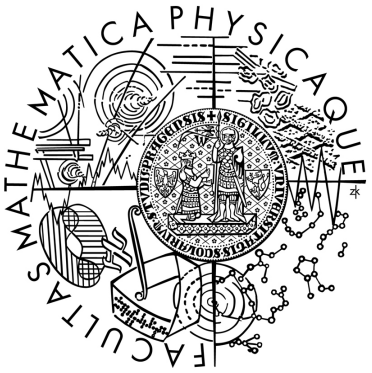
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Michal Mižišin

Analyzátor útokov na webový server

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Miroslav Novotný PhD.

Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2012

Poďakovanie

Moje poďakovanie patrí predovšetkým vedúcemu tejto bakalárskej práce za vedenie, cenné rady a trpezlivosť.

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne a výhradne s použitím citovaných prameňov, literatúry a ďalších odborných zdrojov.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona v platnom znení, hlavne skutočnosť, že Univerzita Karlova v Prahe má právo na uzavretie licenčnej zmluvy o užití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V …..... dne.....

Název práce: Analyzátor útokov na webový server

Autor: Michal Mižišin

Katedra / Ústav: Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Miroslav Novotný PhD., Katedra softwarového inženýrství

Abstrakt: V tejto práci sa zamieram na najčastejšie formy útokov na webové aplikácie. Predovšetkým na tzv. Injection flaws (útoky, pri ktorých sú dáta od užívateľa interpretované a vykonané), XSS (Cross Site Scripting) a CSRF (Cross Site Request Forgery), ktoré majú pre webovú aplikáciu v prípade kompromitácie fatálne následky. Zmapujem tieto typy útokov, ich históriu, popis, konkrétne príklady zneužitia. Navhrhnem tiež možné spôsoby ochrany a možnosti detekcie týchto útokov.

Klíčová slova: webový útok, sql injection, xss, csrf, injection flaw

Title: Web server attack analyzer

Author: Michal Mižišin

Department: Department of Software Engineering

Supervisor: Mgr. Miroslav Novotný PhD., Department of Software Engineering

Abstract: In this work I will focus on the most common forms of attacks on web applications. My focus will point on so called Injection flaws (attacks where data given by user are interpreted and executed), XSS (Cross Site Scripting) and CSRF (Cross Site Request Forgery), that have for web application in case of compromisation fatal consequences. I will describe these attacks, their history, concrete examples of successful execution. I will propose also possible kinds of protection and possibilities of detection.

Keywords: web attack, sql injection, xss, csrf, injection flaw

1 ÚVOD **7**

2 WEBOVÉ ÚTOKY **9**

2.0 MX INJECTION 11
2.0.1 Popis *11*
2.0.2 IMAP Injection *12*
2.0.3 SMTP Injection *13*
2.1 SSI INJECTION (SERVER SIDE INCLUDE) 15
2.1.1 Popis *15*
2.1.2 Ochrana *16*
2.2 XPATH INJECTION 16
2.2.1 Popis *16*
2.2.2 Ochrana *18*
2.3 PATH TRAVERSAL 19
2.3.1 Popis *19*
2.3.2 Ochrana *20*
2.4 LDAP INJECTION 20
2.4.1 Popis *20*
2.4.2 Ochrana *21*
2.5 OS COMMANDING 22
2.5.1 Popis *22*
2.5.2 Ochrana *23*
2.6 XML INJECTION 23
2.6.1 Popis *23*
2.6.2 Ochrana *25*
2.7 ORM INJECTION 25
2.7.1 Popis *25*
2.7.2 Ochrana *27*
2.8 SQL INJECTION 27
2.8.1 Popis *27*
8.2 Ochrana *29*
2.8.3 Príklady útokov *30*
2.9 CROSS SITE SCRIPTING 31
2.9.1 Same Origin Policy *31*
2.9.2 Popis útoku *31*
2.9.3 Spôsoby vloženia skriptu do webovej stránky *34*
2.9.4 Ochrana *36*
2.10 CROSS SITE REQUEST FORGERY 37
2.10.1 Popis útoku *37*
2.10.2 Ochrana *38*

3 UŽÍVATEĽSKÁ DOKUKEMENTÁCIA **40**

4 PROGRAMÁTORSKÁ DOKUMENTÁCIA **42**

4.0 HĽADANIE ÚTOKOV 43
4.1 POROVNANIE S OSTATNÝMI NÁSTROJMI 49
4.2 MOŽNOSTI ROZŠÍRENIA 50
4.3 PREKLAD PROGRAMU 50

5 ZÁVER **51**

6 ZDROJE **52**

7 PRÍLOHY **61**

PRÍLOHA 1: PRÍKLAD KONFIGURAČNÉHO SÚBORU 61
PRÍLOHA 2: CD S PROGRAMOM A ELEKTRONICKOU VERZIOU BAKALÁRSKEJ PRÁCE 63

1 Úvod

Internet ako celosvetová sieť počítačov, kde zdieľanie dokumentov medzi užívateľmi bez ohľadu na ich polohu, je fenoménom dnešnej doby. Svojím postupným rozmachom zasahuje do takmer všetkých oblastí nášho života. Naše životy sa týmto spôsobom postupne v digitálnej podobe presúvajú do celosvetovej siete. Hlavným dôvodom je pohodlnosť, ktorú to prináša. Mať všetko okamžite dostupné bez toho, aby človek musel vystrčiť čo i len päty z pohodlia svojho domova, byť v kontakte s blízkymi, ktorí sú na druhom konci sveta tak ako keby sa nachádzali vedľa nás.

Za tento rozmach môžeme ďakovať hlavne webovým technológiám. Je síce jedna vec, umiestniť niekam svoje dáta, ale rovnako dôležitou vecou je tieto dáta aj sprístupniť na akomkoľvek mieste. V tomto ohľade tomuto rozmachu pomohli webové technológie. Pod týmto názvom sa skrývajú prostriedky, metódy a postupy, ktoré boli neskôr prevedené do foriem štandardov, protokolov a programového vybavenia. Webové technológie pomáhajú svojimi prostriedkami zvýšiť pohodlnosť pri získavaní týchto informácii odkiaľkoľvek.

Touto pohodlnosťou sme boli tak trochu zaslepení a zabudli sme vziať do úvahy to, či sú naše súkromné dáta v bezpečí. Je síce pravda, že obyčajný užívateľ možným rizikám takmer vôbec nerozumie, keď niekam umiestňuje informácie o sebe, ale na druhej strane, ani odborníci, ktorí celý tento systém navrhli, prípadne spravujú, na to nemysleli dostatočne. Ako jeden z dôvodov môžeme označiť to, že na nich nebol kladený tlak od užívateľov ich aplikácií.

Vo svojej práci som sa zameral na útoky, ktoré sú vedené voči webovým aplikáciám. Konkrétne ide o útoky typu „injection flaws“. Nebezpečenstvom týchto útokov je, že aplikácie, ktoré sú voči nim zraniteľné, sa v podstate správajú „korektne“ voči správnym dátam. Problém ale je, že nedostatočne ošetrujú vstup od užívateľa, ktorý im môže podsunúť vstup taký, ktorý je daným programom vyhodnotený ako príkaz pre neho a urobí niečo iné ako zamýšľal programátor.

Nerobme si ilúzie, útoky na webové aplikácie tu budú stále. Dôležité na tom je, aby sme vzali do úvahy, že aplikácia je potenciálne zraniteľná. Z tohto pohľadu je

veľmi dobré vedieť, ktoré miesta využívajú útočníci, odkiaľ vlastne chodia takéto dotazy a o aké typy útokov sa snažia.

Prvá kapitola popisuje rôzne druhy útokov, ich históriu, technológiu, kde sa uplatňujú, a tiež príklady konkrétnych úspešných útokov.

Druhá kapitola obsahuje uživateľskú dokumentáciu. Konfiguráciu programu a ako ho spustiť, v prílohe č.1 sa potom nachádza konkrétny príklad konfiguračného súboru.

V tretej kapitole sa zameriavam na AttackAnalyzer z pohľadu programátora. Obsahuje popis modulov, ako funguje analýza, kedy sa spúšťa a čo sa vlastne v danej komunikácii hľadá. Nachádza sa v nej tiež časť, ktorá ukazuje ako rozšíriť analýzu o ďalšie typy útokov.

2 Webové útoky

Útok na systém je podľa IETF (Internet Engineering Task Force) úmyselný pokus, ktorého účelom je obísť jeho zabezpečenie. Obyčajne je páchaný niekým so zlými úmyslami, ale existujú aj útoky, ktorých snahou je objaviť zraniteľnosť systému (penetračné testy).

Pod webovými útokmi chápeme útoky, ktoré sa snažia napadnúť systémy, ktoré komunikujú nad protokolom HTTP a užívateľ komunikuje so systémom cez webový prehliadač. Do tejto kategórie patria útoky, ktoré využívajú formátovanie znakových reťazcov. Ako vstupné dáta sa podstrčia aplikácii také reťazce, ktoré potom daný systém vyhodnocuje ako príkaz.

Počet webových útokov neustále narastá, keďže sa stále viac aktivít presúva na internet. Podľa firmy Symantec narástol počet webových útokov v roku 2010 oproti roku 2009 o 93% [86].

Finančné škody, ktoré spôsobia hackeri sú vo väčšine prípadov ťažko vyčísliteľné. Väčšina firiem tieto útoky podceňuje. Často o nich dokonca ani nevedia, keďže veľa organizácii nemonitoruje online aktivitu na úrovni webovej aplikácie. Týmto spôsobom ale vznikajú škody, ktoré môžu mať ťažké finančné dopady:

- strata dôvery zákazníka, reputácie spojené s poškodením značky firmy, čo môže mať za následok zníženie ziskov,
- strata možnosti prijímať určité formy platenia na internete ako napr VISA, Mastercard (kvôli ich politike, ktorá zakazuje prijímať platby od kompromitovaných stránok),
- negatívny dopad na profit, ktorý prichádza kvôli sfalšovaným transakciám,
- nefunkčnosť webovej aplikácie, čo napríklad v prípade internetového obchodu znižuje počet predajov,
- náklady na zamestnancov, ktorí musia pracovať na oprave daného problému,
- právne náklady, prípadne odškodné, ktoré musia firmy zaplatiť obetiam

útokov,

Združenie OWASP[45] vydáva pravidelné „Top 10“ najčastejších webových útokov:

1.) **Injection:** napr. SQL injection, LDAP injection, OS commaning. Objavujú sa vtedy, keď sa neskontrolované dáta posielajú priamo aplikácii, ktorá to môže chápať ako príkaz a nie užívateľské dáta.

2.) **XSS (Cross Site Scripting):** objavuje sa vtedy, ak sa neskontrolované dáta posielajú webovému prehliadaču, ktorý ich chápe ako príkazy.

3.) **Nefunkčný session management a autentikácia:** útočník je schopný kompromitovať heslo, session...

4.) **Nezabezpečené priame interné referencie:** bez akejkoľvek prístupovej kontroly môže útočník manipulovať napr. súbory, priečinky...

5.) **Cross Site Request Forgery:** umožňuje autentikovanému užívateľovi na jednom serveri vykonať bez jeho vedomia akciu na inom serveri.

6.) **Zlá bezpečnostná konfigurácia:** napr. ak aplikácia využíva framework, ktorý nie je aktuálny a sú v ňom známe bezpečnostné chyby.

7.) **Nedostatočné zabezpečené uloženie citlivých dát:** napr. keď heslá v databáze sú uložene ako čistý text.

8.) **Nezabezpečený prístup cez URL:** aplikácia síce kontroluje práva predtým než napríklad vykreslí tlačidlo na stránke, ktoré užívateľa presmeruje na inú stránku, už ale nekontroluje priamy prístup cez URL adresu.

9.) **Nedostatočné zabezpečenie transportu dát:** aplikácia nepoužíva šifrovanie na prenos citlivých informácii.

10.) **Nekontrolované presmerovania na stránky:** aplikácia nekontroluje dostatočne presmerovanie užívateľa na inú stránku, čo môže útočník využiť na presmerovanie na podvrhnutú stránku.

V nasledujúcich kapitolách som sa venoval prevažne útokom typu Injection, XSS a CSRF. Tieto útoky sa dajú detekovať sledovaním komunikácie medzi webovou aplikáciou a útočníkom.

2.0 MX Injection

2.0.1 Popis

Útok MX Injection [1] prvýkrát prezentoval Vicente Aguilera Diaz na konferencii OWASP [45] 16.6.2006. Názov MX Injection pochádza od toho istého autora. Konkrétne podoby zneužitia ale nie sú dokumentované, ale rôzne služby ponúkajúce webové rozhranie na prístup k emailovej schránke boli zraniteľné, napr. Squirellmail[4].

Webové aplikácie používajú protokoly IMAP [2] a SMTP [5] na správu užívateľov a ich e-mailov. Interakcia začína poslaním užívateľových prihlasovacích údajov na overenie totožnosti. V tomto momente posiela webová aplikácia e-mailovému serveru príkaz [2]:

AUTH LOGIN <user> <password>

Na základe odpovede povolí alebo zakáže prístup užívateľa k jeho mailovej schránke.

Akákoľvek užívateľova akcia súvisiaca s jeho emailovou schránkou je potom prekladaná do IMAP prípadne SMTP príkazov. Tento preklad je zneužitý na vloženie iného príkazu ako ma aplikácia v úmysle Na podobnom princípe fungujú aj ostatné známe zraniteľnosti (SQL injection, XPath Injection...).

Výhoda tohto útoku je, že útočník takto môže kompromitovať mail server, ktorý nie je dostupný priamo z internetu. V niektorých prípadoch totiž interné systémy nemajú rovnakú úroveň ochrany infraštruktúry, ktorá je aplikovaná na frontend web server. [6]

Obrázok č.1: MX Injection [3]

Táto technika ponúka možnosť vkladať celé príkazy do zasiahnutého mail serveru, a to nie len vloženie email headerov (From:, Subject:, To:, …) alebo posielanie veľkého množstva emailov, ale dovoľuje aj činnosti, ktoré nie sú inak prístupné webovej aplikácii (napr. buffer overflow emailového servera cez IMAP príkaz).

Postihuje všetky platformy, ktoré používajú protokoly IMAP alebo SMTP.

2.0.2 IMAP Injection

V tomto prípade komunikuje aplikácia so serverom cez protokol IMAP, takže všetky príkazy musia spĺňať špecifikáciu tohto protokolu. Predtým než sú manipulované, musí útočník identifikovať, aké príkazy sú používané na komunikáciu so serverom a aký je ich formát (prihlásenie, operácie so správami, schránkami...).

Výhodou tohto útoku je, že útočník ani nemusí byť autentikovaný aplikáciou, aby zneužil túto zraniteľnosť.

Konkrétny príklad uvádza Diaz [3]:

Útočník zneužije funkcionalitu čítania správy. Predpokladajme, že webová aplikácia používa parameter „message_id“ ako identifikátor správy, ktorú chce prečítať. Potom žiadosť obsahujúca identifikátor správy bude vyzerať takto:

http://<webmail>/read_email.php?message_id=<number>

Ďalej predpokladajme, že webová stránka „read_email.php“, ktorá je zodpovedná za zobrazenie požadovanej správy, predáva tento parameter IMAP serveru bez toho, aby nejako validovala hodnotu <number> od útočníka. Príkaz poslaný serveru bude potom vyzerať takto:

FETCH <number> BODY[HEADER]

V tomto prípade môže útočník podstrčiť aplikácii nechcený príkaz ako parameter message_id. Napríklad IMAP príkaz CAPABILITY by mohol byť zneužitý nasledujúcim spôsobom:

http://<webmail>/read_email.php?message_id=1
BODY[HEADER]%0d%0aZ900 CAPABILITY%0d%0aZ901 FETCH 1

čo bude mať za následok nasledujúcu sekvenciu príkazov na na serveri:

FETCH 1 BODY[HEADER]
Z900 CAPABILITY
Z901 FETCH 1 BODY[HEADER]

Teda stránka vrátená užívateľovi serverom zobrazí výsledok príkazu CAPABILITY na IMAP serveri:

** CAPABILITY IMAP4rev1 CHILDREN NAMESPACE*
THREAD=ORDEREDSUBJECT THREAD=REFERENCES
SORT QUOTA ACL ACL2=UNION
Z900 OK CAPABILITY completed

2.0.3 SMTP Injection

Ako v predchádzajúcom prípade, aj tu musí útočník na zneužitie tejto zraniteľnosti použiť príkazy v súlade so špecifikáciou protokolu SMTP[5], navyše ale musí byť útočník autentikovaný.

Túto zraniteľnosť ukazuje Diaz[3] na jednoduchom posielaní mailov:

Typická žiadosť na poslanie mailu vyzerá takto:

POST http://<webmail>/compose.php HTTP/1.1
...
----------------------------134475172700422922879687252
Content-Disposition: form-data; name="subject"
SMTP Injection Example
----------------------------134475172700422922879687252

...

čo vygeneruje nasledujúcu postupnosť SMTP príkazov:

MAIL FROM: <mailfrom>

RCPT TO: <rcptto>

DATA

Subject: SMTP Injection Example

...

V prípade, že aplikácia nekontroluje dostatočne užívateľom zadané hodnotu „subject“, útočník tam môže podstrčiť nechcený kód.

POST http://<webmail>/compose.php HTTP/1.1

...

----------134475172700422922879687252

Content-Disposition: form-data; name="subject"

SMTP Injection Example

.

MAIL FROM: notexist@external.com

RCPT TO: user@domain.com

DATA

Email data

.

----------134475172700422922879687252

...

čo bude mať za následok nasledujúce príkazy:

MAIL FROM: <mailfrom>

RCPT TO: <rcptto>

DATA

Subject: SMTP Injection Example

.

MAIL FROM: notexist@external.com

RCPT TO: user@domain.com

DATA

Email data

.

Obrana pred útokom

Ochrana pred týmto útokom spočíva v týchto troch bodoch:

- validácia užívateľom zadaných hodnôt
- správna konfigurácia IMAP/SMTP servera
- použitie firewallu

Validácia užívateľom zadaných hodnôt

Všetky hodnoty zadané užívateľom by mali byť kontrolované. Vymazané alebo nahradené musia byť všetky znaky, ktoré by mohli byť chápané ako príkazy serverom.

Na to, aby bolo možné vložiť nový IMAP_SMTP príkaz, je nutné, aby predchádzajúci skončil s CRLF. Takže pomerne dobrou ochranou je vymazať s užívateľovho vstupu tento znak predtým než je poslaný serveru.

Správna konfigurácia IMAP/SMTP servera

Povoliť aplikácii len nevyhnutné príkazy a ostatné zakázať.

2.1 SSI Injection (Server Side Include)

2.1.1 Popis

Útoky tohto typu sa pravdepodobne začali objavovať začiatkom nového milénia [7].

SSI je premenná ako napr. „Last modified“ dátum, ktorú môže web server vložiť do HTML súboru. Predtým než server pošle súbor užívateľovi, prejde tento súbor hľadajúc CGI (common gateway interface) premenné a vloží na miesto nich žiadanú hodnotu. Toto môže útočníkovi dovoliť pridať, nahradiť alebo vymazať HTML súbory na serveri, ale taktiež sprístupniť zdroje web servera [8]. SSI má nasledujúcu syntax[11]:

<!--#element attribute=value attribute=value ... -->

Server Side Includes prinášajú so sebou viacero výhod, ale aj bezpečnostných zraniteľností. SSI zvyšuje zátaž servera. Všetky stránky, ktoré majú povolené SSI musia byť spracované webovým serverom, bez ohľadu na to, či majú v sebe prítomné SSI direktívy alebo nie. Toto zvýšenie záťaže je minimálne, ale v zdieľanom prostredí to navýšenie už môže byť značné [12]. Ďalším a asi

najdôležitejším nebezpečenstvom je to, že pri nesprávnej konfigurácii servera a nesprávnej validácii užívateľom zadaných hodnôt, môže útočník vykonať takmer akýkoľvek príkaz s právami webového servera.

Ako príklad [9] zneužitia uvádza organizácia Webappsec získanie výpisu koreňového adresára na unixových systémoch nasledujúcim vstupom:

<!--#exec cmd="/bin/ls /" -->

Podľa testov je tento útok vykonateľný iba v prostredí, kde je možný aj Persistent XSS[13].

Tento typ útoku sa týka predovšetkým webového servera Apache, ktorý je najpoužívanejším serverom súčasnosti[14], ale zasahuje aj Microsoft Internet Information Server (IIS) [68].

2.1.2 Ochrana

Ochrana pred týmto útokom spočíva v tom, že administrátor servera dovolí SSI iba na stránkach, ktoré to vyslovene potrebujú. Pre stránky, kde je to nutné povoliť je odporúčané urobiť nasledujúce opatrenia [8]:

- povoliť iba tie SSI direktívy, ktoré konkrétna stránka potrebuje
- zakódovať HTML kódovaním vstupné hodnoty užívateľa predtým než sú poslané stránke, ktorá má práva na vykonanie SSI
- (týkajúce sa servera Apache) použiť SUExec[10], aby SSI bolo vykonané s právami vlastníka daného súboru namiesto práv používateľa webového servera

2.2 XPath Injection

2.2.1 Popis

V roku 1999 pod taktovkou konzorcia W3C uzrelo svetlo sveta Xpath 1.0. Od tej doby sa začali používať XML dokumenty aj ako úložiská dát. Jedným z prvých dokumentov hovoriacich o Xpath injection je z roku 2004[19], ale útoky sa vyskytovali s veľmi vysokou pravdepodobnosťou už skôr, len žiaľ o tom nie sú záznamy. V posledných rokoch táto zraniteľnosť postihuje pomerne veľké množstvo webových aplikácií (v roku 2007 to bolo podla zistení spoločnosti WhiteHatSecurity jeden z 30 webov [22]) .

XPath je jazyk, ktorý bol vytvorený primárne na adresovanie časti XML dokumentu. Historicky sú najpoužívanejším spôsobom na ukladanie dát relačné databázy. V posledných rokoch sme svedkami toho, že na organizovanie dát je použitý XML jazyk. Dôvodom môže byť jednoduchá prenositeľnosť, kompatibilita, štruktúrované formátovanie [15]. Tento typ útoku je veľmi podobný SQL injection.

Zatiaľ čo Xpath je štruktúrou veľmi podobný jazyku SQL, je potenciálne omnoho nebezpečnejší, pretože nedokáže diferencovať prístup k jednotlivým tabuľkám (častiam dát) na základe užívateľských práv, v Xpath je prístup k celému úložisku. Xpath 2.0 prináša novú funkcionalitu, kde použitím collection alebo doc funkcie umožňuje získať akýkoľvek XML dokument, ktorého umiestnenie je na tom istom úložišti. V súvislosti s prenositeľnosťou je nutné podotknúť, že to prináša aj malé výhody pre útočníkov. Xpath je štandard, teda neexistujú od neho rôzne dialekty, a aj útoky sú univerzálne v porovnaní s SQL, kde musia byť využité konkrétne vlastnosti tej ktorej implementácie [16]. Na druhej strane ale Xpath v porovnaní s SQL nedokáže meniť dáta.

Uveďme si príklad, ako môže byť nedostatočnou ochranou využitá táto slabina na neautentifikovaný vstup do systému [21]:

Predpokladajme, že aplikácia autentifikuje užívateľov voči užívateľom uloženým v nasledujúcom súbore.

```
<?xml version="1.0" encoding="utf-8"?>
<Employees>
  <Employee ID="1">
    <FirstName>Arnold</FirstName>
    <LastName>Baker</LastName>
    <UserName>ABaker</UserName>
    <Password>SoSecret</Password>
    <Type>Admin</Type>
  </Employee>
  <Employee ID="2">
    <FirstName>Peter</FirstName>
    <LastName>Pan</LastName>
    <UserName>PPan</UserName>
    <Password>NotTelling</Password>
```

```
    <Type>User</Type>
  </Employee>
</Employees>
```

Keď sú zadané username a password, program použije Xpath na vyhľadanie používateľa v databáze.

```
String FindUserXPath;
FindUserXPath = "//Employee[UserName/text()='" + Request("Username") + "'
And
      Password/text()='" + Request("Password") + "']";
```

S normálnym username a password bude všetko v poriadku fungovať, ale útočník môže poslať neplatné username a heslo bez toho, aby vedel správne.

```
Username: blah' or 1=1 or 'a'='a
Password: blah

FindUserXPath becomes //Employee[UserName/text()='blah' or 1=1 or
      'a'='a' And Password/text()='blah']
```

2.2.2 Ochrana

Ochrana pred týmto útokom je podobná ako pred SQL Injection a pozostáva z 3 častí.

1. Validácia [20]: toto je základný kameň ochrany pred všetkými útokmi. Bez ohľadu na prostredie, aplikáciu a programovací jazyk, programátor by mal urobiť nasledujúce kroky:
   - všetky vstupné dáta považovať za podozrivé
   - validovať nie len dátový typ, ale aj formát, dĺžku, rozsah a obsah (napríklad jednoduché regulárne výrazy ako (/^"*^';&<>()), ktorý by mal nájsť najpodozrivejšie znaky).
   - validovať dáta na klientskej a serverovej strane, lebo je jednoduché obísť klienskú validáciu
2. Parametrizácia [20]: na rozdiel od väčšiny databázových aplikácii Xpath nepodporuje koncept parametrizovaných dotazov. Je to ale možné napodobniť použitím iných API ako napr. Xquery. Namiesto budovania

výrazov ako reťazcov a predávania ich Xpath parseru na dynamické vyhodnotenie, je možné parametrizovať dotaz vytvorením externého súboru, ktorý bude obsahovať dané dotazy.

Príklad súboru dologin.xq

declare variable $loginID as xs:string external;

declare variable $password as xs:string external;//users/user[@loginID= $loginID and @password=$password]

3. „Escapovať“ špeciálne znaky ako napr. :, –, zátvorky

2.3 Path Traversal

2.3.1 Popis

Presné korene tohto útoku nie sú známe, ale prvé známky útokov sa objavali už na konci 80-tych rokov. Niektorí bezpečnostní odborníci predpokladajú, že Path Traversal existuje ešte dlhšie. Aj napriek pomerne dlhej existencii tohto útoku, nezískal si veľa pozornosti do roku 2000 [23].

Path Traversal dovoľuje útočníkovi získať prístup k súborom, priečinkom a príkazom, ktoré sa nachádzajú mimo koreňového adresára webového dokumentu. Základná úloha webového servera je poskytovať súbory. Môžu byť statické (obrázky, HTML súbory) alebo dynamické (ASP, JSP...). Ak užívateľ požaduje dynamický obsah, príkazy v súbore server najprv vykoná a potom ho pošle. Aby sa užívateľ nedostal k neautorizovanému obsahu, používa webový server dva hlavné bezpečnostné mechanizmy, a to koreňový adresár a access control list. Všetky súbory, ktoré sa nachádzajú v koreňovom adresári alebo podadresároch sú prístupné užívateľovi. Na obmedzenie prístupu užívateľa k určitým súborom v koreňovom adresári používajú administrátori access control list. Koreňový adresár zabraňuje vykonaniu súborov ako napr cmd.exe na Windows platforme alebo prístupu citlivých údajov ako passwd na Unix systémoch.

Využitím path traversal vulnerability, útočník obíde oba tieto ochranné mechanizmy, čo môže mať za následok získanie prístupu k zakázaným dokumentom alebo vykonanie systémových príkazov, ktoré môžu kompromitovať celý webový server [24].

Najzákladnejší útok používa “../” na zmenu umiestnenia. V dnešnej dobe už

väčšina populárnych webových serverov zabraňuje takýmto spôsobom opustiť koreňový adresár, ale použitím dekódovaní je to niekedy možné obísť. Tieto metódy obsahujú správne a nesprávne Unicode kódovania ("..%u2216" or "..%c0%af") lomítka, spätného lomítka („..\“) na Windows serveroch, URL kódovanie [24].

Aj napriek tomu, že web server správne zabráni Path traversal v ceste URL, webová aplikácia samotná nemusí byť odolná voči tomuto útoku kvôli žiadnej resp. zlej kontrole dát, ktoré zadáva užívateľ aplikácie.

Path Traversal útok na webový server

http://example/../../../../../etc/passwd
http://example/..%255c..%255c..%255cboot.ini
http://example/..%u2216..%u2216someother/file

Path Traversal útok s použitím špeciálnych sekvencií (using special-character sequences).

V tomto prípade získa útočník prístup k zdrojovému kódu súboru foo.cgi, pretože hodnota premennej page bola použitá ako obsah. Aby obišiel kontrolu typu súboru, použil „%00“ sekvenciu.

Original: http://example/scripts/foo.cgi?page=menu.txt
Attack: http://example/scripts/foo.cgi?page=../scripts/foo.cgi%00txt

2.3.2 Ochrana

Ako pri všetkých typoch útokov, ochranou je kontrolovať užívateľom zadaný obsah predtým, než je predaný aplikácii.

Pri kontrole mien súborov je vhodné použiť obmedzenú sadu znakov. Ak je to možné, povoliť len jednu „.“ za sebou v mene súboru, filtrovať „/“ a „\“.

2.4 LDAP Injection

2.4.1 Popis

Prvá verzia štandardu LDAP bola vytvorená v roku 1993 Michiganskou univerzitou. Dnešná verzia (číslo 3) vznikla v roku 1997. Prvé útoky teda môžeme datovať do rokov 1997 (vznik štandardu) až 2003, kedy bol publikovaný [29]. Ku

konkrétnemu útoku sa mi bohužiaľ nepodarilo dopracovať. Keďže tento typ útoku je veľmi podobný SQL injection, ktorého prvý publikovaný útok bol v roku 1999. LDAP injection je teda možné datovať do približne podobného obdobia.

LDAP je skratka pre Lightweight Directory Access Protocol. Je to „odľahčená“ verzia štandardu X.500 od ISO/OSI konzorcia, ktorý je príliš robustný a ťažkopádny. Slúži na ukladanie informácii o užívateľoch, hostoch a iných objektoch. Podľa tohto protokolu sú jednotlivé položky ukladané formou záznamov a usporiadané v stromovej štruktúre (ako v skutočnej adresárovej štruktúre). Aplikácia funguje na princípe klient-server.

Na stránke s formulárom na hľadanie užívateľov je nasledujúci kód zodpovedný za získanie vstupnej hodnoty a vytvorenie LDAP dotazu, ktorý bude použitý v LDAP databáze [33].

<input type="text" size=20 name="userName">Insert the username</ input>

Funkcia v nižšej vrstve by mohla vyzerať takto:

```
String ldapSearchQuery = "(cn=" + $userName + ")";
 System.out.println(ldapSearchQuery);
```

Ak premenná $username nie je validovaná, je možné urobiť LDAP injection:

- ak užívateľ vloží do vyhľadávacieho poľa '*', systém vráti všetkých užívateľov v LDAP databáze
- ak užívateľ vloží 'jonys) (| (password = *))', kód uvedený vyššie vygeneruje kód, ktorý odhalí jonysovo heslo (cn = jonys) (| (password = *))

2.4.2 Ochrana

1.) Validácia užívateľom zadaných dát

Najideálnejšie by bolo, aby boli vyfiltrované všetky znaky, ktoré by teoreticky mohli byť škodlivé. Teda povoliť užívateľovi zadávať len čísla, veľké a malé písmena a ostatné znaky vymazať. To bohužiaľ ale príliš obmedzuje aplikáciu a nie je to preto často použiteľné. V prípade, že nemôže aplikácia obmedziť tieto znaky, je potrebné všetky znaky, ktoré majú význam v LDAP escapovať. Ide o

znaky '&', '!', '=', '<', '>', ',', '+', '-', '“', '", ';', ale predovšetkým znaky '(', ')', '\', '*', '/', 'NUL' a ich zakódované ekvivalenty. V prípade, že je ako programovací jazyk použitá Java je potrebné ešte escapovať akýkoľvek JNDI meta-znak, pretože Java používa JNDI na vykonávanie LDAP dotazov [32][34][39].

2.) **Validácia odchádzajúcich dát**

Všetky dáta vracané užívateľovi by mali byť validované a množstvo dát vracaných dotazmi by malo byť obmedzené ako ďalšia úroveň zabezpečenia [29].

3.) **LDAP konfigurácia**

Implementácia tesnej prístupovej kontroly k dátam v LDAP-e je veľmi odporúčaná, obzvlášť pri nastavovaní práv na užívateľské objekty a hlavne ak LDAP je použitý pre SSO (Single Sign On) [91]. Úroveň prístupu webovej aplikácie k LDAP serveru by mal byť obmedzený na absolútne minimum, ktoré umožňuje chod. Vtedy aj v prípade, že sa podarí útočníkovi prelomiť aplikáciu, jeho prípadné škody budú limitované. LDAP server by nemal byť prístupný priamo z internetu [29].

2.5 OS Commanding

2.5.1 Popis

Je to technika, ktorá umožňuje neautorizovane vykonať príkaz operačného systému. OS Commanding je priamy výsledok mixovania dôveryhodných a nedôveryhodných dát. Je možný v prípade, keď aplikácia prijíma nedôveryhodný vstup na vytvorenie príkazu operačného systému. Príkazy budú spustené s rovnakými právami ako má komponenta, ktorý ten príkaz volá. Pri nesprávnom pridelení práv aplikáciam to môže viesť až k poškodeniu systému [41].

Aplikácie často zahŕňajú súbor, ktorý majú zobraziť alebo použiť ako template.

http://example/cgi-bin/showInfo.pl?name=John&template=tmp1.txt

V prípade, že dáta nie sú validované, môže útočník vložiť cestu k príkazu, ktorý môže byť vykonaný, napr. /bin/ls

http://example /cgi-bin/showInfo.pl?name=John&template=/bin/ls|

2.5.2 Ochrana

1.) **Obmedzenie práv programov operačného systému**

Ak sa odstráni execution bit pre everyone group (-rwxrwzrw-) z príkazu operačného systému, potom účet webového servera nebude schopný vykonať daný príkaz. Čiže aj keď útočník podstrčí aplikácii škodlivý kód, tá ho nebude môcť vykonať. Tento postup je síce užitočný, ale má viacero problémov. Veľmi častým prípadom, keď bola využitá táto vulnerabilita, tak práve webový server mal neprimerané práva, niekedy až administrátorské. Navyše, pri takomto postupe už musí byť zapojený do činnosti aj administrátor systému, čo nie vždy je možné [42].

2.) **Biela listina dovolených znakov**

Tak ako pri takmer každom útoku, je potrebné odfiltrovať znaky, ktoré musí útočník použiť, aby prešiel cez validáciu aplikácie. V tomto prípade by mali byť odfiltrované všetky znaky okrem týchto: ([a-z][A-Z]/_-\.0-9) [42].

3.) **Odfiltrovať priečinky, kde sa nachádzajú príkazy**

To závisí na konkrétnom systéme, ale ak sa nachádza v užívateľom zadaných dátach nejaký priečinok s príkazmi, mal by byť tiež odfiltrovaný. Napríklad pre Linux systémy je potrebné odfiltrovať tieto priečinky: /etc, /bin, /sbin, /tmp, /var, /opt, /dev. Takýto filter zároveň tiež zabráni útočníkovi dostať sa k podpriečinkom [42].

2.6 XML Injection

2.6.1 Popis

Ako už bolo spomínané v časti o Xpath Injection, XML ma široké využitie. XML súbory môžu plniť úlohu dátových zdrojov, popisovať štruktúru dokumentu napríklad na validáciu, ale tiež ako transportný protokol (SOAP [47]). Podľa klasifikácie organizácie OWASP[45] je zaraďovaný tento útok medzi „injection flaws“, ktorá sa posledné roky umiestňuje v prvej desiatke najčastejších útokov.
Tento typ útoku sa prevažne používa na manipuláciu Web Services. Existuje viacero druhov útokov na Web Services [52]:

- **Parameter Tampering –** parametre sa používajú na posielanie pre klienta

špecifických informácií webovej službe, aby boli vykonané určité vzdialené operácie. Inštrukcie sú popísané vo WSDL[51] dokumente. Potenciálne môže útočník pozmeniť parametre, aby získal neautorizované informácie.

- **Recursive Payloads –** XML môže obsahovať elementy na adresovanie komplexných vzťahov v rámci dokumentu. Napr. vytvorenie objednávky, ktorá obsahuje adresu doručenia a fakturačnú adresu. Útočník takto môže službe vyradiť XML parser, keď mu podstrčí dokument s obrovským množstvom vnorených elementov.
- **Oversized Payloads –** XML parser je zraniteľný vočí útoku DoS (Denial of Service) a to tým, že sa mu podstrčí veľmi veľký XML dokument.
- **Schema Poisoning –** XML schema modeluje gramatickú a vzorovú štruktúru XML dokumentu. XML parsery používajú schémy na správne interpretovanie správ webovej služby. Útočník sa môže snažiť nahradiť tento súbor.
- **WSDL Scanning –** WSDL je mechanizmus pre webovú službu, ako dynamicky popísať paremetre používané pri pripájaní k príkazom, ktoré prijímajú vstup z externých zdrojov. WSDL súbory sú typicky generované automaticky nástrojmi, ktoré sú vytvorené na odhalenie a popísanie všetkých dostupných informácií v príkaze. Útočník sa v tomto type útoku snaží rozličnými príkazovými a reťazcovými kombináciami nájsť nepublikované aplikačné rozhrania.
- **Replay Attacks** – snahou útočníka je preťažiť webovú službu alebo XML parser opakovaným posielaním SOAP správ.
- **External Entity Attack** – XML môže vytvárať dokumenty dynamicky nasmerovaním na URI (Uniform Resource Identifier), kde sa nachádzajú aktuálne dáta. Útočník sa tieto dáta snaží nahradiť škodlivými.

Okrem útokov popísaných v časti o Xpath Injection, existuje aj ďalší typ, ktorý zasahuje webové stránky, ktoré používaju XML. Ide o element CDATA. Všetko, čo sa nachádza v tomto elemente je parserom ignorované. Takto môže útočník prepašovať do aplikácie nedovolený obsah a podarí sa mu takto napríklad XSS útok.

<HTML>

<![CDATA[<IMG SRC=http://www.exmaple.com/logo.gif
onmouseover=javascript:alert('Attack');>]]>
</HTML>

2.6.2 Ochrana

Ochrana voči vyššie spomínaným útokom spočíva hlavne v správnej konfigurácií prostredia.

Brániť sa voči "payloads" útokom je možné reštrikciami na veľkosť či štruktúru dokumentu, čo ale business požiadavky nie vždy umožnujú.

Pred External Entity útokom je možnou, ale opäť obmedzujúcou ochranou, zakázanie ich používania.

Replay útoku je možné zabrániť tým, že sa nadefinujú limity požiadaviek pre webové služby a limity adresy odkiaľ požiadavky pochádzajú. Aj keď nie vždy musí isť o útok, môže nastať situácia, že klient nedostal potvrdenie o prijatí požiadavky, a preto ju neustále opakuje. Toto je jeden z mála typov útokov, ktorý môže zablokovať proxy server nachádzajúci sa pred serverom webovej služby.

Požitím rôznych filtrov nevieme zistiť, čí sa útočník pokúša o XML útok. Môžeme nimi ale objaviť ostatné druhy útokov (XSS, CSRF...), aj v CDATA elementoch.

2.7 ORM Injection

2.7.1 Popis

Technika objektovo orientovaného programovania sa snaží čo najvernejšie zachytiť realitu. Objekty reálneho sveta sú v aplikácii reprezentované ako entity. Relačné entity sú riadky, resp. množiny riadkov v databázových tabuľkách. Z toho dôvodu vznikla technika ORM, ktorej účelom je konverzia medzi aplikačnými entitami a databázovými entitami. Vďaka tomu má vývojár uľahčenú prácu, keďže nemusí pracovať s SQL.

Z dôvodu, že dáta od užívateľa nejdú priamo do databázy, ale prechádzajú ešte cez nejaký ORM framework, existuje mylná predstava, že pri použití ORM

nástrojov je aplikácia chránená pred útokmi typu injection. To ale nie je pravda. Nie je to síce tak jednoduché, ale je to možné. Snahou útočníka je využiť slabinu v kóde generovanom ORM nástrojom alebo vtedy, ak programátor použil tento framework na vloženie vlastného SQL príkazu, čo v menšej, či väčšej miere väčšina frameworkov umožňuje.

Útok je veľmi podobný SQL injection s rozdielom, že aplikácia nepoužíva JDBC [92] na priamu komunikáciu s databázou, ale používa vrstvu prístupu k dátam („data access layer“).

Napríklad vezmeme asi najpoužívanejší ORM nástroj súčasnosti, Hibernate [52]. Ten interne používa techniku predpripravených dotazov s naviazanými parametrami, čo je jednou z techník ako zabrániť SQL injection.

Nech máme aplikáciu, ktorá berie parameter id z URL adresy na získanie entity.

http://example.org/ormtest/?id=1

Kód v aplikácii na získanie tejto entity vyzerá takto

entita = entityLoad('Book', url.id);

Pridaním stringu +OR+1=1 by malo byť teoreticky možné získať všetky entity.

Výsledné SQL by malo vyzerať takto:

```
SELECT title, text
   FROM ProtectedContent
   WHERE id = 1
        OR 1=1
```

Hibernate nám ale zahlási chybu

Error Occurred While Processing Request

The value 1 OR 1=1 cannot be converted to a number.

Problém ORM nástrojov je, že síce uľahčujú prácu programátorovi, ale zďaleka nedokážu všetko, čo by mohol potrebovať. Preto napr. v Hibernate existuje jazyk HQL, ktorý je veľmi podobný jazyku SQL.

V jazyku HQL predchádzajúci dotaz vyzerá nasledovne:

entita = ORMExecuteQuery('FROM Boook WHERE id = ' + url.id);

V tomto prípade už ale aplikácia vráti všetky výsledky. Preto by sa mali používať pomenované parametre v predpripravených dotazoch.

entita = ORMExecuteQuery('FROM Boook WHERE :id = ', {id= url.id});

Tento dotaz skončí rovnako chybou ako prvý príklad [53].

2.7.2 Ochrana

Keďže tento útok je veľmi podobný SQL injection, aj detekcia a ochrana bude rovnaká.

V prípade, že aplikácia používa SQL kód (resp. ich rôzne formy špecifické pre daný nástroj) cez ORM vrstvu, je potrebné, aby sa používali naviazané parametre (bind parameters) [54].

2.8 SQL Injection

2.8.1 Popis

Podľa Nikity, Fahima a Santosha[56] je SQL injection technika útoku, ktorá môže byť použitá útočníkom na zneužitie aplikácie. Výsledkom môže byť neautorizovaný prístup do databázy alebo získanie informácií priamo z databázy. Útočník môže využiť zraniteľnosť SQL injection vzdialene bez autentifikovania v aplikácii alebo v databázy. Útoky SQL injection sú priame – útočník len vkladá škodlivý reťazec ako vstup k aplikácii na ukradnutie dôverných informácií.

SQL injection patrí do prvej desiatky najčastejších útokov na internete podľa konzorcia OWASP [45]. Jeho história siaha do konca 90. rokov, keď bol v roku 1998 opísaný [64].

Väčšina webových aplikácií berie vstup od užívateľa na vytváranie dynamických SQL dotazov.

*Query = "SELECT * FROM student WHERE sname = 'studentname' ";*

Predstavme si aplikáciu, ktorá vezme vstup študentov a zobrazí výsledok študenta. Predpokladajme, že útočník podstrčí aplikácii meno študenta takto:

nikita' OR '1'='1

SQL dotaz potom bude vyzerať:

*SELECT * FROM student WHERE sname = 'nikita' OR '1'='1*

Tento dotaz vráti všetky riadky z tabuľky student bez ohľadu na to, či sa študent volá nikita alebo nie, pretože druhá časť klauzule WHERE je vždy pravdivá.

Toto nás privádza k nasledujúcemu deleniu techník SQL injection [62]:

Tautológie:

Jednou z metód ako získať neautorizovaný prístup k dátam je vloženie tautológie do dotazu. V SQL, ak je where klauzula SELECT alebo UPDATE dotazu podmienená tautológiou, je vrátený každý riadok databázovej tabuľky. Príklad dotazu s tautológiou je spomenutý vyššie. Detekcia hľadaním tautológií je náročná, pretože SQL dovoľuje širokú paletu volaní funkcií a hodnôt. Jednoduché hľadania znaku rovnosti (n=n) je nedostatočné. Príkladom sú tautológie, 'greg' LIKE '%gr%' and 'greg'=N'greg' , ktoré sú postavené na operátoroch ako LIKE a =N, ale nie sú typickými matematickými operátormi.

UNION dotazy:

Ďalšia technika v sebe obsahuje kľúčoví slovo UNION. SQL dovoľuje spojenie dvoch dotazov a ich vrátenie ako jednej množiny. To je jeho hlavná výhoda, lebo útočník môže získať prístup k tabuľkám, ktoré inak nemusia byť prístupné. Napríklad *SELECT col1,col2,col3 FROM table1 UNION SELECT col4,col5,col6 FROM table2* vráti jednu množinu zloženú z výsledkov oboch dotazov. UNION dotaz môže byť jednoducho pripojený k pôvodnému dotazu. Problém ale spočíva v tom, že oba dotazy musia mať rovnaký počet stĺpcov a typy stĺpcov musia byť rovnaké.

Získanie počtu stĺpcov dotazu je v normálnom UNION SQL injection takmer nemožné [61]. Toto číslo je možne získať sledovaním chybových hlášok databázového servera (samozrejme len v prípade, ak ich nenahradil nejakými aplikačnými). Jednou z možností je postupné pridávanie stĺpcov do dotazu. Útočník ich pridáva dovtedy, kým sa chybová správa „column number mismatch" (*1222: The used SELECT statements have a different number of columns,* v prípade MySQL) nezmení na „column type mismatch". V tom prípade útočník získal počet stĺpcov v SQL dotaze.

Ďalšou možnosťou je pridať klauzulu ORDER BY. Tá normálne slúži na usporiadanie výsledkov dotazu a môže mať aj číselnú hodnotu, ktorá sa odkazuje na číslo stĺpca. To znamená, že prilepenie klauzuly *order by 1*, znamená, že výsledná množina sa má zotriediť podľa prvého stĺpca. Takto môže postupne pridávať útočník čísla stĺpcov, kým sa nedostane k výsledku spomínanému vyššie.

Zistenie správnych typov je tiež problém. V SQL sú 3 základné typy (číslo, reťazec a dátum). Jednou možnosťou je použitie bruteforce metódy, tu ale vzniká obrovské množstvo možností (dotaz s 8 stĺpcami má 3^8 možností). Preto je výhodnejšie použiť NULL hodnoty a postupne takto získavať dátové typy jednotlivých stĺpcov, pretože použitím NULL nevracia databázový server chybu. Takže útočník pridá nasledujúci dotaz:

UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 --

Viacero príkazov v jednej tranzakcii :

Zaujímavým bodom pri používaní UNION na výber viacerých požiadaviek je, že často je webovou stránkou použitý len prvý riadok výsledku. V tomto prípade útočník nemusí nutne profitovať, získaním viacerých záznamov.

Ďalšou možnosťou je preto použitie UNION príkazu na spojenie s iným, ktorý nevracia žiadne výsledky. Tieto nové dotazy môžu vykonávať špecifické akcie na databáze, ktoré môžu mať katastrofálne následky.

Je možné tiež použiť znak *;*, ktorý slúži na oddelenie SQL príkazov.

*SELECT * FROM users WHERE username='greg' AND password='secret' ;*

DROP TABLE users

Bude to mať za následok zmazanie tabuľky *users*. Takto sa dajú použiť aj ostatné SQL príkazy (UPDATE, DELETE, INSERT..) alebo v databáze uložené procedúry (stored procedures).

Napríklad Microsoft SQL Server má „storované“ procedúry, ktoré môžu napríklad volať systémové príkazy (xp_cmdshell(string)).

Použitie komentárov

Použitím komentárov v dotazoch sa môže útočník vyhnúť ďalším podmienkam, ktoré obsahuje SQL dotaz. Napríklad dotaz

*SELECT * FROM users WHERE username='greg' AND password='secret'*

môže byť útočníkom pozmenený vložením *admin' - -* ako užívateľského mena:

*SELECT * FROM users WHERE username='admin' - - AND password=''*

kde podmienka za – bude ignorovaná.

2.8.2 Ochrana

Existuje viacero techník ako detekovať SQL injection. Väčšina z nich ale vyžaduje, aby všetky dotazy, ktoré idú z aplikácie do databázového servera, šli cez nejakú medzivrstvu.

Jednou z techník je SQL Parse Tree Validation [62]. Z SQL dotazu sa postaví syntaktický strom, kde užívateľom zadávané dáta sa budú nachádzať len v listoch, Detekcia nebezpečného dotazu prebieha porovnaním stromu zamýšľaného dotazu a aktuálneho dotazu. V prípade odlišností je dotaz charakterizovaný ako útok a môže byť odmietnutý.

Ďalšou zaujímavou technikou je postavenie „databázového firewallu“ medzi databázový server a webovú aplikáciu [65], ktorý sa najprv v „učiacom“ móde naučí aké dotazy sú povolené a potom na základe identifikácie administrátorských a citlivých SQL príkazov a vypočítaním hodnoty risku dotazu povolí alebo zakáže dotaz.

Ako už bolo spomenuté, SQL injection sa dá použiť aj na obídenie autentifikácie aplikácie tým, že sa zakomentuje podmienka hesla ako je to vidieť na príklade vyššie. Takémuto útoku sa dá zabrániť tým, že sa do tabuľky užívateľov pridajú dva stĺpce navyše, jedným bude hašovaná hodnota užívateľského mena a druhým hašovaná hodnota jeho hesla. Teda pri prístupe do aplikácie sa najprv vypočíta hašovaná, užívateľom zadaná hodnota a tá sa potom porovná s tým, čo je uložené v databáze. Teda ak by sa útočník snažil obísť autentifikáciu tým, že by napríklad zakomentoval podmienku zhody hesla, tak hašovaná hodnota by nebola rovnaká a teda ani autentifikácia by nebola úspešná [66].

Pre programátorov sa odporúča nepoužívať dynamické SQL dotazy, ak sa už používajú, tak by mali byť hodnoty predávané ako viazané parametre (bind parameters), prípadne používanie uložených procedúr.

Najdôležitejšie, ako pri každom útoku, je vždy validovať užívateľom zadaný vstup.

2.8.3 Príklady útokov

27.3.2011 oficiálna stránka MySQL http://www.mysql.com bola hackerom TinKode kompromitovaná pomocou Blind SQL injection [94].

8.11.2010 bola tým istým hackerom kompromitovaná stránka Britského kráľovského námorníctva [95].

2.9 Cross Site Scripting

2.9.1 Same Origin Policy

Javascript umiestnený na webovej stránke môže manipulovať s objektmi tejto stránky, otvárať nové okná prehliadača, načítať v nich ďalšie stránky alebo načítať ďalšie externé objekty. Ak by skript mohol načítať ľubovoľné objekty z ľubovoľnej externej lokácie, viedlo by to k závažnému bezpečnostnému problému. Z toho dôvodu bol vyvinutý koncept Same Origin Policy. Zavádza obmedzenia, aby skript mohol pracovať len s objektmi, ktoré sú rovnakého pôvodu ako webová stránka. Rovnaký pôvod znamená, že stránka využívajúca daný skript sa zhoduje s objektom, ku ktorému skript pristupuje na základe 3 podmienok [70]:

- **Rovnaký protokol:** časť adresy špecifikujúca použitý protokol, z ktorej je skript aktivovaný sa zhoduje v rovnakej časti URL adresy pristupovaného objektu.
- **Rovnaká doména:** časť adresy špecifikujúca doménové meno URL adresy objektu, z ktorého sa pristupuje, je rovnaká ako doménové meno URL adresy pristupovaného objektu. Prístup objektu na adrese http://example.org/script.html má zakázaný prístup k objektu na adrese http://dev.example.org/script.html.
- **Rovnaký port:** port objektu, z ktorého sa pristupuje, musí byť rovnaký ako port pristupovaného objektu. Objekt na adrese http://example.org/script.html má zakázaný prístup k objektu na adrese http://example.org:8080/script.html.

2.9.2 Popis útoku

XSS je technika útoku, ktorej podstatou vloženie kódu útočníka do inštancie užívateľovho prehliadača a jeho vykonanie. Kód je v rámci prehliadača vykonaný v rámci bezpečnostného kontextu hostujúcej webovej stránky. Škodlivý kód je zvyčajne napísaný v HTML/JavaScript, ale môže byť použitý tiež VBScript, ActiveX, Java alebo Flash, v podstate v akejkoľvek technológii, ktorú podporuje prehliadač [71].

Podstatou tejto zraniteľnosti, je zobrazenie nedôveryhodných dát od užívateľa. Nastáva vtedy, ak stránka zobrazuje dáta, ktoré zadal užívateľ bez toho, aby ich nejako validovala.

Dôsledkom, v prípade úspešného útoku, môže byť ukradnutie session (ukradnutie cookie), presmerovanie užívateľa na inú adresu alebo zobrazenie podstrčeného obsahu. Cross-site scripting útoky kompromitujú vzťah dôvery medzi užívateľom a webovou stránkou [71].

Existujú 3 typy útokov:

- Perzistentný XSS
- Non-perzistentný XSS (reflected)
- DOM XSS

Perzistentný XSS

Tento typ útoku dovoľuje útočníkovi vložiť obsah do zraniteľnej webovej stránky a stáva sa trvalou súčasťou stránky. Z toho dôvodu môže byť zasiahnutý obrovský počet užívateľov (závisí to na návštevnosti stránky), pretože podvrhnutý skript je vykonaný v prehliadači bez ohľadu na typ užívateľa.

V praxi sú zasiahnutými stránkami rôzne diskusné fóra, webové rozhrania pre prístup k elektronickej pošte, sociálne siete a pod.

Nech máme nejaké diskusné fórum, ktoré vezme príspevok od užívateľa, bez filtrácie a kontroly ho uloží do databázy a potom ho zobrazuje iným užívateľom. Keď útočník vloží ako svoj príspevok

*<script>new Image().src=“*http://attacker.org/store?
cookie=*"+encodeURI(document.cookie);</script>*

ostatným užívateľom, ktorý si zobrazia tento diskusný príspevok budú ukradnuté a poslané cookies útočníkovi na jeho server. Tento sa potom môže vydávať za daného uživateľa.

Non-persistentný XSS

Princíp spočíva v tom, že aplikácia berie dáta od užívateľa a zobrazuje ich spätne v jeho prehliadači, bez kontroly alebo filtrácie (napríklad služba

internetového vyhľadávača, ktorá zobrazuje výsledky a zároveň hľadaný výraz).

V praxi sa vyskytuje veľmi často. Je to ale menej nebezpečná varianta, pretože vyžaduje spoluúčasť užívateľa, teda útočník musí vynaložiť väčšie úsilie. Dôvod je, že musí presvedčiť obeť, aby na infikovaný link klikla. Na tento účel sa využíva obvykle sociálne inžinierstvo.

Nech na stránke http://example.org/vyhladavac.php beží internetový vyhľadávač, ktorý vezme od užívateľa výraz a následne ho aj s výsledkami zobrazí v prehliadači. Nevykonáva žiadnu validáciu ani filtráciu. V prípade, že útočník podstrčí ako výraz

<script>alert('XSS');</script>

pošle aplikácie obsah adresy

http://example.org/vyhladavac.php?vyraz=*<script>alert('XSS');</script>*

na zobrazenie v prehliadači. Tam sa užívateľovi zobrazí dialógové okno s textom *XSS*.

DOM XSS

DOM XSS útok, pri ktorom je útočníkov skript vykonaný ako výsledok zmeny DOM [93] prostredia v prehliadači obete, ktorý používa originálny skript na strane klienta. Klientsky skript potom beží neočakávane [72]. To znamená, že stránka ako taká sa nezmení, ale klientský kód je vykonaný odlišne, kvôli útočníkovým zmenám.

V tom sa DOM XSS líši od predchádzajúcich dvoch typov, kde je útok umiestnený v stránke, ktorou odpovedá server na požiadavku klienta.

Nasledujúci kód je použitý na vytvorenie formulára, v ktorom si užívateľ vyberá preferovaný jazyk:

Select your language:
<select><script>
document.write("<OPTION
value=1>"+document.location.href.substring(document.location.href.indexOf("defa
ult=")+8)+"</OPTION>");
document.write("<OPTION value=2>English</OPTION>");
</script></select>
Stránka je vyvolávaná adresou URL :

http://www.some.site/page.html?default=French

Útočník podvrhne obeti nasledujúcu URL:

http://www.some.site/page.html?default=<script>alert(document.cookie)</script>

Po kliknutí obete na link odpovie server stránkou, pre ktorú prehliadač vytvára DOM objekt, v ktorom premenná document.location je nastavená na podvrhnutú stránku. Javascript kód ale neočakával, aby default parameter obsahoval HMTL tagy, a preto útočníkov skript jednoducho vypíše do stránky (DOM). Prehliadač teda pri vykresľovaní stránky vykoná kód:

alert(document.cookie)

2.9.3 Spôsoby vloženia skriptu do webovej stránky

Pridanie prostredníctvom atribútu webového objektu

Udalosti a akcie JavaScriptu je možné priamo definovať na objekte webovej stránky pomocou pridania atribútu udalosti JavaScriptu na objekte. Výhodou je jeho jednoduchosť, naopak nevýhodou je, že udalosti sa vzťahujú iba ku konkrétnym objektom, kde sú definované. Ak by sme chceli tú istú udalosť na viacerých objektoch, museli by sme ju definovať pre každý žiadaný objekt [70].

```
<html>
 <body>
  <h1 OnMouseOver="this.innerHTML='Zmenený nadpis'">
   Povodny nadpis
  </h1>
 </body>
</html>
```

Pridanie v samostatnej sekcii webovej stránky

Je možné definovať skript v samostatnej časti stránky pomocou značiek *<scrip>* a *</script>*. Každý text definovaný medzi týmito značkami je chápaný ako kód JavaScriptu a je spracovávaný pomocou JavaScript engine v prehliadači.

Výhodou je, že takto definovaná akcia môže byť uplatnená na viacerých

objektoch webovej stránky [70].

```
<html>
 <body>
  <h1>Povodny nadpis</h1>
 </body>
 <script>
  var pole = document.getElementsByTagName('h1');
  for (var i = 0; i &lt; pole.length; i++) {
   pole[i].setAttribute('onmouseover','this.innerHTML="Zmeneny nadpis"');
  }
 </script>
</html>
```

Pridanie z externého zdroja

Táto metóda je veľmi podobná predchádzajúcej. Využíva tie isté značky HTML, ale súbor je pripojený pomocou atribútu *src*, ktorý je pridaný k značke <script> [70].

```
<html>
 <body>
  <h1>Povodny nadpis</h1>
 </body>
<script src="http://example.org/script.js"></script>
</html>
```

Aktivácia pomocou bookmarkletu

Moderné prehliadače podporujú vyvolanie akcie na objekte pomocou tzv. bookmarkletu [73]. Je to krátky kód napísaný vo forme URL adresy. Sú uvedené kľúčovým slovom *javascript:*, za ktorým nasleduje reťazec kódu. K jeho aktivácii dochádza načítaním bookmarkletu ako URL adresy. Keď prehliadač zistí adresu začínajúcu kľúčovým slovom *javascript:*, až do konca reťazca ho spracováva ako JavaScript.

Keďže je to forma URL, podľa špecifikácie RFC 1738, nesmie tento kód obsahovať medzery ani iné špeciálne znaky, ktoré sú využívané v JavaScripte. Táto

restrikcia sa obchádza dekódovanám, napr. URL kódovaním [70].

```
<html>
  <body>
    <h1>Povodny nadpis</h1>
    <a href="javascript:var%20
      pole=document.getElementsByTagName(%27h1%27);
      for(var%20i=0;i%3Cpole.length;i++)
{pole[i].setAttribute(%27onmouseover%27,
     %27this.innerHTML=%22Zmˇnˇený%20nadpis%22%27);}">
          Aktivacny odkaz
    </a>
  </body>
</html>
```

Vlastnosť expression kaskádových štýlov CSS

Proprietárne riešenie spoločnosti Microsoft CSS expressions [74] (známe tiež ako dynamic properties) umožňuje pomocou vlastnosti výrazu *expression(javascript expression)* nastavovať CSS vlastnosti na dynamické hodnoty, nie na konštanty.

Tento spôsob bol predstavený s prehliadačom Internet Explorer 5, ale s príchodom Internet Exploreru 8 sa už štandardne nepodporuje. Tento spôsob fungoval len prehliadačoch od Microsoftu.

```
<html>
  <body>
      <DIV STYLE="width: expression(alert('XSS'));">
    </body>
</html>
```

2.9.4 Ochrana

Organizácia OWASP[45] navrhuje tieto kroky ako ochranu pred XSS útokmi [75]:

Nikdy nevkladať nedôveryhodné dáta okrem povolených lokácii

V prvom rade platí, že je potrebné zakázať všetko, okrem lokácii definovaných v dalších bodoch. Dôvodom je, že v HTML je veľa kontextov, kde by boli escapovacie pravidlá veľmi komplikované. Neexistuje žiaden dobrý dôvod, prečo povoliť vkladať nedôveryhodné dáta do týchto kontextov:

- priamo do <*script*> tagu - <*script*>***data***</*script*>,

- do HTML komentára - <!-- **data** -->,
- do mena atribútu - <**data** href="/test" />,

Ak už je potrebné vkladať užívateľské dáta do DOM objektu, je nutné, aby sa dodržali nasledujúce pravidlá:

- Escapovať HTML pred vložením do obsahu HTML elemntu,
- Escapovať atribúty pred vložením do HTML atribútu,
- Escapovať JavaScript pred vložením do HTML JavaScript dátových hodnôt,
- Escapovať CSS pred vložením do HTML vlastnosti štýlov,
- Escapovať URL pred vložením do URL parametrov,

Znak	Escapovaný znak
&	&amp
<	&lt
>	&gt
"	&quot
'	&#x27
/	&#x2F

2.10 Cross Site Request Forgery

2.10.1 Popis útoku

Cross Site Request Forgery funguje na opačnom princípe ako XSS. Zneužíva dôveru stránky, ktorú má voči užívateľovi narozdiel od zneužitia dôvery užívateľa k stránke ako je to pri XSS [78] (verí tomu, že požiadavok príchádza z legitímneho zdroja).

Úlohy stránky sú obyčajne späté s nejakou URL adresou, napr. http://www.example.org/delete?user=123, ktoré dovoľujú vykonať určité akcie pri požiadavke. Podstatou útoku je, že užívateľ navštívi napadnutú stránku, ktorá urobí nejakú akciu bez toho, aby o tom užívateľ vedel.

Dopad tohto útoku záleží na tom, akú rolu ma daný užívateľ v aplikácii. V

prípade normálneho užívateľa dochádza ku kompromitácii koncových dát, ale ak sa to podarí uskutočniť s administrátorským užívateľom, môže útočník získať kontrolu nad celou aplikáciou. Typicky sú napádané komunitné webové stránky alebo banky, či stránky ponúkajúce platobné akcie. Tento útok je dokonca možné previezť na stránke, ktorá používa zabezpečené pripojenie [77].

Nech na stránke http://www.example.org existuje nejaká databáza užívateľov. Pre zmazanie užívateľa je potrebné zavolať funkciu delete s identifikátorom užívateľa:

http://www.example.org/delete?user=123

Útočník podstrčí užívateľovi stránku, ktorá bude vyzerať takto:

```
<html>
  <body>
    <img src=“http://www.example.org/delete?user=123 “>
    .....
  </body>
</html>
```

Pri spracovaní stránky prehliadačom požiada o obrázok, ktorý sa má nachádzať na adrese. Tým ale s užívateľskými právami prihláseného užívateľa vyvolá akciu delete bez toho, aby o tom užívateľ vedel. Táto požiadavka pre aplikáciu vyzerá správne, keďže užívateľ je prihlásený.

2.10.2 Ochrana

Ochrana pred týmto útokom je dosť ťažká, keďže pre aplikáciu sa akcia, ktorá je od neho požadovaná, zdá úplne v poriadku, pretože užívateľ je autentifikovaný. Preto sú navrhované nasledujúce kroky:

- Predávať parametre aplikácii pomocou metódy POST nie GET,
- Kontrolovať hlavičku Referer HTTP protokolu,
- Používať autorizačný token,

Predávanie parametrov aplikácii pomocou metódy POST, nie GET

Toto odporúčanie nezabraňuje CSRF útoku, ale pomáha mu aspoň trochu predísť. Jeho nefunkčnosť spočíva v tom, že útočník môže podstrčiť skrytý formulár

so skrytými hodnotami a pomocou JavaScriptu ho aj potvrdiť bez užívateľovej interakcie [79][80].

Kontrola hlavičky Referer HTTP protokolu

Je to jedna z možností, ktorá obmedzuje účinnosť útoku (nezabraňuje úplne). Neoddeliteľnou súčasťou tohto útoku je, že ide cez užívateľov prehliadač, inak sa nepošle autorizačné cookie. Túto hodnotu nastavuje prehliadač a keby bolo možné to podvrhnúť v prehliadači, šlo by o bezpečnostnú chybu prehliadača. Nevýhodou je, že znemožňuje všetky prístupy z vonku, vrátane tých oprávnených [79].

Používanie autorizačného tokenu

Táto metóda je asi najúčinnejšia, len trochu implementačne zložitejšia. Pred prevedením každej operácie si vygenerujeme náhodný reťazec a pri jej prevedení tento reťazec skontrolujeme. Táto metóda znemožňuje používanie aplikácie vo viacerých oknách súčasne, čo môžeme riešiť dvoma spôsobmi. Buď generovať autorizačný token až v momente, keď platnosť predchádzajúceho skončila alebo ho generovať pre každú novú požiadavku. Pri jeho ukladaní do databázy namiesto do session premennej je umožnený prístup z vonku do aplikácie [79][80].

3 Užívateľská dokukementácia

Popis

Program AttackAnalyzer bol navrhnutý tak, aby dokázal detekovať čo najviac bežných útokov na webové servery. Jeho cieľom je nájsť, informovať a prípadne zablokovať potenciálny útok.

Analyzuje komunikáciu medzi webovým serverom a klientom. V tejto komunikácii vyhľadáva výrazy, ktoré by mohli indikovať, že sa klient pokúša o útok. Výrazy sú definované v programe. Boli zvolené na základe známych pokusov [76][85]. V regulárnych výrazoch vyhľadáva špeciálne znaky systémov, ktoré sa vo webovej aplikácii používajú.

Funguje ako transparentný proxy server sprostredkujúci komunikáciu medzi klientom a serverom. Môže ju pri tom analyzovať v reálnom čase alebo keď bude mať voľné systémové prostriedky.

Program tiež definuje, ako sa má správať v prípade detekcie útoku:

- Môže útok zablokovať a užívateľovi zobraziť správu o tom, že sa pokúsil o útok. V tomto prípade požiadavka klienta nedôjde na server.
- Môže daný pokus len zaznamenať a pustiť požiadavku ďalej na webový server.
- Môže sa pokúsiť „escapovať“ špeciálne znaky danej technológie, ktoré potom nebudú mať pre webovú aplikáciu formu príkazov. Napr. nahradiť jeden apostrof dvoma pre jazyk SQL.

V prípade, že je server pomerne vyťažený, je možné nastaviť analýzu komunikácie len keď na to budú voľné systémové prostriedky. Ak je v tomto prípade použitý proxy server, celá komunikácia je uložená na disk, vo vhodnú dobu vyzdvihnutá a analyzovaná. V prípade útoku je program schopný poslať mail administrátorovi o tom, kto, odkiaľ a kde sa pokúsil o útok a o aký parameter šlo.

Systémové požiadavky

Pre beh programu je nevyhnutná inštalácia Java JRE SE 1.6 [90].

Nastavenie

Nastavenie programu sa nachádza v súbore *attack_analyzer.config*, ktorý je v priečinku *./configurationFiles*. V tomto súbore sa je uložená prednastavená konfigurácia. Pre každé nastavenie obsahuje popis a všetky prípustné hodnoty daného nastavenia. Znak # na začiatku riadku určuje, že ide o komentár a bude ignorovaný.

V prílohe č.1 sa nachádza príklad konfiguračného súboru.

Spustenie programu

Program je distribuovaný v zip formáte. Pred spustením, je potrebné najprv rozbaliť archív. Adresárová štruktúra rozbaleného archívu musí byť zachovaná, aby program fungoval korektne.

Program sa spúšťa príkazom v rozbalenom adresári:

java -jar AttackAnalyzer.jar

Predtým musia byť nastavené všetky hodnoty v konfiguračnom súbore. V prípade chybnej konfigurácie bude užívateľ informovaný v konzole. Dôležité je, aby mal program write práva do priečinka, kam má zapisovať informácie o útokoch.

4 Programátorská dokumentácia

Popis

Program je napísaný v programovacom jazyku Java, čo mu umožňuje, aby bol nasadený na rôznych platformách, kde je nainštalované behové prostredie Javy. Nutnou podmienkou pre beh sú tiež príslušné práva do priečinka, kam majú byť zapisované záznamy o útokoch a práva nutné na počúvanie na definovanom porte.

Okrem štandardných knižníc Java SE 6 [90] vuyžíva balíček JavaMail API [88] verzie 1.4.4 na e-mailové notifikácie týkajúce sa útokov a Hyperic SIGAR [89], ktorý umožňuje zisťovať vyťaženosť procesora a zaplnenosť RAM, pre spúšťanie analyzátora. Tento modul je využívaný len v prípade, že je zvolená analýza komunikácie, keď sú voľné systémové prostriedky.

Moduly

Program je logicky členený do balíkov, ktoré sú zodpovedné za jednotlivé úlohy v programe.

Hlavnou triedou je trieda Main.java v balíku com.attackanalyzer.main. Má za úlohu spustiť načítanie konfigurácie a na základe nej potom spustiť server na danom porte a adrese.

Trieda Configuration ako singleton z balíka com.attackanalyzer.configuration je zodpovedná za načítanie nastavenia programu a dodávanie nastavení iným triedam počas celého programu. V prípade chybne zadanej konfigurácie je užívateľ upozornený a program nie je spustený.

V balíku com.attackanalyzer.attacks sú definované typy útokov, ktoré majú byť vyhľadávané v komunikácii. Hlavnou súčasťou týchto útokov sú výrazy, ktoré ich definujú.

Balík com.attackanalyzer.decoders obsahuje triedy, ktoré dekódujú rôzne druhy kódovaní, aby sa tak odhalili útoky, ktoré sú v inom kódovaní.

Prácu so súbormi a sieťovými „streamami“ uľahčujú triedy, ktoré sa nachádzajú v balíku com.attackanalyzer.fileoperations.

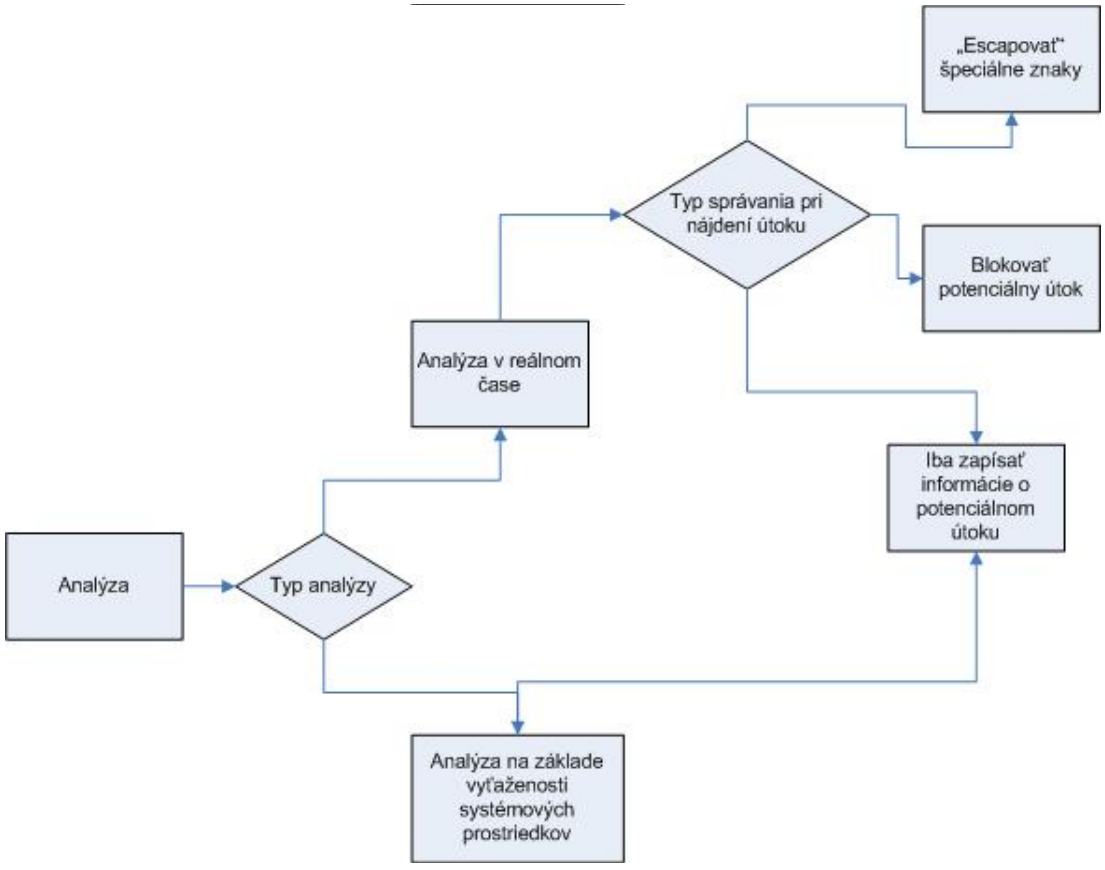
Administrátor aplikácie môže byť notifikovaný o prípadných potenciálnych útokoch e-mailom, čo má na starosti balík com.attackanalyzer.mailsending.

V prípade, že sa má analýza spúšťať až pri nižšej vyťaženosti servera, pomocou tried v com.attackanalyzer.systemusage a Hyperic SIGAR API sa zisťuje aktuálna vyťaženosť prostriedkov.

4.0 Hľadanie útokov

AttackAnalyzer je postavený pred webový server, kde zachytáva všetku komunikáciu prichádzajúcu na server. Tú na základe konfiguračného súboru buď okamžite analyzuje v reálnom čase alebo ju ukladá na pevný disk. Odtiaľ si ju vezme v prípade, ak sa trochu uvoľnia prostriedky servera.

Analýza komunikácie je popísaná v nasledujúcom obrázku č.1



Obrázok č.1: Priebeh analýzy

Pri hľadaní potenciálneho pokusu o útok sa najprv odstránia nadbytočné biele znaky ako medzery, tabulátory a pod. Potom sa použijú dekódery ako URL dekóder, Base64 dekóder atd. a až následne sa hľadajú v texte regulárne výrazy definované

v jednotlivých typoch útokov.

Obrázok č.2: Hľadanie výrazov v komunikácii

Hľadané výrazy

Najdôležitejšou úlohou bolo nadefinovať správne regulárne výrazy, ktoré budú hlásiť čo najviac potenciálnych útokov, ale zároveň budú detekovať čo najmenej falošných útokov, to znamená korektnú komunikáciu.

Hlavný problém spočíval v tom, ako detekovať úspešný a neúspešný pokus o útok. V prípade XSS útoku je to pomerne jednoduché, v odpovedi na dotaz sa výhľadáva výraz, ktorý bol označený ako potenciálny útok.

Horšia situácia nastáva s inými typmi útoku. Detekovať úspešný SQL injection útok je pomerne zložité. Problém nastáva v tom, že AttackAnalyzer nevie, aký dotaz posiela aplikácia do databázy. V prípade, že by stál ešte medzi aplikáciou a databázou, mohol by zistiť podľa tvaru dotazu, či aplikácia dokázala zabrániť útoku. Toto ale využiť nemôžeme. Ďalšou možnosťou je skúmať odpoveď servera, či v sebe neobsahuje nejakú chybovú správu databázového servera. Toto ale tiež nehovorí nič o tom, či bol útok úspešný alebo nie. Dozvieme sa len, že aplikácia je potenciálne zraniteľná voči SQL injection. Podobná situácia je s ostatnými typmi útokov.

Vzniknuté výrazy sú kombináciou príkazov, ktoré v danej technológii majú význam nejakého príkazu, úspešne prevedených útokov a potenciálnych zraniteľností, ktoré dané technológie obsahujú.

SMTP injection

Hľadanie útokov spočíva v hľadaní príkazov protokolu SMTP.

helo|ehlo|mail from:([:alnum:])|rcpt to:([:alnum:])|vrfy [:alnum:]| expn [:alnum:]

IMAP injection

Hľadanie útokov je opäť v hľadaní príkazov protokolu IMAP.

l*ist|status|(examine|select|create|delete|rename|store|copy|getquotaroot|getacl) [:alnum:]|fetch|close|expunge|capability*

SSI Injection

Detekcia spočíva vo vyhľadávaní SSI direktív [69]. Syntax SSI direktívy vyzerá takto:

<!--#element attribute=value attribute=value ... -->

Detekovať v tomto prípade sa budú SSI príkazy:

- include: prijíma parametre file, direct, virtual.
  - <!--#include virtual=“file.html“ -->
  - <!--#include file=“file.test“ -->
  - <!--#include direct=“test“ -->
- exec: prijíma parametre cgi, cmd
  - <!--#exec cmd=“ls -l“ -->
  - <!--#exec cgi=“foo.cgi“ -->
- echo: prijíma parameter var
  - <!--#echo var="REMOTE_ADDR" -->
- config:prijíma parametre timefmt, sizefmt, errmsg
  - <!--#config timefmt="%y %m %d" -->
  - <!--#config sizefmt="bytes" -->
  - <!--#config errmsg="SSI command failed!" -->
- flastmod: prijima file, virtual
  - <!--#flastmod virtual="index.html" -->
  - <!--#flastmod file="index.html" -->
- fsize: prijima file, virtual
  - <!--#fsize virtual="index.html" -->
  - <!--#fsize file="index.html" -->

- printenv
  - <!--#printenv -->

Regulárny výraz na detekciu bude nasledovný:

<!--#((include (virtual|file|direct)|exec (cgi|cmd)|echo var|config(timefmt|sizefmt|errmsg)|flastmod(file|virtual)|fsize(file|virtual))=\".+\" -->)|printenv) -->

XPath Injection

Xpath nemá komentáre, takže útočník nemôže použiť komentáre, aby odstránil ostatné podmienky. Musí skúsiť pomocou boolean výrazov odstrániť ostatné podmienky, ktoré by zabránili útoku.

Na detekciu útoku budeme vyhľadávať výrazy s or a funkcie jazyka Xpath. Tento jazyk má ale veľké množstvo funkcií a len niektoré z nich sa dajú využiť na získanie nejakých informácii alebo booleanizáciu.

or \'?[:alnum:]+\'?(<|>|<=|>=|=|)\'?[:alnum:]+|(current|node|position|count|string|compare|concat|join|substring|string-length\upper-case|lower-case|contains|replace|true|false|name|local-name|namespace|empty|reverse|max|min|sum|last)\([:alnum:]\)

Path Traversal

V tomto prípade je detekcia dosť nachýlna k tomu, že bude hlásiť príliš veľa falošných pokusov o útok.

Výraz, ktorý sa bude vyhľadávať:

[^\.]\.\.[^\.](\/\)**|

OS Commanding

V tomto pripade sa zameriam len na Linux, teda výraz bude vyhľadávať pokusy o prístup k priečinkom, kde sa nachádzajú spustiteľné programy.

/(etc|bin|sbin|tmp|var|opt|dev)

SQL Injection

Hľadanie vzorov v užívateľom zadaných dátach

Metódou na detekciu SQL injection bez možnosti prístupu k databáze a aplikácii je hľadanie určitých vzorov vo vstupných dátach klienta. Na základe toho sa vyhodnocuje, či sa útočník nepokúša o nejaký útok. Vstupné dáta od útočníka ale môžu byť rôzne kódované, aby sa vyhli týmto filtrom. Takže je potrebné zároveň zo vstupu vymazať špeciálne znaky (napríklad null byte %00) a vstup tiež dekódovať z HEX, URL, BASE64 kódovaní, odstrániť nadbytočné biele znaky a potom v ňom nájsť vzory nasledujúcich regulárnych výrazov.

Tautológie: ako už bolo spomínané pri popise tejto techniky, je pomerne náročné urobiť výčet všetkých tautologických výrazov. Nestačí iba nájsť znamienko rovnosti. Vždy pravdivé sú aj výrazy používajúce LIKE, prípadne <,>,<=,>=. Na detekciu preto poslúži nasledujúci regulárny výraz:

\' or \'?[:alnum:]+\'?(<|>|<=|>=|=|like)\'?[:alnum:]+

Union a viacero príkazov v jednej transakcii: táto technika útoku je rozpoznávaná spôsobom, že vo vstupe bude vyhľadávať SQL príkazy INSERT, UPDATE, DELETE, SELECT, DROP, CREATE, TRUNCATE, EXEC:

(union)? (
((drop|create) (table|index|database|procedure)) [:alnum:]+|
truncate table [:alnum:]+|
update .+ set ([:alnum:]+=[:alnum:]+)(,[:alnum:]+=[:alnum:]+)|
select ([:alnum:]+|) from [:alnum:]+|*
delete +from +[:alnum:]+|
exec +[:alnum:]+)

Tento regulárny výraz vyhľadá vo vstupe všetky SQL príkazy a snahy o spustenie uložených procedúr.

Použitie komentárov: Komentáre sa nachádzajú na konci užívateľom zadaných dát, aby zakomentovali ostatné podmienky, ktoré má v sebe dotaz v aplikácii. Teda regulárny výraz vyzerá takto:

(- -|/)?*

Celkový výraz na detekciu SQL injection bez vymazania nadbytočných bielych znakov vyzerá takto:

(\' or \'?[:alnum:]+\'?(<|>|<=|>=|=|like)\'?[:alnum:]+)?
(union)?[:blank:]+(
((drop|create)[:blank:]+(table|index|database|procedure))[:blank:]

+[:alnum:]+|

truncate[:blank:]+table[:blank:]+[:alnum:]+|

update[:blank:]+.+[:blank:]+set[:blank:]+([:alnum:]+[:blank:]+=[:blank:]

+ [:alnum:]+)([:blank:]+,[:blank:]+[:alnum:]+[:blank:]+=[:blank:]

+[:alnum:]+)|

select[:blank:]+[:alnum:]+[:blank:]+from[:blank:]+[:alnum:]+|

delete[:blank:]+from[:blank:]+[:alnum:]+|

exec[:blank:]+[:alnum:]+)

(- -|/)?*

Cross Site Scripting

Detekcia spočíva vo vyhľadávaní možností ako je možné kód do stránky vložiť.

Samostatná sekcia webovej stránky

<script .>|</script>*

Attribút webového objektu

(onblur|onchange|onclick|onerror|onfocus|onkeydown|onkeypress|onkeyup|onload|

onmousedown|onmousemove|onmouseout|onmouseover|onmouseup|onresize|

onselect|onunload)=\".+\"|

(altkey|button|clientx|clienty|ctrlkey|metakey|relatedtarget|screenx|screeny|shiftkey)

**=\".+\"*

Aktivácia pomocou bookmarkletu

=[:space:]"?javascript[:space:]*:|<img src=.*>*

Vlastnosť expression kaskádových štýlov

*:[:space:]*expression[:space:]+*

Cross Site Request Forgery

Detekcia útoku je pomerne náročná, mala by prebiehať tak, že filter má k dispozícii iba HTTP komunikáciu. V tom prípade mu ostáva jedine kontrolovať hodnotu hlavičky Referer. To ale prináša nevýhody, zabraňuje prístupom z vonku aplikácie, prípadne ich hlási ako útok, aj keď to nemusí byť pravda.

Menším zlepšením tejto metódy je použitie schválených hodnôt Referer. Kde

by administrátor stránky nadefinoval filtru umiestnenia, odkiaľ je možné prijať HTTP požiadavku. V tomto prípade nastáva problém, kedy útok bude prichádzať z bezpečnej stránky a v tom prípade nebude detekovaný.

4.1 Porovnanie s ostatnými nástrojmi

Primárnym cieľom práce bolo vytvoriť systém, ktorý poskytne administrátorovi webového servera možnosť sledovať pokusy o útok na server bez toho, aby zasahoval do samotnej aplikácie (tzv. blacbox metódu). Existuje viacero nástrojov, ktoré sú určené na to, aby analyzovali „logy" webových serverov (apache-scalp [81], Logalizer Pro [82]). Tieto nástroje v logoch webového servera hľadajú určité regulárne výrazy. Ďalšou skupinou, s ktorou by sme mohli porovnávať sú proxy servery, ktoré dokážu detekovať, či komunikácia, ktorá cez neho prechádza, vykazuje známky nejakého známeho útoku. Medzi tieto nástroje by sme mohli zaradiť proxy server WebScarab[83] od viackrát spomínaného združenia OWASP [45]. Problém je, že bol určený primárne na zachytávanie komunikácie a nie ako transparentný proxy server. Cieľom programu bolo sa čo najviac priblížiť nástrojom, ktoré fungujú v rámci aplikácie, majú prístup k zdrojovému kódu. Tie poskytujú výhodu toho, že autori programu vedia, aké dotazy majú očakávať (na aké tabuľky a pod.). Týmto programom som sa snažil priblížiť technikám, ktoré dokážu pomerne presne nájsť pokus o útok, ako som popisoval v kapitole o SQL Injection.

Najvhodnejším nástrojom (z pohľadu porovnania funkcionality), ktorý blokuje a analyzuje komunikáciu je technika popísaná R. Sekarom [84]. Podľa jeho meraní zabránila všetkým pokusom o útok definovaným v AMNESIA [85] a aplikácii WebGout [86]. Tento nástroj využíva možnosť pridať pluginy do serveru Apache a IIS. Analyzuje všetku komunikáciu smerujúcu na server, v ktorej vyhľadáva určité výrazy. Potom je nasadený medzi webovou aplikáciou a databázou, kde analyzuje SQL dotaz, ktorý je vytvorený v aplikácii z užívateľových dát a poslaný do databázy. Na toto je využitý vlastný parser, ktorý s pomocou užívateľom zadaných hodnôt parsuje dotaz do databázy. Nevýhodou v porovnaní s AttackAnalyzer je, že sa dá nasadiť len na servery Apache a IIS. Výhodou je presnejšie nachádzanie pokusov o útok a vyššia rýchlosť.

Na základe môjho hľadania môžem povedať, že neexistuje nástroj, ktorý by v sebe spájal funkcionalitu, o ktorú som sa snažil ja, nástroj, ktorý bude

analyzovať, „escapovať“, blokovať, informovať administrátora webového servera o tom, že sa nejaký útočník pokúša kompromitovať aplikáciu, ktorú má na starosti a zároveň nebol závislý na žiadnom programovacom jazyku ani webovom serveri. V tomto bode je dôležité podotknúť, že je dosť pravdepodobné, že väčšina administrátorov má určité nástroje pomocou, ktorých sa snažia monitorovať situáciu na ich serveroch. Problém ale je, si takéto nástroje administrátori musia vytvárať svojpomocne a ich možnosti v detekcii rôznych typov útokov sú často značne obmedzené.

4.2 Možnosti rozšírenia

Pridanie nového typu útoku

Pre pridanie nového typu útoku je potrebné vytvoriť triedu, ktorá bude dediť od AbstractAttack.java a v tejto triede nadefinovať „patterny“, ktoré budú daný typ útoku identifikovať. Je nutné tiež pridať nový záznam vo výčtovom type AttackType.java, ktorý daný typ identifikuje podľa mena pri načítaní konfigurácie útokov.

4.3 Preklad programu

Priložený archív obsahuje súbor build.xml a pomocou programu Ant je program možné preložiť a vygenerovať dokumentáciu.

5 Záver

Veľké množstvo organizácii nesleduje online aktivity na úrovni webovej aplikácie, pritom vyzkumy ukazujú firiem zaoberajúcich sa počítačovou bezpečnosťou, že až 70% útokov sa deje na tejto úrovni. Firmy sa snažia, aby ich aplikácie neboli napadnuteľné, frameworky, ktoré používajú, sa snažia týmto útokom zabrániť, problém ale je, že stačí jedno neošetrené miesto vstupu do aplikácie, a tá môže byť celá kompromitovaná. Najhoršie na tom je, keď organizácia o kompromitácii jej služby nevie a táto zraniteľnosť je zneužívaná dlhodobo. To môže mať katastrofálne dôsledky pre firmu.

Je dôležité, aby správca aplikácie mal prehľad o tom, či sa ju niekto snaží kompromitovať, aký vstupný bod do aplikácie sa snaží zneužiť a či je úspešný, aby takto mohol vykonať preventívne kroky a minimalozovať škody, ktoré môžu nastať.

Veľa administrátorov používa svoje nástroje, ktoré monitorujú aplikáciu a snažia sa detekovať útoky. Tieto nástroje, ale analyzujú aplikáciu ad hoc, teda zabrániť prípadným útokom nevedia. Navyše nejednotnosť môže spôsobiť problém, že na nejaký typ útoku sa zabudne a opäť sme v situácii, kedy správca o nebezpečenstve nevie.

Z tohto dôvodu, je dôležitý nástroj, ktorý sa bude komplexne zameriavať na najčastejšie typy útokov, aby sa tak predišlo potenciálnym škodám.

6 ZDROJE

[1] AGUILERA DÍAZ, Vicente. Capturando y explotando servidores de correo ocultos. In Capturando *y explotando servidores de correo ocultos* [online]. [s.l.] : [s.n.], 2006/6/12 [cit. 2011-05-14]. Dostupné z WWW: <https://www.owasp.org/images/1/12/OWASP-MX_Injection.pdf>.

[2] CRISPIN, M. . The *Internet Engineering Task Force (IETF)* [online]. 2003/3 [cit. 2011-05-14]. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. Dostupné z WWW: <http://www.ietf.org/rfc/rfc3501.txt>.

[3] AGUILERA DÍAZ, Vicente. *MX Injection - Capturing and Exploiting Hidden Mail Servers*. In MX *Injection - Capturing and Exploiting Hidden Mail Servers* [online]. [s.l.] : Internet Security Auditors, S.L., 2006/12 [cit. 2011-05-14]. Dostupné z WWW: <http://www.webappsec.org/projects/articles/121106.pdf>.

[4] SquirrelMail *webmail for nuts* [online]. 2006/02/15, 2007/07/03 [cit. 2011-05-14]. IMAP injection in sqimap_mailbox_select mailbox parameter. Dostupné z WWW: <http://www.squirrelmail.org/security/issue/2006-02-15>.

[5] POSTEL, J.B. . *SIMPLE MAIL TRANSFER PROTOCOL*. 1982/8 [cit. 2011-05-14]. SIMPLE MAIL TRANSFER PROTOCOL. Dostupné z WWW: <http://www.ietf.org/rfc/rfc0821.txt>.

[6] OWASP *The Open Web Application Security Project* [online]. 2009/2/9 [cit. 2011-05-14]. Testing for IMAP/SMTP Injection. Dostupné z WWW: <https://www.owasp.org/index.php/Testing_for_IMAP/SMTP_Injection_(OWASP-DV-011)>.

[7] Cgisecurity.com [online]. 2002/1 [cit. 2011-05-14]. Header Based Exploitation: Web Statistical Software Threats. Dostupné z WWW: <http://www.cgisecurity.com/papers/header-based-exploitation.txt>.

[8] SearchSoftwareQuality.com [online]. 2006/7 [cit. 2011-05-14]. SSI injection. Dostupné z WWW: <http://searchsoftwarequality.techtarget.com/definition/SSI-injection>.

[9] The *Web Application Security Consortium Project* [online]. 2009/5, 2010/1 [cit. 2011-05-17]. SSI injection. Dostupné z WWW: <http://projects.webappsec.org/w/

page/13246964/SSI-Injection>.
[10] The *Apache Software Foundation* [online]. 1998/6 [cit. 2011-05-17]. Apache suEXEC Support. Dostupné z WWW: <http://httpd.apache.org/docs/1.3/suexec.html>.
[11] The *Apache Software Foundation* [online]. 2000/3 [cit. 2011-05-17]. Apache suEXEC Support. Dostupné z WWW: <http://httpd.apache.org/docs/2.0/suexec.html>.
[12] The *Apache Software Foundation* [online]. 1998/6 [cit. 2011-05-17]. Apache Tutorial: Introduction to Server Side Includes. Dostupné z WWW: <http://httpd.apache.org/docs/1.3/howto/ssi.html>.
[13] The *Apache Software Foundation* [online]. 2005/12 [cit. 2011-05-17]. Security Tips. Dostupné z WWW: <http://httpd.apache.org/docs/current/misc/security_tips.html#ssi>.
[14] GROSSMAN, Jeremiah. Jeremiah *Grossman* [online]. 2006/8 [cit. 2011-05-17]. SSI Injection instead of JavaScript Malware. Dostupné z WWW: <http://jeremiahgrossman.blogspot.com/2006/08/ssi-injection-instead-of-javascript.html>.
[15] COBB, Michael. SearchSecurity.com [online]. 2007/1 [cit. 2011-05-17]. How do XPath injection attacks differ from SQL injection attacks? Would SQL injection mitigating techniques protect against XPath injection attacks?. Dostupné z WWW: <http://searchsecurity.techtarget.com/answer/Do-XPath-injection-attacks-require-the-same-response-as-SQL-injections>.
[16] OWASP *The Open Web Application Security Project* [online]. 2006/10, 2009/05/27 [cit. 2011-05-17]. Testing for XPath Injection. Dostupné z WWW: <https://www.owasp.org/index.php/Testing_for_XPath_Injection_(OWASP-DV-010)>.
[17] W3C. W3C [online]. 1999/11/16 [cit. 2011-05-17]. XML Path Language (XPath) Version 1.0 . Dostupné z WWW: <http://www.w3.org/TR/xpath/>.
[18] W3C. W3C [online]. 2010/12/14 [cit. 2011-05-17]. XML Path Language (XPath) 2.0 (Second Edition) . Dostupné z WWW: <http://www.w3.org/TR/xpath20/>.
[19] KLEIN, Amit . Packet *storm* [online]. 2004/02/15 [cit. 2011-05-17]. Blind XPath Injection. Dostupné z WWW: <http://dl.packetstormsecurity.net/papers/bypass/Blind_XPath_Injection_20040518.pdf>.
[20] SEN, Robi . IBM [online]. 2007/07/17 [cit. 2011-05-17]. Avoid the dangers of

XPath injection. Dostupné z WWW: <http://www.ibm.com/developerworks/xml/library/x-xpathinjection/index.html>.

[21] OWASP *The Open Web Application Security Project* [online]. 2006/06/26, 2009/05/27 [cit. 2011-05-17]. XPATH Injection. Dostupné z WWW: <https://www.owasp.org/index.php/XPATH_Injection>.

[22] GROSSMAN, Jeremiah . WhiteHat *Security* [online]. 2007/04 [cit. 2011-05-18]. WhiteHat Security Web Application Security Risk Report . Dostupné z WWW: <https://www.whitehatsec.com/home/assets/WP041907statsreport.pdf>.

[23] Cyberarmy.net [online]. 2009/30/06 [cit. 2011-04-17]. Directory Traversal Fundamentals. Dostupné z WWW: <http://securityoverride.com/infusions/pro_download_panel/file.php?did=60&file_id=62&rct=j&q=CYBERARMY UNIVERSITY directory traversal&ei=P07-TaSHLpGD-wbAs6nZAw&usg=AFQjCNEkwripPBqUxR1E5f6GEJB7rc4DnA&sig2=OHCPwOPnX6MI-Fviu2pzmQ >.

[24] Imperva. Imperva *Protecting the Data that Drives Business* [online]. 2003/3 [cit. 2011-04-17]. Directory Traversal . Dostupné z WWW: <http://www.imperva.com/resources/glossary/directory_traversal.html>.

[25] The *Web Application Security Consortium* [online]. 2009/05, 2009/12 [cit. 2011-04-17]. Path Traversal. Dostupné z WWW: <http://projects.webappsec.org/w/page/13246952/Path-Traversal>.

[26] Cgisecurity.com [online]. 2008 [cit. 2011-04-17]. Direct SQL Commands. Dostupné z WWW: <http://www.cgisecurity.com/owasp/html/ch11s03.html>.

[27] OWASP *The Open Web Application Security Project* [online]. 2006/06/15, 2009/05/27 [cit. 2011-04-17]. Path Traversal. Dostupné z WWW: <https://www.owasp.org/index.php/Path_Traversal>.

[28] Common *Weakness Enumeration* [online]. 2010/02/16 [cit. 2011-04-25]. CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'). Dostupné z WWW: <http://cwe.mitre.org/data/definitions/22.html>.

[29] FAUST, Sacha. SPI *Dynamic* [online]. 2003 [cit. 2011-04-25]. LDAP Injection. Dostupné z WWW: <http://ebook.security-portal.cz/book/hacking_method/LDAP/LDAPinjection.pdf>.

[30] DONNELLY, Michael . LDAPman *home page* [online]. 2007 [cit. 2011-04-25]. Dostupné z WWW: <http://ldapman.org/>.

[31] HOWES, T. . The *Internet Engineering Task Force (IETF)* [online]. 2006/06 [cit. 2011-04-30]. A String Representation of LDAP Search Filters. Dostupné z WWW: <http://www.ietf.org/rfc/rfc1960.txt>.

[32] SHANKAR . Shankar's *Blog* [online]. 2008/07/08 [cit. 2011-05-17]. LDAP Injection. Dostupné z WWW: <http://blogs.oracle.com/shankar/entry/what_is_ldap_injection>.

[33] OWASP *The Open Web Application Security Project* [online]. 2006/06/09, 2009/07/04 [cit. 2011-04-29]. LDAP Injection. Dostupné z WWW: <https://www.owasp.org/index.php/LDAP_injection>.

[34] OWASP *The Open Web Application Security Project* [online]. 2006/06/30, 2009/06/03 [cit. 2011-04-19]. Preventing LDAP Injection in Java. Dostupné z WWW: <https://www.owasp.org/index.php/Preventing_LDAP_Injection_in_Java>.

[35] OWASP *The Open Web Application Security Project* [online]. 2006/10/16, 2009/06/03 [cit. 2011-05-11]. Testing for LDAP Injection (OWASP-DV-006). Dostupné z WWW: <https://www.owasp.org/index.php/Testing_for_LDAP_Injection_(OWASP-DV-006)>.

[36] The *Web Application Security Consortium* [online]. 2009/05/19, 2009/12/30 [cit. 2011-04-11]. Testing for LDAP Injection (OWASP-DV-006). Dostupné z WWW: <http://projects.webappsec.org/w/page/13246947/LDAP-Injection>.

[37] ALONSO, Chema, et al. The *Web Application Security Consortium* [online]. 2008 [cit. 2011-04-11]. LDAP Injection & Blind LDAP Injection. Dostupné z WWW: <http://www.blackhat.com/presentations/bh-europe-08/Alonso-Parada/Whitepaper/bh-eu-08-alonso-parada-WP.pdf>.

[38] http://www.directory-applications.com/ldap3_files/frame.htm

[39] RAVIKANTH, D. Ravikanth [online]. 2005/12/27 [cit. 2011-04-11]. LDAP Injection overview. Dostupné z WWW: <http://weblogs.asp.net/dvravikanth/archive/2005/12/27/434036.aspx>.

[40] TWILLMAN, Tymm. Neohapsis [online]. 1999/09/20 [cit. 2011-04-11]. Exploit for proftpd 1.2.0pre6. Dostupné z WWW: <http://archives.neohapsis.com/archives/bugtraq/1999-q3/1009.html>. (jeden z prvych format string exploits)

[41] The *Web Application Security Consortium* [online]. 2009/12/30, 2009/05/29 [cit. 2011-04-10]. OS Commanding. Dostupné z WWW: <http://projects.webappsec.org/w/page/13246950/OS-Commanding>.

[42] BARNETT, Ryan C. . Preventing *Web Attacks with Apache*. [s.l.] : Addison Wesley Professional, 2006. 624 s. Dostupné z WWW: <http://www.opensourceproject.org.cn/article.php?id=095>. ISBN 0-321-32128-6.

[43] OWASP *The Open Web Application Security Project* [online]. 2006/10/16, 2009/05/27 [cit. 2011-04-11]. Testing for Command Injection (OWASP-DV-013). Dostupné z WWW: <https://www.owasp.org/index.php/Testing_for_Command_Injection_(OWASP-DV-013)>.

[44] Fortify *Software* [online]. 2011 [cit. 2011-05-14]. XML Injection . Dostupné z WWW: <https://www.fortify.com/vulncat/en/vulncat/java/xml_injection.html>.

[45] OWASP *The Open Web Application Security Project* [online]. 2006/10/16, 2010/01/31 [cit. 2011-05-14]. Testing for XML Injection (OWASP-DV-008). Dostupné z WWW: <https://www.owasp.org/index.php/Testing_for_XML_Injection_(OWASP-DV-008)>.

[46] The *Web Application Security Consortium* [online]. 2009/05/14, 2009/12/30 [cit. 2011-05-14]. XML Injection . Dostupné z WWW: <https://www.owasp.org/index.php/Testing_for_XML_Injection_(OWASP-DV-008)>.

[47] W3C. W3C [online]. 2007/04/27 [cit. 2011-05-14]. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Dostupné z WWW: <http://www.w3.org/TR/soap12-part1/>.

[48] OWASP *The Open Web Application Security Project* [online]. 2010/04/27 [cit. 2011-05-14]. Top 10 2010-Main. Dostupné z WWW: <https://www.owasp.org/index.php/Top_10_2010-Main>.

[49] OWASP *The Open Web Application Security Project* [online]. 2007/05/12 [cit. 2011-05-14]. Top 10 2007. Dostupné z WWW: <https://www.owasp.org/index.php/Top_10_2007>.

[50] OWASP *The Open Web Application Security Project* [online]. 2007/06/05 [cit. 2011-05-14]. Top 10 2004. Dostupné z WWW: <https://www.owasp.org/index.php/Top_10_2004>.

[51] W3C. W3C [online]. 2001/03/15 [cit. 2011-05-14]. Web Services Description Language (WSDL) 1.1. Dostupné z WWW: <http://www.w3.org/TR/wsdl>.

XML *Threats and Web Services Vulnerabilities: Understanding Risk and Protection* [online]. [s.l.] : [s.n.], 2005 [cit. 2011-05-14]. Dostupné z WWW: <http://www.soahub.com/Architecture/PDF/

XML_Threats_Web_Services_Vulnerabilities.pdf>.
[52] Hibernate [online]. 2011 [cit. 2011-05-14]. Dostupné z WWW: <http://www.hibernate.org/>.
[53] DEAN, Jason . 12Robots.com [online]. 2009/11/19 [cit. 2011-05-14]. ORM (Hibernate) SQL Injection - Security Series #14. Dostupné z WWW: <http://www.12robots.com/index.cfm/2009/11/19/ORM-Hibernate-Injection--Security-Series-14>.
[54] Common *Attack Pattern Enumeration and Classification* [online]. 2009/01/12 [cit. 2011-05-14]. CAPEC-109: Object Relational Mapping Injection. Dostupné z WWW: <http://capec.mitre.org/data/definitions/109.html>.
[55] OWASP *The Open Web Application Security Project* [online]. 2006/10/16 [cit. 2011-05-14]. Testing for ORM Injection (OWASP-DV-007). Dostupné z WWW: <https://www.owasp.org/index.php/Testing_for_ORM_Injection_(OWASP-DV-007)>.
[56] Patel, Nikita; Mohammed, Fahim; Soni, Santosh. *International Journal on Computer Science & Engineering*, 2011, Vol. 3 Issue 1, p199-203, 5p, 5 Color Photographs
[57] The *Web Application Security Consortium Project* [online]. 2009/07/13 [cit. 2011-07-18]. SQL Injection. Dostupné z WWW: <http://projects.webappsec.org/w/page/13246963/SQL-Injection>.
[58] OWASP *The Open Web Application Security Project* [online]. 2006 [cit. 2011-05-18]. SQL Injection. Dostupné z WWW: <https://www.owasp.org/index.php/SQL_Injection>.
[59] OWASP *The Open Web Application Security Project* [online]. 2006 [cit. 2011-05-18]. Guide to SQL Injection . Dostupné z WWW: <https://www.owasp.org/index.php/Guide_to_SQL_Injection>.
[60] LITWIN, Paul . MSDN *Magazine* [online]. 2004 [cit. 2011-05-18]. Stop SQL Injection Attacks Before They Stop You. Dostupné z WWW: <http://msdn.microsoft.com/cs-cz/magazine/cc163917(en-us).aspx>.
[61] Imperva [online]. 2009 [cit. 2011-05-18]. Blindfolded SQL Injection. Dostupné z WWW: <http://www.imperva.com/docs/WP_Blindfolded_SQL_Injection.pdf>.
[62] G.T. Buehrer, B. W. Weide. and P. A. G. Sivilotti (2005). Using parse tree validation to prevent SQL injection attacks. *Proceedings of the 5th international*

workshop on Software engineering and middleware. Lisbon, Portugal, ACM: pp. 106-113

[63] Ferruh *mavituna* [online]. 2007 [cit. 2011-05-18]. SQL Injection Cheat Sheet. Dostupné z WWW: <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>.

[64] Rain Forest Puppy [online]. 1998/12/08 [cit. 2011-05-18]. NT Web Technology Vulnerabilities. Phrack. Dostupné z WWW: <http://www.phrack.org/issues.html?id=8&issue=54>.

[65] GreenSQL [online]. 2010 [cit. 2011-05-19]. Dostupné z WWW: <http://www.greensql.net/community/docs>.

[66] Ali, Shaukat; Rauf, Azhar; Javed, Huma. SQLIPA: *An Authentication Mechanism Against SQL Injection. European Journal of Scientific Research*, 2009/12, Vol. 38 Issue 4, p604-611.

[67] BURGHATE, Nilesh; MOOKHEY, K. K. Symantec [online]. 2010/11/02 [cit. 2011-05-19]. Detection of SQL Injection and Cross-site Scripting Attacks. Dostupné z WWW: <http://www.symantec.com/connect/articles/detection-sql-injection-and-cross-site-scripting-attacks>.

[68] Microsoft *IIS* [online]. [cit. 2011-05-20]. Dostupné z WWW: <http://www.iis.net/>.

[69] SSI *(Server Side Include)* [online]. [cit. 2011-05-20]. Dostupné z WWW: <http://www.ssi.su/>.

[70] DOBIÁŠ, Jaromír . Vybrané *aspekty bezpečnosti c webových technologií* . Brno, 2009. 45 s. Diplomová práce. Masarykova Univerzita.

[71**]** The *Web Application Security Consortium Project* [online]. 2009/05/19 [cit. 2011-05-21]. Cross Site Scripting. Dostupné z WWW: <http://projects.webappsec.org/w/page/13246920/Cross-Site-Scripting>.

[72] OWASP *The Open Web Application Security Project* [online]. 2009/01/30 [cit. 2011-05-21]. DOM Based XSS. Dostupné z WWW: <https://www.owasp.org/index.php/DOM_Based_XSS>.

[73] RFC 1738 – URL, <http://www.ietf.org/rfc/rfc1738.txt>.

[74] MSDN [online]. [cit. 2011-05-21]. About Dynamic Properties. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms537634(v=vs.85).aspx>.

[75] OWASP *The Open Web Application Security Project* [online]. 2009/01/16

[cit. 2011-06-21]. XSS (Cross Site Scripting) Prevention Cheat Sheet . Dostupné z WWW: <https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting) _Prevention_Cheat_Sheet>.

[76] Ha.ckers [online]. 2008 [cit. 2011-06-23]. XSS (Cross Site Scripting) Cheat Sheet Esp: for filter evasion. Dostupné z WWW: <http://ha.ckers.org/xss.html>.

[77] OWASP *The Open Web Application Security Project* [online]. 2009 [cit. 2011-06-23]. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet . Dostupné z WWW: <https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF) _Prevention_Cheat_Sheet>.

[78] Cgisecurity.com [online]. 2010 [cit. 2011-06-23]. The Cross-Site Request Forgery (CSRF/XSRF) FAQ . Dostupné z WWW: <http://www.cgisecurity.com/ csrf-faq.html>.

[79] VRÁNA, Jakub. PHP *triky* [online]. 2006 [cit. 2011-06-23]. Cross-Site Request Forgery . Dostupné z WWW: <http://php.vrana.cz/cross-site-request-forgery.php>.

[80] SHIFLETT, Chris. Chris *Shiflett* [online]. 2003 [cit. 2011-06-23]. Foiling Cross-Site Attacks . Dostupné z WWW: <http://shiflett.org/articles/foiling-cross-site-attacks>.

[81]: *Apache-scalp* [online]. 2008 [cit. 2011-12-04]. Apache-scalp. Dostupné z WWW: <http://code.google.com/p/apache-scalp/>.

[82]: *Software.informer* [online]. 2009 [cit. 2011-12-04]. Logalizer Pro. Dostupné z WWW: <http://logalizer-pro.software.informer.com/>.

[83]: *OWASP The Open Web Application Security Project* [online]. 2006 [cit. 2011-12-04]. OWASP WebScarab Project. Dostupné z WWW: <https://www.owasp.org/ index.php/Category:OWASP_WebScarab_Project>.

[84]: SEKAR, R. *An Efficient Black-box Technique for Defeating Web Application Attacks* [online]. [s.l.], 2008. 17 s. Stony Brook University. Dostupné z WWW: <http://seclab.cs.sunysb.edu/seclab/pubs/ndss09.pdf>.

[85]: William Halfond and Alessandro Orso. AMNESIA: Analysis and monitoring for neutralizing sql-injection. In

IEEE/ACM International Conference on Automated Software Engineering (ASE),

2005.

[86]: *OWASP The Open Web Application Security Project* [online]. 2006 [cit. 2011-12-04]. OWASP WebGoat Project. Dostupné z WWW: <http://www.owasp.org/index.php/Category: OWASP WebGoat Project>.

[87]: *Symantec* [online]. 2010/04/16 [cit. 2011-12-06]. The Symantec Internet Security Threat Report (ISTR) Volume 16 Is Here!. Dostupné z WWW: <http://www.symantec.com/connect/Internet_Security_Threat_Report_16_Now_Here>.

[88] JavaMail API. *Oracle* [online]. [cit. 2012-05-22]. Dostupné z: <http://www.oracle.com/technetwork/java/javamail/index.html>

[89] Hyperic SIGAR. *Hyperic SIGAR API* [online]. [cit. 2012-05-22]. Dostupné z: http://sourceforge.net/projects/hyperic-hq/

[90] Java SE 6. *Oracle* [online]. [cit. 2012-05-22]. Dostupné z: <http://www.oracle.com/technetwork/java/javase/jrereadme-182762.html>

[91] SSO Single sign on. The Open Group [online]. [cit. 2012-05-22]. Dostupné z: <http://www.opengroup.org/security/sso/>

[92] JDBC. Oracle [online]. [cit. 2012-05-22]. Dostupné z: <http://www.oracle.com/technetwork/java/overview-141217.html>

[93] Document Object Model (DOM). W3C[online]. [cit. 2012-05-22]. Dostupné z: <http://www.w3.org/DOM/>

[94] MySQL.com hacked via... SQL injection vuln. The Register[online]. [cit. 2012-05-22]. Dostupné z: <http://www.theregister.co.uk/2011/03/28/mysql_hack/>

[93] Royal Navy website attacked by Romanian hacker. BBC [online]. [cit. 2012-05-22]. Dostupné z: <http://www.bbc.co.uk/news/technology-11711478>

7 Prílohy

Príloha 1: Príklad konfiguračného súboru

definuje port na ktorom ma pocuvat AttackAnalyzer

Port: 9999

definuje adresu servera, pred ktorym ma AttackAnalyzer stat a ktoreho komunikaciu sledovat , kam ma preposielat kominukaciu

ServerAddress: localhost

definuje port servera, pred ktorym ma stat

ServerPort: 800

definuje adresar, kde maju byt ukladane logy o utokoch, defaultne je to priecinok programu

vystupne subory su formatu 'attackDATUM.log'

DirectoryLog: /var/www/

ATTACK MODE

#definuje ako sa ma spravat server pri prichadzajucom poziadavku a detekovanom utoku

#block - zablokuje utok a urobi o tom zaznam

#escape - escapuje specialne znaky, posle dalej a urobi o tom zaznam

#log_only - iba zaloguje

#check_when_system_free - ulozi komunikaciu a vrati sa k nej, ked bude mat server volne prostriedky

AttackMode: block

PROGRAM MODE

#definuje ako ma pracovat program, ci ma vyuzit vlastny proxy server alebo mu budu na standardny

vystup posielane poziadavky, ktore bude analyzovat

#analyze_with_server

#analyze_without_server

ProgramMode: analyze_with_server

ANALYZER MODE

#definuje chod analyzatora

#learning - webova aplikacie bezi v testovacom chode a vsetky hodnoty ziskane od uzivatela sa beru

ako bezpecne a su pridavane ako vynimky k filtrom, aby sa takto minimalizoval pocet falosnych alarmov

#analyze - na zaklade filtrov v aplikacii a ziskanych udajov z learning modu analyzuje utoky

AnalyzerMode: analyze

SMTP SERVER

#definuje udaje o smtp serveri , vyzaduje sa toto nastavenie v pripade, ze administrator chce dostavat denne statistiky alebo byt informovany o utoku

#SMTP_encryption - pripusta hodnoty none (ziadne sifrovanie), tls, ssl

#SMTP_authentication_required - hodnoty true, false; definuje ci SMTP server vyzaduje autentikaciu

#SMTP_userName - definuje prihlasovanie meno na SMTP server

#SMTP_password - definuje heslo na SMTP server

#SMTP_address - definuje adresu SMTP servera

#SMTP_port - definuje port SMTP servera

SMTP_encryption: ssl

SMTP_authentication_required: true

SMTP_userName: misom1810@gmail.com

SMTP_password: Mm5954360

SMTP_address: smtp.gmail.com

SMTP_port: 465

MAILING

#definuje kedy, komu a s akym obsahom ma posielat emaily

#posle sa mail vzdy ked program vyhodnoti, ze utok bol uspesny.

mail_on_successful_attack: true

#definuje, ci sa maju na uvedene adresy posielat denne statistiky ako priloha emailu

mail_send_statistics: false

#adresy, na ktore maju byt maily poslane oddelene medzerou

mail_addresess: mizisin.michal@gmail.com

ATTACKS

#definuje utoky, ktore sa maju analyzovat oddelene medzerou

#pripustne hodnoty su (SQL_INJECTION, XSS, CSRF , LDAP_INJECTION, MX_INJECTION, OS_COMMANDING, PATH_TRAVERSAL, SSI_INJECTION, XML_INJECTION, XPATH_INJECTION)

attacks: SQL_INJECTION XSS CSRF

Príloha 2: CD s programom a elektronickou verziou bakalárskej práce

1.) archív s programom zabalený v archíve vo formáte zip

2.) elektronická verzia bakalárskej práce