernel Je zodpovedný za správu všetkých inštancií rendering enginu ktorým poskytuje svoje API rozhranie. Jedna inštancia predstavuje v prehliadači Chrome jeden tab. Spravuje perzistentné dáta (záložky, cookies, heslá) a obsahuje aj networking modul, ktorý sťahuje internetové dokumenty. Ako proces komunikuje z jednej strany s operačným systémem (napríklad win32 api) a z druhej strany poskytuje rozhranie rendering enginu. Takáto architektúra dokáže zastaviť až 70% kritických útokov. [4]

Rendering engine Interpretuje a zobrazuje obsah webu. Proces začína parsovaním dostupných HTML, CSS, XML súborov, pokračuje stavbou Document Object Model stromu, nastavením layoutu a zapísaním bitmapy do browser kernela, ktorý ju skopíruje na obrazovku. Proces Rendering enginu beží v sandbox prostredí z dôvodu, že pri parsovaní môže naraziť na škodlivé kusy kódu (JavaScript injection, regulárne výrazy).



Obr. 2.4: Referenčná architektúra prehliadača Chrome

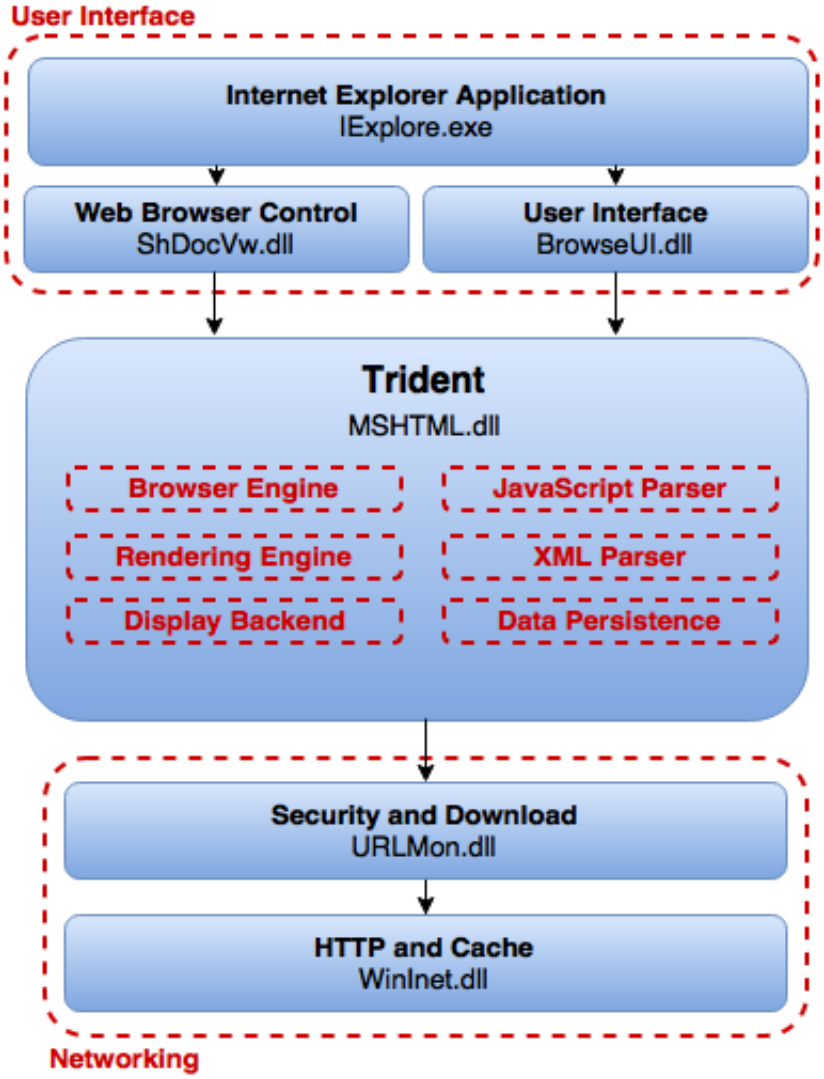
2.4 Internet Explorer 9

Je webový prehliadač vyvíjaný firmou Microsoft dodávaný spolu s ich operačnými systémami už od roku 1995. V rokoch 2002-2003 dosahoval svoju najvyššiu popularitu s vyše 95% podielom na trhu. Charakteristickou črtou IE je využívanie Component Object Model (COM). Jedná sa o nezávislý systém vyvinutý práve Microsoftom pre vytváranie binárnych komponentov širokej škály programovacích jazykov a umožnenie medziprocesovej komunikácie medzi týmito objektmi. Z hľadiska bezpečnosti je tento model nevyhovujúci, kedže kód komponentov beží ako natívny. Riešením bol príchod ActiveX frameworku, ktorému bola zverená starosť o používanie komponent z neoverených zdrojov a z internetu. ActiveX priniesol prísne defaultné bezpečnostné nastavenia, digitálne podpisovanie komponent a čiernu listinu škodlivých komponent. Architektúra IE sa opäť veľmi úzko podobá na tú referenčnú, pričom každý modul je obsiahnutý v práve jednej DLL knižnici. Hlavným modulom IE je Trident v MSHTML.dll, ktorý by sa dal prirovnať k modulu Gecko v prehliadači mozilla.[7, 36] Spája totiž funkcionalitu parserov, renderov, JavaScript interpretu. Sieťový modul je však riešený v IE samostatne, a to v knižnici WinINET.dll. Na schéme IE sú pre názornosť prerušovanou čiarou vyznačené jednotlivé submoduly ktoré odpovedajú referenčnej architektúre. Všetky dll sú spúštané a spravované z hlavného spustiteľného súbora iexplore.exe. Explorer povoľuje dva typy rozšíriteľnosti.

- *Browser extensibility* – pridávanie prvkov do užívateľského rozhrania (toolbars, menu buttons atd.)

- *Content extensibility* – podpora nenatívnych MIME formátov, zobrazovanie netypických súborov alebo sieťových protokolov

Prehliadač IE ako produkt skončil a nahradil ho Edge, ktorý posúva úroveň bezpečnosti vyššie používaním sandbox prostredí. Vývojári oznámili, že Edge nebude podporovať zásuvné moduly tretích strán. O vnútornej architektúre bohužiaľ neexistujú žiadne oficiálne informácie.



Obr. 2.5: Referenčná architektúra prehliadača Internet Explorer

Kapitola 3

Techniky útokov na webové prehliadače

S rastúcou popularitou internetu rastie zároveň aj používanosť webových prehliadačov. Pre väčšinu užívateľov je webový prehliadač jediným médiom, cez ktoré pristupujú k internetu. Preto je logické, že prehliadače sa dostali do pozornosti útočníkom snažiacich sa zneužiť bezpečnostné diery alebo dôveru užívateľa k svojím nekalým praktikám. Na šírenie osvety bezpečnosti webových aplikácií bola založená nezisková nadácia OWASP (Open Web Application Security Project) ktorá uverejňuje na svojích stránkach voľne dostupné články, dokumentácie a nástroje. Útoky snažiace sa využiť webové prehliadače ako vstupné médium by sa dali kategorizovať do troch skupín popísaných v nasledovných podkapitolách.

3.1 Útoky na užívateľa

Útoky zamerané na zlyhanie ľudského faktora pri prehliadaní. Využívajú neznalosť a neopatrnosť mnohých užívateľov, ktorý dôverujú falošným e-mailom, správam alebo škodlivému obsahu. Touto oblasťou sa zaoberá samostatná disciplína nazvaná sociálne inžinierstvo, ktoré však viac zasahuje do psychológie ako do informatiky. Jedným z najčastejších útokov spoliehajúcich sa na dôveru užívateľa je phishing.

3.1.1 Phishing

Phishing je technika získavania citlivých informácií priamo od užívateľa typicky so zámerom neskoršieho zneužitia. Najčastejšie sa jedná o legitímne vyzerajúce e-maily ktoré požadujú od užívateľa poskytnutie hesla, čísla bankovej karty alebo iných citlivých údajov. V praxi sa vyskytujú dva typy phishingu, hromadný, ktorý je adresovaný čo najväčšiemu množstvu príjemcov z ktorých štatisticky aspoň určité percento sa vždy nechá nachytať. Druhý typ je cielený phishing, kedy útočníci majú presne vytipovanú obeť a vďaka vedomostiam o zvykoch, záľubách obete vedia phishing prispôsobiť presne "na mieru". V súčastnosti existujú proti tomuto typu útokov nástroje od viacerých výrobcov, bohužiaľ proti sofistikovanejším útokom nemajú vysokú účinnosť.[9]

3.2 Útoky na webové aplikácie

Kategória útokov zameriavajúcich sa na webové aplikácie. Útočník vďaka chybám v zabezpečení webovej aplikácie alebo užívateľovho klienta dokáže do kódu podstrčiť svoj škodlivý obsah. Najznámejším útokom je Cross-Site Scripting, ktorý má ale mnoho podôb. V tejto sekcii sú popísané aj dalšie typy útokov, menovite Cross-Site Request Forgery 3.2.2, Clickjacking 3.2.4 a SQL injection 3.2.3.

3.2.1 Cross-Site Scripting

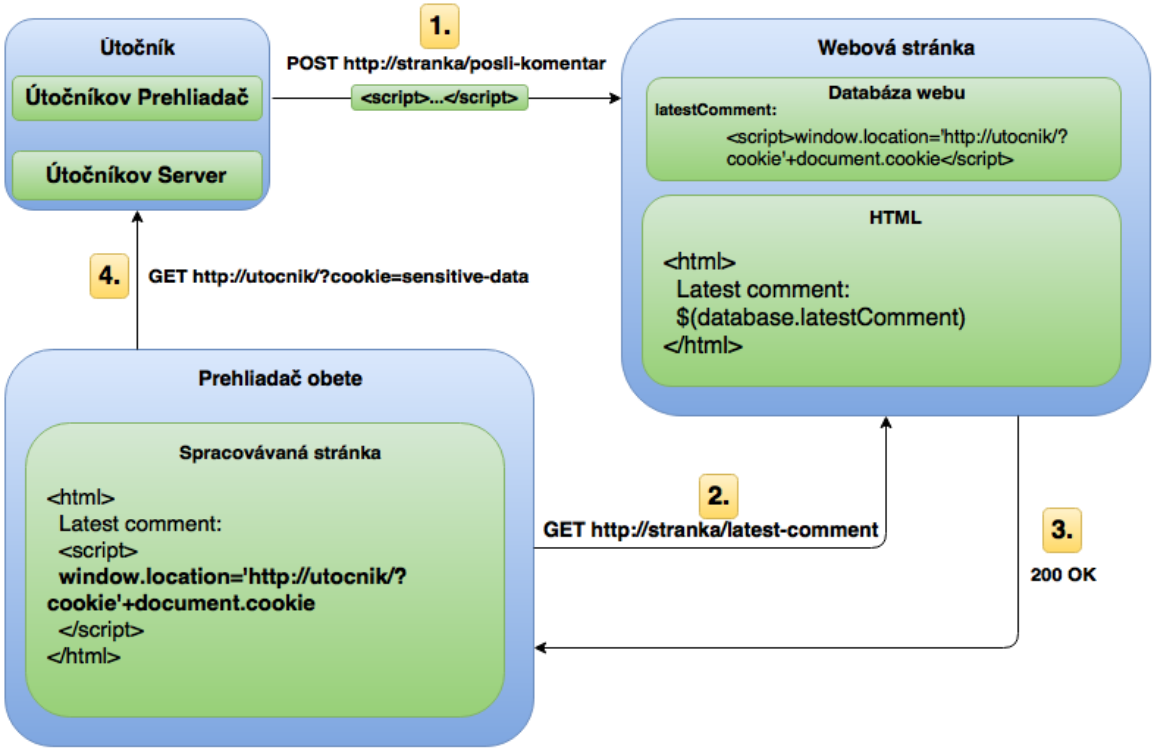
Základnou myšlienkou XSS je to, že útočník sa snaží umiestniť svoj škodlivý klientsky skript do dôveryhodných miest na server, odkial bude neskôr pri prehliadaní webu načítaný a interpretovaný v prehliadači obete.[21] Táto zraniteľnosť môže nastať, ak napríklad webová aplikácia prijíma akékoľvek vstupné dáta bez toho, aby ich vopred skontrolovala. Takáto kontrola sa nazýva sanitizing vstupu. Využitím XSS útočník neútočí priamo na svoju obeť, ale využíva slabo zabezpečené webové stránky ako prostriedok na doručenie škodlivého kódu k obeti. XSS útoky sa rozdeľujú podľa dvoch vlastností do štyroch kategórií. Jednak podľa toho, či škodlivý obsah je perzistentne uložený v svojom cieli alebo je iba dynamicky reflektovaný.

Non-persistent XSS Najčastejší typ útoku Cross-Site scripting, škodlivý kód nieje nikde uložený persistentne a preto nevyžaduje prístup k žiadnym diskovým médiam alebo databáze. Vyskytuje sa na stránkach dynamicky generujúcich odozvu klientovi.

Persistent XSS Princíp útoku je rovnaký s jediným rozdielom, že škodlivý kód je v tomto prípade uložený persistentne na cieľovom serveri, napríklad v databáze. Škodlivý kód sa teda spúšta viackrát, nezáleží ani na aktuálnej session v prehliadači.

Druhý sledovaný atribut je miesto, kde sa škodlivý kód vykonáva. To je zvyčajne u klienta, ale nájdu sa aj prípady, kedy je postihnutá server strana - webová aplikácia, napríklad modifikáciou štýlu stránky pri zmene definície príslušného HTML tagu. V každom XSS útoku sa nachádzajú traja účastníci: útočník, webová stránka a obeť. Nasleduje popis typického neperzistentného útoku, ktorý spôsobí krádež cookies súborov obete.[1] Znázornené na obrázku 3.1.

1. Útočník použije metódu POST na uploadovanie škodlivého kódu do databázy webovej stránky.
2. Obeť narazí na postihnutú webovú stránku a požiada teda o jej HTML kód.
3. HTML kód webovej stránky poslaný obeti obsahuje škodlivý reťazec uploadovaný od útočníka.
4. Webový prehliadač obete vezme prijaté dáta, ktoré považuje za dôveryhodné, začne ich parsing, pričom interpretovaním škodlivého reťazca uploaduje svoje súbory cookies na útočníkov pripravený server.



Obr. 3.1: Cross-Site Scripting útok

3.2.2 Cross-Site Request Forgery

Falšovanie požiadavku je typ útoku, ktorého cieľom je podviesť užívateľa a prinútiť ho tým k vykonaniu istej akcie na webovej aplikácii, v ktorej je aktuálne autentizovaný.[23, 15] Útočník tak môže využiť privilégia obete. Zvyčajne sa útokmi XSRF nekradnú dáta, kedže útočník nemá ako vidieť odozvu na vykonanú akciu. Najčastejšie sa teda jedná o akcie, ktoré menia stav systému, napríklad bankové transakcie, nákupy v e-shope, zmeny statických dát a podobne. Informácie o užívateľovi ako sú cookies, IP adresa, prihlasovacie údaje sa počas útoku nemenia, preto je takmer nemožné, aby webová aplikácia vedela rozoznať legitímnu požiadavku od tej škodlivej. Takéto donútenie k nevyžiadanej činnosti sa väčšinou dosahuje kliknutím na vopred vytvorený URL odkaz, ktorý sa vydáva za legitímny. Úspech týchto útokov preto vo veľkej miere záleží aj na sociálnom inžinierstve, kedže hlavné je presvečiť obeť kliknúť na útočníkov odkaz.

Príklad Uvažujme, že požiadavok o prevod na istý bankový účet sa vykonáva v jednoduchej forme dotazu.

GET http://bank.com/transfer.do?acct=USER& amount=100 HTTP/1.1

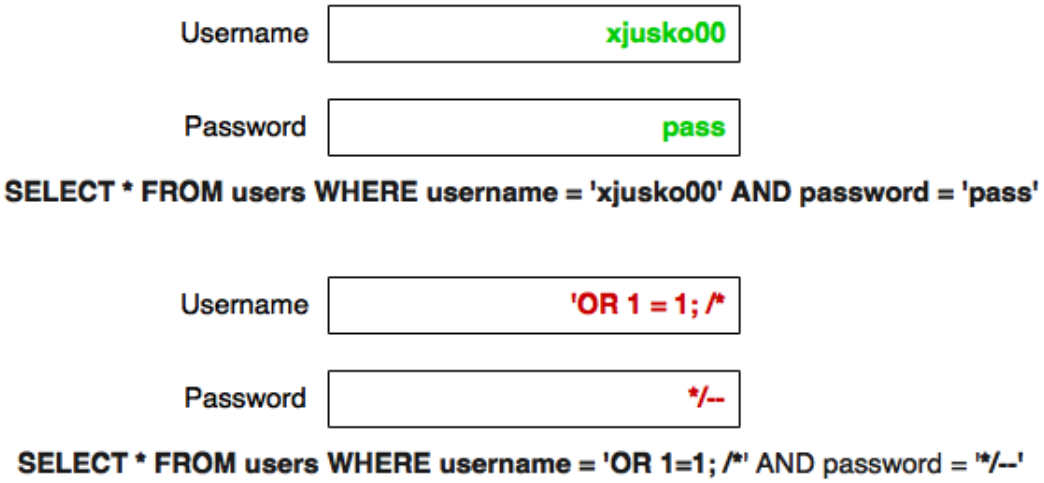
Útočník vytvorí odkaz v prospech svoj bankový účet a tento odkaz ukryje za element, ktorý vzbudzuje legitímnosť.

<a href="http://bank.com/transfer.do?acct=UTOCNIK& amount=100000» View my Pictures!

V prípade, že obeť nedávno pracovala s internet bankingom, má ešte uložené súbory cookies ktoré ju autorizujú vykonať po kliknutí túto transakciu.

3.2.3 SQL injection

SQL injection je podľa OWASP (Web Application Security Project) najviac využívanou zranitelnosťou webových aplikácií v roku 2013[26]. Tak ako názov napovedá, tento útok spočíva v injekcii SQL query príkazu do aplikácie cez ľubovoľný neošetrený vstupný bod.[25] SQL injection funguje iba v prípade, že vstupy od užívateľa nie sú dostatočne preverené sanitizing procesom. Po úspešnom útoku sa SQL príkaz automaticky vykoná a útočník tak dostáva spôsob na odhalenie citlivých dát z databázy alebo manipuláciu až vymazanie interných dát. Na obrázku 3.2 je znázornené, ako sa útočník dvuokáže dostať k všetkým prihlasovacím menám v databáze.



Obr. 3.2: SQL Injection cez prihlasovací formulár

3.2.4 Clickjacking

Tento útok spočíva v umiestňovaní priehľadných vrstiev ponad frekventované elementy na webových stránkach. Obeť tak nekliká na zamýšľaný obsah ale na útočníkove odkazy ktoré môžu viesť na úplne inú aplikáciu alebo doménu. Clickjacking sám o sebe nieje škodlivý a vedie jedine k otráveniu užívateľa. Hrozbou sa stáva ak odkazy vedú na útočníkom vopred pripravené webové stránky infikované napríklad s XSS alebo XSRF útokom.

3.3 Útoky na webový prehliadač a jeho súčasti

Dalšia kategória útokov, tentokrát využívajúca bezpečnostné nedostatky priamo v kóde webového prehliadača, jeho plug-inov a rozšírení. Pluginy a rozšírenia sú v dnešnej dobe veľmi populárne, štatisticky má až 30 percent prehliadačov Firefox nainštalovaný aspoň jeden takýto zásuvný model.[14] Zatiaľ čo webové prehliadače, vyvíjané niekedy aj stovkami profesionálov, vykazujú vysokú spoľahlivosť a bezpečnosť, ich rozšírenia sú častým problémom a príčinou mnohých zneužití. Problém spočíva najmä v tom, že vývojári rozšírení často nie sú expertmi na písanie bezpečného kódu, pričom práve ich kód prichádza do priameho kontaktu s obrovským množstvom prehliadaných webových stránok.[3] Stret rozšírení, ktoré

majú v systéme vysoké privilégia, s neoverenými zdrojmi z internetu častokrát končí tak, že útočník získava kontrolu nad cieľovým počítačom.

Plug-in vs. rozšírenie Na prvý pohľad vyzerá, že plugin aj rozšírenie je len rôzne pomenovanie pre jednu a tú istú vec. V skutočnosti je však rozdiel medzi týmito dvoma typmi softwaru veľký. Obe pracujú bok po boku s prehliadačom, pridávajú mu na funkcionalite alebo personalizujú zážitok z prehliadania. Plug-in je samostatný program s vlastým procesom využívajúcim NPAPI (viď kapitola 4.3), slúži na rendering súborov, ktoré v prehliadači nie sú podporované natívne (pdf, flash). Invokácia plug-inu prebieha explicitne pri narazení na špecifický typ média (Multipurpose Internet Mail Extension), následne sa hľadá vhodný plug-in, ktorý spracováva tento typ. Jeden plug-in môže typicky podporovať aj viacero MIME typov. Interakcia s plug-inom prebieha pomocou JavaScriptu.
Rozšírenie na druhej strane nemodifikuje webový obsah, ale iba užívateľské rozhranie prehliadača. Rozšírenia sú zvyčajne vyvíjané v jazyku JavaScript alebo XUL, nedisponujú vlastným procesom, pre vyvolanie rozšírenia nieje potrebná žiadna akcia, zvyčajne pracujú počas celej doby života prehliadača.

3.3.1 Škodlivé plug-iny a rozšírenia

Pluginy a rozsírenia do webových prehliadačov sú v operačnom systéme plnohodnotné programy s plným prístupom do pamäte a privilégiami vytvárať a ukončovať procesy. Zatiaľ čo v iných aplikáciach to nespôsobuje žiadne bezpečnostné riziká, rozšírenia prichádzajú do styku so širokou škálou nedôveryhodných webových stránok, ktoré môžu využiť ich zraniteľnosť. Rôzne štúdie rozšírení poukazujú na to, že pluginy dostávajú privysoké privilégia, ktoré v 88 percent prípadoch ani nevyužívajú naplno. [3] Najhorším príkladom je určite plug-in Adobe Flash obsluhujúci rendering vektorovej grafiky, animácií, hier atd. Do apríla 2016 je v oficiálnej databáze exploitov (CVE) zaznamenaných vyše 770 zraniteľností iba v tomto plug-ine.[8] Medzi typické exploity takýchto aplikácií patrí buffer overflow, integer overflow.

Buffer overflow Anomália v počítačových programoch, ktorá nastáva pri zápise do istej časti pamäte (buffer). Avšak, ak program neporovná veľkosť bufferu s veľkosťou zapisovaných dát, môže dôjsť k zápisu mimo alokovaný buffer a následne k porušeniu konzistencie dát, zastaveniu programu alebo aj k spusteniu vlastného škodlivého kódu. [22]

Integer overflow Nastáva ak rozsah interných číselných premenných nie je dostatočne kontrolovaný a pri sčítaní s externými hodnotami prekročí svoju maximálnu hodnotu a spôsobí neočakávané chovanie systému. Ak takáto premenná figuruje v inicializácii bufferov alebo pri kopírovaní pamäte, môže dôjsť k javu buffer overflow.[24]

Táto práca bude adresovať práve túto skupinu útokov, v implementovaných testoch bude simulovať škodlivú aktivitu ku ktorej by mohlo dôjsť v prípade získania kontroly nad pluginom. Takáto aktivita zahŕňa pristupovanie do pamäte, spúštanie a rušenie procesov, odchytávanie kláves, práca v registroch atd.

V súčastnosti ale vzrastá snaha obmedziť používanie pluginov tým, že prehliadače už samé o sebe podporujú funkcie ako flash rendering alebo zobrazovanie pdf súborov. Najráznejšie k tomuto kroku pristúpil prehliadač Chrome, ktorý zrušením podpory NPAPI

rozhrania úplne zamedzil používanie pluginov. Nástupca Exploreru, prehliadač Edge už takisto nepodporuje pluginy. Jediný z veľkých prehliadačov, Firefox, stále podporuje NPAPI pluginy a bude tomu tak minimálne do konca roka 2016.

Kapitola 4

Použité technológie

V teto kapitole sú popísané všetky technológie, frameworky a jazyky použité pri implementácii testov. Technológie boli vyberané so zreteľom na multiplatformnosť, stabilitu a zároveň jednoduchosť. Jadro práce, plugin a server, sú tvorené v jazyku C++ využívajúc win32 api. Webové stránky tvorené v HTML jazyku používajú štýlový predpis CSS a JavaScript na interakciu so zásuvným modulom.

4.1 Robot Framework

Robot Framework je generický nástroj na automatizované testovanie softwaru používaný pri akceptačnom testovaní. [29] Testy implementované v Robot Frameworku majú jednoduchú syntax ktorá je čitateľná aj pre ľudí neznalých programovacie jazyky. Prívetivá syntax je dosiahnutá využitím keyword-driven testovania.

Keyword–driven testing Jedná sa o metodológiu pri testovaní softwaru využívajúcu kľúčové slová pre vyjadrenie testovanej funkcionality. Za každým kľúčovým slovom sa ale skrýva postupnosť viacerých akcií definovaných v testovacích knižniciach. Kľúčové slovo sa teda dá predstaviť ako istá funkcia. Takéto oddelenie definície testov od definície kľúčových slov umožnuje zapojiť do testovania aj ľudí ktorí neovládajú programovací jazyk. Rovnako je prínos tejto metodológie aj v možnosti definovať zložitejšie kľúčové slová pomocou iných kľúčových slov. Tým sa dá ľahko zvyšovať abstrakcia testovania.

Robot Framework je implementovaný v jazyku Python. Knižnice s definíciami kľúčových slov sa dajú vytvárať v jazykoch Python a Java. Samotný Robot Framework má širokú škálu vstavaných knižníc s kľúčovými slovami. Po spustení testov má tento framework aj dobre prepracovaný formát výstupných informácií. Po každej sade testov sa vygeneruje HTML report, kde sú podrobné informácie o každom jednotlivom testovacom prípade.

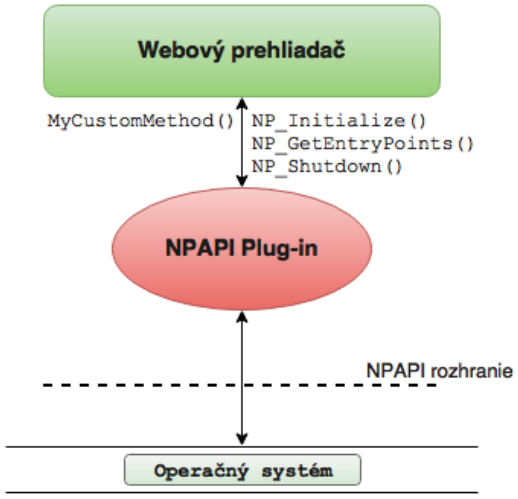
4.2 FireBreath framework

FireBreath je framework umožnujúci jednoduché vytváranie pluginov do prehliadačov šírený pod licenciou LGPL. Je multiplatformný z hľadiska operačného systému aj webového prehliadača. [27] Podporuje vývoj na operačných systémoch Windows, Linux aj Mac OS. Vytvorené pluginy využívajú NPAPI rozhranie a zároveň ActiveX, takže sú použíteľné na širokej škále webových prehliadačoch. Vývojárovi poskytuje dopĺňanie jednotlivých metód,

ktoré následne bude plugin poskytovať prehliadaču vo forme JavaScript volaní. Implementácia týchto metód je umiestnená v súboroch *myplugin*API.cpp a *myplugin*API.hpp. Vývojár je takto ušetrený nutnosti zoznamovať sa so samotným NPAPI rozhraním. Framework je napísaný v jazyku C++, na platforme Windows sa kompiluje vo vývojovom prostredí Visual Studio 2005 až 2013. Je aktívne udržiavaný úzkou komunitou vývojárov, prešiel už mnohými vylepšeniami a aktuálna verzia 2.0 sa snaží adaptovať do post-NPAPI sveta prehliadačov pomocou natívneho zasielania správ medzi operačným systémom a plug-inom.

4.3 NPAPI

Netscape Plug-in API je rozhranie pre programovanie pluginov do webových prehliadačov. NPAPI bolo navrhnuté pre jednotné sprístupnenie služieb operačného systému, preto sú pluginy vytvorené touto technológiou prenositeľné. Pôvodne bola táto architektúra navrhnutá v roku 1995 pre zásuvné moduly prehliadača Netscape Navigator. Technológiu NPAPI prevzala aj väčšina nových prehliadačov, podporujú ju všetky s výnimkou Internet Exploreru, ktorý používa ActiveX. NPAPI rozhranie umožňuje modifikovať lokálne súbory a využívať systémové volania. Každý plugin postavený na NPAPI architektúre definuje typ súborov, ktoré dokáže obsluhovať.



Obr. 4.1: NPAPI rozhranie

4.4 HTML

HyperText Markup Language (HTML) je značkovací jazyk pre tvorbu webových stránok. K jednoduchému textu pridáva štruktúru a sémantiku. HyperText predstavuje možnosť odkazovať sa na dalšie dokumenty. Štandard spravuje World Wide Web Consortium (W3C) [34], aktuálna verzia je HTML 5.

4.5 Cascade Style Sheets

Pretože jazyk HTML sám o sebe je relatívne jednoduchý jazyk na zakódovanie webových stránok, používajú sa rôzne pomocné technológie, ktoré zlepšujú vizuálny vzhľad a tým aj celkový zážitok užívateľa. Cascading Style Sheets (CSS) dovoľuje programátorovi pridať informácie o vzhľade stránky bez zásahu do kostry dokumentu definovanej v HTML jazyku. [33] Oddelenie definície vzhľadu od obsahu stránky umožňuje takisto uplatniť rozdielny štýl pre rozdielne renderovacie metódy. Napríklad webová stránka sa zobrazí rozdielne na obrazovke ako vytlačená na papieri. Takisto je možné, aby užívateľ uplatňoval na všetky stránky explicitne svoj predpis štýlu. To je prínosné pre ľudí trpiacim daltonizmom alebo inou poruchou zraku.

4.6 JavaScript

Dynamicky typovaný, interpretovaný programovací jazyk ktorý je základným prvkom dnešných webových stránok. [10] Pomocou JavaScript kódu je možné tvoriť interaktívne webové stránky a animácie. Narozdiel od jazyka PHP, kde sa kód vykonáva na strane servera, JavaScript skript sa interpretuje u klienta až po stiahnutí. Parsovanie a následné vykonanie ľubovolného kódu tretích strán na lokálnom počítači prináša mnoho bezpečnostných rizík spomenutých v kapitole 3.2.

4.7 Jazyk C++ s Win32 API

Imperatívny, objektovo-orientovaný programovací jazyk vyvinutý z jazyka C. Oproti jazyku C sa snaží implementovať vyššiu abstrakciu, jednoduchšiu modularitu a objektové paradigma. Jazyk C++ bol štandardizovaný v roku 1998. V prostredí Windows je v tomto jazyku možné využívať Win32 volania. Windows API je aplikačné programové rozhranie ponúkajúce základné systémové funkcie dostupné na operačnom systéme Windows. Je implementované na rôznych platformách, preto existujú rôzne verzie rozlišujúce šírku slova, v ktorej procesor pracuje - (win16, win32, win64). V tejto práci bolo využité Win32 API.

Kapitola 5

Návrh a implementácia

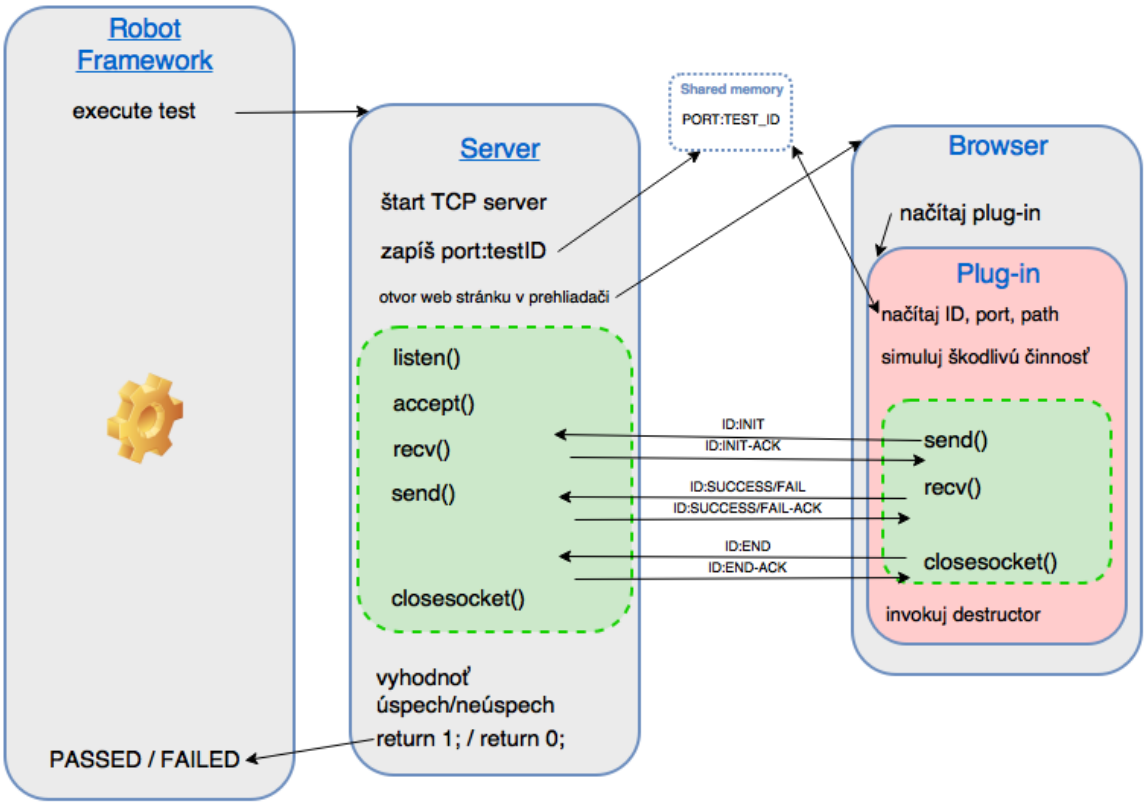
Ako bolo načrtnuté v minulých kapitolách, moderné webové prehliadače už disponujú vysokou mierou bezpečnosti, zásuvné moduly už bežia výlučne v sandbox prostredí. Tieto technológie umožnujúce bezpečnejšie prehliadanie sú však stále relatívne novinkou. Je všeobecne známe, že mnoho užívateľov odkladá upgrade softwaru a rovnako to platí aj pre webové prehliadače. Podľa globálnych štatistík globalcounter, v minulom roku používalo ešte stále vyše 20% užívateľov Chrome verzie nižšej ako 45.0, 2.41% Firefox a 2.45% Internet explorer 8. [20] Všetky tieto verzie webových prehliadačov ešte podporujú pluginy s bezpečnostne nevyhovujúcou architektúrou NPAPI. Tieto čísla nie sú zanedbateľné a práve preto táto práca adresuje testovanie zraniteľností spojených s NPAPI pluginmi. V tejto kapitole je podrobne popísaná architektúra implementovaných testov, zoznam použitých technológií a nástrojov.

5.1 Architektúra

Troma základnými prvkami každého vykonaného testu sú škodlivý plugin, server obsluhujúci plugin a Robot framework, ktorý spúšta jednotlivé testy. Ako je vidieť na schéme X, Robot framework je na najvyššej úrovni a beží od spustenia testu až po koniec. Beh jedného testu zahrňuje mnoho akcií ktoré sú chronologicky popísané v nasledujúcich bodoch.

- Každý test sa začína v Robot Frameworku kľúčovým slovom ExecuteTest a 5 ciferným parametrom identifikuj[cim test.
- Po spustení, program server ihneď otvára v novom vlákne TCP schránku čakajúcu na príchádzajúcu komunikáciu zo strany pluginu.
- Server otvára pripravenú webovú stránku v žiadanom prehliadači. Táto webová stránka má v sebe zakomponovaný media type prvok obsluhovaný plug-inom. Z toho vyplýva že plug-in sa automaticky vyvolá prehliadačom.
- Plugin naviaže komunikáciu so serverom vyvolaním metódy *.launched()*.
- Plugin simuluje škodlivú činnosť vyvolaním jednej zo svojích metód, vyhodnotí či sa škodlivá aktivita podarila spustiť a príslušne odpovedá serveru *SUCCESS/FAIL*.
- Server príjme odpoveď o úspešnosti škodlivej činnosti a končí s príslušným návratovým kódom.

- Robot framework vyhodnotí návratový kód serveru a rozhodne o úspešnosti testu.



Obr. 5.1: Test flow

5.2 Plug-in

Hlavným prvkom tejto práce je multiplatformný plug-in do webových prehliadačov implementovaný pomocou architektúry NPAPI. Okrem povinných 16 metód, ktoré musí poskytovať plugin prehliadaču k správnemu fungovaniu, bolo implementovyných dalších 7. Dve z nich sú pomocné metódy ktoré obsluhujú komunikáciu so serverom a dalšie 5 pri vyvolaní simulujú istú škodlivú činnosť. Plugin bol implementovaný pomocou frameworku Firebreath. Nasleduje popis jednotlivých metód, ktoré poskytuje ako JavaScript API.

launched() Metóda, ktorá sa volá hneď na začiatku činnosti pluginu. Z pamäte zdieľanej so serverom prečíta číslo portu, identifikátor testu a absolútnu cestu k priečinku kde sú umiestnené potrebné súbory. Následne nadväzuje TCP komunikáciu so serverom iniciačným reťazcom <seqnum>.<TESTID>:INIT

finalize() Metóda volaná na konci života pluginu, stará sa o dealokáciu zdrojov, odoslanie ukončovacieho reťazca <seqnum>.<TESTID>:END a bezpečné ukončenie pluginu.

RunGeneralTest(string TestExe) Táto metóda implementuje spúšťanie binárie, ktoré vykonávajú škodlivú činnosť. Konkrétny spustitelný súbor je definovaný parametrom funkcie. Úspešnosť sa vyhodnocuje na základe návratového kódu. Do tejto kategórie útokov patrí práca so súborovým systémom a registrami. Simuluje sa otváranie, čítanie, zapisovanie, mazanie a premenovanie.

keylogger() Metóda simulujúca odchytávanie stisnutých kláves. Jedine v tomto prípade je úspešnosť prevedenia vyhodnocovaná v programe Server, ktorý jednak simuluje stisnuté klávesy a zároveň kontroluje súbor, do ktorého ich keylogger zapisuje. Samotné odchytávanie kláves je implementované vytvorením neviditeľného okna, na ktoré je pripojený WindowHook. Takéto okno potom dostáva od systému všetky stisnuté klávesy aj v prípade, že na okno nieje zaostrený pohľad. Implementácia keyloggeru sa skladá z viacerých podporných funkcií a metód. Funkcia handlekeys je nastavená ako obsluha kláves, ktorá sa automaticky vyvolá pri prijatí signálu klávesy. Na zaistenie toho, aby odchytávanie kláves neblokovalo proces pluginu a prehliadača, je vytvorené vlákno v metóde keylogger_thread.

ReadMemory(string TestExe) Simuluje čítanie z pridelenej pamäťe iného procesu. Najprv spúšťa hosťujúci proces ktorý zapíše premennú na istú adresu, túto adresu zapíše na štandardný vstup metóde ReadMemory(). Následne sa snaží prečítať spomínanú premennú pomocou programu definovanom vstupným argumentom *TestExe*.

WriteMemory(string TestExe) Vykonáva zapisovanie do pamäte pridelenej inému procesu. Takisto na začiatku spúšťa hosťujúci proces, ktorý vracia adresu zo svojho pamäťového priestoru. Následne pomocou programu definovaého v argumente sa na túto adresu snaží zapísovať.

terminateProcess(string TestExe) Generická metóda, ktorá spúšťa množinu testov s rovnakou architektúrou. Obsluhuje činnosti spojené s otváraním, vytváraním a ukončovaním procesov a vlákien. Postup simulácie týchto činností je podobný ako pri práci s pamäťou s tým rozdielom, že po spustení hosťujúceho procesu sa nezavolá jednoduchá funkcia ale spustí sa další proces. Tento následný proces sa už snaží vykonať škodlivú činnosť v hosťujúcom procese.

5.2.1 Trieda HOST

V implementácii pluginu sa mnohokrát vyskytuje spúšťanie procesu s ktorým sa následne komunikuje pomocou štandardného vstupu a výstupu. Pre jednoduchšiu a prehľadnejšiu prácu s takýmito child procesmi bola vytvorená trieda HOST. Táto trieda má na starosti inicializovať komunikačné rúry (pipes) a prepojiť štandardný výstup pluginu so štandardným vstupom synovského procesu a príslušne aj vstup pluginu s výstupom spúšťaného procesu. Ďalej má trieda HOST implementované členské metódy ReadFromPipe, WriteToPipe ktoré obsluhujú komunikáciu a GetExitCode, ktorá čaká na ukončenie procesu a následne ukladá jeho návratový kód.

5.2.2 Trieda COMM

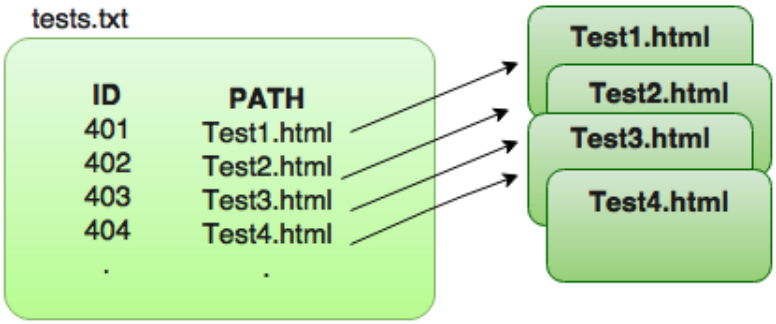
Ďalšia samostatná trieda je zodpovedná za oblushu TCP komunikácie so serverom. Konštruktorom triedy sa nastavia potrebné premenné - číslo portu a string na odoslanie. Me-

tódou .communicate() sa reťazec odošle a čaká sa na odpoveď s potvrdením ACK. Komunikácia prebieha vždy na adrese localhost.

5.3 Control Server

Samostatný C++ program ktorý obsluhuje činnosť pluginu. Zohráva dôležitú úlohu pri parsovaní údajov o spustenom teste. Pracuje ako mediátor medzi Robot frameworkom a škodlivým pluginom.

Argument Ako argument program prijíma 5 ciferné číslo ktoré prvými dvoma ciframi určuje aký webový prehliadač sa má testovať. Zvyšné tri cifry určujú jedinečné číslo testu, teda aká škodlivá činnosť sa bude testovať. To, aká škodlivá činnosť sa má vykonať je definované v kóde spúštanej webovej stránky. Preto má každý test svoj vlastný html súbor, kde v tagu *<script>* je invokovaná príslušná JavaScript metóda pluginu. Príslušnosti identifikátorov testov k súborom sú zapísané v súbore tests.txt.



Obr. 5.2: Identifikácia testu

Program je konštruovaný v jednej triede CNTR. V konštruktore sa zo vstupného argumentu parsuje identifikátor testu a typ webového prehliadača ktorý sa má použiť. Zo súboru tests.txt sa zistí cesta k webovej stránke. Po zistení všetkých potrebných údajov o teste (ID testu, prehliadač, cesta k html súboru) otvára metóda Launch_Browser() webovú stránku v žiadanom prehliadači. Paralelne s tým, v samostatnom vlákne sa metódou Launch_Server() otvára TCP schránka, ktorá očakáva komunikáciu od pluginu. Správy od pluginu sa spracovávajú v metóde Parse_Response() a na všetky sa následne odpovedá ack reťazcom. Po obdržaní správy o úspešnosti alebo neúspešnosti škodlivej činnosti program Server končí s príslušným návratovým kódom, 0 - úspech, 1 - neúspech.

5.4 Nadviazanie spojenia

Zahájenie komunikácie medzi pluginom a serverom nastáva hneď po načítaní plug-inu vo webovom prehliadači. Plugin je však spúštaný automaticky pri narazení na príslušný media typ ktorý obsluhuje, preto server nemá možnosť ako by pluginu priamo zaslal potrebné informácie a to identifikátor testu, komunikačný port a cestu k aktuálnemu priečinku. Tak isto bolo potrebné, aby každý spustený test komunikoval na inom porte. Program server ešte pred spustením webového prehliadača s pluginom otvára TCP komunikáciu na porte

pridelenom od systému Windows. Nasledujúci voľný port sa prideľuje automaticky ak sa v kóde zavolá funkcia bind() s nulovou hodnotou portu. Plugin je v komunikačnom postavení klient, ktorý iniciuje komunikáciu s čakajúcim serverom. Preto je potrebné aby získal číslo portu, na ktorom čaká server. To je dosiahnuté pomocou vytvorenia zdieľanej pamäte s identifikátorom, ktorý poznajú aj server aj plugin. Tam server zapíše reťazec v tvare

<testID>:<port no.>:<CWD_path>

Z neho už plugin dokáže parsovať všetky potrebné informácie k zahájeniu komunikácie.

5.5 Komunikačný protokol

Pre komunikáciu medzi serverom a pluginom bol navrhnutý jednoduchý komunikačný protokol. Každý odoslaný reťazec je v tvare

<seqnum>.<testID>:<message>

pričom dalšie prídavné informácie môžu byť pripojené na koniec. Spojenie vždy nadväzuje ako prvý plugin, server vždy odpovedá potvrdzujúcim reťazcom v tvare

<seqnum>.<testID>:ACK

Zo strany pluginu bolo implementovaných niekoľko správ popísané v nasledujúcej časti.

INIT týmto reťazcom sa zo strany pluginu inicializuje komunikácia, server odpovedá s dodatočnou informáciou o ceste k priečinku, kde sa nachádza spustiteľný súbor serveru. To je potrebné aby plugin vedel pristupovať k spoločným súborom, najmä pri vykonávaní odchytávaní kláves.

INIT_FAILED ak sa plug-in v prehliadači nenačíta úspešne, server obdrží správu a test sa končí neúspechom

SUCCESS správa indikujúca úspešnosť prevedenia škodlivej činnosti pluginom

FAILURE správa indikujúca neúspech prevedenia škodlivej činnosti pluginom

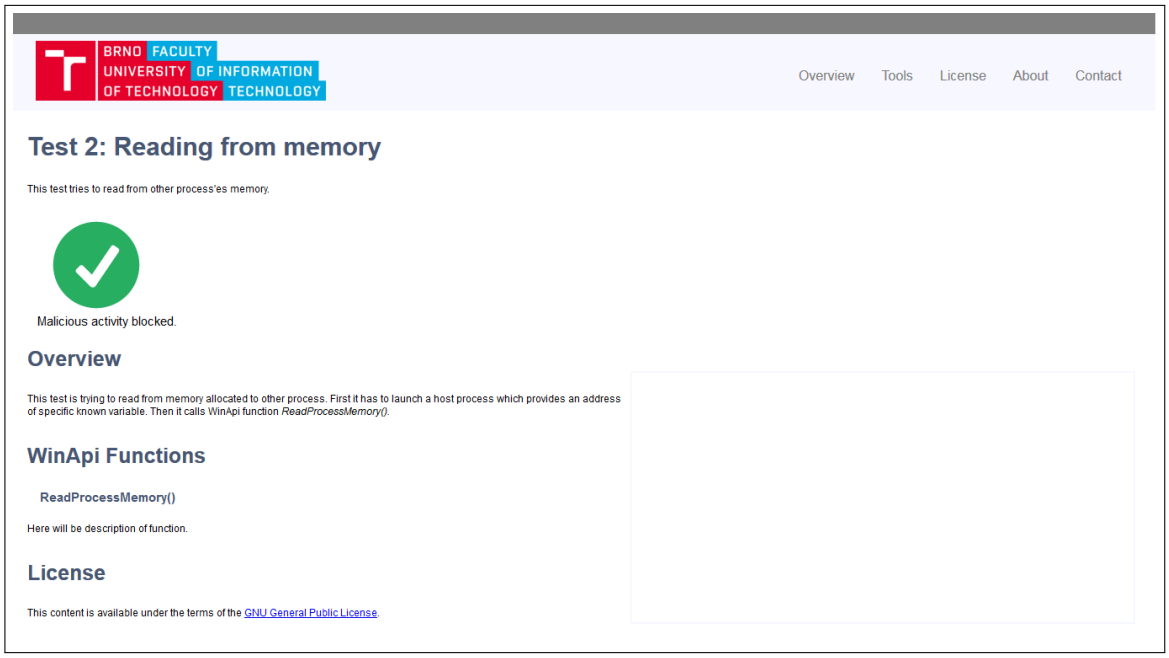
KEYLOG_START informuje server o začiatku odchytávania kláves, to vyvolá na strane servera funkciu, ktorá overuje úspešnosť tohto útoku. Najprv sa simuluje stlačenie kláves winapi funkciou SendInput() a následne sa kontroluje súbor, do ktorého má keylogger zapisovať zaznamenané klávesy.

END týmto reťazcom plugin indikuje koniec svojej činnosti, komunikácia sa následne ukončuje a server môže predať návratovú hodnotu.

Kapitola 6

Testovanie a analýza výsledkov

V záverečnej fáze tejto práce ostávalo implementovaný systém spustiť a overiť tak zabezpečenie voči útokom na rôznych platformách. Boli vytvorené testovacie sady pre tri prehliadače Firefox, Chrome a Explorer. Jediný prehliadač Firefox bol otestovaný v svojej aktuálne najnovšej verzii 45. Prehliadač Chrome bol použitý v zastaralej verzii 23, ktorá podporuje NPAPI plug-iny a Explorer bol použitý vo verzii 9. Testy boli spúštané na operačných systémoch Windows 7 a Windows XP a zároveň v rôznych situáciách podľa nasadenia bezpečnostných aplikácií. Spúštané boli s aktívnou anti-exploit ochranou spoločnosti TrustPort, ďalej bez akéhokoľvek zabezpečenia a nakoniec s ochranou anti-exploit od firmy Malwarebytes. Anti–exploit Malwarebytes sa ukázal ako nevhodný nástroj na ochranu, kedže nezabránil žiadnemu z implementovaných útokov a preto nebude daľej analyzovaný. Simulovanými činnosťami by bol útočník v reálnej situácii schopný prebrať úplnú kontrolu nad cieľovým počítačom, pristupovať k dátam, vykonávať svoj kód alebo sledovať činnosť obete. Preto sa tieto útoky dajú zaradiť medzi veľmi závažné. Implementované testy sa vyznačujú vysokou prehľadnosťou a prívetivým užívateľským rozhraním. Každý test spúšťa webovú stránku s plugin containerom a zároveň krátkym popisom, akú činnosť test simuluje. Rovnako je po vykonaní testu interaktívnou formou znázornený výsledok s vysvetlením, či sa škodlivá činnosť podarila alebo nie. Znázornené na obrázku 6.1. V tejto kapitole budú predvedené dosiahnuté výsledky z testovania na systéme Windows XP, Windows 7. V oboch prípadoch s ochranou TrustPort anti–exploit a bez.



Obr. 6.1: Vizualizácia výsledku.

6.1 Priebeh testovania

Ná základe toho, že implementované útoky sú iba simulované, t.j. reálne neohrozujú chod operačného systému, testovanie nemuselo prebiehať vo virtualizovanom prostredí. Pre zaistenie toho, aby výsledky boli naozaj relevantné, bola každá testovacia sada spustená trikrát a to pri rôznom stupni zaťažení procesora. To však na výsledky nemalo žiadny vplyv. Spustenie testovacej sady prebieha v prostredí príkazoveho riadku cmd príkazom *robot Firefox-Tests.rst* (pre testovanie na prehliadači Firefox) a výsledky je možné zapisovať priebežne priamo z terminálu alebo po skončení posledného testu zo súboru *report.html*.

6.2 TrustPort Internet Security

Produkt TrustPort Internet Security poskytuje užívateľovi kompletnú ochranu jeho počítača a dát proti útokom a iným hrozbám. Okrem tradičnej antivir a antispyware ochrany poskytuje aj ochranu na sieťovej úrovni monitorovaním e-mailov a sieťovej prevádzky so snahou zamedziť vírusom ešte pred tým ako sa dostanú na pevný disk. Do balíka kompletnej ochrany patrí aj inteligentný firewall ktorý dokáže samostatne kategorizovať nové aplikácie podľa dôveryhodnosti, anti-exploit schopný detekovať *zero-days* útoky, rodičovský zámok ktorý blokuje webové stránky podľa obsahu a nástroj na ochranu USB pamäťových médií. [32] Tento produkt je dostupný zadarmo k stiahnutiu v trial verzii. [31] Podporuje operačné systémy Windows XP a novšie.

Obr. 6.2: TrustPort Internet Security 2016.

6.3 Simulované činnosti

V práci sa podarilo implementovať 31 testov simulujúcich širokú škálu útokov na cieľový počítač. Väčšina testov spočíva v spustiteľných súboroch dodaných firmou TrustPort, ktoré sú spúšťané z plugina a následne je vyhodnocovaná ich úspešnosť. Úspešnosť prevedenia sa vyhodnocuje najmä podľa návratového kódu programu vykonávajúceho simulovanú činnosť. V sade sa vyskytuje skupina testov s prívlastkom *Ask*, to je varianta testu, kedy sa pri spustení otvára užívateľské okno ktoré informuje o útoku a zároveň vyžaduje ďalšiu akciu od užívateľa. Útoky by sa podľa toho, akú časť systému napádajú, kategorizovať do piatich skupín.

6.3.1 Útoky na proces

Do tejto skupiny spadá 9 testov, ktoré rôznymi spôsobmi pracujú s iným procesom v operačnom systéme. Jedná sa o vytváranie, ukončovanie, otváranie procesov a vytváranie vlákna v cudzom procese. Patrí sem aj volanie príkazoveho riadku s príkazom priamo z kódu plug-inu pomocou win api funkcie *CreateProcess()*. Princíp týchto testov spočíva v spustení samostatného host procesu, ktorý sa následne dalším procesom snažíme modifikovať.

- *TestCreateCase* – Spustiť nový proces
- *CMDproc* – Spustiť cmd príkaz
- *TestTerminateCase* – Ukonči proces v systéme
- *TestOpenCase* – Otvor proces v systéme
- *TestThreadCase* – Vytvor vlákno v cudzom procese

6.3.2 Útoky na operačnú pamäť

Ďalšia skupina útokov sa zaoberá útokmi na operačnú pamäť cudzieho procesu. Opäť je najprv potrebné spustiť host proces, ktorý po spustení vracia adresu svojho adresového

priestoru. Následne sa z neho snažíme čítať alebo doň zapisovať. Týchto testov je celkom 6, pretože sú implementované aj vo verzii *Ask* a *Dependency*.

- *TestReadCase* – Čítanie z operačnej pamäte
- *TestWriteCase* – Zapisovanie do operačnej pamäte

6.3.3 Keylogger

Samostatná kategória obsahujúca jediný test. Jedná sa o zložitejší prípad útoku kde sa využíva Windows Hook pripojený na handle plug-inu. Tým pádom proces pluginu dostáva správy o stlačených klávesách bez ohľadu na to, ktoré okno je označené ako aktívne. Tento test vytvára dočastný súbor *log.txt* ktorý sa neskôr kontroluje a vyhodnocuje sa úspešnosť odchytávania kláves.

6.3.4 Útoky na súborový systém a registre

Testy ktoré simulujú prácu so súborovým systémom, vytváranie a modifikáciu súborov na disku a zároveň testy na modifikáciu Windows registrov by sa momentálne dali zaradiť do jednej kategórie, pretože aktuálna verzia poskytnutých spustiteľných súborov ešte nefunguje správne a preto sa táto aktivita anti–exploitom neodchytáva. Využitie týchto testov sa uplatní až po dokončení testovacích programov vo firme TrustPort.

- *TestFileReadCase/TestRegReadCase* – Čítanie zo súboru/registra
- *TestFileWriteCase/TestRegWriteCase* – Zapisovanie do súboru/registra
- *TestFileCreateCase/TestRegCreateCase* – Vytváranie súboru/registra
- *TestFileOpenCase/TestRegOpenCase* – Otváranie súboru/registra
- *TestFileRenameCase/TestRegRenameCase* – Premenovanie súboru/registra
- *TestFileDeleteCase/TestRegDeleteCase* – Zmazanie súboru/registra

6.4 Platforma Windows XP

Operačný systém Windows XP je od roku 2014 už neudržiavaný a zastaralý. [16] Jeho zabezpečenie sa ukázalo ako veľmi slabé s 0% úspešnosťou zabrániť útokom z testovacej sady. S nainštalovaným programom TrustPort sa zachytilo vykonanie 13 testov. Práve tieto testy boli z tých viac závažných, jednalo sa o pristupovanie do operačnej pamäte cudzích procesov, spúšťanie alebo modifikáciu cudzích procesov a prácu s registrami. V tabuľke 6.1 sa nachádza kompletný report o úspešnosti jednotlivých testov.

Windows XP						
TEST	Bez ochrany			TrustPort anti-exploit		
	Firefox	Explorer	Chrome	Firefox	Explorer	Chrome
Spustiť nový proces	✗	✗	✗	✓	✓	✓
Čítať z pamäte	✗	✗	✗	✓	✓	✓
Zapisovať do pamäte	✗	✗	✗	✓	✓	✓
Spustiť CMD príkaz	✗	✗	✗	✓	✓	✓
Ukončiť proces	✗	✗	✗	✓	✓	✓
Ukončiť proces (Ask)	✗	✗	✗	✓	✓	✓
Otvor proces	✗	✗	✗	✗	✗	✗
Otvor proces (Ask)	✗	✗	✗	✗	✗	✗
Spustiť nové vlákno v procese	✗	✗	✗	✗	✗	✗
Spustiť nové vlákno v procese (Ask)	✗	✗	✗	✗	✗	✗
Odchytávanie kláves	✗	✗	✗	✗	✗	✗
Vytvoriť súbor*	✗	✗	✗	✗	✗	✗
Otvoriť súbor*	✗	✗	✗	✗	✗	✗
Čítať zo súboru*	✗	✗	✗	✗	✗	✗
Zapisovať do súboru*	✗	✗	✗	✗	✗	✗
Premenovať súbor*	✗	✗	✗	✗	✗	✗
Vymazať súbor*	✗	✗	✗	✗	✗	✗
Vytvoriť register*	✗	✗	✗	✗	✗	✗
Otvoriť register*	✗	✗	✗	✗	✗	✗
Čítať z registra*	✗	✗	✗	✗	✗	✗
Zapisovať do registra*	✗	✗	✗	✗	✗	✗
Premenovať register*	✗	✗	✗	✗	✗	✗
Vymazať register*	✗	✗	✗	✗	✗	✗
Čítať z registra (Dep)*	✗	✗	✗	✗	✗	✗
Zapisovať do registra (Dep)*	✗	✗	✗	✗	✗	✗
Čítať zo súboru (Dep)*	✗	✗	✗	✗	✗	✗
Zapisovať do súboru (Dep)*	✗	✗	✗	✗	✗	✗
Čítať z pamäte (Dep)	✗	✗	✗	✓	✓	✓
Zapisovať do pamäte (Dep)	✗	✗	✗	✓	✓	✓
Čítať z pamäte (Ask)	✗	✗	✗	✓	✓	✓
Zapisovať do pamäte (Ask)	✗	✗	✗	✓	✓	✓

Tabuľka 6.1: Testovanie na Windows XP
Legenda: ✓– neúspešný útok, ✗– úspešný útok, * – útok nepodporovaný TrustPort ochranou

6.5 Platforma Windows 7

Microsoft Windows 7 je momentálne najpoužívanejší operačný systém na svete. Skoro polovica bežných užívateľov používa práve tento systém a preto je ešte stále udržiavaný vývojármi pravidelnými aktualizáciami. Jeho bezpečnosť je oproti staršej verzii XP lepšia, len samotný operačný systém je schopný zabrániť nelegálnemu prístupu do registrov. To v

našej testovacej sade znamená stále iba vyše 10% ochranu. S použitím systému anti–exploit je situácia rovnaká ako na verzii XP, a to 13 zachytených testov. V tabuľke 6.2 je výpis úspešnosti jednotlivých testovacích prípadov.

Windows 7						
TEST	Bez ochrany			TrustPort anti-exploit		
	Firefox	Explorer	Chrome	Firefox	Explorer	Chrome
Spustiť nový proces	✗	✗	✗	✓	✓	✓
Čítať z pamäte	✗	✗	✗	✓	✓	✓
Zapisovať do pamäte	✗	✗	✗	✓	✓	✓
Spustiť CMD príkaz	✗	✗	✗	✓	✓	✓
Ukončiť proces	✗	✗	✗	✓	✓	✓
Ukončiť proces (Ask)	✗	✗	✗	✓	✓	✓
Otvor proces	✗	✗	✗	✗	✗	✗
Otvor proces (Ask)	✗	✗	✗	✗	✗	✗
Spustiť nové vlákno v procese	✗	✗	✗	✗	✗	✗
Spustiť nové vlákno v procese (Ask)	✗	✗	✗	✗	✗	✗
Odchytávanie kláves	✗	✗	✗	✗	✗	✗
Vytvoriť súbor*	✗	✗	✗	✗	✗	✗
Otvoriť súbor*	✗	✗	✗	✗	✗	✗
Čítať zo súboru*	✗	✗	✗	✗	✗	✗
Zapisovať do súboru*	✗	✗	✗	✗	✗	✗
Premenovať súbor*	✗	✗	✗	✗	✗	✗
Vymazať súbor*	✗	✗	✗	✗	✗	✗
Vytvoriť register*	✓	✓	✓	✓	✓	✓
Otvoriť register*	✓	✓	✓	✓	✓	✓
Čítať z registra*	✗	✗	✗	✗	✗	✗
Zapisovať do registra*	✗	✗	✗	✗	✗	✗
Premenovať register*	✗	✗	✗	✗	✗	✗
Vymazať register*	✓	✓	✓	✓	✓	✓
Čítať z registra(Dep)*	✗	✗	✗	✗	✗	✗
Zapisovať do registra(Dep)*	✗	✗	✗	✗	✗	✗
Čítať zo súboru(Dep)*	✗	✗	✗	✗	✗	✗
Zapisovať do súboru(Dep)*	✗	✗	✗	✗	✗	✗
Čítať z pamäte(Dep)	✗	✗	✗	✓	✓	✓
Zapisovať do pamäte (Dep)	✗	✗	✗	✓	✓	✓
Čítať z pamäte (Ask)	✗	✗	✗	✓	✓	✓
Zapisovať do pamäte (Ask)	✗	✗	✗	✓	✓	✓

Tabuľka 6.2: Testovanie na Windows 7
Legenda: ✓– neúspešný útok, ✗– úspešný útok, * – útok nepodporovaný TrustPort ochranou

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo zoznámiť sa s nástrojmi na automatizované testovanie a zároveň so základnými technikami útokov na webové prehliadače. Následne na základe prevedeného návrhu bola dalším krokom implementácia automatizovaných testov, ktoré preverujú efektivitu bezpečnostných aplikácií. Postup práce by sa dal rozdeliť do troch častí. V prvej bolo potrebné sa zoznámiť so základmi automatizovaného testovania, princípmi fungovania webových prehliadačov a útokov na ne. Potom nasledoval návrh, ako sa bude testovanie implementovať, výber množiny útokov ktoré sa budú testovať a spôsob vyhodnocovania výsledkov. Spôsob fungovania jednotlivých modulov medzi sebou sa počas implementácia viac krát pozmenil, aby sa dospelo k najviac generickému riešeniu s možnosťou neskoršieho rozšírenia sady testov. V poslednej fáze práce prebiehalo samotné testovanie v spolupráci so spoločnosťou TrustPort a vyhodnotenie testov.

Výsledkom práce je trojica komunikujúcich programov, plug-in spúšťajúci škodlivú aktivitu, obsluhujúci server a Robot Framework. Spolu vytvárajú sadu 31 testov simulujúcich škodlivé činnosti spúšťané z NPAPI plug-inu ktoré sa často vyskytujú v reálnom svete. Testovaním sme zistili, že pri NPAPI plug-inoch nezáleží na spúšťajúcom webovom prehliadači, pretože plug-in beží ako samostatný proces. Ďalším zistením bolo, že používanie anti-exploitu spoločnosti TrustPort dokáže zachytiť vyše 30 percent implementovaných útokov. Dá sa preto považovať za efektívne riešenie proti útokom na webové prehliadače.

Prínos tejto práce pre spoločnosť TrustPort je, že táto sada bude integrovaná do testovania anti-exploit systému firmy TrustPort. Napriek tomu, že práca naplňuje ciele zadania, je ešte priestor na vylepšenia. Jedným z nich by mohlo byť rozšírenie TCP komunikácie medzi serverom a pluginom tak, aby fungovala v rámci internetu. To by umožnilo vzdialené testovanie. Pokračovať by sa dalo aj v pridávaní testov, napríklad implementáciou útokov buffer owerflow alebo integer overflow. Pre testovanie väčšieho množstva systémov by sa práca dala rozšíriť o automatickú detekciu operačného systému a následne podľa toho kategorizovať výsledné log súbory pre jednoduchšiu analýzu.

Literatúra

[1] Acunetix: *Cross-site Scripting Attack*. [Online; navštíveno 5.5.2016].
URL http://www.acunetix.com/websitesecurity/cross-site-scripting/

[2] Barth, A.: HTTP State Management Mechanism. RFC 6265, RFC Editor, April 2011, [Online; navštívené 5.5.2016].
URL http://www.rfc-editor.org/rfc/rfc6265.txt

[3] Barth, A.; Felt, A. P.; Saxena, P.; aj.: Protecting Browsers from Extension Vulnerabilities. In *NDSS*, Citeseer, 2010.

[4] Barth, A.; Jackson, C.; Reis, C.; aj.: The security architecture of the Chromium browser. 2008.

[5] Brereton, J.: *Conceptual Architecture of Mozilla Firefox 6 [online]*. Queen's University at Kingston, 2011, [Online; navštívené 5.5.2016].
URL https://fullyoptimized.files.wordpress.com/2011/09/conceptualarchitectureoffirefox6.pdf

[6] Burstyn, J.: *Conceptual Architecture of Google Chrome [online]*. Queen's University at Kingston, 2009, [Online; navštívené 5.5.2016].
URL https://archrometects.files.wordpress.com/2009/10/assignment-01-conceptual-architecture-of-google-chrome-archrometects.pdf

[7] Crowley, M.: *Pro Internet Explorer 8 & 9 Development: Developing Powerful Applications for the Next Generation of IE*, kapitola Internet Explorer Architecture. Berkeley, CA: Apress, 2010, ISBN 978-1-4302-2854-7, s. 1–37.
URL http://dx.doi.org/10.1007/978-1-4302-2854-7_1

[8] CVE Details: *Adobe Flash Player vulnerabilities*. [Online; navštíveno 5.5.2016].
URL https://www.cvedetails.com/vulnerability-list/vendor_id-53/product_id-6761/Adobe-Flash-Player.html

[9] Dhamija, R.; Tygar, J. D.; Hearst, M.: Why Phishing Works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, New York, NY, USA: ACM, 2006, ISBN 1-59593-372-7, s. 581–590.
URL http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/1124772.1124861

[10] Flanagan, D.: *JavaScript: The Definitive Guide*. O'Reilly Media, 2006, ISBN 0596101996.

[11] Google: *Google Chrome Web Browser*. [Online; navštívené 5.5.2016].
URL https://www.google.com/chrome/

[12] Grosskurth, A.; Godfrey, M. W.: Architecture and evolution of the modern web browser. *Preprint submitted to Elsevier Science*, 2006.

[13] Irish, P.: *Behind the scenes of modern web browsers [online]*. 2011, [Online; navštívené 5.5.2016].
URL http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/

[14] Kovash, K.: *How Many Firefox Users Customize Their Browser?* Mozilla Blogs, 2009-07-11 [cit. 2016-05-07], [Online; navštíveno 5.5.2016].
URL https://blog.mozilla.org/metrics/2009/08/11/how-many-firefox-users-customize-their-browser/

[15] Mao, Z.; Li, N.; Molloy, I.: *Financial Cryptography and Data Security: 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, kapitola Defeating Cross-Site Request Forgery Attacks with Browser-Enforced Authenticity Protection. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ISBN 978-3-642-03549-4, s. 238–255.
URL http://dx.doi.org/10.1007/978-3-642-03549-4_15

[16] Microsoft: *Windows XP support has ended*. [Online; navštíveno 5.5.2016].
URL http://windows.microsoft.com/en-us/windows/end-support-help

[17] Mozilla Foundation: *Mozilla Add-ons in use for last 7 days*. [Online; navštívené 5.5.2016].
URL https://addons.mozilla.org/en-US/statistics/addons_in_use/?last=7

[18] Mozilla Foundation: *Mozilla Firefox*. [Online; navštívené 5.5.2016].
URL https://www.mozilla.org/en-US/firefox

[19] Mozilla Foundation: *XUL Structure*. [Online; navštíveno 5.5.2016].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/Tutorial/XUL_Structure

[20] NetMarketShare: *Desktop Browser Version Market Share*. [Online; navštívené 5.5.2016].
URL https://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustomd=0

[21] OWASP: *Cross-site Scripting (XSS)*. [Online; navštíveno 5.5.2016].
URL https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[22] OWASP Foundation: *Buffer Overflow*. [Online; navštíveno 5.5.2016].
URL https://www.owasp.org/index.php/Buffer_Overflow

[23] OWASP Foundation: *Cross-site Scripting Request Forgery*. [Online; navštíveno 5.5.2016].
URL https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[24] OWASP Foundation: *Integer Overflow*. [Online; navštíveno 5.5.2016].
URL https://www.owasp.org/index.php/Integer_overflow

[25] OWASP Foundation: *SQL Injection*. [Online; navštíveno 5.5.2016].
URL https://www.owasp.org/index.php/SQL_Injection

[26] OWASP Foundation: *Top 10 Web vulnerabilities of 2013* . [Online; navštíveno 5.5.2016].
URL https://www.owasp.org/index.php/SQL_Injection

[27] Richard Bateman: *FireBreath Framework*. [Online; navštíveno 5.5.2016].
URL http://www.firebreath.org/

[28] Roberts, E.: *How Anti-Virus Software Works [online]*. Stanford University, 2001, [Online; navštívené 5.5.2016].
URL http://cs.stanford.edu/people/eroberts/cs201/projects/viruses/index.html

[29] Robot Framework Foundation: *Robot Framework*. [Online; navštíveno 5.5.2016].
URL http://robotframework.org/

[30] Rogers, R.; Lagarde, K.: Web browser system. 1998, uS Patent 5,793,964.
URL https://www.google.com/patents/US5793964

[31] TrustPort: *Download TrustPort Internet Security 2016*. [Online; navštíveno 16.5.2016].
URL http://www.trustport.com/en/home-users

[32] TrustPort: *TrustPort Internet Security 2016*. [Online; navštíveno 25.5.2012].
URL http://www.trustport.com/sites/default/files/product-sheets/productsheet_trustport_internet_security_en.pdf

[33] W3C: *Cascade Styling Sheets*. [Online; navštíveno 5.5.2016].
URL http://www.w3schools.com/css/

[34] W3C: *HTML5 Standard*. [Online; navštíveno 5.5.2016].
URL https://www.w3.org/TR/html5/

[35] WWW stránky: *AdBlock*. [Online; navštívené 5.5.2016].
URL https://getadblock.com/

[36] WWW stránky: *Internet Explorer Architecture [online]*. Microsoft Developer Network, 2016, [Online; navštíveno 25.5.2012].
URL https://msdn.microsoft.com/en-us/library/aa741312(v=vs.85).aspx

Prílohy

Zoznam príloh

Príloha A

Obsah DVD

V prílohe sa nachádza DVD nosič so všetkými potrebnými súbormi k skompilovaniu projektu a spusteniu testov. Obsahuje implementáciu modulov Control Server, Plug-in, definíciu testov a vytvorenú knižnicu kľúčových slov. Rovnako aj software tretích strán– Firebreath Framework, Robot Framework a vzhľadom na to, že je obtiažne získať staršiu verziu prehliadača Chrome, na nosiči sa nachádza priečinok s inštalovaným Chrome verzie 23. z roku 2013.

A.1 Priečinky a súbory

- ***src/*** Visual Studio projekty modulov Control Server a Plug-in.
- ***thirdParty/*** FireBreath Framework, Robot Framework, Google Chrome 23.
- ***testFiles/*** Definície testovacích sád, definícia kľúčových slov, webové stránky, TrustPort binárky.
- ***TechnickaSprava/*** Zdrojové súbory tchnickej správy a PDF súbor.
- ***bin/*** Preložené programy server (.exe) a plugin (.dll).
- **MANUAL.txt** Inštalačný manuál.

Príloha B

Preklad a použitie

B.1 Control Server

Na preklad tohto modulu je potrebné vývojové prostredie Visual Studio 2013.

- Navigovať k súboru *src/Control Server/cntrl_server.sln*.
- Otvoriť programom Visual Studio 2013.
- build -> build solution.
- Výsledný program je generovaný ako *src/Control Server/Debug | Release/cntrl_server.exe*.

B.2 Plug-in

Na vytvorenie projektu s pripravených zdrojových súborov je potrebné mať nainštalovaný cmake a zároveň pridaný do system PATH. Rovnako aj ľubovolnú verziu Visual Studio 2005 a novšiu.

- Navigovať do priečinka *src/Plug-in*.
- Podľa verzie Visual Studia spustíme dávkový súbor *(.bat)*.
  - *BuildForVisualStudio2005.bat*
  - *BuildForVisualStudio2008.bat*
  - *BuildForVisualStudio2010.bat*
  - *BuildForVisualStudio2012.bat*
  - *BuildForVisualStudio2013.bat*
- V aktuálnom priečinku sa vygeneruje priečinok *build20xx* v ktorom nájdeme projekt pre Visual Studio.
- Otvoriť programom Visual Studio.
- build -> build solution[1].

[1] Môže trvať aj niekoľko minút.

- Výsledný plugin je generovaný ako *src/Plug-in/build20xx/bin/PRCS/Debug | Release/npPRCS.dll*.
- Tento súbor je potrebné z príkazoveho riadku inštalovať do registrov príkazom *regsvr32 npPRCS.dll*.
- Plug-in je pripravený na použitie.

B.3 Robot Framework

Na inštaláciu Robot Framework je potrebný mať nainštalovaný Python 2.7 a mať ho pridaný v system PATH. Rovnako je potrebné mať pridaný v system PATH aj priečinok *Python27/Scripts*.

- Navigovať k súboru *thirdParty/Robot Framework/install.bat*.
- Otvoriť
- Robot Framework by mal byť nainštalovaný a dostupný z príkazového riadku príkazom *robot*.

B.4 Spustenie testov

- Preložený súbor *cntrl_server.exe* premenovať na *server.exe* a skopírovať do priečinka *testFiles/sut*.
- Inštalovať plug-in podľa pokynov v sekcii B.2.
- Cez príkazový riadok navigovať do *testFiles*.
- Podľa toho v akom prehliadači chceme spustiť testy, vyberieme príslušný súbor.
  - *robot FirefoxTests.rst*
  - *robot ChromeTests.rst*
  - *robot IExplorerTests.rst*