Universitatea Sapientia din Cluj-Napoca
Facultatea de Științe Tehnice și Umaniste, Târgu-Mureș
Specializarea Calculatoare

Sistem pentru Testarea Penetrabilității și Descoperirea Vulnerabilităților

Proiect de Diplomă – Extras

Coordonator științific:
Dr. Vajda Tamás

Absolvent:
Bogosi Roland

2016

UNIVERSITATEA "SAPIENTIA" din CLUJ-NAPOCA **Viza facultății:**
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Specializarea: **Calculatoare**

LUCRARE DE DIPLOMĂ

Coordonator științific:	Candidat: **Bogosi Roland**
ș.l. dr. ing. Vajda Tamás	Anul absolvirii: **2016**

a) Tema lucrării de licență:

Sistem pentru Testarea Penetrabilității și Detectarea Vulnerabilităților

b) Problemele principale tratate:

- Noțiuni de scanarea rețelelor
- Identificarea serviciilor
- Detectarea vulnerabilităților
- Identificarea pachetelor vulnerabile

c) Desene obligatorii:

- Schema bloc al aplicației
- Scheme secvențiale

d) Softuri obligatorii:

- Aplicație pentru scanarea rețelelor și detectarea vulnerabilităților

e) Bibliografia recomandată:

[1] Jason Bau, Elie Bursztein, Divij Gupta, John Mitchell. *State of the art: Automated black-box web application vulnerability testing*. In *Security and Privacy (SP), 2010 IEEE Symposium* on. IEEE. 2010. pp. 332–345.

[2] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, J. Alex Halderman. *A Search Engine Backed by Internet-Wide Scanning*. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*. Oct 2015.

[3] Jon Erickson. *Hacking: The Art of Exploitation, 2nd Edition*. No Starch Press Series. No Starch Press. 2008. ISBN 9781593271442.

f) Termene obligatorii de consultații: săptămânal

g) Locul și durata practicii: Universitatea Sapientia,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Primit tema la data de: 31.03.2015
Termen de predare: 27.06.2016

Semnătura Director Departament **Semnătura coordonatorului**

Semnătura responsabilului programului de studiu **Semnătura candidatului**

Cuprins

Sistem pentru Testarea Penetrabilității și Descoperirea Vulnerabilităților

Extras

Scopul Proiectului

Scopul proiectului este de a dezvolta o aplicație, care poate să scaneze o rețea și să descopere vulnerabilitățile rețelei scanate.

Aplicația trebuie să fie complet autonomă în procesul de descoperirea vulnerabilităților, fără a depinde de actualizări de la dezvoltator pentru a identifica servicii și vulnerabilități noi.

Publicul țintă al aplicației trebuie să fie cât mai largă. Aplicația trebuie să fie un instrument util pentru cercetători în domeniul de securitate, pentru consultanți, dar și pentru administratori, chiar în cazul în care administratorii rețelei nu au experiențe semnificative în domeniul de securitate.

Achiziție de Date

Pentru a fi folositor cercetătorilor independenți în domeniul de securitate, aplicația trebuie să conțină metode de "OSINT" ("open-source intelligence"). Serviciile prezentate în secțiunea **6.2**, anume Shodan[**1**], Censys[**2**] și Mr Looquer[**3**], sunt candidate perfecți pentru componentul al aplicației care achiziționează date prin metode pasive.

Prin suportarea metodelor pentru a achiziționa date fără o scanare activă, cercetătorii în domeniul de securitate pot să lanseze o scanare pentru care vor primi rezultate instante, fără necesitatea de a avea o infrastructură largă pentru scanare.

În cazul contrar, consultanții în domeniul de securitate în general trebuie să lanseze o scanare împotriva unei infrastructuri private, pentru care date nu sunt disponibile pe serviciile menționate anterior. Din acest motiv, aplicația trebuie să conțină și metode active pentru achiziționarea datelor.

Indiferent de faptul că aplicația implementează mai multe moduri de a achiziționa date, ca un mijloc de redundanță, utilizatorii pot să lanseze scanarea cu o aplicație exterioară, importând rezultatele pentru procesare în final. În mod implicit, aplicația de scanare *nmap* este suportat, dar orice aplicație se poate folosi care generează reporturi în format XML care sunt compatibile cu nmap; un exemplu pentru un astfel de aplicație ar fi *masscan*.

Analizarea Datelor

Bannerele de serviciu ("service banner") care au fost colectate trebuie să fie analizate într-un mod complet autonom. Contrar cu aplicațiile prezentate în secțiunea **4**, aplicația dezvoltată nu trebuie să depindă pe extensii pentru a identifica servicii și descoperi vulnerabilități.

Aplicația trebuie să folosească bazele de date publicate de instituția *NIST*, discutat în secțiunea **2.3**. Secțiunea **6.8** prezintă componentele care procesează bannerele de serviciu folosind bazele de date menționate anterior, și asociază o nume *CPE* într-un mod complet autonom.

Ca un mijloc de redundanță, secțiunea **6.7** prezintă o metodă alternativă pentru procesarea bannerelor de serviciu. Această metodă nu folosește bazele de date menționate anterior, ci o bază de date care conține nume CPE asociate cu expresii regulate, creat și menținut de dezvoltatorul aplicației sau comunitatea.

Sugestii de Soluționare

După ce serviciile scanate au fost identificate, descoperirea vulnerabilităților este o simplă operațiune de căutarea în baza de date cu vulnerabilități *CVE*.

Din punctul acesta, aplicația trebuie să prezinte cât mai multe informații utile utilizatorului despre serviciile și vulnerabilitățile descoperite, ca și cum ar fi punctajul de CVSS al vulnerabilităților în descoperite în infrastructură pentru analiza de risc.

Ca să fie folositoare și pentru administratorii de rețea care nu au experiențe semnificative în domeniul de securitate, aplicația trebuie să încerce să genereze o listă simplă cu pașii pentru a soluționa vulnerabilitățile descoperite pe fiecare server.

În cazul serverurilor unde sistemul de operare a fost identificat cu succes, aplicația poate să genereze o linie de comandă care când este rulat pe serverul cu vulnerabilități va actualiza programele vulnerabile.

Soluții Aferente

Soluții Comerciale

Proiectul *nmap* menține o listă cu instrumentele de securitate, ordonat pe baza popularității acestora[**4**]. În această secțiune vor fi prezentate trei instrumente de penetrare dintre cele mai populare din lista menționată ulterior.

Nessus *Nessus Vulnerability Scanner*[**5**] este dezvoltat și menținut de Tenable Network Security. Inițial a fost gratuit și open-source, cu funcții opționale contra cost, dar în 2005 sursa a fost închisă. SecTools.org pune Nessus pe poziția numărul unu pe popu-

laritatea instrumentelor din această categorie. Momentan există o ediție "Community" care e gratuit, dar are câteva funcții limitate și se poate folosi numai pentru uz personal. Abonamentul anual pentru versiunea plătită începe de la $2,190 per utilizator.

OpenVAS *Open Vulnerability Assessment System*[**6**] este un fork al proiectului Nessus, din perioada în care era open-source. Momentan este dezvoltat și menținut de Greenbone Networks. SecTools.org pune OpenVAS pe poziția numărul doi pe popularitate instrumentelor din această categorie, iar popularitatea acestuia este într-o creștere continuară din cauza sursei deschise.

Nexpose *Nexpose Vulnerability Scanner*[**7**] este dezvoltat și menținut de Rapid7, o companie care este cunoscut în domeniul de securitate din cauza unui alt proiectul dezvoltate de ei, anume *Metasploit*. SecTools.org pune Nexpose pe poziția numărul trei pe popularitate instrumentelor din această categorie. Abonamentele anuale încep de la $2,000, dar există și o variantă "Community" care se poate folosii gratis pentru uz personal.

Lucrări Științifice

Anumite articole studiază eficiența al soluțiilor comerciale existente. Cele mai citate dintre aceste articole sunt [**8, 9, 10**].

Publicația [**11**] detaliază planificarea și realizarea unei sistem de teste de penetrare pentru web, iar dintr-o altă perspectivă, articolul [**12**] studiază securitatea dispozitivelor conectate la o rețea publică.

ShoVAT[**13**] prezintă o metodologie care este folosită și în aplicația dezvoltată în scopul licenței cu mici modificări, componentul fiind detaliat în secțiunea **6.8**, vorbind despre identificarea serviciilor într-un mod autonom, folosind baza de date publicat de instituția NIST.

Realizarea

Aplicația realizată în cadrul tezei și scripturile asociate sunt gratuite și open source.

Repertoriul git al aplicației principale se poate descărca de pe adresa `https://github.com/RoliSoft/Host-Scanner`, și se poate folosi sub termenii al licenței GNU General Public License version 3[**14**].

Scripturile dezvoltate pentru experimentare și prelucrarea datelor se pot descărca de pe adresa `https://github.com/RoliSoft/Host-Scanner-Scripts`, și se pot folosi sub termenii al licenței MIT license[**15**].

Scanare Activă

Scanare cu ICMP 'EchoRequest'

Protocolul *Internet Control Message Protocol* (*ICMP*) definește un pachet de tip 'EchoRequest', la care destinatarii vor răspunde automat cu un pachet de tip 'EchoReply'. Acest pachet conține seria originală a cererii și octeți atașați, pe care inițiatorul poată să folosească să facă analize statistice.

Implementația componentului folosește socluri de tip "raw" pentru a trimite pachete de tip 'EchoRequest'. Din acest motiv, utilizatorul care lansează aplicația trebuie să aibă drepturi de administrator pe diverse sisteme de operare, de exemplu Linux, Windows, și diferite distribuții de BSD.

Pachetele de acest tip nu se pot adresa la un port exact, numai la o gazdă. Din acest motiv, componentul nu poate să descopere servicii pe o gazdă, numai gazdele online într-o rețea.

Scanare cu ARP 'WhoHas'

Mecanismul *Address Resolution Protocol* (*ARP*) este responsabil pentru traducerea adreselor IPv4 în adrese MAC (și vice versa) într-o rețea locală de calculatoare.

Similar cu ICMP, acest component nu poate să descopere serviciile a unei gazde, numai gazdele online într-o rețea locală.

Scanare TCP

Scanarea porturilor TCP este realizată de componentul `TcpScanner`. Implementația curentă nu necesită drepturi de administrator de la utilizatorul care lansează aplicația, fiindcă aplicația folosește interfața standard de rețea al sistemului de operare pentru a conecta la destinatari.

Figura **1** prezintă strângerea de mână în trei al protocolului TCP la procesul de conectare urmat de procesul de deconectare. După ce aplicația cheamă funcția `connect()` al sistemului, sistemul de operare va începe procesul cerut într-un mod asincron, și va returna cu o eroare sau un soclu deschis.

După o conectare cu succes, aplicația va chema funcția `close()`, care va începe procesul de a încheia conexiunea, altfel ambele terți sunt expuse la problema de epuizarea resurselor, o problemă altfel categorizată în categoria de atacuri[**16**].

Sunt metode alternative pentru scanarea porturilor TCP, de exemplu cele bazate pe 'SYN', 'ACK' și 'FIN'[**17**], care folosesc socluri de tip "raw" pentru crearea și trimiterea pachetelor la destinatari, analizând răspunsul (sau lipsa răspunsului) în final.

Aceste metode sunt folosite în general pentru a evada logurile și a evita paravanele de protecție. Aplicația realizată în cadrul tezei este orientată spre utilizatori care sunt

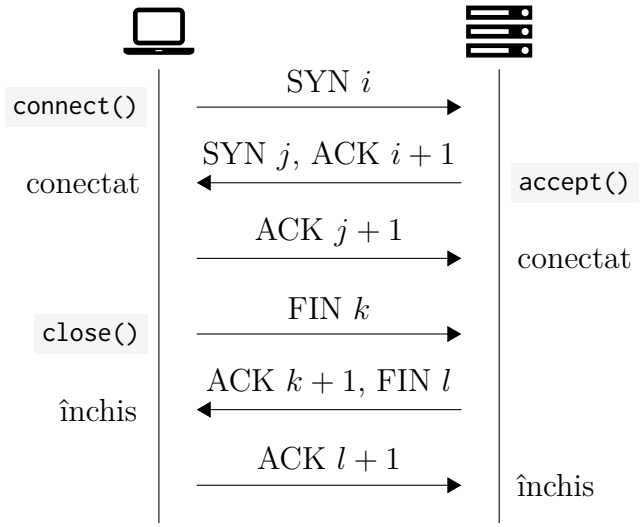


Figura 1. Strângere de mână în trei al TCP la conectare și deconectare

autorizate. Din acest motiv, aplicația nu conține metodele alternative menționate în paragraful anterior.

Scanare UDP

Protocolul UDP, contrar protocolului TCP, oferă comunicație fără conexiune. Din acest motiv, nu există o metodă sigură pentru a determina dacă există o serviciu în spatele unui port, sau nu. Chiar dacă este o aplicație asociată la portul UDP scanat, e posibil ca aceasta să nu răspundă la mesajele care nu sunt formatate corect corespunzând specificațiilor al protocolului folosit de server. Cazul preferabil este ca serviciul să răspundă cu un mesaj de eroare, care poate fi analizat și folosit la identificarea protocolului și potențial aplicației.

Anumite sisteme răspund cu un ICMP 'PortUnreachable' pachet în cazul în care în spatele portul UDP scanat nu există nici o aplicație asociată, dar majoritatea firewall-urilor filtrează mesajele de acest tip în modul explicit pentru a combate posibilitatea de a scana porturile.

Metoda cea mai bună pentru descoperea serviciilor pe porturile UDP e ca să trimitem mesaje la care este garantat că serverul (dacă rulează în spatele portului contactat) va răspunde. Componentul `UdpScanner` conține o bază de date alcătuit de numere de porturi asociate cu pachete care conțin un mesaj corect format al specificațiilor protocolului care e în general în spatele portului asociat. Ca un exemplu, fragmentul de cod **1** prezintă reprezentația hexazecimală al unui pachet UDP generat de `xxd`, care când e trimis pe portul 53 al unui gazdă care rulează un server DNS pe acest port, aceasta va răspunde cu versiunea aplicației.

```
00000000: 34ef 0100 0001 0000 0000 0000 0756 4552  4............VER
00000010: 5349 4f4e 0442 494e 4400 0010 0003       SION.BIND.....
```

Fragment de cod 1: Pachet UDP pentru solicitarea versiunii unei server DNS

Din păcate această metodă limitează lista de servicii detectabile la lista de servicii la care există un pachet de exemplu asociat în baza de date al componentei. Această metodă nu va funcționa nici în cazul în care serviciul a fost mutat de pe portul în care ar funcționa în mod implicit.

În cazul în care utilizatorul solicită scanarea a unui port care nu are un pachet asociat în baza de date al componentului, aplicația generează un pachet alcătuit de 16 octeți de `NULL`, dar acest pachet nu este garantat ca o să genereze un răspuns de la serverul în spatele portului scanat.

Dacă vrem să descoperim numai gazdele care sunt online într-o rețea, un răspuns de tip ICMP 'PortUnreachable' la un mesaj UDP la un port la întâmplare înseamnă că gazda e online, numai portul nu e. În caz contrar, un răspuns de tip ICMP 'HostUnreachable' înseamnă că gazda nu e online sau accesibilă. În general pachetul menționat anterior este generat de un router pe ruta între utilizator și rețeaua destinată.

Scanare Pasivă

Acest capitol prezintă componentul care realizează funcția de a iniția scanări pasive, prin care se înțelege o scanare care nu trimite pachete către gazdele scanate, deci scanarea astfel nu este detectabil de țintele.

Sunt servicii care scanează în continuu toate adresele publice al protocolului IPv4, și publică rezultatele într-un format accesibil. Astfel, când se solicită o scanare pasivă, aplicația nu trimite pachete către serverele numite, ci descarcă datele disponibile pentru adresele specificate de la serviciile menționate anterior.

Servicii există și pentru protocolul IPv6, dar din cauza numărului mărit de adrese de la 2^{32} până la 2^{128}, aceste servicii nu pot să scaneze toate adresele, ci trebuie să folosească trucuri creative pentru a obține liste cu adrese active.

Aplicația momentan suportă serviciile Shodan[**1**], Censys[**2**] și Mr Looquer[**3**].

Pentru a folosi aceste servicii, utilizatorul trebuie să se înregistreze la adresa fiecărui serviciu, și să obțină cheile de acces pentru API-urile fiecăruia, după ce îl poate furniza aceste chei prin argumentele `--[shodan|censys|looquer]-key` în linia de comandă.

Utilizatorul are posibilitatea de a folosi toate trei servicii concomitent, o funcție realizată în componentul `PassiveScanner`, care descarcă datele pentru adresele solicitate de la toate trei servicii.

Amalgamarea datelor este o funcție utilă, fiindcă în majoritatea cazurilor nu toate serviciile au date disponibile pentru fiecare adresă, sau în câteva cazuri mai obscure, unele dintre servicii nu au date utile din cauza a unei restricții impuse de administratorul

serverului.

Scanare Externală

Acest capitol prezintă componentul `NmapScanner`, care lansează o aplicație externală pentru a scana o gazdă și citește rezultatul scanării de la aplicația externală.

Deși aplicația realizată în cadrul licenței are componente implementate pentru scanări active, ele fiind detaliate de la secția **6.1.1** până la secția **6.1.4**, acest component asigură o redundanță în cazul în care utilizatorul vrea o a doua opinie.

Componentul suportă nativ rularea aplicației *nmap*, dar se poate folosi oricare aplicație care generează un report de XML compatibil cu cel generat de nmap. Ca exemplu, o alternativă ar fi *masscan*, pentru aplicațiile care generează reporturi compatibile și utilizabile.

Pentru folosirea aplicației nmap, executabilul `nmap` trebuie să fie accesibilă din variabila de mediu `PATH`.

Identificare Software cu Expresie Regulată

Componentul `ServiceRegexMatcher` primește ca input un banner de serviciu, și îl testează împotriva unei baze de date conținând expresii regulate asociate cu nume CPE.

```
1 ^HTTP/1\.[01] \d{3}.*\r\nServer: nginx(?:/([\d.]+))?
```

Fragment de cod 2: Expresie regulată pentru `cpe:/a:nginx:nginx`

Fragmentul de cod **2** conține o expresie regulată care se potrivește pentru răspunsul generat de serverul de HTTP numit "nginx". Expresia regulată mai conține și un grup opțional, care încearcă să extragă versiunea aplicației. În cazul în care acest grup a potrivit, componentul returnează, de exemplu, `cpe:/a:nginx:nginx:1.9.12`, sau dacă expresia s-a potrivit numai parțial, componentul returnează numai `cpe:/a:nginx:nginx`.

Această metodă diferă de la cel discutat în secțiunea **6.8**, fiindcă acest component, printre altele, poate să identifice și aplicații care nu anunță versiunea.

Identificare Software pe Bază de CPE

Institutul *National Institute of Standards and Technology* menține o bază de date sub numele de *National Vulnerability Database*, care conține vulnerabilitățile aplicaților mai populare. Componentul prezentat în acest capitol, și anume `CpeDictionaryMatcher`, folosește baza de date menționat anterior pentru identificarea aplicațiilor.

CPE este o schemă standardizată pentru a numi hardware, aplicații și sisteme de operare[**18**].

Versiunea al 2.2 este `cpe:/tip:proprietar:produs:versiune:actualizări:ediție:limbă`, unde componentul 'tip' poate să fie `h` pentru 'hardware', `o` pentru 'sistem de operare' și `a` pentru 'aplicație'.

Ca un exemplu, numele pentru "nginx 1.3.9" este `cpe:/a:igor_sysoev:nginx:1.3.9`.

Numele CPE nu sunt listate în bannerele de serviciu, și nici nu există o metodă sigură pentru a extrage o nume CPE dintr-un protocol. Publicația [**13**] abordă această problemă, iar soluția prezentată de autori este cea folosită și în aplicația realizată în cadrul licenței.

Componentul folosește dicționarul CPE publicat de NIST, pre-procesat pentru a-l converti într-un format mai avantajos la utilizarea de acest tip, și pentru a înlătura datele neutilizate din motive de performanță și economie de memorie. Fragmentul de cod **3** prezintă structura încărcată în memorie pentru o nume de CPE.

```
CpeEntry {
    // cpe:/o:cisco:ios
    "ProductSpecificTokens": ["cisco", "ios"],
    "Versions": [
        CpeVersionEntry {
            // cpe:/o:cisco:ios:12.2sxi
            "VersionNumber": "12.2",
            "VersionSpecificTokens": ["sxi"]
        }
        // [celelalte versiuni omise]
    ]
}
```

Fragment de cod 3: Structura aproximativă în memorie pentru `cpe:/o:cisco:ios:12.2sxi`

Prin procesul de identificare pe baza de CPE, structura prezentată cu token-urile al numelor CPE este reiterată în primul rând până cuvintele specifice al aplicației se potrivesc în totalitate. Dacă se găsește o potrivire completă, versiunile asociate aplicației sunt reiterate, cautând o potrivire. Dacă s-a găsit o potrivire și printre numerele de versiuni, dar există o listă de cuvinte specifice ale versiunii, și aceste trebuie să se potrivesc în totalitate ca numele CPE să fie returnat ca o potrivire.

Validarea Vulnerabilităților

Aplicația realizată suportă identificarea și folosirea al sistemelor de operare **Debian**, **Ubuntu**, **Red Hat**, **CentOS** și **Fedora**, cu suport parțial pentru **Windows**.

După scanare și analiza bannerelor de serviciu, aplicația încearcă să identifice sistemul de operare, și versiunea al sistemului de operare pentru fiecare gazdă scanată. Pentru sistemele menționate anterior cu suport complet, aplicația poate să valideze vulnerabilitățile descoperite prin rezolvarea numelor CPE identificate în nume actuale de pachete în

distribuție, urmat de descărcarea al informațiilor disponibile pentru CVE-urile descoperite pentru pachetele rezolvate.

Informațiile folosite pentru validarea vulnerabilităților astfel provin de la echipa de securitate al sistemului de operare sau de la baza de date de urmărirea erorilor al menținătorul pachetului din distribuție.

Rezultate

Pentru a testa aplicația implementată, am lansat o scanare în ambele moduri (activ și pasiv) contra trei universități. Am mai lansat încă o scanare în modul pasiv contra cinci instituții financiare, unde mă așteptam ca aceste instituții să reprezinte standardul "gold" în test.

	S. Activă			**Scanare Pasivă**							
Instituția	u_1	u_2	u_3	u_1	u_2	u_3	b_1	b_2	b_3	b_4	b_5
Servicii	165	178	455	201	269	623	41	19	69	31	11
Identificabil	143	145	402	112	148	352	24	8	58	9	9
Identificat	140	137	394	109	139	348	24	8	58	9	9

Tabela 1. Nume de CPE identificabile și identificate de aplicația realizată

Tabelul **1** prezintă numărul serviciilor descoperite și identificate. Un "serviciu" reprezintă un port deschis pe gazdă. Rândul "Identificabil" reprezintă numărul serviciilor care au fost descoperite, și au trimis un service banner care să conțină informații utilizabile la identificare, de exemplu numele și versiunea softului de server. În ultimul rând, "Identificat" reprezintă numărul serviciilor care au fost identificate cu succes de aplicația realizată.

Din cele **1,410** service banner-uri valide colectate, aplicația a identificat **1,374** servicii corect, care reprezintă o rată de succes de **97.45%**.

Tabelul **2** prezintă vulnerabilitățile descoperite pentru serviciile identificate cu succes de aplicația realizată. Rândurile "Critic", "Înalt", "Mediu" și "Scăzut" reprezintă severitatea a fiecărei vulnerabilități descoperite, iar rândul "AV:N" reprezintă numărul vulnerabilităților care pot fi abuzate prin rețea.

	S. Activă			**Scanare Pasivă**							
Instituția	u_1	u_2	u_3	u_1	u_2	u_3	b_1	b_2	b_3	b_4	b_5
Servicii	165	178	455	201	269	623	41	19	69	31	11
Critic	183	121	160	161	131	230	8	0	30	6	6
Înalt	675	414	645	583	446	826	7	0	67	21	5
Mediu	2545	1441	2775	2308	1589	3247	19	0	299	133	9
Scăzut	231	151	275	195	170	335	7	0	26	13	4
AV:N	3296	1970	3493	2961	2173	4248	40	0	393	153	22

Tabela 2. Numărul vulnerabilităților descoperite pentru serviciile identificate

Software	**CVE#**
Host Scanner[1]	166
OpenVAS	107
Nessus	68
Nexpose	311

Tabela 3. Comparație între vulnerabilitățile descoperite de diverse aplicații

Tabelul **3** prezintă o comparație între aplicația realizată în cadrul tezei și diversele aplicații comerciale prezentate în secțiunea **4.1**. Aplicațiile au fost lansate contra o rețea virtuală alcătuită de diverse mașini virtuale folosite pentru pregătirile la concursuri de tip CTF ("Capture the Flag").

Aplicațiile "OpenVAS" și "Nessus" au identificat mai puține vulnerabilități, fiindcă nu au avut extensii pentru identificarea a serviciilor mai obscure pe gazdele din rețeaua scanată.

În caz contrar, "Nexpose" a abuzat una dintre vulnerabilitățile folosite și a obținut acces shell. Având acces, aplicația a folosit managerul de pachete al sistemului de operare pentru a obține o listă cu numele și versiunea aplicațiilor instalate pe gazdă. Așa, a ajuns să descopere vulnerabilități care nu sunt accesibile/abuzabile de pe rețea, numai dacă utilizatorul are acces local.

Aplicația realizată nu are ca scop abuzarea vulnerabilităților, fiindcă acest comportament nu este în general preferabil în cazul sistemelor în producție și e ilegal pentru cercetători în domeniul de securitate fără autorizație. Validarea vulnerabilităților se face prin metoda prezentată în secțiunea **6.11**, în scurt prin identificarea sistemului de operare urmat de folosirea managerului de pachete a distribuției identificată, fără penetrare.

[1]Numele aplicației realizată în cadrul tezei.

Sapientia Erdélyi Magyar Tudományegyetem
Marosvásárhelyi Kar
Számítástechnika Szak

Behatolástesztelő és Sebezhetőségfelderítő Rendszer

Diplomadolgozat – Kivonat

Témavezető:
Dr. Vajda Tamás

Diák:
Bogosi Roland

2016

Tartalomjegyzék

Behatolástesztelő és Sebezhetőségfelderítő Rendszer

Kivonat

Projekt Célja

A projekt célja egy olyan alkalmazás fejlesztése, amely egy bizonyos hálózatot fel tud térképezni és a rajta található sebezhetőségeket fel tudja deríteni.

Az alkalmazásnak teljesen autonómnak kell lennie a felderítési folyamatában, anélkül hogy jövőbeli frissítésektől függjön az új szolgáltatások beazonosítása, illetve ezen lévő sebezhetőségek felderítése.

A projekt célközönsége a lehető legszélesebbnek kell lennie. Hasznos eszköznek kell lennie biztonsági kutatók, tanácsadók, illetve rendszergazdák számára is, még abban az esetben is, ha az utóbbi csoport nem rendelkezik jelentős biztonsági tapasztalattal.

Adatgyűjtés

Ahhoz, hogy hasznos legyen független biztonsági kutatók számára, az alkalmazásnak alkalmaznia kell „OSINT" („open-source intelligence") módszereket. A **6.2** fejezetben bemutatott szolgáltatások, név szerint Shodan[**1**], Censys[**2**] és Mr Looquer[**3**], tökéletes jelöltek az alkalmazás adatgyűjtési komponenseihez passzív feltérképezési célokra.

Ezáltal, a kutatók lekérdezéseket futtathatnak azonnali választ kapva rájuk, attól függetlenül, hogy lenne egy infrastruktúrájuk a feltérképezések lefuttatására, és be kelljen vállalják ezeknek a következményeit, mint például az „abuse" jelentéseket.

Ezzel ellentétben, a biztonsági tanácsadók általában egy belső infrastruktúrát kell feltérképezniük, amely nem elérhető az előbb említett szolgáltatások számára. Emiatt az alkalmazás egy vagy több aktív adatgyűjtési komponenst is kell tartalmazzon.

Attól függetlenül, hogy az alkalmazás implementál széleskörű protokoll-letapogatókat egyesített felülettel a feladatok párhuzamosításának céljából, redundanciaként a felhasználók számára fel van ajánlva a külső letapogatók választási lehetősége adatgyűjtés céljából. Alapértelmezetten a legnépszerűbb letapogató szoftver *nmap* van támogatva, viszont bármilyen más alkalmazás felhasználható amely nmap-kompatibilis XML-alapú reportokat generál; egy ilyen példa a *masscan* nevű szoftver.

Adatelemzés

Az összegyűjtött szolgáltatási sávok („service banner”) teljesen autonóm módon kell legyenek elemezve. Tehát a **4**. fejezetben bemutatott szoftverekkel ellentétben az alkalmazásnak nem kellene különböző kiterjesztésekre támaszkodnia a protokollok, illetve azután a protokoll mögött lévő szerverszoftverek beazonosítására.

A *NIST* által közzétett adatbázisokat (ahogyan a **2.3**. fejezetben van tárgyalva) kellene felhasználni a legújabb sebezhetőségek beazonosítása céljából. A **6.8**. fejezet bemutatja azon komponenst, amely ezt az adatbázist felhasználva önműködő módon a szolgálati sávokat a nekik megfelelő *CPE* nevükhöz társítja.

Redundanciaként, a **6.7**. fejezet bemutat egy alternatív módszert a szolgáltatási sávok CPE nevükhez való társítására, viszont ez a módszer nem támaszkodik az előzőleg említett adatbázisra, ehelyett egy a fejlesztő vagy közösség által készített/kibővített adatbázisra alapul, amely szolgáltatási sávok illeszkedésére tartalmaz reguláris kifejezéseket.

Megoldási Javaslatok

A CPE-név társításokat követően a sebezhetőségek felderítése a beazonosított szolgáltatásokban már csak egy egyszerű keresési operáció a *CVE* adatbázisban.

Ettől a ponttól követően, az alkalmazás minél több információt kell a felhasználó számára bocsátson a megtalált sebezhetőségeket illetően, beleértve a CVSS pontozást a sebezhetőség kockázatelemzése iránt a feltérképezett infrastruktúrában.

Ahhoz, hogy a szolgáltatás hasznos legyen olyan rendszergazdák számára, akiknek nincs előző jelentős biztonsági tapasztalatuk, az alkalmazás egyszerű lépéseket kell meghatározzon a sebezhetőségek kijavítására – az alapján, hogy a feltérképezett környezetről mit tudott származtatni.

Tehát, azon operációs rendszerek számára amelyek, támogatottak a szoftver által és sikeresen be is voltak azonosítva az infrastruktúrában, a szoftver tényleges parancssori utasításokat térít vissza, amelyet a rendszergazdák lefuttatva a cél rendszeren kijavíthatják ezek sebezhetőségeit.

Szoftver Sebezhetőségek

Az *RFC 2828* és számos *NIST* publikáció úgy definiálja a „sebezhetőséget,” mint „egy hiba vagy gyengeség a rendszer biztonsági procedúráiban, tervezésében, implementálásában, vagy belső ellenőrzései között, amelyet ki lehet játszani (véletlenül kiváltva vagy szándékosan kihasználva) és így az eredmény egy biztonsági rés vagy a rendszer biztonságpolitikájának megsértése.”[**19, 20**] Egyszerűen átfogalmazva ez annyit jelent, hogy a sebezhetőség egy hiba a szoftver vagy webszolgáltatás kódjában, amely kihasználáskor (például a felhasználó egy olyan bemenetet ad meg, amely szándékosan úgy lett formázva, hogy kiváltsa az ismert hibát) megengedi a felhasználó számára, hogy olyan

tevékenységet hajtson végre, amelyet különben nem szabadna, vagy olyan információhoz férjen hozzá, amelyhez különben nem kéne.

Sebezhetőség Adatbázisok

A CERT/CC által egyik szponzorált projekt a *CVE* (*Common Vulnerabilities and Exposures*), amely egy módszert biztosít a publikusan ismert sebezhetőségek címkézésére és követésére. A *NIST* (*National Institute of Standards and Technology*) alapítvány futtat egy weboldalt, *NVD* (*National Vulnerability Database*) néven, amelyen keresztül karbantartanak egy adatbázist, amely a publikusan ismert sebezhetőségeket tartalmazza, illetve információkat ezekről, olyan strukturált formátumban, amelyet számítógépes alkalmazások is önműködően olvashatnak[**21**].

Az említett adatbázis, illetve egyéb komponenseinek felhasználása bővebben a sebezhetőségkereső komponens implementációjáról szóló fejezetben van tárgyalva.

Sebezhetőségek Felderítése

A *sebezhetőségek felderítése* (angolul *vulnerability assessment*) egy olyan folyamat, amely során az infrastruktúrában lévő sebezhetőségeket beazonosítjuk és megállapítjuk a súlyosságaikat kockázatelemzés során.

Behatolás Tesztelés

A legoffenzívabb módszere a sebezhetőségek felderítésének a *behatolás tesztelés*, (angolul *penetration testing*) amely egy valós támadást szimulál az infrastruktúrán.
A behatolástesztelést a következő módokon lehet elvégezni:

- port a szerveren – egy bizonyos szolgáltatás tesztelése biztonsági frissítések megléte és helyes konfiguráció iránt (például egy SMTP szerver);
- web alkalmazás – egy teljes webes alkalmazás bejárása és tesztelése az ismert biztonsági rések ellen környezettől függően;
- teljes szerver – az összes port letapogatása és sebezhetőségkeresés indítása a válasz alapján beazonosított szolgáltatások ellen;
- teljes hálózat – egy teljes hálózat letapogatása és tesztelése, például abból a célból, hogy a hálózaton belül az összes szerver, amely titkosított információhoz juthat nem-e sebezhető.

A behatolástesztelést több megközelítésből is lehet elvégezni:

- publikus – azon támadási vektorok beazonosítására, amelyek kihasználhatóak egy kívülálló által (például a publikus IP-címeken lévő szolgáltatások);

- kliens – azon támadási vektorok beazonosítására, amelyek kihasználhatóak egy bejelentkezett, de nem emelt privilégiumokkal rendelkező felhasználó által (például egy web-banking felhasználó);
- belső – azon károk beazonosítása, amelyet egy belső személy képes okozni (például egy rossz-indulatú alkalmazott).

Behatolást Megelőző Rendszerek

A behatolástesztelés egy *aktív* módszere a sebezhetőségek felderítésének, amely egy valós támadást szimulál a hálózaton, viszont ennek *passzív* megfelelője is létezik *IDS/IPS* (*Intrusion Detection/Prevention System*) név alatt.

Ezek a rendszerek a felhasználó és az alkalmazás között helyezkednek el, és a hálózati adatforgalmat figyelik bizonyos lehetséges támadási indikátorokért. Ez a módszer eléggé korlátozott, ugyanis nem tud olyan sebezhetőségeket észlelni a rendszerben, amelyeket a felhasználó még nem próbált kihasználni, ugyanis az egyetlen adatforrása a rendszernek a valósidejű hálózati adatforgalom.

Egy másik hátránya az IDS típusú rendszereknek, hogy csak figyelnek és nem nyúlnak az adatforgalomhoz. Így értesítést tudnak küldeni egy támadásról, de azt nem előzik meg, amely bizonyos esetekben azt is jelentheti, hogy a támadó már le is töltötte a bankkártya számokat az adatbázisból.

Kapcsolódó Termékek

Kereskedelmi Megoldások

Az *nmap project* karban tart egy listát a biztonsági eszközökről, népszerűségük szerint rendezve őket[**4**]. A három legnépszerűbb sebezhetőségfelderítő termék kerül bemutatásra ebben a fejezetben.

Nessus A *Nessus Vulnerability Scanner*[**5**] a Tenable Network Security által van fejlesztve. Eredetileg ingyenes és nyílt-forrású volt, fizetős opciókkal, viszont ez 2005-ben megváltozott, amikor az alkalmazás teljesen zárt-forrású lett. A SecTools.org a Nessus-t a legnépszerűbb sebezhetőségfelderítő terméknek véli. Jelenleg egy „Community" kiadás ingyenesen kipróbálható, viszont a funkcionalitások korlátozva vannak, és csak személyes használatra használható. Éves előfizetések $2,190-tól kezdődnek felhasználónként.

OpenVAS Az *Open Vulnerability Assessment System*[**6**] egy fork-ja az utolsó Nessus verziónak, amely még nyílt-forrású volt, mielőtt zárt-forrású lett volna. Jelenleg a Greenbone Networks fejleszti. A SecTools.org az OpenVAS-t a második legnépszerűbb sebezhetőségfelderítőnek véli, és növekvőben van a népszerűsége a nyílt-forrása miatt.

Nexpose A *Nexpose Vulnerability Scanner*[**7**] a Rapid7 által van fejlesztve, akik híresek a biztonsági világban egy másik biztonsági szoftver, a *Metasploit* miatt. A SecTools.org a Nexpose-t a harmadik legnépszerűbb sebezhetőségfelderítő terméknek véli. Az éves előfizetések $2,000-tól kezdődnek, viszont van egy korlátozott „Community" kiadás is, amely ingyenes személyes használatra.

Tudományos Kutatások

Bizonyos cikkek a meglévő termékek hatékonyságát tanulmányozzák. A legidézettebb cikkek ezek közül a [**8, 9, 10**].

Új sebezhetőségfelderítő rendszerek készítéséről is találhatók cikkek, mint például a [**11**] cikkben leírt webes alkalmazások sebezhetőségfelderítője, vagy a [**12**] cikk, amely az előzővel ellentétben egy hálózaton lévő csatlakozott eszközök biztonságát vizsgálja.

A *ShoVAT*[**13**] cikk bemutat egy metodológiát, amelyet az ezen dolgozat céljából fejlesztett alkalmazás **6.8**. fejezetében tárgyalt CPE-alapú azonosító komponense is használ, kisebb változásokkal.

Kivitelezés

A dolgozat céljából fejlesztett alkalmazás és a hozzátartozó szkriptek ingyenesek és nyílt forráskódúak.

A főalkalmazás git repója elérhető a `https://github.com/RoliSoft/Host-Scanner` cím alatt, amelyet szabadon fel lehet használni, módosítani illetve terjeszteni a GNU General Public License version 3[**14**] licencszerződés feltételei alatt.

A különböző hozzátartozó szkripteket, amelyek főként adatfeldolgozási és kísérletezési célokból voltak fejlesztve, a `https://github.com/RoliSoft/Host-Scanner-Scripts` címről lehet elérni. Ezeket szabadon fel lehet használni, módosítani illetve terjeszteni az MIT license[**15**] licencszerződés feltételei alatt.

Aktív Feltérképezés

Ez a fejezet bemutatja az alkalmazás azon komponenseit, amelyek aktív feltérképezést végeznek, azaz csomagokat küldenek a feltérképezni kívánt kiszolgáló felé letapogatás céljából.

ICMP 'EchoRequest' Kérések

Az *Internet Control Message Protocol* (*ICMP*) definiál egy 'EchoRequest' csomagtípust, amelyre válaszként a címzett kiszolgáló kernele, amennyiben nincs ez explicit módon kikapcsolva, egy 'EchoReply' típusú csomagot küld vissza a küldőnek. Ebben a csomag-

ban a kapott bájt sorozat illetve a csomag sorozatszáma van visszaküldve, és ezt a küldő felhasználhatja statisztikai analízis célokból.

A `ping` parancssori program e ICMP 'EchoRequest' típusú csomagokat küld a megcímzett kiszolgáló felé a kommunikációs körút (round-trip time) meghatározása érdekében. A letapogatás kikerülése végett, az `IcmpPinger` komponens egy 32-bájtos véletlenszerű csomagot generál minden letapogatási alkalommal.

A feltérképező megvalósítás nyers csatlakozókat („raw sockets") használ az 'EchoRequest' csomagok küldése végett. Ez a funkcionalitás bizonyos operációs rendszereken, mint például Linux, Windows és különböző BSD-ken csak adminisztrátori joggal rendelkező felhasználók által érhetőek el.

Mivel az ilyen típusú „ping" csomagokat nem lehet egy bizonyos portnak megcímezni, csak egy bizonyos kiszolgálónak, ezért ezt a komponenst nem lehet használni a kiszolgálón lévő szolgáltatások feltérképezésére, csak egy hálózaton belül lévő online kiszolgálók felderítésére. Mivel az ilyen csomagok nem létfontosságúak egy bizonyos hálózat működéséhez, különböző tűzfalak bekonfigurálhatóak az ezen típusú csomagok szűrésére.

ARP 'WhoHas' Kérések

Az *Address Resolution Protocol* (*ARP*) mechanizmus feladata a hálózati réteg által használt IPv4 címek lefordítása az adatkapcsolati rétegben használt MAC címekre.

Mivel ez a mechanizmus létfontosságú egy IPv4 alapú hálózat működéséhez, ezért a tűzfalak nem avatkozhatnak be megakadályozni ezen csomagok továbbítását a hálózaton[2].

Az ICMP-hez hasonlóan, ez a funkcionalitás nem használható szolgáltatások letapogatására, csak egy hálózaton lévő kiszolgálók és kliensek feltérképezésére.

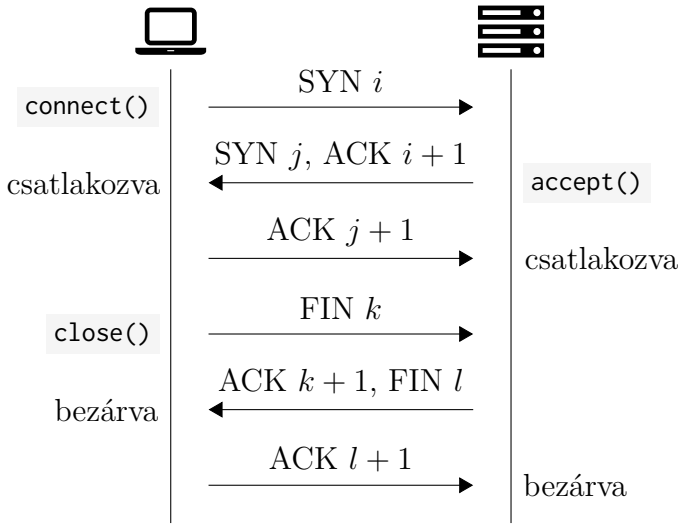
TCP Letapogatás

A TCP port letapogató funkcionalitást a `TcpScanner` komponens implementálja. A jelenlegi implementáció felhasználásához nem szükséges a felhasználónak adminisztrátori jogokkal rendelkeznie, ugyanis a letapogatás az operációs rendszer által támogatott hálózati API hívásokkal történik.

Ahogy az **1**. ábrán is látható, a TCP feltérképező komponens új kapcsolatot próbál nyitni a `connect()` függvényhívással, amely során az operációs rendszer hálózati kivitelezése egy háromutas kézfogást próbál kezdeményezni a címzettel, és visszatéríti a megnyitott csatlakozót.

Sikeres csatlakozás után a feltérképező komponens meghívja a `close()` függvényt, amely elkezdi a protokoll által előírt kapcsolatbontási folyamatot, ugyanis ellenkezőképpen a feltérképező és a feltérképezett fél is sebezhetővé válhat szolgáltatásmegtagadási támadások iránt az erőforrások kimerülése miatt[**16**].

[2]Kivéve olyan esetekben, ahol a kliensek el vannak szigetelve egymástól a hálózaton, azért, hogy egy vagy több kijelölt szerver kivételével ne tudjanak egymással kommunikálni.



1. ábra. TCP háromutas kézfogása csatlakozás és kapcsolat bontáskor

Vannak alternatív módszerek a TCP portok letapogatására, mint például 'SYN', 'ACK' és 'FIN' alapú letapogatás[**17**], amelyek esetén ahelyett, hogy az operációs rendszer hálózati kivitelezését használják a kapcsolatok létrehozására, a szoftver maga készít egy TCP csomagot és nyers csatlakoztatókon keresztül küldi, illetve fogadja a válaszokat. A nyers csatlakozók használata adminisztrátori jogokat kér a felhasználó számára Linux, Windows illetve különböző BSD disztribúciók esetén, beleértve a Mac OS X-et is.

Az ilyen alternatív letapogatási módszerek általában arra vannak használva, hogy kikerüljék a naplózást vagy átjárjanak egy tűzfalon, ugyanis amennyiben a kiszolgáló által használt tűzfal nem implementálja helyesen az állapotok nyilvántartását, megengedheti a nem-'SYN' csomagok fogadását illetve az 'RST'/'FIN' típusú csomagok küldését a védendő kiszolgáló által. Hasonlóan, amennyiben a kapcsolat nem jött létre teljesen a háromutas kézfogás során, megeshet, hogy a támadó letapogatási kísérlete nem ért olyan fázisba, ahol naplózva lenne, viszont már elég információ kiszivárgott ebben a pontban ahhoz, hogy hasznos legyen a támadó számára.

Mivel a dolgozat keretén belül kivitelezett alkalmazásnak teljes szolgáltatási sávokra van szüksége (a port állapotok listája egymagában nem elég) illetve felhatalmazott felhasználók számára volt szánva (mint például hálózati adminisztrátorok vagy biztonsági kutatók), ezért a tűzfalak átjárása illetve naplózás elkerülése nem célja az alkalmazásnak.

UDP Letapogatás

Az UDP protokoll, a TCP-vel ellentétben, kapcsolat nélküli. Ez azt vonja maga után, hogy a letapogató nem tudja megbízhatóan megállapítani, hogy a letapogatott port mögött van-e valamilyen szolgáltatás. Ha van is egy bizonyos szerver hozzárendelve az adott UDP porthoz, lehetséges, hogy a saját protokollja szerint szűri azon üzeneteket, amelyek nem felelnek meg a protokoll által előírt specifikációknak, ahelyett, hogy egy hibaüzenetet küldjön vissza, amelyet fel lehetne használni a protokoll beazonosítása céljából.

Bizonyos rendszerek válaszolnak egy ICMP 'PortUnreachable' csomaggal abban az esetben, amelyben a letapogatott port mögött nincs bármilyen szerver. Ezt a csomagot viszont a legtöbb tűzfal szűri kifejezetten a letapogatási kísérletek elhárítása miatt, vagy valahol a letapogató és letapogatott fél közötti úton egy ICMP üzenet korlátozási politika eldobja a port elérhetetlenségi üzenetet.

Emiatt a legésszerűbb módszer az ilyen típusú letapogatásra az, hogy a szolgáltatásoknak megpróbálunk olyan üzeneteket küldeni, amelyekre biztosan válaszolni fognak. Az `UdpScanner` komponense az alkalmazásnak egy adatbázist használ, amelyben bizonyos UDP példacsomagok találhatóak azon portok listájával, amelyek mögött általában ezen szolgáltatások vannak. Például az **1**. kódrészlet egy olyan UDP csomag hexadecimális reprezentációját mutatja `xxd` által generálva, amely, amikor egy 53-as portra van küldve, amely mögött egy DNS szerver található, ezen szerver válaszol a saját verziószámával.

```
1 00000000: 34ef 0100 0001 0000 0000 0000 0756 4552  4.............VER
2 00000010: 5349 4f4e 0442 494e 4400 0010 0003       SION.BIND.....
```

Kódrészlet 1: Példa UDP csomag DNS szerver verziószámának kérésére

Sajnos ez a módszer leszűkíti a letapogatható szolgáltatások számát csak azon szolgáltatásokra, amelyekhez hozzá van rendelve egy ismert csomag, amely választ vált ki. Illetve azon esetekben is csak akkor, amikor a szolgáltatás az alapértelmezett portján található. Abban az esetben, amelyben a felhasználó egy olyan UDP port letapogatását kéri, amely nem található az adatbázisban, a letapogató komponens generál egy 16 `NULL` bájtból álló csomagot, viszont annak az esélye, hogy ez a csomag választ fog kiváltani a letapogatott szolgáltatásból nagyon kevés.

Amennyiben csak azt szeretnénk kideríteni, hogy egy hálózaton belül mely kiszolgálók online-ok, egy ICMP 'PortUnreachable' válaszüzenet egy véletlenszerű UDP portra küldött csomagnak azt jelenti, hogy a kiszolgáló online, de maga a port nem, míg az ICMP 'HostUnreachable' üzenetet ezzel ellentétben, általában a letapogató és letapogatott felek közötti forgalomirányító eszköz küldhet a letapogatónak, hogy értesítse a letapogatott fél állapotáról.

Passzív Feltérképezés

Ez a fejezet az alkalmazás azon komponenseit mutatja be, amelyek passzív feltérképezést tesznek lehetővé, azaz úgy térképeznek fel egy hálózatot és a kiszolgálókon lévő szolgáltatásokat, hogy nem küldenek aktívan csomagokat a kiszolgálók felé.

Ez úgy lehetséges, hogy olyan szolgáltatásokat használnak, amelyek biztonsági kutatóknak vannak szánva, és ezek a publikus IPv4 címtartományt rendszeresen letapogatják, majd az eredményt valamilyen automatikusan felhasználható módon elérhetővé teszik, mint például letölthető adatbázis mentéssekként vagy publikusan fogyasztható API-n keresztül.

Az első ilyen támogatott szolgáltatás a Shodan[**1**], amelyik egy olyan projekt, amely folyamatosan letapogatja az IPv4 teljes címtartományát és a letapogatási eredményt egy webes keresőmotor felületen keresztül lehet lekérdezni. Egy API hozzáférés is van biztosítva, amely során a fejlesztők és biztonsági kutatók saját lekérdezéseket indíthatnak, és ezek eredményeit strukturált formátumban olvashatják be.

A második szolgáltatás a Censys[**2**], amely egy Michigani Egyetem Régensei által készített és karbantartott szolgáltatás. Hasonlóan az előző szolgáltatáshoz, az adatait Internet-szerte letapogatási műveletek során gyűjti. Az adatokhoz fejlesztők és biztonsági kutatók hozzáférhetnek strukturált lekérdezések folyamán a projekt webes felületén vagy a REST API-ján keresztül.

Végül, de nem utolsó sorban, az újonnan induló Mr Looquer[**3**] szolgáltatás is támogatva van, amely az IPv6 címtartomány feltérképezésére fókuszál. Az IPv4 címtartománnyal ellentétben, az IPv6 címtartomány 2^{32} címről 2^{128} címre nő, amely ellehetetleníti a tartomány teljes letapogatását. A szolgáltatásnak így különböző adatgyűjtési módszerekhez kell folyamodnia, amelyen heurisztikák segítségével próbál kapcsolatokat kikövetkeztetni.

Ahhoz, hogy az alkalmazásban a felhasználó használhassa a három említett szolgáltatás adatait, a felhasználónak regisztrálnia kell egy ingyenes fiókot a használni kívánt szolgáltatás weboldalán, és az ott található API kulcsot specifikálnia kell az alkalmazásnak a `--[shodan|censys|looquer]-key` parancssori argumentumon keresztül.

A felhasználónak meg van adva az a lehetőség is, hogy egy letapogatás során felhasználja mindhárom szolgáltatás adatait egyszerre. Ezt a funkcionalitást a `PassiveScanner` komponens implementálja, amely egy letapogatás során mindhárom szolgáltatás implementációját példányosítja, majd kérést kezdeményez és a visszatérített adatokat összefűzi bizonyos szabályok szerint, amelyek arra ügyelnek, hogy az a kiválasztott szolgáltatási sáv minél több információt tartalmazzon.

Az adatok egyesítése egy hasznos funkcionalitás, ugyanis a legtöbb esetben a különböző szolgáltatásokon a letapogatandó IP címekről csak parciális adat érhető el, vagy akár az sem, így előnyös, ha mindhárom szolgáltatás információit összefűzzük, és ezt az egyesített adatot elemezzük. Bizonyos esetekben az is fellép, hogy az egyik szolgáltatás ki lett tiltva a szerver rendszergazdája által a letapogatott szolgáltatásról, és így nem sikerül elemezhető adatot összegyűjtenie, a másik szolgáltatásnak viszont igen.

Külső Feltérképezés

Ez a fejezet bemutatja az alkalmazás `NmapScanner` komponensét, amely egy külső alkalmazást indít el a kért portok letapogatása érdekében, majd a külső alkalmazás által generált eredményeket beolvassa további elemzési célokból a jelenlegi alkalmazásba.

Habár az alkalmazás tág körű letapogató komponenseket implementál, amint ezek tárgyalva vannak a **6.1.1**. fejezettől a **6.1.4**. fejezetig, egyesített interfésszel amely engedélyezi a feladatok könnyű és hatékony párhuzamosítását, redundanciaként a felhasználóknak fel van ajánlva az a lehetőség is, hogy egy külső alkalmazást használjanak

a letapogatási célokra.

A komponens, amint a neve is sugallja, az *nmap* alkalmazást támogatja, viszont bármilyen más alkalmazás is használható amely nmap-kompatibilis XML-alapú jelentéseket képes generálni. Egy ilyen alternatív alkalmazásra példa a *masscan* nevű szoftver lenne.

Ahhoz, hogy az nmap-et használni lehessen letapogatásra, az `nmap` alkalmazás elérhető kell legyen a `PATH` környezeti változón belül.

Szerverek Minta-alapú Beazonosítása

A `ServiceRegexMatcher` komponens feladata, hogy bemenetként fogadja a teljes szolgáltatási sávot, és futtassa le egy adatbázis ellen, amely reguláris kifejezéseket tartalmaz CPE nevekhez társítva.

```
1 ^HTTP/1\.[01] \d{3}.*\r\nServer: nginx(?:/([\d.]+))?
```

Kódrészlet 2: Példa reguláris kifejezés `cpe:/a:nginx:nginx` szoftverhez

A **2**. kódrészletben található reguláris kifejezés illeszkedik az „nginx” HTTP szoftver által generált válaszfejlécekre. Továbbá, a reguláris kifejezés tartalmaz egy opcionális csoportot, amely a verziószámot próbálja megkeresni. Amennyiben a kifejezés illeszkedik, a komponens visszatéríti a `cpe:/a:nginx:nginx:1.9.12` nevet, míg ha a kifejezés parciálisan illeszkedik, azaz a verziószám csoport nélkül, a `cpe:/a:nginx:nginx` név lesz visszatérítve.

Ez különbözik a **6.8**. fejezetben leírt módszertől, ugyanis itt az azonosítás sikeres verziószám nélkül is, így a jövőbeli verziószámok is könnyen felismerhetőek a letapogató szoftver frissítése nélkül.

Szerverek CPE-alapú Beazonosítása

A *National Institute of Standards and Technology* karban tart egy *National Vulnerability Database* nevű adatbázist, amely tartalmazza a publikusan elismert sebezhetőségek listáját a népszerűbb szoftvereknek. Ebben a fejezetben bemutatott komponens, a `CpeDictionaryMatcher` felhasználja ezen intézet által szabadon közzétett és naponta frissített *Common Platform Enumeration Dictionary* listáját.

A *CPE* egy elnevezési rendszer hardver, szoftver illetve operációs rendszerek számára[**18**]. A 2.2-es formátumja: `cpe:/típus:tulajdonos:termék:verzió:frissítés:kiadás:nyelv` ahol a típus' komponens lehet `h` mint 'hardver', `o` mint 'operációs rendszer' és `a` mint 'alkalmazás'.

Példaként, az „nginx 1.3.9” CPE neve `cpe:/a:igor_sysoev:nginx:1.3.9`.

Sajnos a CPE nevek nincsenek a szolgáltatási sávban feltüntetve, és nem létezik egy jól definiált módszer arra, hogy a szolgáltatási sávok alapján megkeressük a hozzátartozó CPE neveket. A [**13**] cikkben a szerzők hasonló problémát tárgyaltak, és az általuk

bemutatott módszeren alapszik az alkalmazásban implementált CPE-alapú beazonosító komponens is, amely memóriába tölti a teljes CPE szótár egy előre feldolgozott változatát, amelyben a CPE nevek tokenizálva vannak és a felesleges információ el van hanyagolva memóriahatékonysági okok miatt. A **3**. kódrészlet bemutatja a memóriába betöltött struktúrát egy példa CPE névnek.

```
CpeEntry {
    // cpe:/o:cisco:ios
    "ProductSpecificTokens": ["cisco", "ios"],
    "Versions": [
        CpeVersionEntry {
            // cpe:/o:cisco:ios:12.2sxi
            "VersionNumber": "12.2",
            "VersionSpecificTokens": ["sxi"]
        }
        // [további verziók kivágva]
    ]
}
```

Kódrészlet 3: Megközelítő belső reprezentációja a `cpe:/o:cisco:ios:12.2sxi` bejegyzésnek

A tokenizációs folyamat során, a CPE név 'tulajdonos' és 'termék' komponensei a `([a-z][a-z0-9]+)` reguláris kifejezés ellen vannak illesztve azon okból fogva, hogy azokat a szavakat, amelyek több mint egy karakter hosszúak és nem számmal kezdődnek, egy külön token listába legyenek helyezve.

Így példaként, a `cpe:/a:apache:tomcat:4.1.36` CPE név alapján készül egy tömb, amely a `["apache", "tomcat"]` tokeneket tartalmazza.

A CPE verzió komponense bizonyos esetekben tartalmaz elhanyagolható karaktereket vagy non-standard verzió jelölést. Ezen esetek kiküszöbölése végett, a verziószám a `\d+\.(?:\d+\.)*\d+` reguláris kifejezés ellen van illesztve, amelynek szükséges legalább egy ponttal elválasztott két szám.

Amennyiben a verzió komponens tartalmaz ezen kívül szavakat, ezek az első tokenizációs lépésben bemutatott reguláris kifejezés ellen vannak illesztve, és egy verzió-specifikus token tömbbe tárolva. Példaként ilyen esetekben, a `cpe:/o:cisco:ios:12.2sxi` CPE név esetén, a verzió-specifikus tokenek tömb tartalma `["sxi"]` lesz.

CPE-alapú azonosítás során, a teljes adatbázis átiterálódik, először a termék-specifikus tokeneket próbálva. Amennyiben az összes token megtalálható a bemeneti szövegben, a termékhez csatolt verziószámok vannak kipróbálva. Amennyiben legalább egy verziószám talált, és van neki egy verzió-specifikus token tömbje is, az összes ilyen további tokennek is találnia kell.

Operációs Rendszerek Beazonosítása

Az alkalmazás jelenleg a **Debian**, **Ubuntu**, **Red Hat**, **CentOS** és **Fedora** disztribúciókat támogatja teljesen, ami alatt az értetendő, hogy képes ezen operációs rendszerek és verzió számuk beazonosítására, illetve a felderített CPE neveket és CVE sebezhetőségeket egy egész pontos csomag nevéhez kötni az operációs rendszeren belül.

Windows támogatása jelenleg korlátozott: az operációs rendszer felismerése támogatott, viszont egy központi csomagkezelő rendszer hiánya miatt a további funkciók nem érhetőek el ezen kiszolgálók részére.

Mindegyik operációs rendszer felismeréséjért egy külön osztály gondoskodik, amelyek algoritmusai egy nagyszámú szolgáltatási sávok tanulmányozásai folytán lettek kitervelve.

A kivonat terjedelmi korlátai miatt a különböző osztályok és azok tervezés döntési okai nem lesznek tárgyalva, viszont példaként megemlíthető, hogy az SSH protokoll előírja, hogy az SSH szervernek azonosítania kell magát kompatibilitási okok miatt, így ez a funkcionalitás egyik disztribúcióban sem kapcsolható ki.

Debian illetve Debian-alapú disztribúciók esetén az azonosítás el van túlozva, és az `openssh` csomag ki van egészítve egy `DebianBanner` opcióval, amely alapértelmezetten be van kapcsolva, és kihirdeti a feltelepített csomag egész pontos patch szintjét is.

Az „Enterprise Linux” alapú rendszerek ugyan ezzel nem rendelkeznek, viszont mivel több év is eltelik a disztribúciók között, ezért amennyiben az osztály biztos abban, hogy Red Hat/CentOS fut a kiszolgálón, a disztribúció verziója kinyerhető az SSH szerver által kihirdetett verzió generációjának illesztésével.

Sebezhetőségek Érvényesítése

A támogatott operációs rendszerek esetén, a szoftver képes a CPE nevek illetve CVE sebezhetőségek állapotát kikeresni a bizonyos disztribúció csomagkezelőjében. Ezen disztribúciók mindegyike rendelkezik egy „security tracker” csapattal vagy nyílt „build system”-mel, amelyek nyilvántartják a disztribúcióba csomagolt alkalmazások biztonsági és a bizonyos sebezhetőségek javítási állapotát.

A sebezhetőségek lekérdezése során az alkalmazás el tudja vetni azon sebezhetőségeket, amelyek nem érvényesek a beazonosított disztribúcióra (például `NOT-FOR-US` címke a Debian Security Tracker-en) vagy a beazonosított szolgáltatásra már fel lett telepítve az a biztonsági frissítés, amely kijavítja az adott sebezhetőséget.

Továbbá, ezen osztályok gondoskodnak a sebezhető csomagok nevüknek összegyűjtésére és a frissítési parancs összeállítására, amely feltelepíti a megfelelő biztonsági frissítéseket.

Utolsó Frissítési Dátum Megbecslése

Bizonyos szerverszoftverek a szolgáltatási sávjukban a verziószámuk mellett (például `PHP/5.5.12`) a biztonsági frissítés szintjét is kihirdetik (például `PHP/5.5.12-2ubuntu4.4`).

Az előző fejezetekben tárgyalt komponensek képesek ebből beazonosítani az operációs rendszert, annak verzióját, illetve annak a verziónak a csomagkezelőjéből letölteni a PHP csomag biztonsági frissítéseinek listáját.

A kiszolgáló utolsó teljes rendszerfrissítésének dátuma megbecsülhető kisebb-nagyobb pontossággal úgy, hogy egy csomag beazonosított biztonsági frissítésének kiadási dátumát, illetve ugyanazon csomagkövetkező biztonsági frissítés dátuma közötti intervallum közé helyezzük. Amennyiben több szoftver biztonsági frissítése is beazonosítható, ez leszűkíti az utolsó rendszerfrissítés dátumintervallumát.

A példában említett csomag biztonsági frissítése 2015 április 17-én lett publikálva, míg a következő frissítés meg `5.5.12-2ubuntu4.6` verziószámmal 2015 július 6-án, amint ez a changelog-ból is leolvasható: `https://launchpad.net/ubuntu/utopic/+source/php5/+changelog`

Az utolsó frissítési dátum megbecslése azért hasznos funkcionalitás, mivel megengedi azon sebezhetőségek elvetését, amelyek ki lettek javítva (azon beazonosított szolgáltatásokban, amelyek nem hirdetik ki a feltelepített biztonsági szintjüket) az utolsó rendszerfrissítés óta. Természetesen ez a funkcionalitás csak becslés, és szabadon kikapcsolható.

Eredmények

A szoftver teszteléséhez három felsőoktatási intézményt teszteltem aktív letapogatással, majd ugyan azokat passzívval is összehasonlításként. Továbbá öt egyéb pénzügyi intézményt is, amelyektől elvártam volna, hogy ezek képviseljék a „gold standard”-et a tesztben.

	Aktív L.			**Passzív Letapogatás**							
Intézmény	u_1	u_2	u_3	u_1	u_2	u_3	b_1	b_2	b_3	b_4	b_5
Szolgáltatások	165	178	455	201	269	623	41	19	69	31	11
Beazonosítható	143	145	402	112	148	352	24	8	58	9	9
Beazonosítva	140	137	394	109	139	348	24	8	58	9	9

1. táblázat. Beazonosítható illetve a megvalósított rendszer által beazonosított CPE nevek

Az **1**. táblázatban látható a letapogatás által felfedezett szolgáltatások száma intézményenként. „Szolgáltatás” alatt egy nyitott port értetendő. A „Beazonosítható” sor azon szolgáltatások számát képviseli a felfedezett szolgáltatások közül, amelyek tartalmaznak olyan információt a szolgáltatási sávjukban, amely során be lehet azonosítani a

szolgáltatás mögötti szoftvert, mint például szoftver neve és verziószáma. Végül a „Beazonosítva” sor mutatja meg a beazonosítható szolgáltatások azonosítási sikerarányát.

Az **1,410** beazonosítható szolgáltatási sávból a dolgozat körében fejlesztett alkalmazás **1,374** szolgáltatási sávot azonosított be helyesen, amely egy **97.45%** sikerességet reprezentál.

	Aktív L.			**Passzív Letapogatás**							
Intézmény	u_1	u_2	u_3	u_1	u_2	u_3	b_1	b_2	b_3	b_4	b_5
Szolgáltatások	165	178	455	201	269	623	41	19	69	31	11
Kritikus	183	121	160	161	131	230	8	0	30	6	6
Magas	675	414	645	583	446	826	7	0	67	21	5
Közepes	2545	1441	2775	2308	1589	3247	19	0	299	133	9
Alacsony	231	151	275	195	170	335	7	0	26	13	4
AV:N	3296	1970	3493	2961	2173	4248	40	0	393	153	22

2. táblázat. Megvalósított rendszer által beazonosított szolgáltatások sebezhetőségei

A **2**. táblázatban található a helyesen beazonosított szolgáltatásoknak a megvalósított alkalmazás által felderített sebezhetőségei. A „Kritikus”, „Magas”, „Közepes” és „Alacsony” sorok a felfedezett sebezhetőségek súlyosságát mutatják, míg az „AV:N” sor azon sebezhetőségek számát mutatja, amelyek távolról kihasználhatóak.

A pénzügyi intézmények IP tartományában felfedezett sebezhetőségek nem meglepőek, ugyanis egy kapcsolódó projektem[3] során mérem bizonyos intézmények válaszidejét frissen felfedezett biztonsági sebezhetőségek kijavítására, és sajnos azt sikerült levonni, hogy egyes bankok esetén a biztonsági frissítések telepítése kritikus hibákra betelhet akár 3 hónapba is.

Szoftver	**CVE#**
Host Scanner[4]	166
OpenVAS	107
Nessus	68
Nexpose	311

3. táblázat. Felderített sebezhetőségek szoftverenkénti összehasonlítása

A **3**. táblázatban a dolgozat körében fejlesztett alkalmazás, illetve a **4.1**. fejezetben listázott kereskedelmi alkalmazások le lettek futtatva egy CTF („Capture the Flag”) versenyfelkészítőre használt virtuális hálózaton összehasonlításként.

[3] `https://github.com/RoliSoft/Bank-Security-Audit`

[4] A dolgozat körében megvalósított rendszer megnevezése.

Az „OpenVAS” illetve „Nessus” gyenge eredményei azzal magyarázhatóak, hogy a kiterjesztés-alapú architektúrájuknak köszönhetően nem sikerült beazonosítaniuk bizonyos szolgáltatásokat, ugyanis nem volt kiterjesztés írva azon kevésbé népszerű szoftverek beazonosítására.

Ezzel ellentétben, a „Nexpose” kihasználva az egyik felfedezett sebezhetőséget, bejelentkezett a tesztelt kiszolgálókra, és a kiszolgálón található csomagkezelő rendszertől lekérte a feltelepített csomagokat illetve azok verziószámát. Így elérte, hogy olyan sebezhetőségeket is sikerült felderíteni, amelyek nem szerverszoftverekhez kapcsolódnak és hálózaton keresztül nem érhetőek el.

A megvalósított rendszer hatálya alá nem esik a sebezhetőségek kihasználása, ugyanis ez nem kívánatos éles rendszerek esetén, illetve törvénytelen felhatalmazás nélküli biztonsági kutatók számára. A sebezhetőségek érvényesítését a **6.11**. fejezetben leírt módszer alapján végzi az alkalmazás, a beazonosított operációs rendszer csomagkezelőjén keresztül, behatolás nélkül.

Sapientia Hungarian University of Transylvania
Faculty of Technical and Human Sciences, Târgu-Mureș
Computer Engineering

Black-Box Penetration Testing and Vulnerability Management Platform

Bachelor's Thesis

Supervisor:
Dr. Tamás Vajda

Student:
Roland Bogosi

2016

Table of Contents

List of Figures

List of Tables

List of Listings

Black-Box Penetration Testing and Vulnerability Management Platform

Abstract

Nowadays, Internet-based companies and Internet-of-Things devices are a booming business. For such services and devices, the security of their product is a crucial aspect, in order to prevent both financial damage and damage to the brand's reputation. Companies providing penetration testing and miscellaneous security services are growing with the needs of the market, as more and more small to large enterprises are outsourcing the security assurance of their infrastructure to 3^{rd}-party security contractors. This thesis presents an autonomous black-box vulnerability assessment system, which can adapt to the ever-changing landscape of network infrastructures and security policies, automatically satisfying the needs of the user in regards to their public or private infrastructure setups, and keeping up with the latest vulnerabilities and 0-day attacks straight from the source of emergency incident response teams. While the existing penetration testing tools are extensive but also expensive, the presented solution is autonomous, open-source, and free for everyone to use regardless of purpose, be it home users, security researchers or system administrators at large enterprises.

Keywords: software vulnerabilities, network scanning, penetration testing, vulnerability assessment, vulnerability validation

1 Introduction

As the world is embracing digital technologies more and more, many our services are keeping up with this trend and have started being digitally available for the purposes of easier and faster accessibility. Digital data can be processed far more quickly and reliable than its analog equivalents, which benefits the service providers by keeping the costs of providing their services low. Customers are also saved the headache of experiencing routine unpleasantries which are now alleviated by digital services, such as queuing up in order to pay the bills for various services one has subscribed to.

Service providers, however, are not the only consumers of the benefits handed to us by the Internet. Advancements made in the past decade in the fields of software, hardware, and network connectivity, have led to an exponential growth in the number of devices connected to the Internet. Due to the boom of *IoT*, ("*Internet of Things*") *Cisco* predicts there will be 25 billion devices connected to the Internet by 2015, which number is expected to increase to 50 billion by 2020.[**22**] The pool of unallocated IPv4 addresses was completely depleted when *LACNIC* has allocated their last block on the 10th of June, 2014.[**23**] This event symbolically signifies that the Internet has become way too big for what it was initially envisioned in the 1960's.

Our increasing reliance on digital services and data are not without its drawbacks, however. With more and more sensitive information and transactions being done over the Internet, one must question the security of it all. As one idiom states, "*a chain is as strong as its weakest link*," and in the context of Internet-enabled applications, there are a lot of these aforementioned figurative "*links*." Starting from the possibility of there being an attack vector on any layer of the *OSI* between the two communicating devices, down to the vulnerabilities which are specific to the application being executed over the network.

1.1 Security Awareness of IT Professionals

As one study done by *BSIMM-V* (*Building Security in Maturity Model*) which included 51 top-tier firms (mostly fortune 500) has concluded[**24**], a large percentage of product managers, software developers, and system administrators have never received security awareness training nor been instructed to lay an emphasis on security, and as a result they do not prioritize it. One of the reasons why one might *not* consider security awareness to be a priority is due to bogusly thinking that security aware application development and deployment is too costly. A study done by the *Aberdeen Group* has shown that companies who invest in security end up with up to a four-fold *ROI* ("*Return on Investments*").[**25**]

Companies have a long history of showing ignorance instead of security awareness. They are continuously claiming that they are untouchable, however, this is simply not true, and these companies have been proven wrong time and time again. One recent

example would be *General Motor*'s take on their Internet-connected cars, where their spokesperson claimed that their vehicles are not "hackable" due to the fact that their entertainment system is separate from the rest of the control systems. A vulnerability was found by two security researchers[**26**] which allowed them to inject control messages into the car's CAN bus, effectively allowing them to take control of the car via either Wi-Fi or GSM.

Software developers who would like an introduction to security-aware development can look up many of the freely available resources, however, this is not a perfect substitute for the proper training and the appropriate mindset. In some of the times, a vulnerability that was introduced by a developer might not even be their own fault, as more factors come into play in such cases. A developer might be familiar with the quirks and security issues of a specific language/environment/stack, but then be required by circumstance to move out of their comfort zone and either implement a given task in a non-standard way or a language they are not familiar enough with.

One such case would be a developer of a managed language (for example, C#) be required to interface with a non-managed language (for example, a library written in C) in which case the developer is now expected to do memory management in a language that does not promote its language management tools for it is a managed language. If said library is more complex, there are very high chances, that the developer will overlook some aspects (or not be aware they must be considered in the first place), and undesirable consequences may start happening, such as memory leaks and buffer over/underflows. In such case, it is recommended that a seasoned developer should write a wrapper for the specific library in *C++/CLI* (successor of *Managed-C++*), a language designed by Microsoft for the intent purpose[**27**] of bridging the managed and unmanaged worlds.

There are other special circumstances under this argument, for example, situations in which the developer of the application might be security-aware, or a proper penetration testing/code review was done, but the proposed security fixes might impact performance, ease of use, or even cause minor to major discomfort for the users. In such cases, risk analysis will be done, and some of these proposed solutions will be rejected as being unjustifiably too obtrusive. It is not the *right* course of action in such situations, but risk analysis[**2.4**] is a pretty standard tool for decision makers.

Another particular case would be IoT device ideas that have been funded through the use of crowdfunding platforms. These platforms, such as Kickstarter or IndieGoGo, are the hotbed of young entrepreneurial spirits with fresh ideas wanting to procure enough money in order to have a chance at executing them. Early prototypes of these projects are generally crafted using existing prototyping boards, such as Arduinos and Raspberry PIs. Later, when the proof of concept has proven to be viable, and the first batch is moved to manufacturing, a similar architecture is built so as to allow reuse of existing code. This generally means that experimental code is written in an environment in which the developer might not be comfortable enough in, is now being shipped as production code. This practice has spawned in the past lots of devices that are vulnerable and might

present a huge security risk.[**28**]

1.2 Security Awareness of Home Users

As Internet-connected devices are becoming more and more affordable and easier to use, home users have started adopting more and more of these devices. Unfortunately, home users never receive any formal security awareness training, and as such are very easily persuaded to fall for any type of attack which could have been easily prevented had they received a minimal amount of training. Even though, best-practices are usually laid out in the manual of the software being used, users tend to apply ignorance[**29**] even in cases when important notices and warnings are being displayed on the interface of the software and automatically click the "OK"/"Next" buttons, whatever gets them to what they wanted to be doing in the first place.

A study done by the *Colorado State University* has had some interesting[**30**] results. When testing a group of people self-identified as having *high* to *very high* security awareness, 68% of them ignored certificate errors on an HTTPS connection, and tried to determine the legitimacy of the said site based only on its content.

A similar survey done in the U.K. has determined[**31**] that 93% of the surveyed users has had some sort of anti-virus software installed, but only 50% have installed it for themselves, and while this last point would not be an issue in and of itself, only 37% of the respondents were applying security patches to their operating system on an at least weekly basis.

When it comes to devices designed for, and services provided to home users, the balance of ease of use and security always comes into play. Users would like an easy-to-use device or application and would rather sacrifice security in order for experiencing a mild discomfort. In one study, in the context of online banking, a user mentioned that two-factor authentication was "*not* worth spending 5 minutes for $1.99 purchases"[**32**]. In the same study, it is shown that only 27% of the users have *voluntarily* opted to use multi-factor authentication, when available.

1.3 Security Assurance in Enterprise Environments

Security is of paramount importance in enterprise environments. In such an environment, hundreds if not thousands of users (employees) depend on the IT infrastructure, and it is business-critical for these services to be functioning and data to be accessible 24/7, otherwise minor to major losses might be incurred. These hundreds or thousands of employees will all have different levels of access, controlled by strict security policies on what they can and cannot do or access.

Administrators of large organizations, however, generally have to face large amounts of bureaucracy, and need to justify every little change and downtime to their superiors. They are also on a fixed budget, which means they might not be able to use the proper

tools for a requested change, or even not being able to upgrade to a more secure version of a software or operating system, as it is "not in the budget" and their justification is rejected, as the superiors do not deem it important enough. The fixed budget is generally kept at a given level for unforeseen expenditures, such as having to adapt to possible legislation changes (for example, regarding storing sensitive customer data) or coping with new business roadmaps, trends and visions.[**33**]

Developing, deploying, and administering mission critical projects, are also faced with a few dilemmas, as their project has to stay afloat and secure for years to come, all while the world around them is evolving at an exponential pace.

Both administrators of large organizations and developers of mission critical projects therefore opt for versions of software/operating system/libraries usually annotated with *LTS* ("*Long-Term Support*") or *LTSB* ("*Long-Term Servicing Branch*"). These special versions are "feature-frozen", meaning no new features will be added, and no bugfixes will be made that break backward compatibility, but they continue to receive security patches for a long period of time, usually in the terms of years. Unfortunately, there is no known and widely adopted standard as to the expected longevity of such LTS branches, which results in the practice of each project making up its own rules that it deems reasonable enough.

Use of these special LTS versions are crucial in these cases, as *without* them, one would have to face a choice as to risk using up-to-date versions, or risk using the version with which it was originally deployed with:

- Using *up-to-date* software might break the project due to modified behavior or deprecation of functionality. As such, mission critical projects might start misbehaving or breaking completely, which might result in damages.
- Using the version that was originally *deployed with*, means that the software is not receiving any updates, which solves the issues that might present themselves down the road as outlined in the first point. However, no software is perfect, and without applying any security patches, the system now becomes vulnerable to any attacks that might be disclosed during the lifetime of the project.

One "celebrity" example for the aforementioned second point would be the *Heartbleed* vulnerability in *OpenSSL*. The project being a universal, highly platform-agnostic and trustworthy cryptography and PKI toolkit was the basis of the majority of daemon software which were using SSL/TLS sockets in the *NIX userland, and was also embedded in IoT devices that had any sort of web interface or have done cryptography in any way, shape or form. Every embedded device deployed with a vulnerable version of OpenSSL is now live and vulnerable, with no easy fix that can be applied by the vendor remotely, in most cases. This has been acknowledged by multiple leading networking gear vendors in statements saying "Cisco and Juniper cannot just press a button and immediately replace the vulnerable software running on the machines."[**34**]

1.4 Security Assurance in the Financial Sector

Figure 1. PCI Logo[**35**]

In the financial sector, major card vendors (such as Visa and MasterCard) have joined forces in order to form a council called *Payment Card Industry Security Standards Council* which would then set forth a list of strict rules and requirements for parties handling sensitive financial data. These are outlined in a document called *PCI DSS*. (*Payment Card Industry Data Security Standard*)

Any company handling sensitive cardholder information (such as merchants, payment gateways, banks, etc.) of the participating card schemes (such as Visa and MasterCard) are required to be *PCI Compliant*, which means they follow the rules and regulations set forth in the *PCI DSS* document. Compliance is then assessed either by an external *QSA* (*Qualified Security Assessor*) or, when the company is processing smaller volumes of transactions, a *SAQ* (*Self-Assessment Questionnaire*) can be filled out by the responsible persons within the enterprise. Compliances are re-assessed every year, and businesses processing sensitive cardholder information which are *not* compliant will be fined by the card schemes for every month and occurrence of non-compliance.[**36**]

The aforementioned document lays out 12 major requirements[**35**] each with its own set of checklists, in order to determine full compliance with them. For the purposes of this thesis, the following requirements will be examined and discussed:

- Requirement 6: “Develop and maintain secure systems and applications”
- Requirement 11: “Regularly test security systems and processes”

Both of these requirements have rules whose execution goes beyond the complexity that a few check boxes can convey, and as such have supplemental information available on the website of the council, in order to further clarify what the compliance requirements mean, address frequently asked questions and close eventual loopholes.

The section regarding the development and maintenance of secure systems (6^{th} requirement, further clarifications in [**37**]) requires merchants to firewall and conduct a code review of their public-facing web applications. These code reviews can either be done manually by developers, or through automated tools, which are generally known as *static source code analyzers*. After the deployment of the reviewed code to production, another requirement is the use of *WAF* (*Web Application Firewall*) whose job is to stand behind the user and the web application, trying to catch and identify any known attack vectors and common vulnerabilities.

The second section of interest, regarding the regular testing of secure systems (11^{th} requirement, further clarifications in [**38**]) requires merchants to conduct a penetration test at least yearly (and after every significant change to the infrastructure) by qualified personnel. The penetration test has to cover the entire environment storing and working with the sensitive data, has to include the testing of the *Network* and *Application layers*,

and be conducted both from an *internal* and *external perspective*, so that sensitive customer data is protected both from attacks from outside sources (such as hackers), and attacks from inside sources (such as rogue employees).

While strict rules are set forth for protecting sensitive data, merchants tend to try and get away with the least amount of strictness they can still get compliance certified for, by skipping rules marked as "best practice, not requirement" and rules which they can tick on a *technicality* due to the phrasing of the sentence. One such example would be the fact that for a long period of time, it was possible to *trick* the code review requirement by substituting automated source code analyzers with automated penetration testing, however, since version 3.0 of the security standard, it has been clarified that penetration testing is a mandatory requirement in a different section, and as such they are not a substitute for code reviews.

1.5 Government Mandated Security

The standards discussed in previous sections were all *best practices* where implementing bodies would do their *best effort* on complying, none of it was mandated[5].

Besides various "computer access acts" around the world, making unauthorized computer access punishable by law, there is not much regulation for the relevant parties regarding storing and handling sensitive information. This results in a shifting of the liability from the data holder with low security to the person accessing the system without authorization. The problematic part in this scenario is that this means once the infrastructure has been penetrated, the data is freely available for unauthorized consumption, whereas if stricter rules were mandated regarding storing sensitive data, it might not have been available in the first place.

There is, however, at least one sector in which there is prominent government mandated security. In the USA, the field of public health is regulated by *HIPAA* (*Health Insurance Portability and Accountability Act*), which was signed into law[**40**] in 1996, and amongst others, sets forth rules and requirements about the confidential handling of protected health information.

2 Software Vulnerabilities

The *RFC 2828* and many *NIST* publications define "vulnerability" as "a flaw or weakness in system security procedures, design, implementation, or internal controls that could be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system's security policy."[**19, 20**] What this means in

[5]In 2009, Nevada (USA), then a year later, the State of Washington (USA), incorporated *PCI DSS* into state law[**39**], however, the incorporated standard's sole purpose is to shield compliant parties from liability in case of a data breach.

simplified terms is, that a vulnerability is essentially a bug in the software's or web service's code, which when *exploited*, (e.g. specifying an input which was specifically crafted to be bogus and known to trigger the bug) allows users to perform actions they would otherwise not be allowed to, or access data they would otherwise not be privy to.

2.1 Disclosure Procedures and Policies

Dealing with vulnerabilities found by a 3^{rd}-party in software or web service is a complex topic, which does not have a standard procedure defined. The person, company or responsible industry body may have a *vulnerability disclosure policy* in place if they have dealt with vulnerability disclosure in the past, however, as there is no industry standard in place, the terms in the policy will be uniquely defined as to how the vulnerability finder sees it fit.

A common *responsible* vulnerability disclosure practice is that the discovering body immediately notifies the vendor of the product of the vulnerability, and then either waits for a given amount of days before making the vulnerability public, sometimes this may be cut short or prolonged depending on the reaction of the vendor.

There are highly debated topics on what can be considered a *responsible* vulnerability disclosure. One of such debate topics include whether the disclosure should be made public immediately after discovery, (a) but with as little information as possible until the vendor fix comes, so as to protect vulnerable users from potential attackers, or (b) it should include the full report, so as to make vulnerable users fully aware, in which situation they can actively try to defend themselves against it.

Last, but not least, in case the discovering body withheld information about the vulnerability from the public, the other debate falls on what should the grace period be for the vendor fix, and what should a vulnerability discoverer do in case the vendor does not fix it in time or cannot be contacted. If the discovered vulnerability was not made public at all, or just partially, the situation can get infinitely more complex in cases where multiple vendors are involved.

Some vendors will react faster, and will try to push the update to their users, detailing the vulnerability, while others might react slower, and will now be vulnerable, because the competing product leaked the vulnerability details with their update. In such cases, the discovering body may attempt a *coordinated disclosure* and ask the vendors to withheld the fixes and updates for a few days, or until all the vendors came up with a fix.

Depending on the morality of the discovering person, however, one might choose to go down the illegal route and sell the newly discovered vulnerability. On the deep web, there are marketplaces for so-called "zero-day exploits," whose prices range based on the severity and impact of the vulnerability. It can go from several hundred dollars to half a million, averaging around a few thousands generally[**41**].

In order for high-profile companies to encourage security researchers to take the responsible disclosure route instead of the alternative, they have started creating programs

called "bug bounties." These programs reward vulnerability reports by offering monetary compensation to the founder person (whose value fluctuates based on the severity and impact of the reported vulnerability) and recognition for their feat.

One such notable example is Google's *Vulnerability Reward Program*[**42**], which pays up to tens of thousands of dollars in rewards for a vulnerability in their own services, but they have also extended the bug bounty to several high-risk open-source software, for which they independently pay $500 to $3,133.7. Google also has a related program called *Patch Reward Program*, where they pay volunteers who have fixed a security issue in a high-risk open-source application. Rewards for that program range from $500 (for "one-liner fixes") to $10,000 (for "complicated, high-impact fixes.")

2.2 Regulatory Industry Bodies

One of the responsible industry bodies in the United States is *CERT/CC* (*Computer Emergency Response Team Coordination Center*) which was created by *DARPA* (*Defense Advanced Research Projects Agency*) in November 1988 in order to be the first organization of its kind. The need for the organization was created by the first ever computer virus distributed over the Internet, namely the *Morris worm*[**43**], which was the first of its kind, generating notable mainstream media attention, and resulting in the first ever arrest related to computer abuse/fraud laws in history.

Figure 2. CERT/CC Logo[**43**]

CERT/CC has their own policy regarding vulnerability disclosure, namely: notify the vendor as soon as possible, and disclose the report within 45 days, regardless whether the vendor has fixed the issue, or not.

In Romania, *CERT-RO* (*Centrul Național de Răspuns la Incidente de Securitate Cibernetică*) was established in the May of 2011, through the government decision H.G. 494/2011[**44**]. Its mission is to provide a national infrastructure for identification, analysis, and prevention of cybernetic incidents.

In Hungary, *GovCERT-Hungary* (*Kormányzati Eseménykezelő Központ*) was established in the April of 2013[**45**], assuming the role of the responsible industry body for providing relevant services.

Similarly, a response team is available at the European Union level. *CERT-EU* (*Computer Emergency Response Team European Union Task Force*) was established on the November of 2012[**46**], via mandated EU Commission decision. This body acts as a separate response team, meaning it does not exist solely to amalgamate reports of member countries' response teams.

2.3 Vulnerability Databases

One of the projects sponsored by CERT/CC is *CVE* (*Common Vulnerabilities and Exposures*), which provides a reference method for publicly known and tracked vulnerabilities. *NIST* (*National Institute of Standards and Technology*) runs a website called *NVD* (*National Vulnerability Database*) which is a repository of public vulnerabilities represented in such a way as to allow for automatic consumption by vulnerability management software[**21**].

The feed of public vulnerabilities available for consumption in the aforementioned paragraph is called *SCAP* (*Security Content Automation Protocol*) and contains multiple components for each entry:

- *Common Vulnerabilities and Exposures* (*CVE*) – Vulnerability reference standard.
- *Common Configuration Enumeration* (*CCE*) – Unique identifier provider for system configuration in order to facilitate easy misconfiguration testing.
- *Common Platform Enumeration* (*CPE*) – Standardized method for classifying and identifying applications, operating systems, and hardware devices present on a system.
- *Common Weakness Enumeration* (*CWE*) – Provides a common language for identifying the type and source of the vulnerability.
- *Common Vulnerability Scoring System* (*CVSS*) – Standard metric which allows measurement of the vulnerability's impact.
- *Extensible Configuration Checklist Description Format* (*XCCDF*) – XML-based format for providing security checklists.
- *Open Vulnerability and Assessment Language* (*OVAL*) – Provides language for encoding vulnerability details in order to aid internationalization.

These components will be further discussed in the section regarding the implementation of the vulnerability assessment engine.

2.4 Vulnerability Risk Analysis

During the project planning phase or later down the line when a vulnerability is discovered either during development or via penetration testing, risk analysis may be performed in order to assess the severity of the risk using the standard risk model formula shown in equation **1**. Generally, if its fix would be too expensive, (e.g. it would require breaking changes, full infrastructure rebuild, or other miscellaneous major inconveniences) the results of the risk analysis are carefully evaluated by managers, and if no immediate

major threat is imminent with a tremendous impact that can be caused by the discovered bug, an alternative route may be chosen which is not as costly.

$$\text{Risk} = \text{Probability} \cdot \text{Impact} \tag{1}$$

The *Open Web Application Security Project* has defined a risk rating methodology[**47**] specifically for vulnerabilities. The steps of risk analysis for a project are:

1. Identification of Risks
2. Estimation of Probability
3. Estimation of Impact
4. Estimation of Severity
5. Decision Regarding Risk Elimination

For the first step, in the case of a web application or a project with a web application subcomponent, we can start with OWASP's Top 10[**48**] list of most common security risks. After all the possible risks have been listed, we will need to go through them one-by-one and go through steps $2-5$ individually for each.

$$\text{Threat Agent} = \text{Skill Level} + \text{Motive} + \text{Opportunity} + \text{Size} \tag{2}$$

$$\text{Vulnerability} = \text{Discovery} + \text{Exploitation} + \text{Awareness} + \text{Detection} \tag{3}$$

$$\text{Probability} = \frac{(\text{Threat Agent} + \text{Vulnerability})}{8} \tag{4}$$

In step 2, we estimate the probability of the risk occurring. In order to do this, we go through each factor in equations **2** and **3**, assigning them each a number on a scale of 0..9. For example, the "Motive" factor is directly proportional to the reward a user might get from exploiting the vulnerability, as such, we rate 0 if no significant reward can be obtained, or 9 if major financial gain can be obtained from exploiting the vulnerability. After all the factors have been scored, we calculate the arithmetic mean as seen in equation **4**.

$$\text{Technical} = \text{Confidentiality} + \text{Integrity} + \text{Availability} + \text{Accountability} \tag{5}$$

$$\text{Business} = \text{Financial} + \text{Reputation} + \text{Noncompliance} + \text{Privacy Violation} \tag{6}$$

$$\text{Impact} = \frac{(\text{Technical} + \text{Business})}{8} \tag{7}$$

Moving on to step 3, we calculate the impact of the risk, should it occur, from both a technical and business side. Equations **5** and **6** list the factors which should be considered.

One important example of these factors would be "Noncompliance," which might further damage the company both financially and reputation-wise. Non-compliance can lead to loss of audits, licenses, which in turn can lead to fines, loss of liability protections and inability to continue business until the company can be re-audited. For example, as discussed in section **1.4**, the violation of *PCI compliance* can result in fines of $\$5,000$ to $\$100,000$ per month[**49**] by the credit card company at their own discretion. If the company has no such compliance obligations, we can rate non-compliance as a non-issue with a score of 0, or if credit card data might be leaked, a maximum score of 9 should be provided.

Score	Severity
0 to $2.\bar{9}$	Low
3 to $5.\bar{9}$	Medium
6 to 9	High

Table 1. Score Classification

		Probability		
		Low	Medium	High
Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Note	Low	Medium

Table 2. Vulnerability Risk Severity

The semifinal step of the risk analysis is determining the severity of the risk/vulnerability. This can be done by taking the values obtained from the aforementioned equations, and computing the final score from the standard risk formula as shown in **1**.

Alternatively, a more simplified visual guide can be obtained by taking the value of the "Probability" and "Impact" factors, and classifying them according to table **1**. After the severity of both factors is obtained, table **2** allows an easy lookup of the final severity of the risk.

In the final step of the process, we make a decision regarding the mitigation of the risk. In general risk management theory, if the costs of mitigating a risk outweigh the costs of incurring the losses caused by its occurrence, the risk is not mitigated.

2.5 Common Vulnerability Scoring System

NIST's *CVE* database has *CVSS* scores associated to their vulnerability entries. This greatly enhances the ability of the tool implemented within this thesis to aid risk analysis, as the detected vulnerabilities can be set up to map automatically onto OWASP's risk analysis (or any other similar methodology in use by the business) for easier analysis of the latest vulnerabilities that may appear on a business's infrastructure.

Figure 3. CVSS Logo[**50**]

The *Common Vulnerability Scoring System*[**50**] (*CVSS*) provides the following factors for each vulnerability with an associated score:

- Base
  - Access Vector (AV)
  - Access Complexity (AC)
  - Authentication (Au)
  - Confidentiality Impact (C)
  - Integrity Impact (I)
  - Availability Impact (A)
- Temporal
  - Exploitability (E)
  - Remediation Level (RL)
  - Report Confidence (RC)
- Environmental
  - Collateral Damage Potential (CDP)
  - Target Distribution (TD)
  - Confidentiality Requirement (CR)
  - Integrity Requirement (IR)
  - Availability Requirement (AR)

$$\text{Exploitability} = 20 \cdot \text{Access Complexity} \cdot \text{Authentication} \cdot \text{Access Vector} \tag{8}$$

$$\text{Impact} = 10.41 \cdot (1 - (1 - \text{Conf.Impact}) \cdot (1 - \text{Integ.Impact}) \cdot (1 - \text{Avail.Impact})) \tag{9}$$

$$\text{Score} = (0.6 \cdot \text{Impact} + 0.4 \cdot \text{Exploitability} - 1.5) \cdot \begin{cases} 1.176, & \text{if Impact} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

As with the equations discussed in section **2.4**, the "Probability" factor seen in equation **4** is the functional equivalent of the "Exploitability" factor in equation **8**, similarly so are the "Impact" factors between equations **7** and **9**.

The final score given by equation **10** is on the scale of 0..1, and its components can be used after scaling to 0..9 in order to determine the severity of a risk by using table **1** to classify it, which can then be looked up in table **2**.

Since the application implemented in the scope of this thesis reports CVSS scores for automatically detected fresh CVE vulnerabilities, it speeds up the risk analysis and decision-making process, as managers can determine the severity of a newly discovered vulnerability in their systems and can act fast, if required, in cases where involving a security engineer would take much more time than desired in a critical situation.

3 Vulnerability Assessment

Vulnerability assessment is the process of identifying and quantifying (see section **2.4** for methodology) of the vulnerabilities present within an infrastructure.

In order to ensure the continuous security of an application, server or whole infrastructure, proactive measures need to be taken by the competent bodies. These measures generally include periodic scanning of the infrastructure in order to identify possible attack surfaces and test against newly discovered vulnerabilities.

3.1 Static Code Analysis

One of the ways this can be carried out is through *static analysis* of the source code. This is generally done by an automated tool, and then a software developer goes through the results, reviewing and evaluating the possibly erroneous code in the meantime.

Static analyzers process the source code of the application, doing lexical analysis, semantic analysis, type inference, constraint analysis, and so on. Simply put, they are looking for common mistakes there were either introduced by accidentally overlooking things on the developer's part – such as variable value assignment (`=`) instead of comparison (`==`) in `if` statements – or parts that would possibly behave differently than the developer intended due to various language features (such as loose comparison type juggling/inference issues.)

Depending on their feature set, static analyzers may also evaluate the *control flow* of code in order to try and determine if branches exist that would result in an unwanted behavior. Another feature can be *taint analysis*, which observes the evolution of variables that have user-provided input in order to determine if any input can cause issues later on in the code. In any case, the code is not executed, only the source is analyzed, which introduces some limitations.

Using static analysis tools have their pros and cons, for example, they can be enforced by integrating it into the build process, but they also have cons, for example, not being able to analyze binaries that have been provided without a source code or the fact that pattern matching cannot account for everything, resulting in *false negatives*. (Also known as, malicious code that was analyzed but not deemed malicious.)

3.2 Fuzz Testing

A similar software testing technique that has proven useful in the field of security is *fuzz testing*. This technique feeds random malformed data to the software as input and observes the behavior of the software as a result.

While this technique is not necessarily popular, it has been used quite successfully in the past by security researchers to weed out bugs in numerous applications automatically.[**51**] The use of randomly generated data which breaks assumptions is very successful in confusing/breaking the state machines of the parser being tested and potentially leading to instability during runtime or software crash. This is essentially "vulnerability scanning by brute force."

3.3 Vulnerability Scanning

3.3.1 Penetration Testing

The most offensive vulnerability assessment technique is *penetration testing*, which simulates real-life attacks on an infrastructure.

Vulnerability scanning works well with binaries whose source is not provided, as it only simulates *malicious* user interaction, and does not do code analysis. This technique also works well against live production servers, and as such fully deployed infrastructures can be scanned for vulnerabilities.

Testing may be done against:

- port on the server – testing for applied patches and misconfiguration of a single service (e.g. a SMTP server);
- web application – crawling and analyzing the security of a whole web application through blind exploitation;
- entire server – scanning all the ports and running the proper vulnerability analysis for each protocol based on their service banner;
- entire network – scanning all the servers within the network, to ensure none of them are vulnerable or compromised in an environment in which sensitive data might be transferred.

Testing may also be done from various perspectives:

- public – in order to determine attack vectors from the perspective of an outsider (e.g. vulnerability of public-facing IPs);
- consumer – to determine attack surfaces for an authenticated non-privileged user (e.g. logged-in consumer of a banking web application);
- insider – to determine the damages that can be caused by insiders (e.g. rogue employees).

3.3.2 Intrusion Detection/Prevention Systems

Penetration testing is an *active* vulnerability scanning technique which simulates real-life attacks on the infrastructure, however a *passive* version of this technique exists under the name of *IDS/IPS*. (*Intrusion Detection/Prevention System*)

These systems sit between the user and the application, sniffing and analyzing traffic for potential red flags. This method is rather limited, as it cannot detect vulnerabilities in the system, with which a user has not yet interacted with due to its limited data source being only the network traffic.

Another downside of this technique is that an exploitation might slip through even in the strictest setting of the IPS as the analyzed network traffic might simply be identified as false negative. As such, preventive measures were not taken against the traffic, and the exploitation was successful.

4 Related Work

As discussed in sections **1.3**, **1.4** and **1.5**, vulnerability assessment is a booming business due to the huge market demand which has been created by the rapid outbreak of digitalization of various services.

Whether organizations care about their security or just want to pass the yearly PCI audits, the manual analysis of the existence of all known vulnerabilities within the infrastructure would be a tedious process.

As such, *vulnerability scanners* have been developed by various companies in order to automate this process by checking against all known vulnerabilities and various configurations which are known to be substandard or are requirements on the PCI checklist.

4.1 Commercial Solutions

The *nmap project* has been cataloging various security products, such as vulnerability scanners, and ranking them based on user feedback and their popularity based on usage[**4**]. The three most popular vulnerability scanners will be presented in this section.

Nessus The *Nessus Vulnerability Scanner*[**5**] is developed by Tenable Network Security. Initially, it was released as a free and open-source vulnerability scanner with paid registered options available, however, this was changed in 2005 when the application went closed-source.

SecTools.org lists Nessus as the most popular vulnerability scanner today. Currently, a "Community" edition can be obtained at no charge, however, it is limited to personal and non-commercial usage. Annual subscriptions otherwise are available starting from $2,190 per user per year.

The internal architecture of the vulnerability scanner works based on plugins. It also implements its own embedded scripting language "NASL" to be used when writing plugins.

While it boasts about having 70,000+ plugins, the scanner works by running all of the probe scripts on a service banner, and separate plugins are required in order to identify different software products. After identification, separate scripts are run in order to test vulnerabilities on the identified services.

This means that the developers of the scanner need to keep up-to-date with all of the existing server software and their newest vulnerabilities. This means releasing a new plugin as soon as possible to the users of the scanner in order to catch them.

This generally results in a large software with high maintenance fees, as the annual subscription cost shows.

OpenVAS The *Open Vulnerability Assessment System*[**6**] is a fork of the last open-source Nessus edition before that went closed-source. It is currently developed and maintained by Greenbone Networks.

The vulnerability scanner, as a result of the shared heritage, has the same architecture as Nessus, and even continues to use the Nessus NASL scripting language for its plugins.

SecTools.org lists OpenVAS as the second most popular vulnerability scanner today, and it continues to be free and open-source despite its growing popularity.

Nexpose The *Nexpose Vulnerability Scanner*[**7**] is developed by Rapid7, who are also famous for developing and maintaining another popular security tool *Metasploit*.

As opposed to the previous solutions, Nexpose aims to be much more than just a vulnerability scanner, fulfilling the roles of the entire vulnerability management lifecycle, from detection to risk analysis and mitigation.

SecTools.org lists Nexpose as the third most popular vulnerability scanner today. Annual subscriptions start from $2,000, however, a limited "Community" edition is also available to be used for non-commercial purposes.

Behind-the-scenes, Nexpose has powerful integration with tools such as Metasploit for exploit scanning and validation purposes.

4.2 Scientific Papers

There is a wealth of research available on the various topics within the field of network security. On the defensive side of research topics, as every vulnerability scanner with an active reconnaissance component has to contact the target hosts, paper [**52**] experiments with evaluating whether port scans can be used as an alert for an impending attack.

The component discussed in section **6.1.1** uses the industry-standard ICMP Echo Requests packets presented within in the aforementioned paper, therefore being susceptible to this kind of analysis.

A wide range of comparative studies are also available, examining different scanning methods, techniques, and the various commercial tools available to do so, some of which are mentioned in section **4.1**.

The most cited papers doing such comparative studies on black-box web and/or network vulnerability scanners with the inclusion of at least one of the aforementioned commercial tools are [**8, 9, 10**].

These studies generally highlight the strengths and weaknesses of each studied application, all of them ending in the conclusion that no single application can be used as a one-stop shop for detecting the latest vulnerabilities or the most obscure misconfigurations, no matter how much is the application marketed as being "state-of-the-art" by its developer.

As for creating new vulnerability scanners as part of the research, paper [**11**] presents

the implementation challenges faced when writing a web application vulnerability assessor. Similarly, but taking on a new perspective, paper [**12**] focuses on the same goal, but extending the vulnerability assessment to networked devices, as opposed to limiting to just web applications.

The paper *ShoVAT*[**13**] presents a methodology for mapping service banners to CPE names for the purposes of vulnerability assessment. The implementation of the CPE-matcher component discussed in section **6.8** is loosely based on this paper, with some slight implementation differences.

5 Project Aim

The aim of this project is to develop an application which can scan a requested network and determine the vulnerable services within it.

The application should be completely autonomous in its process, without relying on future updates from the developer in order to add support for new services and associated vulnerabilities.

The target audience of the project should be as wide as possible. It should be a useful tool for security researchers, security consultants and system administrators alike, even without requiring much security experience from the former group.

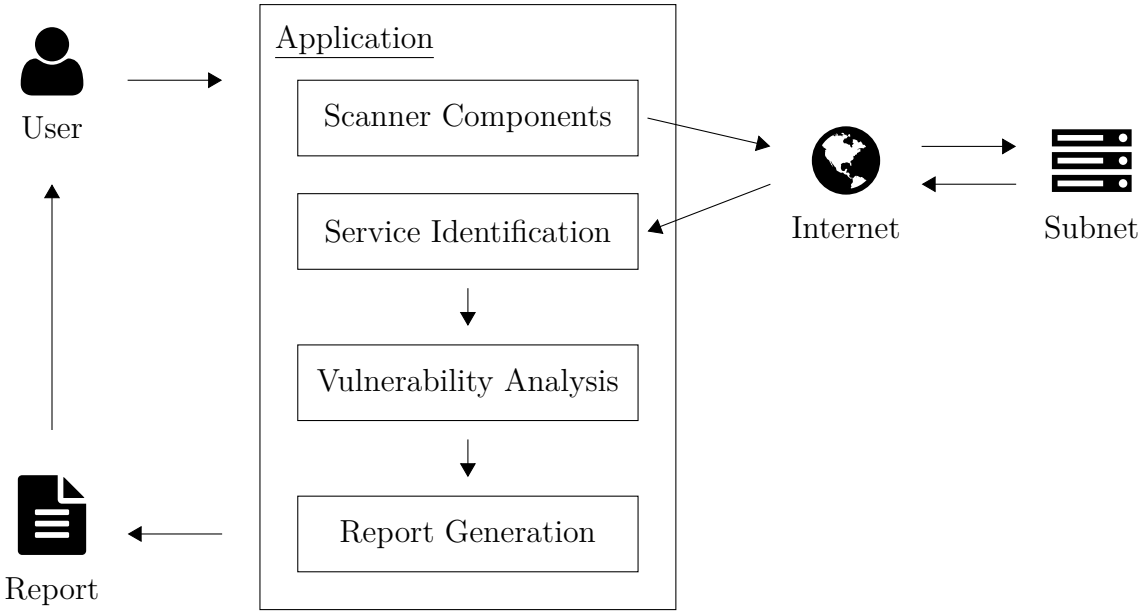


Figure 4. Block diagram of the proposed application

5.1 Data Acquisition

In order to be useful for independent security researchers, the application needs to be able to utilize open-source intelligence. The three services presented in sections **6.2.1**, **6.2.2** and **6.2.3**, namely Shodan[**1**], Censys[**2**] and Mr Looquer[**3**], are all candidates for usage within the passive data reconnaissance component.

This way, researchers may execute queries and get instantaneous results, without having to have an infrastructure on which to launch the scans from, and then deal with the consequences, such as abuse emails.

On the other hand, security consultants generally need to scan an internal infrastructure, one which is not accessible to the aforementioned services. As such, the project needs to implement active data reconnaissance components as well.

While the application implements a wide range of protocol scanners with a unified interface for task parallelization, as a measure of redundancy, the users are given the choice of using an external scanner for their reconnaissance purposes. By default, *nmap* is supported out-of-the-box as an alternative scanner, however, any 3^{rd}-party application which generates nmap-compatible XML-based reports can be used, one example of which is *masscan*.

5.2 Data Analysis

The collected service banners should be analyzed in an entirely autonomous manner. That is, as opposed to the software and services discussed in section **4**, the application should not rely on separate plugins for identifying the protocols and afterward identifying the servers speaking those protocols.

The databases published by *NIST* (as discussed in section **2.3**) should be consumed in order to identify the newest vulnerabilities. Section **6.8** presents the component consuming the database and autonomously mapping service banners to their equivalent *CPE* names.

As a redundancy, section **6.7** presents a different way of mapping service banners to CPE names, however, that does not rely on the aforementioned published databases, it instead relies on a database published by the developer or written/extended by the user/community.

5.3 Resolution Suggestions

Following the CPE associations, discovering the vulnerabilities within the identified services is just a simple lookup away in the *CVE database*.

At this point, the application should provide as many information as possible about the found vulnerabilities, including CVSS scoring in order to determine the risk severity within the context of the scanned infrastructure.

However, in order to be useful for system administrators without pre-existing security knowledge, the application should provide simple tips – based on what it could deduce about the scanned environment – on how to fix the issues in layman's terms.

That is, when the operating system is identified and supported, actual command lines which the administrator can run on the target host in order to ameliorate the vulnerabilities.

6 Implementation

The application and scripts developed within the scope of this thesis are free and open-source software.

The git repository for the main application is available at `https://github.com/RoliSoft/Host-Scanner` and can be freely used, modified and redistributed under the terms of the GNU General Public License version 3[**14**].

The various scripts written mainly for the purposes of data processing and experimentation are available at `https://github.com/RoliSoft/Host-Scanner-Scripts` and can be freely used, modified and redistributed under the terms of the MIT license[**15**].

6.1 Active Reconnaissance

This section presents the components implemented within this application which engage in active target probing, to be more exact, those that are actively sending packets towards the scanned targets in order to determine port states and grab service banners for analysis.

It should be noted that the legality of this practice varies by region, and one should generally not aim towards targets for which they previously have not received authorization to do so.

6.1.1 ICMP Echo Requests

The *Internet Control Message Protocol* (*ICMP*) has an 'EchoRequest' packet type, to which the kernels of the intended destinations, if not explicitly disabled, will reply with an 'EchoReply' packet, sending the received payload back, including its sequence number, which can then be used for statistical analysis purposes by the requesting party.

The command line utility `ping` uses this ICMP 'EchoRequest' mechanism in order to "ping" a remote destination and compute the round-trip time of the communication.

As shown in listings **1** and **2**, the `ping` implementations that come bundled with both Windows and various Linux[6] distributions utilize a static payload. This opens up the

[6]Only Linux distributions which come bundled with GNU userland tools.

scanning party for fingerprinting, and firewalls may choose to drop such packets if they are configured based on filtering these static payloads.

```
00000000: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
00000010: 7172 7374 7576 7761 6263 6465 6667 6869  qrstuvwabcdefghi
```

Listing 1: Static payload sent by the Windows implementation of `ping`

```
00000000: aa4a e956 0000 0000 9b2c 0c00 0000 0000  .J.V.....,......
00000010: 1011 1213 1415 1617 1819 1a1b 1c1d 1e1f  ................
00000020: 2021 2223 2425 2627 2829 2a2b 2c2d 2e2f   ....%&.()*+,-./
00000030: 3031 3233 3435 3637                      01234567
```

Listing 2: Static payload sent by the GNU implementation of `ping`

In order to work around the fingerprinting issue, the `IcmpPinger` component generates a random 32-byte payload, as shown in listing **3**.

```
00000000: 54e4 d362 3644 a768 147b 9538 2fcb 0fa2  T..b6D.h.{.8/...
00000010: 5810 8906 fcaa cbe7 430b dcc9 09a7 f39a  X.......C.......
```

Listing 3: Example random payload generated as the 'EchoRequest' data

The implementation of the scanner uses raw sockets in order to craft the 'EchoRequest' packets, the use of which is restricted to super-user or administrator users on most operating systems, including Linux, Windows, and various BSD flavors.

Such "ping" requests cannot be sent on a per-port basis, as such it cannot be used to discover the available services on a host, and can only be used for host discovery within a network. While this is the most standard way of probing a host for availability state, most firewalls are configured by default to drop such packets, as they are not crucial for the normal operation of a network. They are generally only used for troubleshooting purposes or by outside actors to try and map a given network.

6.1.2 ARP Solicitation

The *Address Resolution Protocol* (*ARP*) is the mechanism responsible for mapping network-layer IPv4 addresses to link-layer MAC addresses.

When a client would like to receive the MAC address of an IP address within a network, as shown in figure **5**, it will craft an ARP packet consisting of its own data, and the requested data set to zeros.

The aforementioned ARP packet is then broadcasted throughout the network, and if a client has the information which the packet is requesting, it will reply back with the zeroed fields filled out, and source-target fields swapped, as shown in figure **6**.

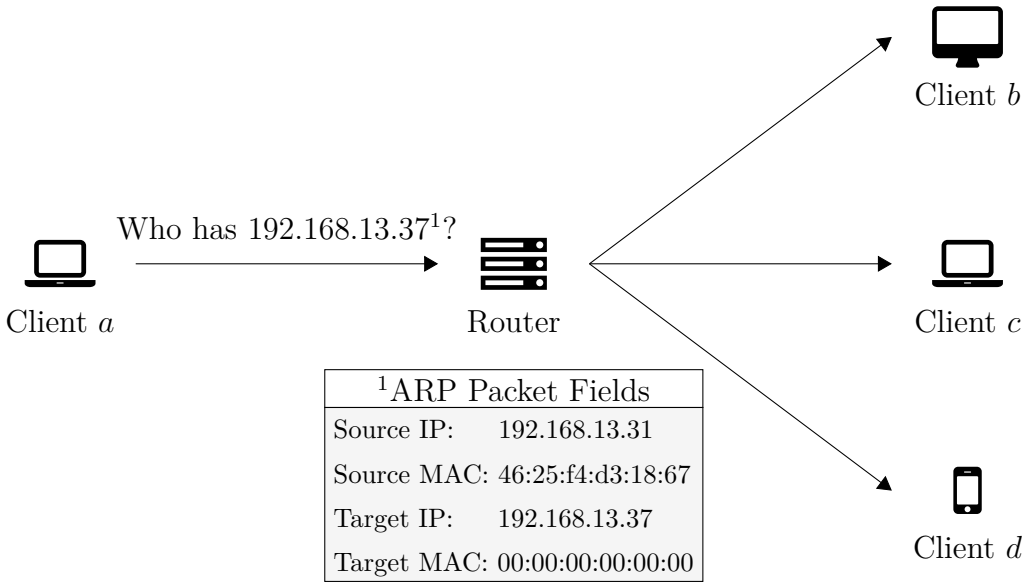


Figure 5. ARP Request for MAC address of IP 192.168.13.37

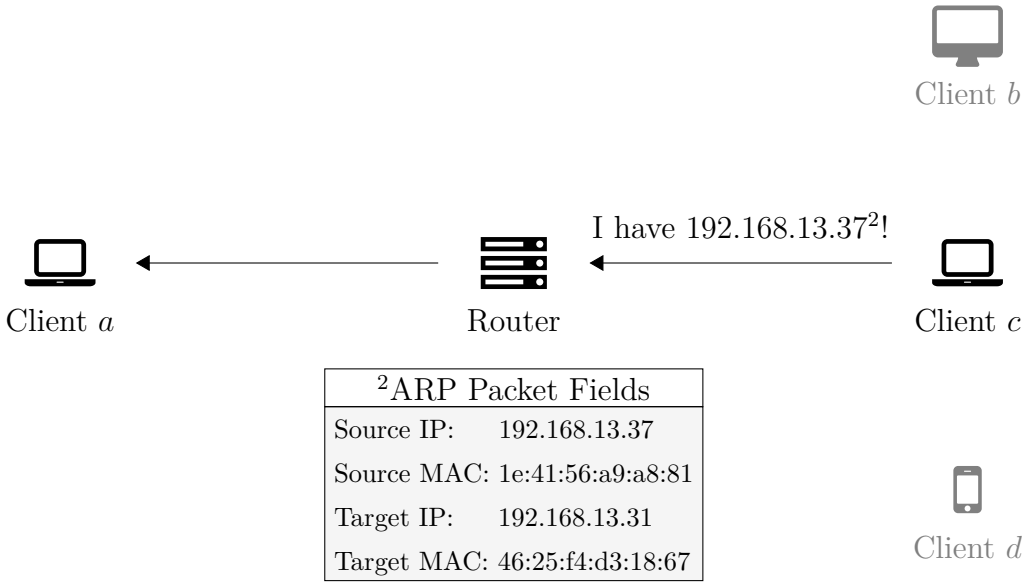


Figure 6. ARP Response with MAC address of IP 192.168.13.37

This mechanism is crucial to the proper functioning of an IPv4 network, and as such firewalls cannot interfere with it[7].

Similarly to ICMP, this functionality is only useful for discovering the hosts connected to a specific network, and not to determine the port states on the hosts themselves.

6.1.2.1 Cross-Platform Implementation Challenges

The implementation of the `ArpPinger` component was one of the most challenging and time-consuming components of the application developed within the scope of this thesis. None of the targeted operating systems provides out-of-the-box support for crafting, sending and receiving ARP packets. Documentation is also scarce on the subject, and the ones that do exist are generally too platform-specific.

The bird's-eye overview of the implementation is that a raw socket is opened which can send link-layer packets, as opposed to network-layer communication which is generally the default mode for raw sockets. After this, the `ArpPinger::sendRequest(Host* host)` function manually constructs an Ethernet frame, with the underlying ARP 'WhoHas' request, which is filled with the known information and the information sought after left as zeros.

After the packet is sent through the opened socket, the component will invoke the `ArpPinger::sniffReplies(Interfaces* ifaces, Hosts* hosts)` function in a new thread, which will listen for any incoming ARP responses and checks if any of them is a response to any of the specified sent requests in argument `hosts`. The implementation of this phase greatly varies on every operating system, and will be discussed in the following sections.

6.1.2.2 Linux-Specific Implementation

The Linux-specific implementation was the easiest, as the kernel does not try to impose various senseless limitations upon the developers.

The `getifaddrs(struct ifaddrs **ifap)` function returns the list of connected interfaces, from which the `AF_PACKET` type interfaces are being saved for later use.

In order to send the crafted ARP packet, a link-layer socket can be opened with the `socket(PF_PACKET, SOCK_RAW, ETH_P_ALL)` call.

Similarly, the same socket can be used to listen for incoming link-layer packets. In order to speed up the process, the Linux kernel has support for Berkeley Packet Filter byte-codes, which allow filtering of the received packets on the socket in kernel-space. This will be further discussed in the BSD-specific implementation, as the BSD kernel is where it is originating from.

[7]Except in cases where the clients are isolated on the network, and as such are not supposed to communicate with each other.

6.1.2.3 BSD/Darwin-Specific Implementation

The BSD/Darwin-specific implementation was largely based on the Linux-specific implementation, as most function calls shared the same behavior. However, some subtle argumentation and structure differences have crept up here and there, creating the need for a different implementation.

As with Linux, the `getifaddrs(struct ifaddrs **ifap)` function returns the list of connected interfaces, from which the `AF_LINK` type interfaces are being saved for later use. While both `AF_PACKET` and `AF_LINK` represent the same interface, they use a different underlying structure to represent information about them, and as such require differing implementations.

Unfortunately, BSD does not allow the creation of a link-layer socket with a simple `socket()` call as Linux does, however, it does have support for Berkeley Packet Filter devices, which can be used to, amongst other things, send and receive link-layer packets.

In order to open a BPF device, the scanner first tries to open the next available descriptor in the range of `/dev/bpf0` to `/dev/bpf1000`. Once the device is open, it will be bound to a network interface with the `ioctl(bpf, BIOCSETIF, &inf)` call. At this point, link-layer packets can be sent by writing Ethernet frames to the `bpf` descriptor.

Similarly, the same device can be read in order to listen for incoming link-layer packets. In order to speed up the process, the BSD kernel has support for parsing and evaluating Berkeley Packet Filter byte-codes, which allow for kernel-space filtering of packets before they reach the BPF device to be handled by the application running in user-space.

An assembly-like BPF instruction list can be seen in listing **4**, as generated by Wireshark, to filter packets other than ARP.

```
1 ldh [12]              ; skip 12 bytes
2 jeq #0x806 jt 2 jf 3 ; if Eth type is ARP goto 2 else 3
3 ret #262144          ; return packet [when ARP]
4 ret #0               ; return null
```

Listing 4: Berkeley Packet Filter instructions to filter packets other than ARP

In order to apply this instruction list to a BPF device, the code first has to be built into a list of `bpf_insn` type array, as shown in listing **5**. This can then be wrapped into a `bpf_program` type structure, which in turn is ready to be attached to a device via the `ioctl(bpf, BIOCSETF, &bfprog)` call.

```
struct bpf_insn bfcode[4];
bfcode[0] = BPF_STMT(BPF_LD + BPF_H + BPF_ABS, 12); // skip 12 bytes
bfcode[1] = BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, ETH_P_ARP, 0, 1); // if Eth type is
 ↪  ARP goto +0 else +1
bfcode[2] = BPF_STMT(BPF_RET + BPF_K, sizeof(struct EthHeader) + sizeof(struct
 ↪  ArpHeader)); // return packet [when ARP]
bfcode[3] = BPF_STMT(BPF_RET + BPF_K, 0); // return null
```

Listing 5: Specifying the Berkeley Packet Filter instructions shown in listing **4**

The Darwin kernel and Mac OS X distributions inherit the BPF device, as such this implementation of the scanner is fully supported on these platforms.

6.1.2.4 Windows-Specific Implementation

The Windows-specific implementation was a complete rewrite of the algorithm. Even though the Windows networking stack is based on BSD sockets, various senseless limitations were introduced, which make it impossible to send and receive link-layer packets, without resorting to extreme measures.

In order to get the list of interfaces on Windows, there is an exposed API, which can be used, namely `GetAdaptersInfo(PIP_ADAPTER_INFO pAdapterInfo, PULONG pOutBufLen)`. The filtering is then done based on the `MIB_IF_TYPE_ETHERNET` type in order to only retain virtual and physical Ethernet interfaces.

However, the list of functions that are supported via exposed APIs ends here, since Windows does not allow the sending and receiving of link-layer packets by user-space applications. In order to get this type of scanner working on Windows, I had to resort to the aforementioned 'extreme measures.'

In order to send and receive link-layer packets on Windows, as shown in [**53**], the WinPcap library can be utilized, which is a 3^{rd}-party port of the libpcap interface to the Windows operating system, complete with its own virtual driver to allow low-level network access.

After opening the network interface using `pcap_open(...)`, the resulting instance pointer can be used to either send packets at the link-layer level through the driver using the `pcap_sendpacket(pcap_t *p, u_char *buf, int size)` function, or to receive packets by calling the `pcap_next_ex(pcap_t *p, pcap_pkthdr **pkt_header, u_char **pkt_data)` function in a loop until either the sought after packet is read from the device or a specified time frame has elapsed.

The aforementioned Berkeley Packet Filter functionality is also available within WinPcap, and the presented instructions in listings **4** and **5** can be activated on the device using the `pcap_setfilter(pcap_t *p, bpf_program *fp)` function.

As a result of this library usage due to this scanner, in order to compile the application with the ARP scanner included on Windows, the "WinPcap Developer's Pack" has to

be installed and its path specified to CMake on the compiling machine, while the hosts running the application will need to have the WinPcap device driver installed.

6.1.3 TCP Scanning

The TCP port scanning functionality is implemented by the `TcpScanner` component. The current implementation does not require super-user or administrator rights on the host device, as the component uses the operating system's network API to initiate connections.

All the network operations are initiated in a non-blocking mode, after which the task at hand will go back into the queue until a state change has happened, as-is task multiplexing discussed in section **6.5**.

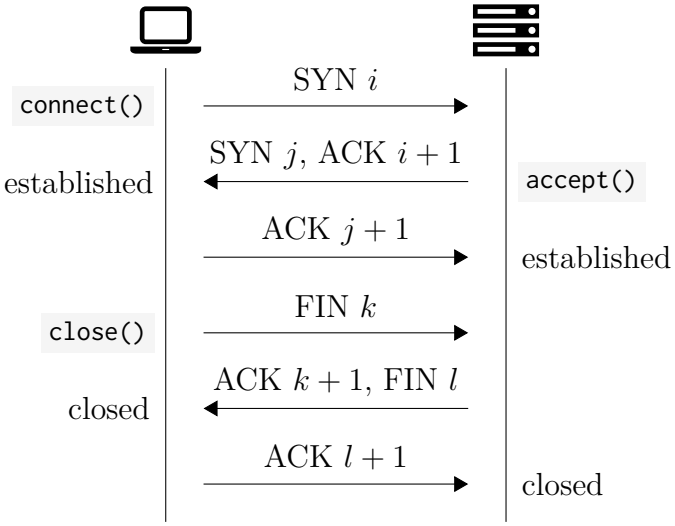


Figure 7. Three-way handshakes of TCP connects and disconnects

As seen in figure **7**, the TCP scanner component initiates a new connection with the `connect()` function call, which in turn leads the operating system's network stack to go through the full three-way handshake in order to open the socket.

After the connection has succeeded, the scanner will call the `close()` function, which initiates the proper sequence for shutting down a TCP connection, as it would otherwise leave the connections open, which would open up both the scanner and targets for denial-of-service attacks by the way of resource exhaustion[**16**].

There are alternative methods for port scanning, for example 'SYN', 'ACK' and 'FIN' scanning[**17**], where instead of using the operating system's network stack in order to initiate connections, the software crafts its own TCP packets, and uses raw sockets in order to send them and listen for replies. However, the use of raw sockets is generally restricted to super-user or administrator users on most operating systems, including Linux, Windows, and various BSD flavors, including Mac OS X.

Such alternative scanning methods are generally used to infiltrate firewalls or evade

logging, as a firewall which does not implement a proper stateful stack may let non-'SYN' packets in and 'RST'/'FIN' packets out. Similarly, if a connection is not opened fully by the way of a three-way handshake, the server software may never end up logging the connection attempt, but enough information is leaked beforehand to the attacker in order to determine whether the port is open, closed or filtered.

As the software implemented within the scope of this thesis requires full service banners (port state information alone is not enough) and was intended to be used only by authorized personnel (such as network administrators and security researchers), it was decided that firewall evasion techniques are irrelevant and beyond the scope of this application.

6.1.3.1 Protocol Probing

When a connection initiates, some protocols dictate that the server will send a "welcome message," otherwise known as a service banner. For protocols, such as SSH, SMTP, and FTP, this is enough to run analysis on them and determine the protocol type, software name, and version.

However, HTTP and other TLS-wrapped services require the client to send its request first, which is problematic, as there is no method at that point for determining what kind of protocol is the server listening for.

One way would be to check the list of "well-known" ports[**54**], which is maintained by *IANA* (*Internet Assigned Numbers Authority*). While most of the port numbers coincide with the ones used in the real world, this is not an accurate method to deduce the service behind a port, as an administrator may choose to run a service on another port for security through obscurity reasons, evasion of firewall for the clients, or simply just due to running multiple instances of a server. Other than that, the list does not contain associations for ports between 49152 and 65535, as these are considered dynamic/ephemeral ports, which makes the identification of services on these ports impossible through this method.

As such, this list is not utilized at all and active protocol probing is done instead. If the connection succeeds and no service banner was sent within a specified timeframe, the scanner sends a `GET / HTTP/1.0 \r\n\r\n` message to the server. If the reply is not satisfactory, the scanner will re-try the connection and send the `HELP\r\n` message.

At this point, if the server still has not sent a satisfactory reply, the scanner sends a TLS 'ClientHello' packet in the hopes of negotiating a TLS connection. If the secure connection established, the previous plaintext protocol probes are retried in the hopes of getting a satisfactory reply message which can by analyzed by the other components of the application.

6.1.4 UDP Scanning

The UDP protocol, as opposed to TCP, is connectionless. This means that it cannot be reliably determined whether a server is listening on the scanned port at all. Even if a server is listening, it might be possible that its implementation drops all packets which are not well-formed, instead of returning an error message which can be used to deduce the protocol.

Some systems will reply with an ICMP 'PortUnreachable' packet, however, a firewall may filter this for the specific purpose of combating scanning attempts, or the ICMP messages are rate-limited somewhere along the route between the scanner and target, leading to dropped port unreachable notification packets.

As such, the best way to get a reply is to craft and send a well-formed message to the specific port in order to get the server application to send a reply. The `UdpScanner` component of the application utilizes a database of known UDP payloads which are mapped to specific port numbers. If the scanned port has an associated payload, the scanner will send this packet. As an example, listing **6** shows the hexadecimal dump generated by `xxd` for a packet which is sent to port 53 and requests a DNS server to reply with its version number.

```
1 00000000: 34ef 0100 0001 0000 0000 0000 0756 4552  4............VER
2 00000010: 5349 4f4e 0442 494e 4400 0010 0003       SION.BIND.....
```

Listing 6: Example binary UDP packet to request DNS server version

Unfortunately, this method limits the number of discoverable services to those with a known payload, and even in those cases, it only works when the server is listening on its default designated port number. In cases where a scan of a port not in the database is requested, the scanner sends a UDP packet consisting of 16 `NULL` bytes, however, the chances of a server replying to this packet are slim.

The payloads database is revisited and discussed in greater detail in section **6.13.3**.

As these packets are static bytes and generally request system information or server metadata, which may not be a part of the general use-case flow of the protocol, some intrusion detection systems actively monitor for such UDP packets and trigger alerts if found.

Listing **7** shows the hexadecimal dump of a UDP packet which, when sent to port 1434, will trigger the popular open-source intrusion detection system Snort[**55**].

```
1 00000000: 02                                       .
```

Listing 7: Example binary UDP packet to trigger Snort rule 2049 in listing **8**

As Snort is open-source, the rule/signature for detecting this type of packet can be accessed by any interested party and is shown in listing **8**.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1434 (msg:"MS-SQL ping attempt";
↪ content:"|02|"; depth:1; reference:nessus,10674; classtype:misc-activity;
↪ sid:2049; rev:4;)
```

Listing 8: Snort rule 2049 for blocking Microsoft SQL ping attempts[**55**]

For the purposes of host aliveness detection, an ICMP 'PortUnreachable' message to a random UDP port informs the scanner that the host is in fact online, but that specific port is not, as opposed to an ICMP 'HostUnreachable' message, which is sometimes sent by a router along the way to the host, informing the scanner that the host is not online.

6.2 Passive Reconnaissance

This section presents the components implemented within the application which allow passive reconnaissance, that is, probing of IP addresses without actively contacting them.

Such feat can be achieved by utilizing 3^{rd}-party services which scan the public IP address space on a regular basis and make the results available either in a downloadable database dump or a publicly consumable API format, as shown in figure **8**.

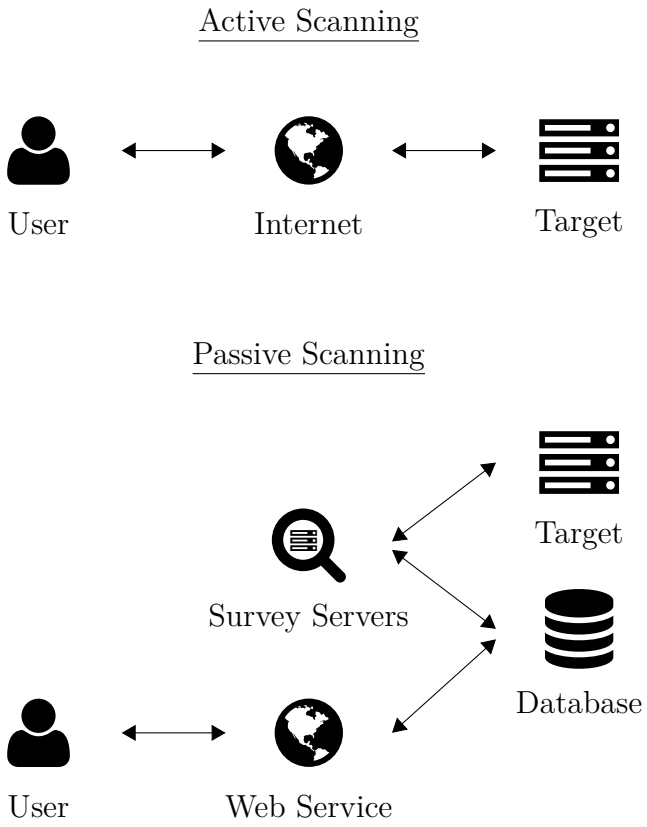


Figure 8. Comparison of reconnaissance methods

6.2.1 Shodan

Shodan[8][1] is a project which scans the entire IPv4 address space continuously and exposes the gathered data on a web interface in the form of a search engine. It also exposes various APIs which allows developers and security researchers to generate reports on queries, or query the database and consume the results themselves. Figure **9** shows the data readily available on the service for a queried IP address.

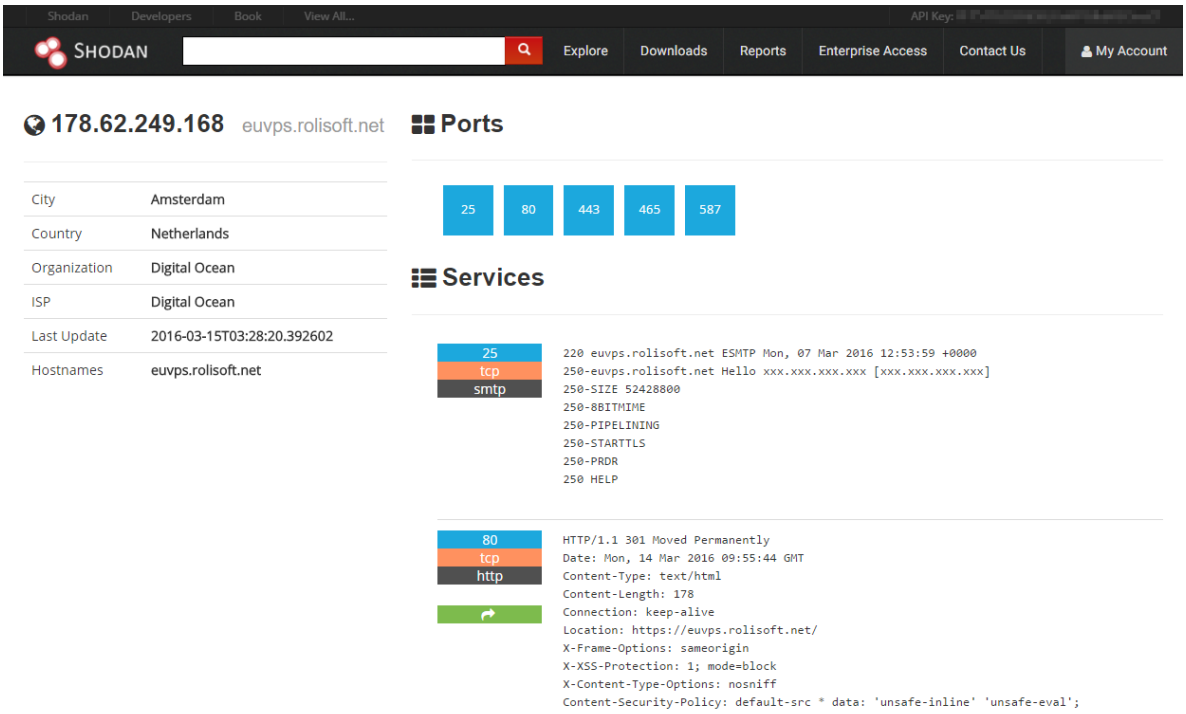


Figure 9. Shodan report on an example IPv4 address

In order to use Shodan data in the application developed within the scope of this thesis, the user has to create a free account at `https://shodan.io/` and specify the generated API key to the application via the `--shodan-key` argument.

Implementation-wise, the `ShodanScanner` component takes care of contacting the Shodan JSON API interface and retrieving the data available for the specified IP addresses. This component extends the `HostScanner` class, and as such it is possible to substitute the default scanner to Shodan data for any scanning needs.

[8]I would like to thank the creator of Shodan for upgrading my account to an unlimited plan, so that I may continue the development of the application with unrestricted API access and scan credits.

6.2.2 Censys

Censys[**2**] is a project created and run by the Regents of the University of Michigan. Similarly to the previously discussed service, it also gathers its data by regular Internet-wide scanning efforts. It allows developers and security researchers to initiate structured queries, full-text search queries or raw SQL queries from their web interface or the exposed RESTful API. Figure **10** shows the data readily available on the service for a queried IP address.

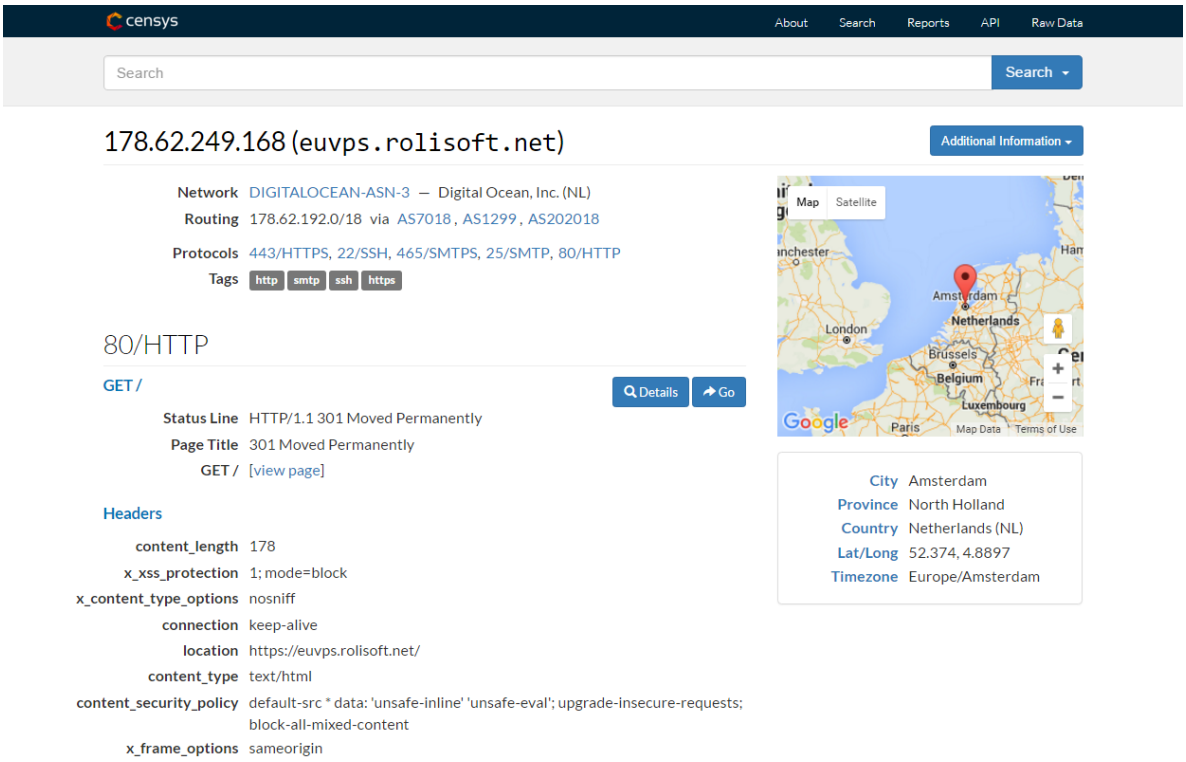


Figure 10. Censys report on an example IPv4 address

The network scanner in use by the service, namely ZMap[**56**], and its various components are open-source. The raw data generated by this scanner during an Internet-wide scan is also available for download at the Censys website for anyone to freely consume.

In order to use Censys data in the application developed within the scope of this thesis, the user has to create a free account at `https://censys.io/` and specify the generated UID and secret key to the application via the `--censys-key` argument.

Implementation-wise, the `CensysScanner` component takes care of contacting the Censys JSON API interface and retrieving the data available for the specified IP addresses. Similarly, this component also extends the `HostScanner` class, and as such it is possible to substitute the default scanner to Censys data for any scanning needs.

6.2.3 Mr Looquer

Mr Looquer[9][3] is a new open initiative with a focus on IPv6 intelligence, launched by two security researchers in the May of 2016.

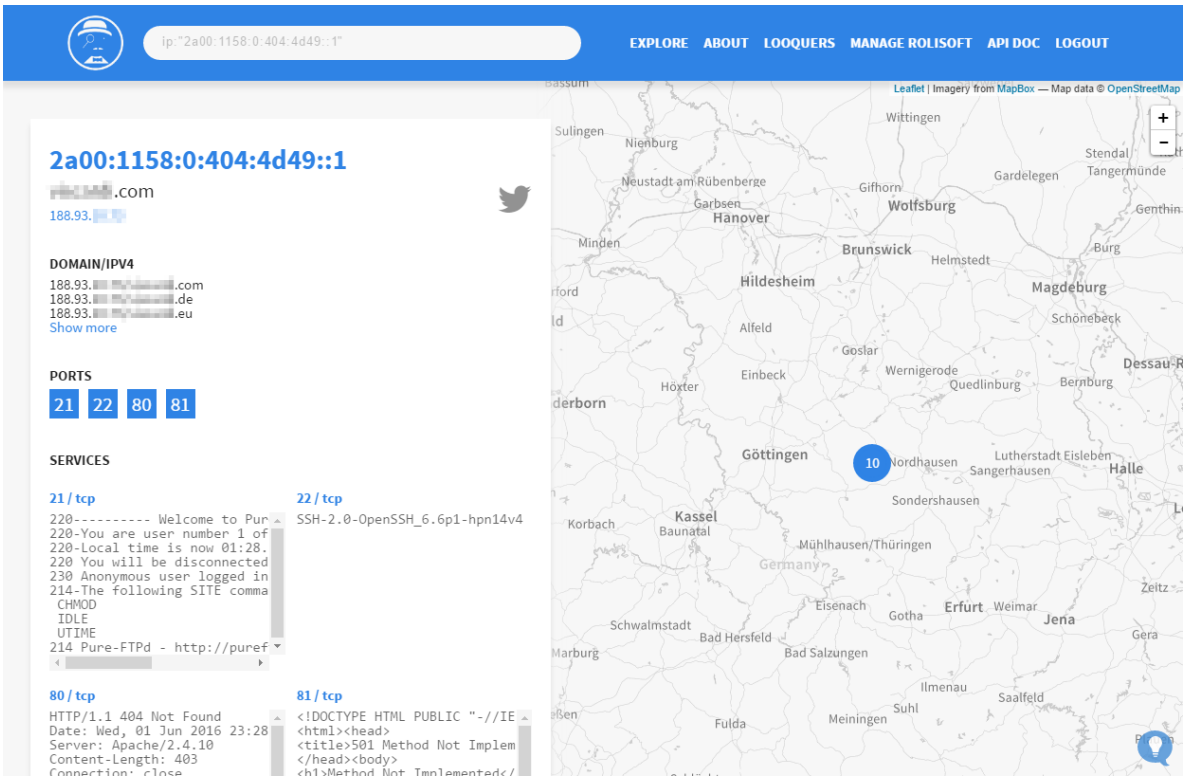


Figure 11. Mr Looquer report on an example IPv6 address

IPv6 has a much larger address space of 2^{128} addresses, as compared to the address space of IPv4 consisting of only 2^{32} addresses. Due to this increased address space, the technique used by the two aforementioned services (*Shodan* and *Censys*) of scanning the whole IPv4 address space, cannot be used in the case of IPv6. While the creators of *ZMap* claim it is possible to scan the whole IPv4 address space under 5 minutes[**56**], this speed would translate to $7.537 \cdot 10^{23}$ years for the IPv6 address space.

As such, this service uses various data acquisition techniques to gather publicly used IPv6 addresses. One example would be the registered domain list various top-level domain NICs publish on request, such as Verisign's *TLD Zone File Access Program*[**57**] for *.com* and *.net* TLDs. With this list, they can perform zone walking and use heuristics to infer relationships between the pointing domains and the reverse domains of both the IPv4 and IPv6 associated addresses.

Similarly to the previously discussed services, it also allows developers and security researchers to initiate structured queries and full-text search queries from their web in-

[9]I would like to thank the creators of Mr Looquer as well, since they were also cooperative and supported the project.

terface or the exposed RESTful API. Figure **11** shows the data readily available on the service for a queried IP address.

In order to use Mr Looquer data in the application developed within the scope of this thesis, the user has to create a free account at `https://mrlooquer.com/` and specify the generated API key to the application via the `--looquer-key` argument.

Implementation-wise, the `LooquerScanner` component takes care of contacting the Mr Looquer JSON API interface and retrieving the data available for the specified IP addresses. Similarly, this component also extends the `HostScanner` class, and as such it is possible to substitute the default scanner to Mr Looquer data for any scanning needs.

6.2.4 Amalgamation

The `PassiveScanner` component of the application initiates the requested scan tasks on all three passive components, Shodan (s. **6.2.1**), Censys (s. **6.2.2**) and Mr Looquer (s. **6.2.3**), merging the results at the end.

During the development phase of the application it was found that generally most services have had data on a requested IP address, but most of the times one of the services discovered more ports, have done much deeper analysis, or the information is more useful than at the competitor service.

One concrete example would be where one service was somehow identified and banned from indexing an IP address, as shown in listing **9**, yet the other one was able to access it freely and has data available as a result, as shown in listing **10**.

```
HTTP/1.1 403 You are banned from this site.  Please contact via a different client
↪  configuration if you believe that this is a mistake.
Content-Type: text/html; charset=utf-8
Date: Wed, 16 Mar 2016 01:17:18 GMT
Retry-After: 5
Server: Varnish
X-Varnish: 2115087
Content-Length: 590
Connection: keep-alive
```

Listing 9: Example response of 54.193.103.xyz for Shodan with a ban message

```
HTTP/1.1 200 OK
Content-Length: 3770
Vary: Accept-Encoding
Server: nginx/1.8.0
Connection: keep-alive
Last-Modified: Fri, 28 Aug 2015 19:43:39 GMT
Content-Type: text/html
Accept-Ranges: bytes

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 ↪  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<!-- rest of the page omitted -->
```

Listing 10: Example response of 54.193.103.xyz for Censys without a ban message

It should also be noted that the underlying server software is not exposed to the banned party in listing **9**, but the non-banned party in listing **10** has their request forwarded from the load-balancer to the backend, revealing itself to be "nginx 1.8.0," a version with at least one **known remotely-exploitable vulnerability**[**58**] at this time.

In cases where both services return a service banner for a specific port, the longer banner is chosen during the amalgamation phase. One of the reasons for this decision was because ban messages are generally shorter[**59**] in order to avoid denial of service attacks. The other reason was that a longer banner generally means that the service in question possibly probed the port further (e.g. `STARTTLS` command sent to SMTP servers by Censys, but not by Shodan) which allows the pattern matching component described in **6.7** to have more data available.

6.3 External Reconnaissance

This section presents the `NmapScanner` component of the application which executes an external application in order to scan the requested ports, and then parses the results of the 3^{rd}-party application for further processing within the 1^{st}-party application.

While the application implements a broad range of protocol scanners as discussed in sections **6.1.1** through **6.1.4**, with a unified interface for task parallelization as discussed in section **6.5**, as a measure of redundancy, the users are given the choice of using an external scanner for their reconnaissance purposes.

The component, as its name suggests, supports *nmap* out-of-the-box as an alternative scanner, however, any 3^{rd}-party application which generates nmap-compatible XML-based reports can be used. One such example is the *masscan* scanner.

In order to use nmap for scanning, the `nmap` executable has to be accessible from within the `PATH` environmental variable.

6.4 Class Hierarchy of Data Reconnaissance Components

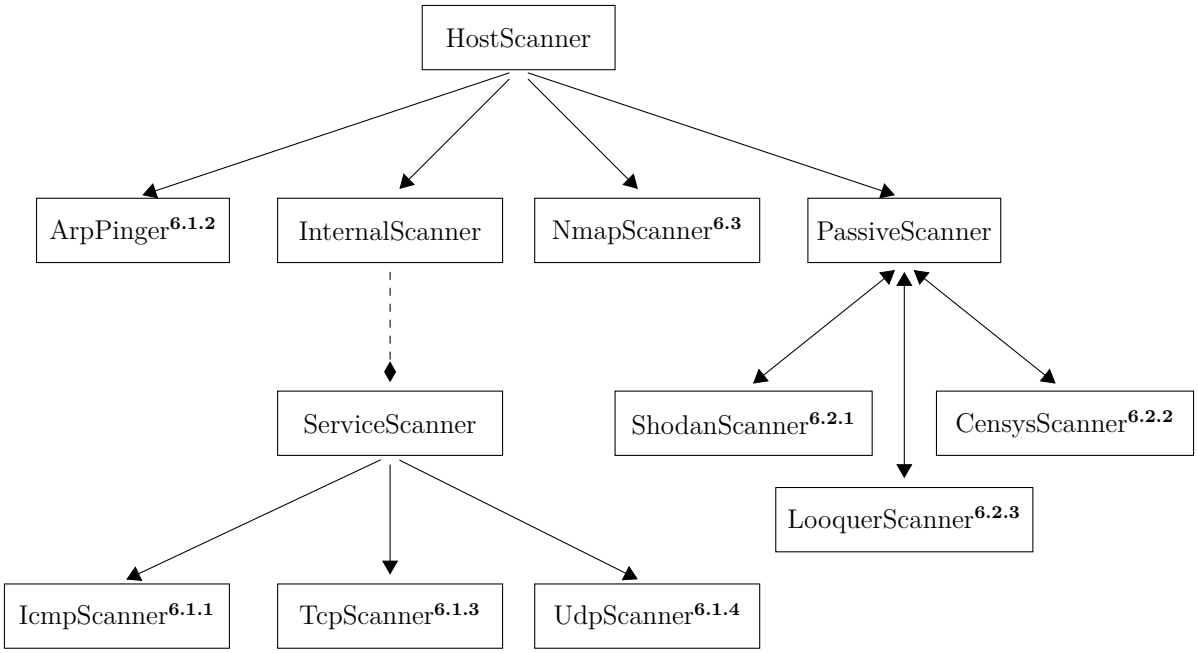


Figure 12. Class hierarchy of Data Reconnaissance components

6.5 Multiplexing Task Runner

A "task" in this context is defined to be the action of scanning a port on an IP address for a specified protocol. Such tasks mostly consist of sending IP packets back and forth followed by waiting for their acknowledgment or lack thereof.

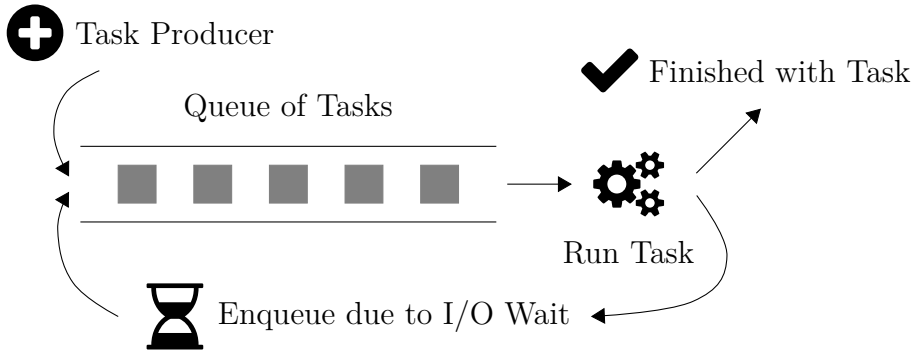


Figure 13. Consumer-producer queue process for Tasks

As such, the parallelization method of using a thread pool and multiple threads is not efficient, since most threads would spend their time sleeping. Given the costly nature of thread spawning and managing, this is not efficient unless more lightweight threads can be spawned, such as 'goroutines' in the Go programming language[**60**].

The application instead implements a consumer-producer queue for storing and retrieving its tasks during scheduling, as shown in figure **13**.

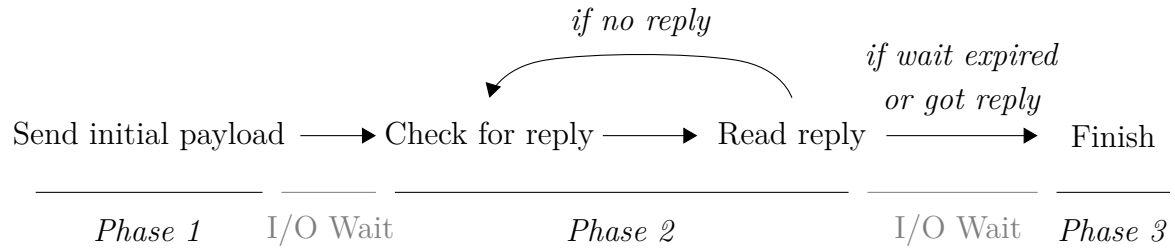


Figure 14. Phases of a UDP scan as implemented by component in section **6.1.4**

Figure **14** presents the phases of a UDP scan task. Each 'phase' shows up in the queue as a different 'task.' As a result of this scheduling and task multiplexing, one thread can handle all the scanning tasks needs.

Figure **15** represents the sequence diagram of the process through which tasks are initiated and placed onto the queue for multiplexed execution.

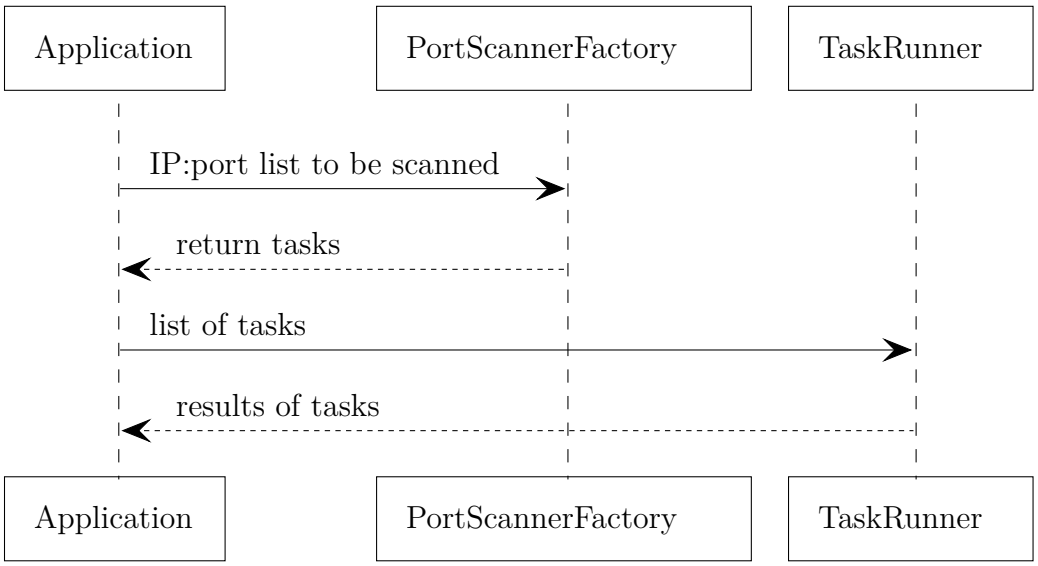


Figure 15. Sequence diagram for Task Instantiation and Multiplexed Execution

All tasks are implemented to use their sockets in a non-blocking mode, as such, as soon as one task has sent an IP packet and is about to wait for the results of its I/O operation, it instead goes into the back of the queue.

Once the whole queue has been looped to collect the results of the previously initiated I/O operations, the aforementioned task will have a chance to collect its own operation results.

The multiplexing task runner uses a lock-free thread-safe underlying queue container, namely the concurrent FIFO implementation in Boost, `boost::lock free::queue<T>`. As

a result, for heavy workloads when excessive resources are available, the task runner can theoretically use multiple consumers on the same queue.

6.6 Protocol Tokenization

In order to improve the quality of the input for the CPE matcher component (further discussed in **6.8**) and thereby minimizing the chances of false positives, the application makes a best effort to try recognizing and tokenizing the extracted service banners.

The server replies are not fully parsed using a protocol-aware parser, instead it aims to either *a)* extract server names, versions and operating system tags, or if that is not possible, *b)* clean any known protocol strings, leaving only implementation-specific strings in the service banner.

Currently, there are two protocol tokenizers implemented, and these are run in order of protocol popularity when a service banner needs to be cleaned. An API is exposed which does this, `ProtocolTokenizer::AutoTokenize(const std::string& banner)`.

Figure **16** represents the sequence diagram of the process through which an unidentified service banner is automatically tokenized via the aforementioned exposed API.

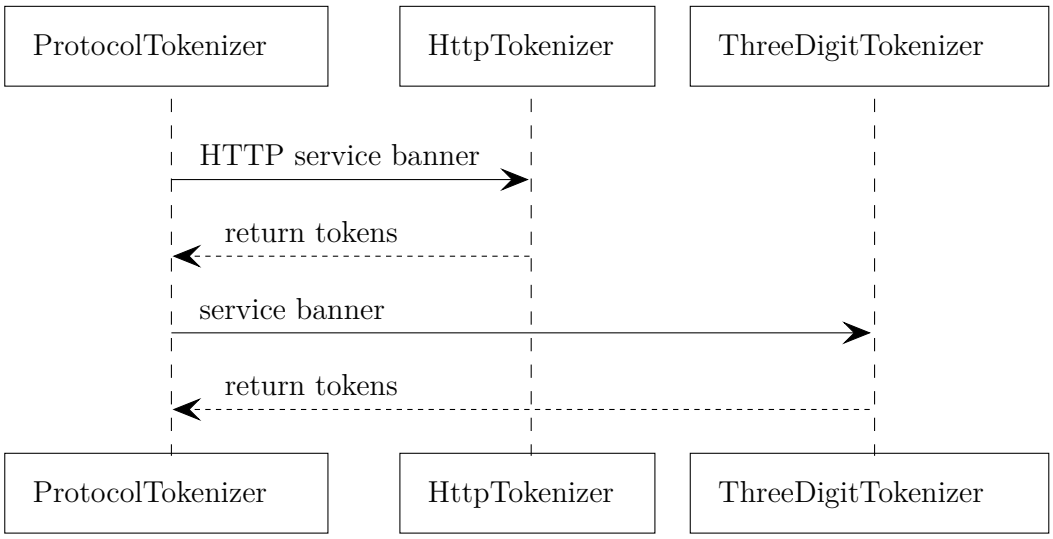


Figure 16. Sequence diagram for Protocol-based Tokenization of Service Banners

6.6.1 Hyper-Text Transfer Protocol Tokenizer

The first tokenizer is `HttpTokenizer`, which decides whether the specified service banner contains a valid HTTP header and if so, proceeds to tokenize it. During tokenization, it will try to extract product names and version numbers from the appropriate places.

The HTTP protocol has the 'Server' and 'X-Powered-By' header fields, which are generally used by software to indicate their name and version number. Unfortunately, the exact listing methodology is not standardized, as such different software may use

different separators and notation to indicate their existence, version number, any vendor patches and possibly the operating system. Since these fields may have multiple products listed, the tokenizer makes sure to extract all the product names including any associated version numbers into separate tokens.

```
HTTP/1.1 200 OK
Server: nginx/1.9.12 (Ubuntu)
X-Powered-By: PHP/5.6.19
Date: Fri, 11 Mar 2016 16:04:07 GMT
Connection: close
```

Listing 11: Example HTTP service banner

The HTTP service banner shown in listing **11** is produced by nginx 1.9.12 with PHP 5.6.19 installed on an Ubuntu distribution. When processed by the implemented tokenizer, it will produce the elements shown in listing **12**.

```
[
    // token,           later mapped to,           by component
    "nginx/1.9.12",  // cpe:/a:nginx:nginx:1.9.12, CpeDictionaryMatcher
    "Ubuntu",        // cpe:/o:canonical:ubuntu,   OperatingSystemIdentifier
    "PHP/5.6.19"     // cpe:/a:php:php:5.6.19,     CpeDictionaryMatcher
]
```

Listing 12: Extracted tokens from banner in listing **11**

However, if the service banner being analyzed does not contain a valid HTTP header, the next tokenizer in order of protocol popularity to be tried is the `ThreeDigitTokenizer`, from now on referred to as the "SMTP tokenizer" for simplicity's sake.

6.6.2 Generic Fallback Tokenizer

The "three-digit" tokenizer is a general purpose solution for parsing protocols which use a three-digit response in their protocol to indicate message type/category of a given server reply. Such protocols include SMTP, NNTP, FTP and a few more. For these protocols, however, unfortunately, there is no standardized way to announce server name and version (like the 'Server' header in the HTTP protocol) and as such the server name is generally casually announced in the informational level welcome message part of the service banner. The informational messages are generally within the range of $200 - 299$. However, this might vary depending on the actual protocol.

```
220 example.com ESMTP Exim 4.87 #2 Fri, 11 Mar 2016 16:05:06 +0000
```

Listing 13: Example SMTP service banner

The SMTP service banner shown in listing **13** is produced by Exim 4.87 listening on port 25. This banner is sent as a 'greeting' as soon as the client connects to the server. This is favorable since no further protocol probes are required to be sent in order to get a processable service banner. When this banner is processed by the implemented tokenizer, it will produce the elements shown in listing **14**.

```
[
    // token,                later mapped to,       by component
    "ESMTP Exim 4.87 #2", // cpe:/a:exim:exim:4.87, CpeDictionaryMatcher
]
```

Listing 14: Extracted tokens from banner in listing **13**

It should be noted that the "perfect" token would be `Exim 4.87`, however as previously mentioned, the server name and version are not clearly advertised. As such, the processing of the banner starts by removing the protocol-specific strings, namely the response code (`220`), the host name (`example.com`), and the date (`Fri, 11 Mar 2016 16:05:06 +0000`). This leaves us with the implementation-specific string of `ESMTP Exim 4.87 #2`.

The tokenizer implementation will try to remove as many non-implementation-specific tokens as possible, however in cases where a specific element cannot be determined with a high degree of certainty whether it is a protocol or implementation-specific string, the tokenizer will instead leave it in. This decision was made in order to ensure that the tokenizer does not remove any elements which are required for the CPE matcher during entry matching. If an element is left in falsely, it does not affect the scoring of the CPE matcher, however, if it was falsely removed, it could completely prevent the matching of that entry.

If all of the implemented tokenizers fail to process a service banner, it will be sent to the next stage of discovery (usually to component described in **6.8**) as one token, containing the whole service banner.

6.7 Pattern Matching of Service Banners

The purpose of the `ServiceRegexMatcher` component of the application is to take the original full service banner (without any tokenization) as an input and match it against a database of regular expressions, which in turn are mapped to their own CPE names.

```
^HTTP/1\.[01] \d{3}.*\r\nServer: nginx(?:/([\d.]+))?
```

Listing 15: Example regular expression to match `cpe:/a:nginx:nginx`

The example regular expression in listing **15** matches against the response headers produced by the HTTP server software "nginx". See the service banner in listing **11** for

an example that would be matched. Furthermore, the listed regular expression contains an optional capture group, which captures the version number. If the version number is listed in the service banner, the matcher component will return the CPE name `cpe:/a:nginx:nginx:1.9.12`, while without a version number listed, it will still return the CPE name `cpe:/a:nginx:nginx`.

This behavior is different from the matcher described in section **6.8**: whereas here a CPE name can be produced with high confidence with or without a known version number, the CPE dictionary-based matcher *requires* a known version number, since it plays a pivotal role in the identifying and scoring process.

Figure **17** represents the sequence diagram for the process of pattern-matching unidentified service banners. The "BannerProcessor" component of the sequence diagram acts as an intermediary, making sure to merge the returned information properly from the "ServiceRegexMatcher" component into the host object, as the latter class only contains the regular expression matching implementation.

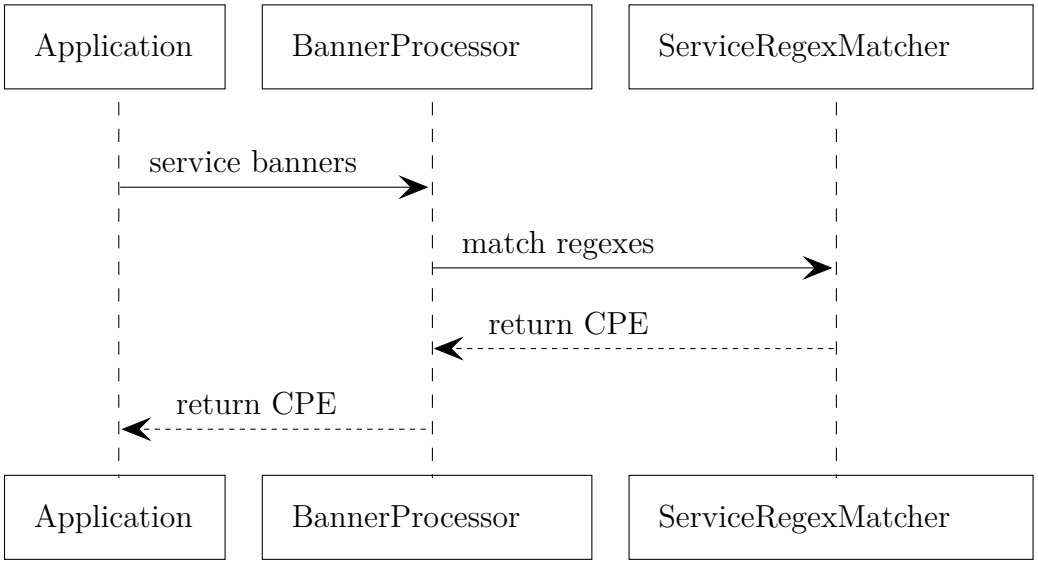


Figure 17. Sequence diagram for Pattern-based Matching of Service Banners

The regular expression database is revisited and discussed in greater detail in section **6.13.4**.

6.7.1 Inferring Products Without Express Announcement

While the example presented in listing **15** is rather straight-forward, the pattern matcher method can be used in a more subtle way, for example, to match software which is not advertising their name or version number.

```
1 ^554 SMTP synchronization error\r\n
```

Listing 16: Example regular expression to match `cpe:/a:exim:exim`

The example presented in listing **16** is a regular expression which matches an error message produced by the SMTP server software "exim". The reason why it is possible to determine this fact with high confidence is that exim is the only software returning this exact error message verbatim. Other SMTP servers will have a similar error message for this problem, with the same error code, but not with this same exact error message, as the messages themselves are not standardized byte-by-byte.

```
220 example-1.com ESMTP Sat, 12 Mar 2016
 ↪  17:14:06 +0000
EHLO client-1.com
250-example-1.com Hello client-1.com
 ↪  [2a02:2f07:d18d:1100::cake]
250-SIZE 52428800
250-8BITMIME
250-PIPELINING
250-STARTTLS
250-PRDR
250 HELP
```

Listing 17: Exim $\geq$ 4.83

```
220 example-2.com ESMTP Sat, 12 Mar 2016
 ↪  16:12:25 +0000
EHLO client-2.com
250-example-2.com Hello client-2.com
 ↪  [2a02:2f07:d18d:1100::cake]
250-SIZE 20971520
250-8BITMIME
250-PIPELINING
250-AUTH PLAIN LOGIN
250-STARTTLS
250 HELP
```

Listing 18: Exim $<$ 4.83

It is even possible to associate a version number to the software behind the port. For example, exim has implemented "SMTP Service Extension for Per-Recipient Data Responses" in version 4.83, and it advertises this capability as 'PRDR' after the handshake, as seen in listing **17** versus the handshake of an older version in listing **18**.

A regular expression can be written to inspect the advertised capability list, and make an educated guess stating that the version of the software is 4.83 or older. This can be further improved by creating a pattern which matches a change which only applies to versions 4.86 and older, at which point it can be deduced that the inspected service banner was produced by exim between the versions of $4.83 - 4.86$.

For open-source software, it is possible to compare the source between different releases and check which publicly visible string changed, in order to write a pattern for detecting that range of versions. Building such a database of patterns, however, is beyond the scope of this project, and would be more suited for a community-sourced project.

6.8 Matching of CPE Tokens in Service Banners

As discussed in **2.3**, the *National Institute of Standards and Technology* runs a *National Vulnerability Database*. The `CpeDictionaryMatcher` component presented within this section makes use of the publicly distributed, freely available and daily updated *Common Platform Enumeration Dictionary*.

The *CPE* is a naming scheme for hardware, software and operating systems[**18**]. Its v2.2 format is `cpe:/type:vendor:product:version:update:edition:language` where the

'type' component can be `h` for 'hardware', `o` for 'operating system' and `a` for 'application', while the rest of the components are self-explanatory.

For example, the CPE name for "nginx 1.3.9" is `cpe:/a:igor_sysoev:nginx:1.3.9`.

The aforementioned *CPE dictionary* is a list of CPE names aggregated and maintained by NIST, which are known to be vulnerable, i.e. have associated entries in the *CVE database*.

In listing **19** an excerpt is shown, presenting an entry from the dictionary for the "nginx 1.9.9" software.

```
<cpe-item name="cpe:/a:nginx:nginx:1.9.9">
    <title xml:lang="en-US">Nginx 1.9.9</title>
    <references>
        <reference href="http://nginx.org/">Product</reference>
        <reference href="http://nginx.org/en/CHANGES">Change Log</reference>
    </references>
    <cpe-23:cpe23-item name="cpe:2.3:a:nginx:nginx:1.9.9:*:*:*:*:*:*:*"/>
</cpe-item>
```

Listing 19: CPE entry for nginx 1.9.9

Unfortunately, the CPE names are not advertised in the service banners, nor is there a direct standard to map CPE names to service banners, or any other straight-forward solution on mapping them. In paper [**13**], the authors have tackled the same issue of processing and mapping the entries to service banners. The method presented within the paper was the basis for the implementation of the CPE matcher component in the application developed within the scope of this thesis.

6.8.1 Implementation Overview

The current implementation of the CPE matcher component loads a preprocessed version of the CPE dictionary into its memory, where the CPE names are tokenized, and irrelevant data (such as product links) are stripped for memory efficiency. Listing **20** presents the tokens loaded into memory for an example CPE name. For the exact implementation, see the structures within the header file of the `CpeDictionaryMatcher` class.

The CPE dictionary is revisited in section **6.13.1**, discussing the preprocessing of the published data into a harmonized binary data format for performance and memory efficiency.

```
CpeEntry {
    // cpe:/o:cisco:ios
    "ProductSpecificTokens": ["cisco", "ios"],
    "Versions": [
        CpeVersionEntry {
            // cpe:/o:cisco:ios:12.2sxi
            "VersionNumber": "12.2",
            "VersionSpecificTokens": ["sxi"]
        },
        CpeVersionEntry {
            // cpe:/o:cisco:ios:12.2sxh
            "VersionNumber": "12.2",
            "VersionSpecificTokens": ["sxh"]
        }
        // [further versions omitted]
    ]
}
```

Listing 20: Approximate internal representation of tokens for `cpe:/o:cisco:ios:12.2sxi`

During the tokenization process, the 'vendor' and 'product' components of the CPE name are matched against the regular expression `([a-z][a-z0-9]+)` in order to extract all words individually, ignoring one character words and those starting with a number.

Doing so, the resulting array will end up looking like `["apache", "tomcat"]` for the example CPE name `cpe:/a:apache:tomcat:4.1.36`.

The version component of the CPE contains in some cases irrelevant characters or non-standard version notation. In order to work around this, the version number is extracted from the component using the regular expression `\d+\.(?:\d+\.)*\d+`, which requires at least two numbers separated by a dot.

If the version number contains any words, these are extracted with the aforementioned regular expression in the tokenization phase. As such, in the case of the CPE name `cpe:/o:cisco:ios:12.2sxi`, there will be an array of 'version-specific tokens' consisting of the word `["sxi"]`.

During CPE matching, the entries are iterated, checking to see if the product-specific tokens match. If all of the tokens match, the entry-specific versions are iterated, checking to see if any of the version numbers are present in the input. If so, version-specific tokens are checked as well. A version is not considered a match if the version number matches, but even just one of the version-specific tokens do not.

There are entries where multiple versions may match within the same product, such edge cases are discussed in subsection **6.8.2**. In situations like these, a single version number is selected as the winner, which is determined by the distance of the tokens to the version number. The further a token is from the version number, the lesser it is considered to be relevant.

For product names, however, if multiple products (including their complete version

numbers with tokens) match, all of the matches will be returned. The reason for this decision is because a service banner may name multiple software in use, but not multiple versions of the same software to be in use. In listing **11**, the service banner reveals the existence of three products, namely "nginx 1.9.12," "PHP 5.6.19" and "Ubuntu."

Figure **18** represents the sequence diagram of the process through which an unidentified service banner is tokenized and matched against the database of known tokens. The "ProtocolTokenizer" component of the sequence diagram has its own sequence diagram visible in figure **16**. The "BannerProcessor" component acts as an intermediary, making sure to properly merge the returned information from the "CpeDictionaryMatcher" component into the host object, as the latter class only contains the token matching implementation.

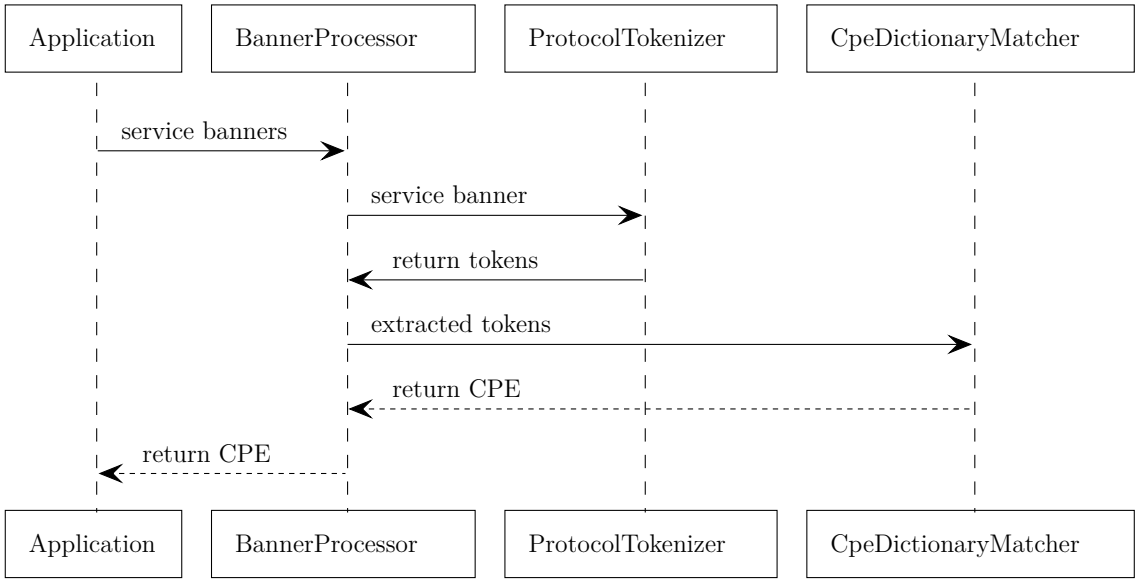


Figure 18. Sequence diagram for CPE Token-based Matching of Service Banners

6.8.2 Handling of Edge Cases

The now-defunct *Sun Microsystems* vendor identifier is "sun", resulting in CPE names such as `cpe:/a:sun:jre`, `cpe:/a:sun:jdk` and `cpe:/o:sun:solaris:10.0`. The problematic part with this token is that most protocols, such as SMTP and HTTP, return the current date in the RFC 1123 format[**61**], which looks like this: "Sun, 14 Mar 2016 16:33:02 GMT".

The first component of the date is the shortened three-letter English name of the day, which on Sundays is "Sun". This introduces a 'random' element into the equation, as the scoring system of the CPE matcher would rank CPEs from the *Sun Microsystems* vendor higher, as the "sun" token is now present in the service banner.

```
1 Cisco IOS Software, s72033_rp Software (s72033_rp-IPSERVICESK9_WAN-M), Version
   ↪  12.2(33)SXI3, RELEASE SOFTWARE (fc2)
2 Technical Support: http://www.cisco.com/techsupport
3 Copyright (c) 1986-2009 by Cisco Systems, Inc.
4 Compiled Tue 27-Oct-09 11:12 by prod_
```

Listing 21: Example telnet service banner of Cisco routers

Another example would be Cisco's version numbering and their telnet service banners. In listing **21**, the service banner of `cpe:/o:cisco:ios:12.2sxi3` is shown. The problematic part arises from the fact that Cisco has the same version number 12.2 with the patch level specified as "by": `cpe:/o:cisco:ios:12.2by`.

If one were to tokenize the two CPE names, they would get two arrays which would look like these: `["cisco", "ios", "sxi"]` and `["cisco", "ios", "by"]`. The version number and the first two tokens from both arrays would match, however so would both "sxi" and "by". On lines 3 and 4 of **21** the copyright and compilation notices both contain the word/token "by".

The aforementioned ShoVAT[**13**] paper solved this issue by weighing tokens around the version number more than those further from the version number, which solution is ultimately what the application written within the scope of this thesis has also settled with on one hand.

However, a different solution was also implemented to combat this. The protocol tokenizer component discussed in **6.6** was developed for the express purpose of extracting product names and versions from a service banner, or if that is not possible, then removing irrelevant parts, such as dates and non-informational messages.

6.9 Identification of Operating Systems

The techniques discussed in the sections **6.7** and **6.8** cannot be used to identify the operating system of the scanned host, as in most of the cases, these services do not advertise the underlying operating system the server software is running on.

The application implemented within the scope of this thesis supports the identification of **Debian**, **Ubuntu**, **Red Hat**, **CentOS** and **Fedora** Linux distributions. For these distributions, the exact version number of the installed operating system can be deduced as well, not just the name.

After successful identification, the CPE names and CVE vulnerabilities discovered on the system can be traced back to a specific package in the distribution's package manager, thus allowing *vulnerability validation* (further discussed in **6.11**) by checking with the vendor, whether the discovered vulnerability actually affects the active installation or not.

Support for **Windows** is currently limited: the operating system itself can be identified, however due to the lack of a centralized package manager, the CPE to package

name and vulnerability validation functions are not available for scanned hosts running such operating systems.

Identification of the operating systems based on the gathered results are made by separate `*Identifier` classes, where one implementation exists for each operating system, and they are invoked in order of popularity until a certain hit is produced.

A common theme within these identifier implementations is analyzing the discovered SSH server, as the SSH protocol requires every server to identify themselves accurately for compatibility reasons.

While most server software have an option to decrease the verbosity of the announced software information (such as name, version, patch level and operating system) due to the aforementioned protocol requirements, no SSH server (at least on the supported Linux distributions) allows silencing the information contained within the service banner of the SSH service.

In Debian or Debian-based (such as Ubuntu) distributions, the `openssh` package is modified to further increase the amount of information announced, as its service banner also contains the installed security patch version number as well. This behavior can be turned off using the `DebianBanner` configuration option. However, it is on by default, and it is not common practice to turn it off after a fresh installation.

If the extended version information announcement is not turned off, the distribution can be deduced from that. Otherwise, the SSH server version number can be looked up against the list of bundled OpenSSH packages in each distribution generation, as shown in table **3** for Debian or table **4** for Ubuntu.

The “Enterprise Linux” distributions do not suffer from this, however, due to the nature of the release lifecycle of these distributions, multiple years may pass between the versions, and as such the SSH server version number will increase in the newer versions, but stay the same in the current generation ones.

As a result, if the software is otherwise certain that the host is actually running one of these distributions, it is possible to tell the exact version number of a host running Red Hat or CentOS, by matching the version number of the discovered SSH service to the list of the version numbers of the SSH packages as shipped between different versions of the specific operating system distribution, as shown in table **5**.

Debian	OpenSSH
Stretch (9)	7.2p2
Jessie (8)	6.7p1
Wheezy (7)	6.6p1 – 6.0p1
Squeeze (6)	5.5p1
Lenny (5)	5.1p1
Etch (4)	4.3p2
Sarge (3.1)	3.8.1p1
Woody (3)	3.4p1

Table 3. Bundled OpenSSH packages in each Debian distribution

Ubuntu	OpenSSH
Xenial (16.04)	7.2p2
Wily (15.10)	6.9p1
Vivid (15.04)	6.7p1
Utpic (14.10)	6.6p1
Trusty (14.04.4)	6.6.1p1
Trusty (14.04)	5.9p1
Lucid (10.04)	5.3p1
Hardy (8.04)	4.7p1
Dapper (6.06)	4.2p1

Table 4. Bundled OpenSSH packages in each Ubuntu distribution

E.L.	OpenSSH
7	6.6.1p1
6	6.6p1, 5.3, 5.2p1
5	4.3
4	3.9

Table 5. Bundled OpenSSH packages in each Red Hat/CentOS distribution

Tables **3**, **4** and **5** were compiled manually, by navigating to the web interface of each distribution's package manager, and looking up historical release information for the "openssh" package. This list is hard-coded into the application, as it is not bound to change anymore.

For operating systems where only identification is supported, without further interactions, such as Windows, the identification is achieved by analyzing the discovered service banners for various tokens unique to the platform. For example, the Windows identifier will look for "Cygwin" or "Win32" in the SSH banner, or the existence of platform-exclusive server software, such as an HTTP server identify itself as "Microsoft IIS".

Figure **19** represents the sequence diagram for the process through which a collection of service banners are processed on a per-host basis, in order to determine the operating system and version of the host. The "*Identifier" component of the sequence diagram represents all the implemented operating system identifier classes, which are enumerated in order of popularity until an implementation successfully returns.

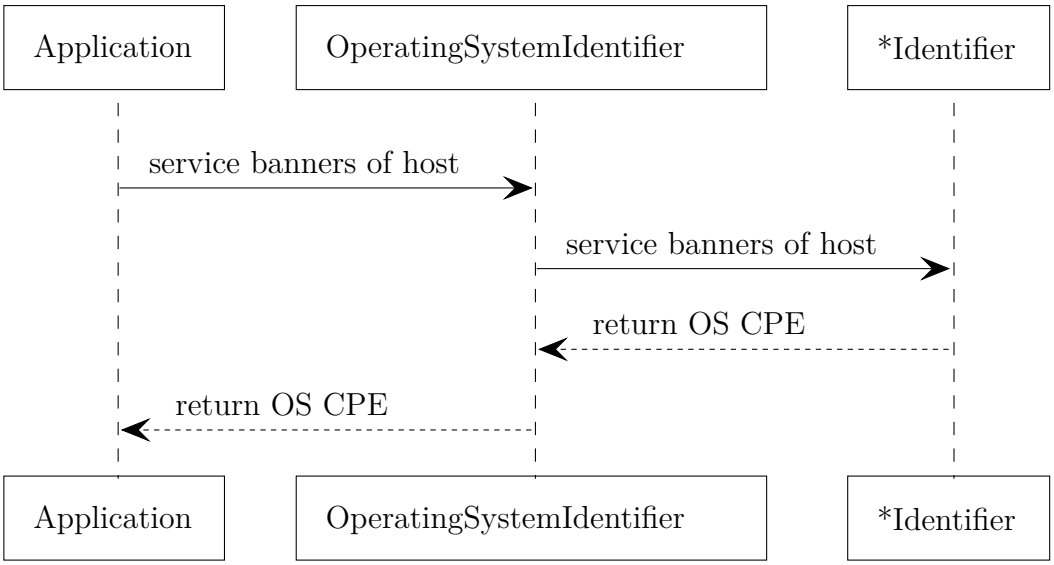


Figure 19. Sequence diagram for Operating System Identification

6.10 Vulnerability Lookup

This section presents the methodology which allows the application to find the vulnerabilities of an identified CPE name.

The vulnerability database in used by the application is *NIST*'s *National Vulnerability Database*, as discussed in section **2.3**, which is an XML file containing *CVE* entries. A CVE entry represents a single vulnerability, containing its description, *CVSS* scoring (as discussed in **2.5**), and a list of the affected CPE names (as briefly mentioned in **6.8**).

When searching for vulnerabilities, the application traverses the list of CVE entries, and checks if any of the CVE entries' `<vulnerable-software-list />` contains the CPE name in question.

When comparing CPE entries, all components of the CPE have to match in order to declare the application vulnerable. It should be noted, that fields which do not have a value are considered to be wildcard components, and match to any value.

For example, `cpe:/o:microsoft:windows:10` refers to *Windows 10*, while the *Hungarian* version of Windows 10 is referred to as `cpe:/o:microsoft:windows:10:::hu`. When referred specifically, it means that the other language versions of the same product are not affected. However, if a CVE entry only refers to `cpe:/o:microsoft:windows:10`, it is assumed that other fields are set to `*`, being equal to `cpe:/o:microsoft:windows:10:*:*:*`, and therefore making the Hungarian version of the product affected as well.

Figure **20** presents the sequence diagram of the process through which CPE names are resolved to CVE entries within the application. The "Database" component of the diagram represents the aforementioned National Vulnerability Database. However, the application itself is not querying the published XML file directly, it instead is querying a SQLite database, which is filled with the relevant fields of the published XML file. This is done in order to increase performance and lower memory consumption of each lookup request.

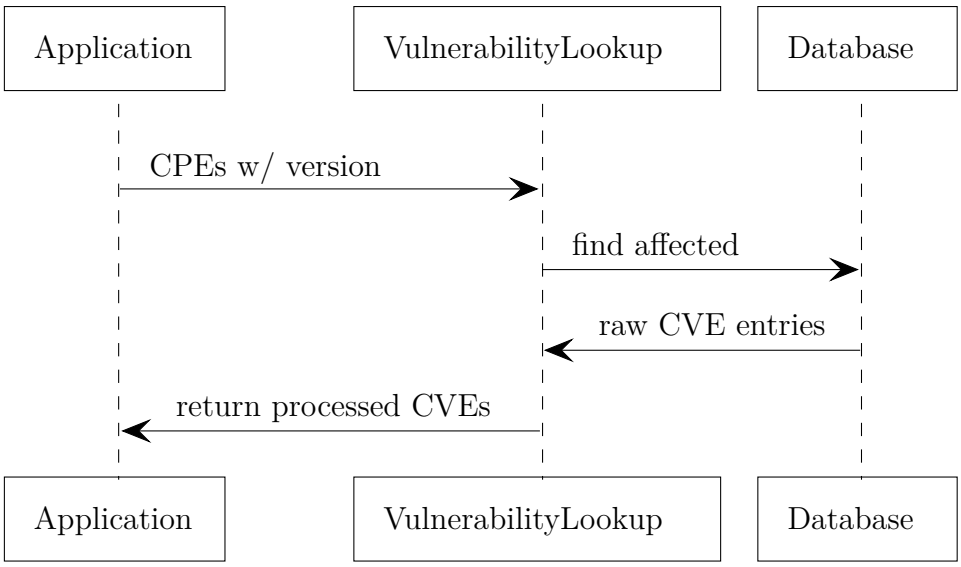


Figure 20. Sequence diagram for Vulnerability Lookup

Section **6.8** briefly introduces the CPE notation, giving an example to "nginx 1.9.9" in listing **19**. Continuing this example, listing **22** presents the raw XML entry for the CVE item "CVE-2016-0742", which affects the aforementioned server software by having its CPE name in the list of vulnerable software.

The CVE database is revisited in section **6.13.5**, discussing the preprocessing of the published data into a relational SQL database format for performance and memory efficiency.

```
<entry id="CVE-2016-0742">
    <summary>The resolver in nginx before 1.8.1 and 1.9.x before 1.9.10 allows remote
    ↪ attackers to cause a denial of service (invalid pointer dereference and worker
    ↪ process crash) via a crafted UDP DNS response.</summary>
    <vulnerable-software-list>
        <product>cpe:/a:nginx:nginx:1.9.9</product>
        <product>cpe:/a:nginx:nginx:1.9.8</product>
        <product>cpe:/a:nginx:nginx:1.9.7</product>
        <product>cpe:/a:nginx:nginx:1.9.6</product>
        <!-- [several other versions omitted] -->
    </vulnerable-software-list>
    <cvss>
        <base_metrics>
            <score>5.0</score>
            <access-vector>NETWORK</access-vector>
            <access-complexity>LOW</access-complexity>
            <authentication>NONE</authentication>
            <confidentiality-impact>NONE</confidentiality-impact>
            <integrity-impact>NONE</integrity-impact>
            <availability-impact>PARTIAL</availability-impact>
            <source>http://nvd.nist.gov</source>
        </base_metrics>
    </cvss>
    <references xml:lang="en" reference_type="UNKNOWN">
        <source>CONFIRM</source>
        <reference href="https://bugzilla.redhat.com/show_bug.cgi?id=1302587"
    ↪ xml:lang="en">Bug 1302587</reference>
    </references>
    <references xml:lang="en" reference_type="UNKNOWN">
        <source>DEBIAN</source>
        <reference href="http://www.debian.org/security/2016/dsa-3473"
    ↪ xml:lang="en">DSA-3473</reference>
    </references>
    <!-- [several other references omitted] -->
    <published-datetime>2016-02-15T14:59:00.107-05:00</published-datetime>
    <last-modified-datetime>2016-02-29T18:40:11.667-05:00</last-modified-datetime>
</entry>
```

Listing 22: CVE-2016-0742 entry affecting nginx 1.9.9

6.11 Vulnerability Validation

The act of vulnerability validation is the phase in which the discovered vulnerabilities are checked whether they actually affect the discovered service or not.

One obvious way to validate a vulnerability is by exploiting it. If the exploitation was successful, then the service is vulnerable. This, however, is unfeasible, as the vulnerability database (NVD) does not contain enough information in order to automatically generate and launch an exploitation attempt against a described vulnerability.

Alternatively, even if it was possible to do so, in production environments it may not be desirable to launch exploits, as in case they may succeed, they may ultimately end up affecting the normal operation of the service, causing minor to major inconveniences on the public-facing service.

However, in such enterprise settings, homogeneous environments are preferred, where the operating system's official package manager is used to install and update server software. This has the added advantage, of having information about the exact installed package, in case the operating system was successfully identified and is supported.

For the operating systems listed as fully supported in section **6.9**, regarding the identification of the various Linux distributions and their versions, the application supports querying the distributions' package repository, miscellaneous build system or dedicated security tracker for any CPE names or CVE vulnerabilities, in order to determine whether they are affected and whether a security fix is available.

After querying an authoritative source, the application can discard vulnerabilities which do not affect the discovered operating system and package combination (e.g. `NOT-FOR-US` tag on the Debian Security Tracker) or otherwise check if the proper security update which addresses the vulnerability was applied or not.

Just like in section **6.9**, the package lookup functionality is laid out in separate `*Lookup` classes, having one implementation for each supported operating system. These classes are also the ones responsible for identifying the specific names of the packages, and generating the command line instructions which apply the specific software upgrades.

Figure **21** presents the sequence diagram of the process through which an identified CVE vulnerability is validated for a host with an identified and supported operating system through its package manager.

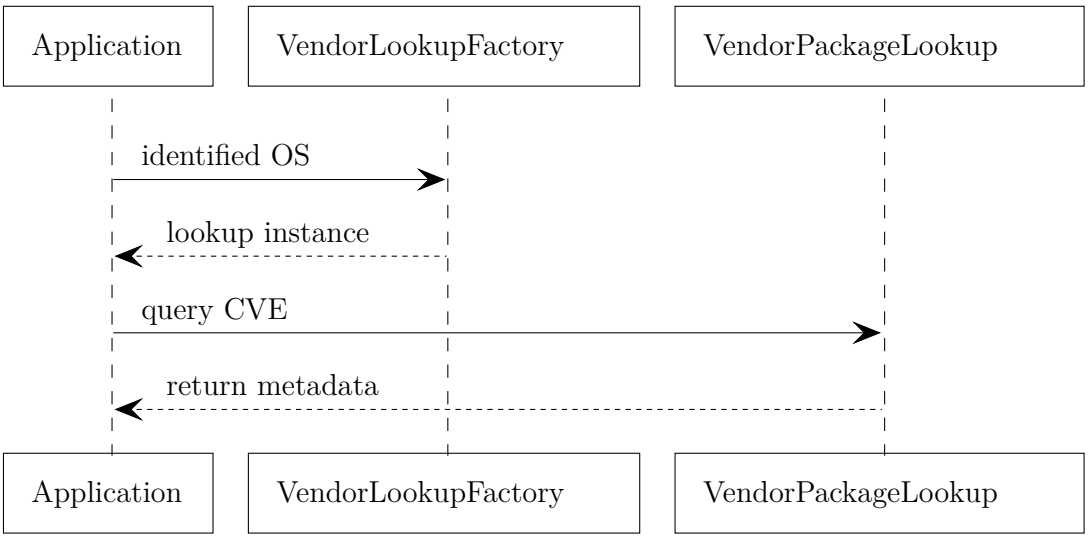


Figure 21. Sequence diagram for Package Manager-based Vulnerability Validation

6.12 Estimation of the Last System Upgrade Date

Some software, besides the usual application name and version number in the service banner (e.g. `PHP/5.5.12`), also announce the vendor-specific (e.g. `PHP/5.5.12-2ubuntu4.4`) patch level, which can be used to identify whether a security fix was applied to the software, or not.

The components discussed in sections **6.9** and **6.11** can work together in order to download the changelog of the identified package from the identified operating system's package manager.

Since these changelogs contain the list of all minor and major releases with their associated dates, it is possible to estimate the date of the last system update of the host to some degree of accuracy, by starting the date range from the date when the installed patch was initially released, and ending the range right before the next patch was released. It can be assumed, that the last system update was initiated in this date range, as this security patch was the last one available for installation at the actual update time.

If multiple packages can be identified to the patch level, the date range can be significantly reduced, thus increasing the accuracy of vulnerability validation on services whose exact patch level is not known.

Continuing the example above, the mentioned security update was released on the April 17th of 2015, while the next security update under the version of `5.5.12-2ubuntu4.6` was released on July 6th of 2015. This information can be easily obtained from the published changelog of the "php5" package in the online interface of the Ubuntu package manager: `https://launchpad.net/ubuntu/utopic/+source/php5/+changelog`

The estimation of the last system update is a useful functionality, as it can provide vulnerability validation for services whose exact patch level is not known. If by the start of the guesstimated update date range, a partially identified service already had a security update for an identified vulnerability, it can be freely discarded, as the corresponding security update was most likely applied later in the date range during the update.

This functionality is, of course, only guesswork, and can be freely turned off in order to avoid false negatives during validation.

6.13 Data Preprocessing

The data being used by the application is first processed and converted into another format from the various sources. This is being done for multiple reasons. First and foremost, to harmonize the databases from the various XML, JSON, CSV and arbitrary plain-text formats to a generic binary format. The binary format is preferred due to performance and memory efficiency, which are important factors considering the wide range of environments the application is planned to be ran in, ranging from small VPSes to being virtualized in mobile devices[**62**].

The preprocessor component is a separate project, comprised of shell and Go scripts,

which download the freshest available data and then convert it into the requested format. As noted in section **6**, these scripts are available from a different repository, and they are under a permissive non-copyleft license, as opposed to the other components. This change was made in order to allow the reuse of the scripts and generated output in 3^{rd}-party projects, without any restrictions.

The file format which the source data is converted to is a generic binary format. All the data file types start with the header shown in listing **23**. The "Package type" field denotes which parser should follow, and "Package version" is useful for allowing future modifications to the format, as clearly denoting the version number allows the parser to decide whether the format is supported, or if any workarounds are required to be used during parsing.

```
+ uint16      Package type
| uint16      Package version
+[uint32      Number of entries]
```

Listing 23: Header of the utilized binary format

All integers are written in little-endian format. This endianness has been chosen due to the fact that Intel x86 and x86-64 architectures are little-endian, thereby excluding the need to convert each integer during read. However, the `DataReader` component has compile-time checks to include endian conversion in case the target architecture is big-endian.

```
+ uint16      String length
+- char       Characters
```

Listing 24: String storage in the binary format

Binary data (such as UDP payloads) and arbitrary strings are stored with a leading length indicator and no trailing `NULL` terminator, as shown in listing **24**. It should be important to note that this convention only allows for 65,535 bytes to be stored per field. At this time, this was not an issue since no fields across the datasets exceeded this limit, and adhering to the YAGNI principle[**63**] it is not implemented, however if it does become one, this will be solved in a backward-compatible way, where a length of `USHRT_MAX` will instruct the parser to read additional fields of `uint16` until a value smaller than the maximum is met, and sum them to create the final length of the string to be read.

6.13.1 CPE Dictionary

The `cpe2hs.go` script converts NIST's official CPE dictionary from the published XML format to the binary format in use by the application.

Listing **25** presents the binary format – including the header and its values – which is used to store the data on the disk.

```
+ uint16      Package type [0x0100]
| uint16      Package version [0x0100]
| uint32      Number of entries
+- string     CPE name
 | uint8      Number of common tokens
 +- string    Token
 | uint32     Number of versions
 +- string    CPE version
  | string    Version token
  | uint8     Number of version-specific tokens
  +- string   Token
```

Listing 25: Binary format of the CPE dictionary

Previously discussed in section **6.8**, listing **19** presents a sample entry from the XML format being converted, while listing **20** presents the approximate internal representation of a sample entry from this file as it is being stored in the memory after being read.

6.13.2 CPE Aliases

Since NIST's CVE database may use multiple CPE names to refer to the same application, the `cpealt2hs.go` script converts a list of known CPE aliases to the binary format in use by the application.

As an example, the server software "nginx" may appear as both `cpe:/a:nginx:nginx` and `cpe:/a:igor_sysoev:nginx` amongst the CVE entries. However, there are more extreme cases, such as "X11", which has 12 CPE names all referring to the same software package:

- `cpe:/a:x:x.org`
- `cpe:/a:x.org:libx11`
- `cpe:/a:x.org:x.org`
- `cpe:/a:x.org:x11`
- `cpe:/a:x.org:x11r6`
- `cpe:/a:x.org:x11r7`
- `cpe:/a:x.org:xorg-server`
- `cpe:/a:x.org:xserver`
- `cpe:/a:x:x11`
- `cpe:/a:xfree86_project:x11r6`
- `cpe:/a:xfree86_project:xfree86`
- `cpe:/a:xorg:x11`

Listing **26** presents the binary format – including the header and its values – which is used to store the data on the disk.

```
+ uint16      Package type [0x0200]
| uint16      Package version [0x0100]
| uint32      Number of entries
+- uint16     Number of aliases in entry
 +- string    CPE name
```

Listing 26: Binary format of the CPE alias list

The internal representation of the CPE alias list is shown in listing **27**. The author of the data is Debian's Security Tracker team, as mentioned in section **6.11**, and can be freely used under the same license as the scripts themselves.

```
[
    [
        "cpe:/a:asterisk:asterisk",
        "cpe:/a:asterisk:open_source",
        "cpe:/a:asterisk:p_b_x",
        "cpe:/a:digium:asterisk"
    ],
    [
        "cpe:/a:clamav:clamav",
        "cpe:/a:cclamav:clamav",
        "cpe:/a:clam_anti-virus:clamav",
        "cpe:/a:clamavs:clamav"
    ],
    [
        "cpe:/a:nginx:nginx",
        "cpe:/a:igor_sysoev:nginx"
    ],
    // [further entries omitted]
]
```

Listing 27: Approximate internal representation of the CPE alias list

6.13.3 UDP Payloads

As discussed in section **6.1.4**, there needs to exist a list of known UDP payloads associated with their standardized port numbers in order to make sure the service behind the port answers the scanning effort. Two scripts are provided which read such data and convert to the appropriate binary format in use by the application, namely `nudp2hs.go` and `zudp2hs.go`.

Listing **28** presents the binary format – including the header and its values – which is used to store the data on the disk.

```
+ uint16      Package type [0x0A00]
| uint16      Package version [0x0100]
| uint32      Number of entries
+- string     Payload data
 | uint16     Number of ports in entry
 +- uint16    Port number
```

Listing 28: Binary format of the UDP payload database

The internal representation of the UDP payloads database is shown in listing **29**, as it is being stored in memory after being read from disk. The example entries were picked to showcase the varying type of payloads that are being stored and used.

```
[
    {
        "Ports": [ 53 ],
        "Data": "4\ufffd\u0001\u0000\u0000\u0001\u0000\u0000\u0000\u0000\u0000\u0000
 ↪  \u0007VERSION\u0004BIND\u0000\u0000\u0010\u0000\u0003"
    },
    {
        "Ports": [ 1900 ],
        "Data": "M-SEARCH * HTTP/1.1\r\nHost:239.255.255.250:1900\r\nST:upnp:rootdev
 ↪  ice\r\nMan:\"ssdp:discover\"\r\nMX:3\r\n\r\n\r\n"
    },
    {
        "Ports": [
            27910,
            27911,
            27912,
            27913,
            27914
        ],
        "Data": "\ufffd\ufffd\ufffd\ufffdstatus"
    },
    // [further entries omitted]
]
```

Listing 29: Approximate internal representation of the UDP payload database

The `nudp2hs.go` script takes a *plain-text file* as an input, where each entry starts on a new line with a comma-separated list of port numbers or ranges, followed by a quoted string containing the payload, which may span across multiple lines for easy readability and maintenance.

The `zudp2hs.go` script takes a *path* as an input, where it will look for `*.pkt` files. The content of the files are used as the payload, and the file name is expected to contain the list of port numbers, separated by an underscore in case there are multiple.

The `get.sh` script may be used to download the UDP database of *nmap* and *zmap*, as

discussed in section **6.3**. The license of both applications, and their respective databases, are compatible with the license of the application developed within the scope of the thesis, as such, they can be freely used.

6.13.4 Regular Expression Database

The database discussed in this section provides the entries for the pattern-matching component discussed in section **6.7**. Similarly to the previous section, two scripts are provided which read such data and convert to the appropriate binary format in use by the application, namely `ncpe2hs.go` and `bsvr2hs.go`.

Listing **28** presents the binary format – including the header and its values – which is used to store the data on the disk.

```
+ uint16      Package type [0x0F00]
| uint16      Package version [0x0100]
| uint32      Number of entries
+- string     Regular expression
 | string     CPE name
 | string     Product
 + string     Version
```

Listing 30: Binary format of the service regular expression database

The internal representation of the regular expression database is shown in listing **31**, as it is being stored in memory after being read from disk.

The first two showcased entries are simple matches to the HTTP responses generated by the server software "nginx". This regular expression is also shown as an example in the section regarding the pattern matcher, in listing **15**. The last entry matches a software-specific string, without the name of the software itself. This specific edge case is discussed in section **6.7.1**, where a different example is shown in listing **16**, for a different server software, however, the underlying concept is the same here as well.

The `ncpe2hs.go` script takes a *plain-text file* as an input, where each entry starts on a new line with a protocol name, followed by an escaped string containing the regular expression. The line may contain additional fields after the regular expression, such as the CPE name and the version of the software, which is either a static string or a variable referencing a capture group from the regular expression. This format is the same as *nmap*'s database format, as such, the "service probes" file may be converted for use within the application.

The `bsvr2hs.go` script is a simplified version, taking a *tab-separated file* as an input, where the first column is the regular expression, the second column denotes the optional number of the capturing group inside the regular expression responsible for capturing the version number, while the third column is the CPE field.

The `get.sh` script may be used to download the database of *nmap*, as discussed in

section **6.3**. The license of the application and its database, are compatible with the license of the application developed within the scope of the thesis, as such they can be freely used.

```
[
    // simple entry to match nginx
    {
        "Regex": "^HTTP/1\\.[01] \\d\\d\\d.*\\r\\nServer: nginx\\r\\n",
        "CPE": "a:nginx:nginx",
        "Product": "nginx",
        "Version": ""
    },
    // match nginx and extract version number from group 1
    {
        "Regex": "^HTTP/1\\.[01] \\d\\d\\d.*\\r\\nServer: nginx/([\\d.]+)\\r\\n",
        "CPE": "a:nginx:nginx:$1",
        "Product": "nginx",
        "Version": "$1"
    },
    // match nginx without its name, based on software-specific string
    {
        "Regex": "^HTTP/1.1 400 Bad Request\\r\\n.*The plain HTTP request was sent to
 ↪  HTTPS port",
        "CPE": "a:nginx:nginx",
        "Product": "nginx",
        "Version": ""
    },
    // [further entries omitted]
]
```

Listing 31: Approximate internal representation of the regular expression database

6.13.5 Vulnerability Database

Initially, the vulnerability database used a binary format similar to the ones discussed in the previous sections. However, loading all the required information into the memory resulted in a memory usage of 2 GB, and when running with a debugger attached, the data loader function took 30 seconds to execute, thereby making debugging more difficult due to the introduced penalty to application restarts.

To combat the memory usage issues, there were attempts at rewriting the reader component to lazily load the data. However, since the data had to be fully iterated sooner or later during the application's runtime, this attempt did not solve any issues, just delayed them.

Ultimately the decision was made to switch to a SQL database for this particular dataset. The other datasets were more optimal being in their own respective binary formats. However, due to the size and query style of this particular dataset, a mature

relational database was more fit for the job.

The SQL database chosen for this project was SQLite3[**64**], a small but mature serverless and embeddable relational database. The database contains two tables, as shown in figure **22**.

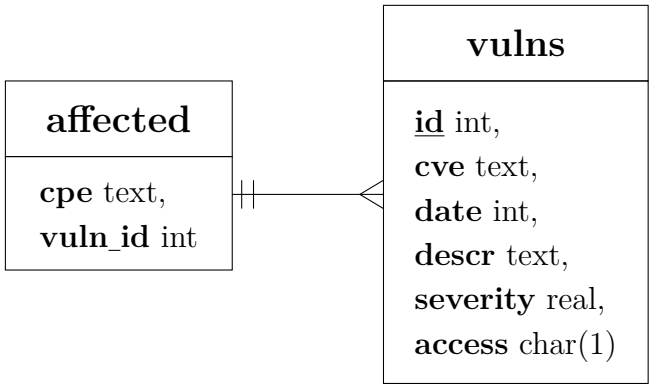


Figure 22. Structure of the Vulnerability Database

The "vulns" table contains the published vulnerabilities imported from the NVD and their relevant information. Its "access" field represents the access vector, which can be:

- `l` for **local**: physical access or local presence is required to exploit.
- `a` for **adjacent**: attacker has to reside on the same local network.
- `n` for **network**: vulnerability is remotely exploitable over the Internet.

Table **6** presents two sample entries from the "vulns" table.

id	cve	date	descr	severity	access
75158	2016-0747	1455566342	The resolver in ...	5.0	n
75157	2016-0746	1455566341	Use-after-free ...	7.5	n

Table 6. Sample entries of the "vulns" table

The "affected" table contains the associations between the published CPE names and which published CVE vulnerability affects them from the "vulns" table. Table **7** presents four example entries in the "affected" table, where the affected CPE names are listed for the example vulnerabilities shown in table **6**.

vuln_id	cpe
75158	cpe:/a:nginx:nginx:1.9.0
75158	cpe:/a:nginx:nginx:1.9.1
75157	cpe:/a:nginx:nginx:1.9.7
75157	cpe:/a:nginx:nginx:1.8.0

Table 7. Sample entries of the “affected” table

The “cpe” field of the “affected” table is indexed in order to provide optimal prefix search amongst the CPE names. The query plan of each SQL statement run by the application was carefully examined in order to optimize the table structure and index usage for the best possible performance.

6.14 Unit Testing

Unit testing is an integral part of every project in order to ensure new changes do not alter or break the functionality of the existing components in a continuous development life cycle.

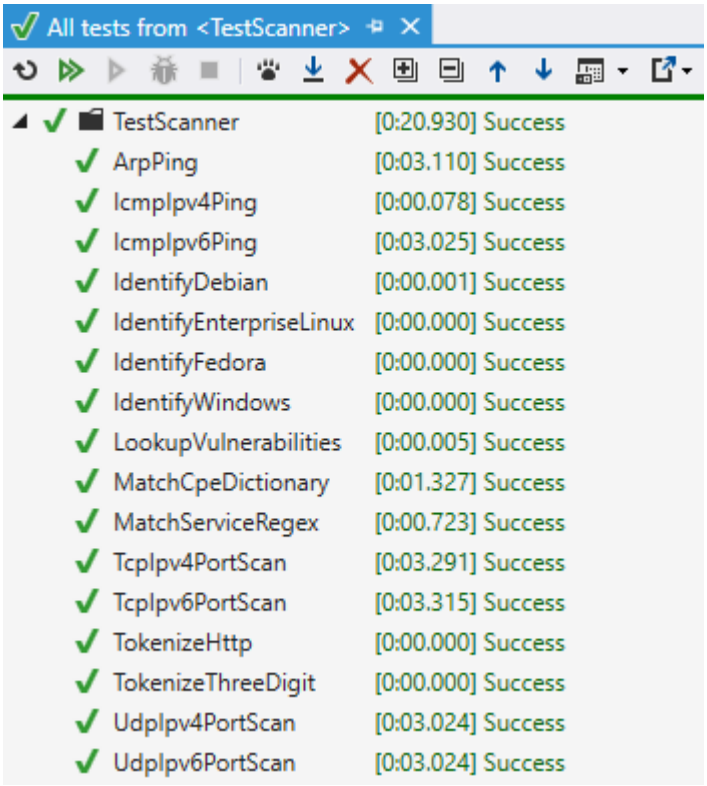


Figure 23. Some of the test cases in the Visual Studio Unit Test runner

Unit testing was also an integral part of the application developed within the scope of this thesis. The application supports multiple platforms – several Linux distributions, BSD, Mac OS X and Windows – multiple compilers – the GNU Compiler Collection, Clang, and Microsoft Visual C++ – all of which required some macro-based tweaks or in some cases, completely new implementations – as was the case with the ARP ping component discussed in section **6.1.2**.

The unit testing framework used by the application is the *Boost Test* framework, which comes bundled with the Boost library, thus saving an extra dependency, also decreasing the unnecessary complexity of the project when porting to new platforms. The framework provides a solid foundation for writing test cases and a CLI application for running them. It has also been integrated into the build system used by the project, *CMake*, providing a fluid testing experience for the users.

The test cases have been written in such a way as to first test normal functionality, covering all major code paths, and then to test all discovered edge cases, both those that are expected to succeed in the face of unusual inputs, and also those that are expected to fail, to ensure a graceful failure.

There are test cases written, such as `MatchCpeDictionary`, which directly test a specific *component* to ensure all edge cases described in section **6.8** are handled correctly, and will not break in the future as a result of a change.

However, there are also test cases which do not test a specific component, rather test an exposed *functionality*. This is called **integration testing**[**65**] and focus testing the interaction of the components as a whole, one such test case being `MatchAuto` in the application.

This test receives a set of service banners as an input and is expected to output a very strict set of CPE names with exact version numbers. In the case of this test, it does not matter which component does the identification. This is used to ensure the functionality as a whole of the application does not break in the face of tweaking a single component, and that fail-overs between redundancies are properly handled.

One of the main challenges of unit testing the application was the network-oriented nature of it. Under normal circumstances, functionality such as file system, database or network access would be *mocked*[**66**]. However, the problem, in this case, is that the implemented components all use the low-level networking API of the operating system.

While strictly speaking it would be possible to "overwrite" the used low-level function calls to the network stack (such as `connect()` and `send()`), this would only ensure that the tested components receive the proper response from the network stack every time, thus removing the element of environmental discrepancies.

```
ArpPinger scan;

Hosts hosts = {
    new Host("192.168.1.1"),    // local, alive
    new Host("192.168.1.254"),  // local, not alive
    new Host("188.166.36.214"), // remote, excelsior.rolisoft.net
};

scan.Scan(&hosts);

BOOST_TEST_CHECK( hosts[0]->alive, "192.168.1.1 should answer.");
BOOST_TEST_CHECK(!hosts[1]->alive, "192.168.1.254 should not answer.");
BOOST_TEST_CHECK(!hosts[2]->alive, "188.166.36.214 should not answer.");

// further checks omitted from listing, such as reason for failure
```

Listing 32: Example test case for the `ArpPinger` component

Since the same functions of the network stack are present and utilized on all three supported platforms, but with the presence of slight changes, such as them exhibiting different behavior even when invoked with the same argumentation. Furthermore, this behavior discrepancy is not only present going from one platform to another, but also between different versions of the same platform.

As a result, it was decided that the best course of action for tackling the testing of the network-dependent components would be not to mock the network stack, but rather, use the native network stack of the operating system directly.

Unfortunately, this introduces another element of uncertainty, since, in order to test the scanners accurately, a known result is needed for the comparison to occur, which means multiple consistent targets are required for the scanners to be pointed at.

This issue was solved by pointing the scanners to public services (such as Google's DNS resolver) and various services on servers owned by me, ultimately also to bogon (unroutable/unallocated) addresses for those that are expected to fail. Listing **32** presents an example test case written using this method.

An added benefit of using the operating system's network stack and actually sending and receiving packets over the Internet is that it is possible to test for unforeseen factors which might affect the scanner components. One such factor would be the MTU size ("maximum transmission unit") of the unit test runner's network, which might inhibit the proper functioning of the ICMP pinger or UDP scanner components.

While there is no code written to gracefully handle this specific issue, this, and other similar factors, can be tracked down if the test case fails on a new platform amongst perfect conditions.

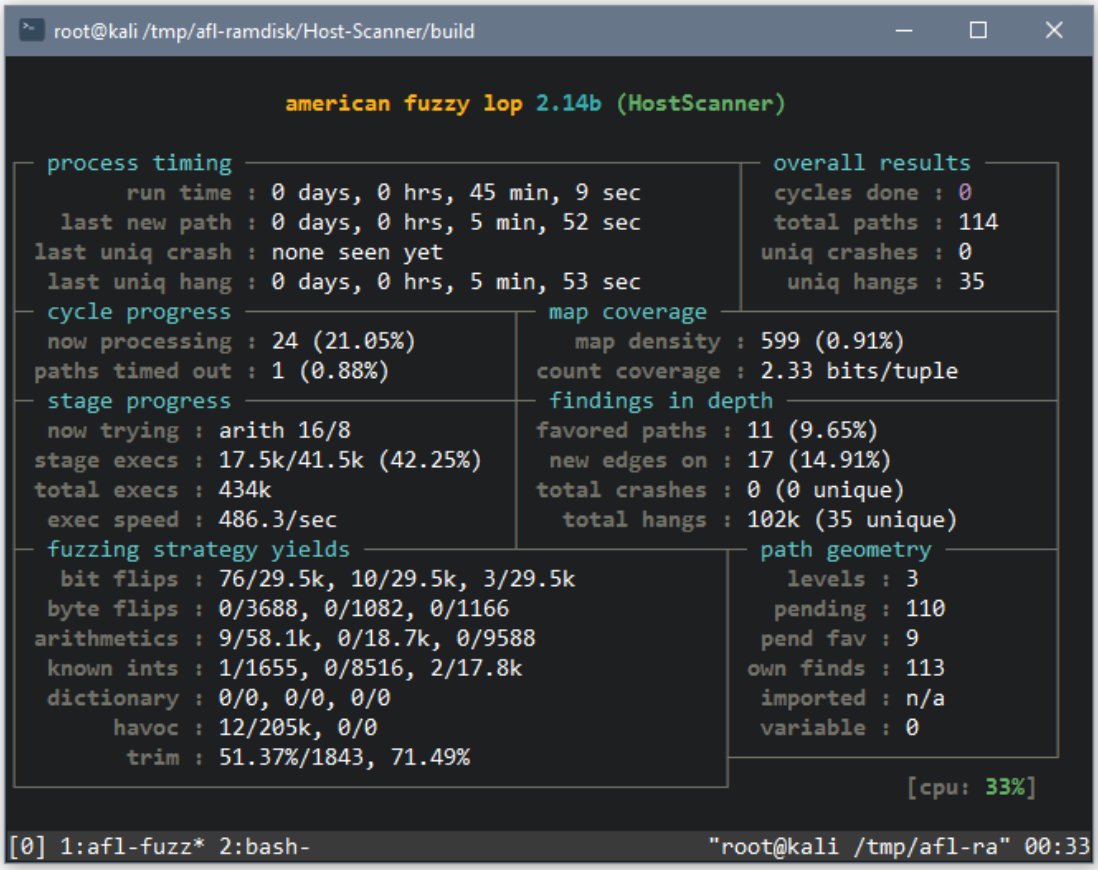


Figure 24. American Fuzzy Lop fuzzing the UDP payloads database loader

Besides the various functional tests present in the application, *non-functional* aspects are also tested, such as *performance* tests – ensuring the use of the fastest implementation between components, *failover* tests – ensuring proper failover where redundancy is available, and *stress* tests – ensuring proper functioning when scanning a large IP space.

Continuous integration was also used during the development of the application, with unit test execution on successful builds and *static code analysis* with various free (**Cppcheck**) and commercial, but free for open-source (**Coverity**) tools. The importance and advantages of checking the source code with static analyzers are discussed in this thesis as a means of white-box vulnerability assessment, in section **3.1**.

Components taking an input (such as the data loader implementations discussed in **6.13**) were tested using **afl**[**51**], a popular *fuzz testing* application, which is mentioned in section **3.2** as a means of black-box vulnerability assessment. Furthermore, **Valgrind** was used during the unit test session runs to ensure no memory leaks occur in the implemented components.

7 User Guide

This section presents how to acquire, build and use the application developed within the scope of this thesis.

7.1 Build Instructions

The source code for the application is available under the link presented in section **6**, and can be freely used, modified and redistributed under the terms of the GNU General Public License version 3[**14**].

To compile and run the application, the dependencies must first be installed, which can be done with the following command lines, depending on the operating system the application is being compiled on:

Debian/Ubuntu/Kali Install the following packages:

```
apt install build-essential cmake libcurl4-openssl-dev libsqlite3-dev
 ↪  libboost-all-dev libz-dev
```

RHEL/CentOS/Fedora Install the following packages:

```
yum install gcc-c++ make cmake libcurl-devel sqlite-devel boost-devel-static
 ↪  zlib-devel
```

FreeBSD Install the following packages:

```
pkg install cmake curl sqlite3 boost-libs
```

Mac OS X Install the following packages using *Homebrew*[10]:

```
xcode-select --install && brew install cmake curl sqlite boost
```

Windows Install the following packages from the project root using *Conan*[11,12]:

```
conan install --build=missing
```

After the dependencies have been installed, the repository can be checked out and compiled with the commands shown in listing **33**.

[10] `http://brew.sh/`

[11] `https://conan.io/`

[12] `https://conan.io/source/WinPcap/4.1.2/RoliSoft/stable` – the WinPcap WDK build script and package was contributed by me

```
git clone https://github.com/RoliSoft/Host-Scanner.git
cd Host-Scanner/build
cmake ..
make
```

Listing 33: Instructions to checkout and build the source

If the compilation was successful, it can be run with the `./HostScanner` command. Unit tests are also available, they may be run through `make test` or directly, by executing `./TestScanner` if the `TestScanner` target was included in the compilation.

The `distrib` folder contains Docker container scripts (`Dockerfile`) to bake fresh `deb` and `rpm` files. However, since this is not part of the regular build workflow, instructions will not be included here and users wishing to go down this route should instead read the `distrib/README` file for more instructions.

7.2 Usage Examples

This section presents various copy-pasteable use cases of the application. More information regarding the command line options can be found in section **7.3**.

Active Scan Scan a network for hosts responding on the top 100 TCP ports and known UDP ports using the internal active scanners, then identify all the services and check them for vulnerabilities:

```
./HostScanner -p t -u t 192.168.1.0/24
```

Passive Scan Scan an IP address or range, using the internal passive scanners, which amalgamate data from *Shodan*, *Censys* and *Mr Looquer*:

```
./HostScanner -x 178.62.192.0/18
```

XML Import Perform service identification and vulnerability analysis on an earlier XML output of *nmap*:

```
./HostScanner -s nmap -f report.xml
```

Package Identification Scan the specified targets, then determine the operating system running on the hosts, identify the services, perform vulnerability analysis, and resolve to a list of packages which are vulnerable on each individual host:

```
./HostScanner -r 192.168.1.66 192.168.1.71
```

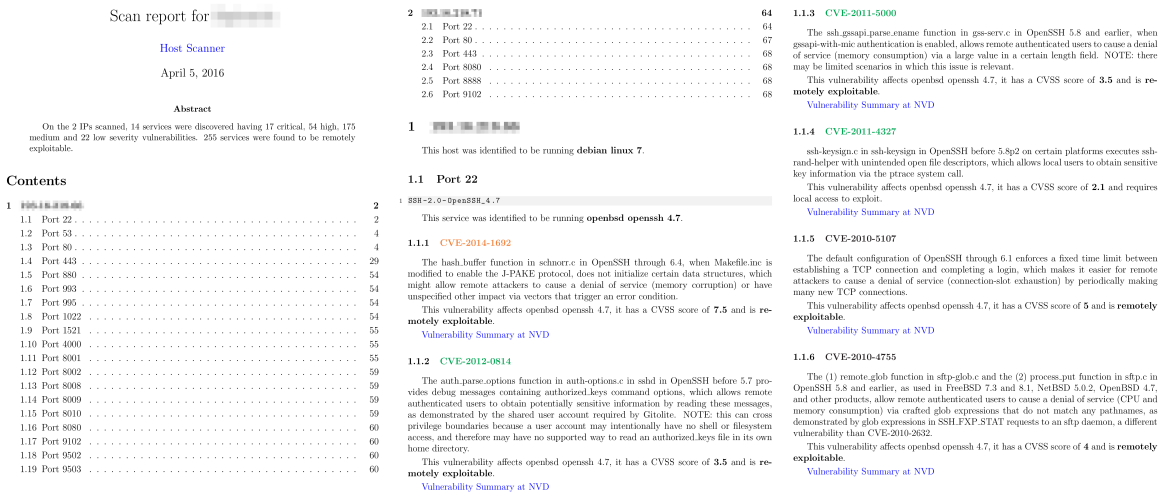
Using the command line above, the application will scan the TCP ports of the listed targets, identifies the exposed services, and then tries to perform operating system identification, as discussed in section **6.9**. If the services and the OS have been successfully identified, any discovered vulnerability will be checked against the OS's package manager, as discussed in section **6.11**. Vulnerabilities found to be valid with a known fix will end up listed in a command line generated on a per-host basis, which when run, applies the security updates on the vulnerable hosts. An example output fragment of such a session can be found in listing **34**.

```
[*] 192.168.1.66 is running cpe:/o:debian:debian_linux:8
[*] 192.168.1.71 is running cpe:/o:redhat:enterprise_linux:7
  ...
[*] 192.168.1.66 -> sudo apt-get install --only-upgrade apache2 php5 python2.7
[*] 192.168.1.71 -> sudo yum update httpd php python27-python
```

Listing 34: Generated command line to update the vulnerable packages on the host

LaTeX Report Generate a LaTeX report after the scan, which contains all the information that has been gathered during the scan, including open ports, service and operating system identification results, discovered vulnerabilities, and when applicable, the command line generated to be run on a per-host basis to update the vulnerable packages:

```
./HostScanner -o report.tex -r 192.168.1.66 192.168.1.71
```


Scan report for

Host Scanner

April 5, 2016

Abstract

On the 2 IPs scanned, 14 services were discovered having 17 critical, 54 high, 175 medium and 22 low severity vulnerabilities. 255 services were found to be remotely exploitable.

Contents

1 2
1.1 Port 22 2
1.2 Port 53 4
1.3 Port 80 4
1.4 Port 443 29
1.5 Port 880 54
1.6 Port 993 54
1.7 Port 995 54
1.8 Port 1022 54
1.9 Port 1521 55
1.10 Port 4000 55
1.11 Port 8001 55
1.12 Port 8002 59
1.13 Port 8008 59
1.14 Port 8009 59
1.15 Port 8010 59
1.16 Port 8080 60
1.17 Port 9102 60
1.18 Port 9502 60
1.19 Port 9503 60

2 64
2.1 Port 22 64
2.2 Port 80 67
2.3 Port 443 68
2.4 Port 8080 68
2.5 Port 8888 68
2.6 Port 9102 68

1

This host was identified to be running **debian linux 7**.

1.1 Port 22

1 SSH-2.0-OpenSSH_4.7

This service was identified to be running **openbsd openssh 4.7**.

1.1.1 CVE-2014-1692

The hash_buffer function in schnorr.c in OpenSSH through 6.4, when Makefile.inc is modified to enable the J-PAKE protocol, does not initialize certain data structures, which might allow remote attackers to cause a denial of service (memory corruption) or have unspecified other impact via vectors that trigger an error condition.

This vulnerability affects openbsd openssh 4.7, it has a CVSS score of **7.5** and is **remotely exploitable**.

Vulnerability Summary at NVD

1.1.2 CVE-2012-0814

The auth_parse_options function in auth-options.c in sshd in OpenSSH before 5.7 provides debug messages containing authorized_keys command options, which allows remote authenticated users to obtain potentially sensitive information by reading these messages, as demonstrated by the shared user account required by Gitolite. NOTE: this can cross privilege boundaries because a user account may intentionally have no shell or filesystem access, and therefore may have no supported way to read an authorized_keys file in its own home directory.

This vulnerability affects openbsd openssh 4.7, it has a CVSS score of **3.5** and is **remotely exploitable**.

Vulnerability Summary at NVD

1.1.3 CVE-2011-5000

The ssh_gssapi_parse_ename function in gss-serv.c in OpenSSH 5.8 and earlier, when gssapi-with-mic authentication is enabled, allows remote authenticated users to cause a denial of service (memory consumption) via a large value in a certain length field. NOTE: there may be limited scenarios in which this issue is relevant.

This vulnerability affects openbsd openssh 4.7, it has a CVSS score of **3.5** and is **remotely exploitable**.

Vulnerability Summary at NVD

1.1.4 CVE-2011-4327

ssh-keysign.c in ssh-keysign in OpenSSH before 5.8p2 on certain platforms executes ssh-rand-helper with unintended open file descriptors, which allows local users to obtain sensitive key information via the ptrace system call.

This vulnerability affects openbsd openssh 4.7, it has a CVSS score of **2.1** and requires local access to exploit.

Vulnerability Summary at NVD

1.1.5 CVE-2010-5107

The default configuration of OpenSSH through 6.1 enforces a fixed time limit between establishing a TCP connection and completing a login, which makes it easier for remote attackers to cause a denial of service (connection-slot exhaustion) by periodically making many new TCP connections.

This vulnerability affects openbsd openssh 4.7, it has a CVSS score of **5** and is **remotely exploitable**.

Vulnerability Summary at NVD

1.1.6 CVE-2010-4755

The (1) remote_glob function in sftp-glob.c and the (2) process_put function in sftp.c in OpenSSH 5.8 and earlier, as used in FreeBSD 7.3 and 8.1, NetBSD 5.0.2, OpenBSD 4.7, and other products, allow remote authenticated users to cause a denial of service (CPU and memory consumption) via crafted glob expressions that do not match any pathnames, as demonstrated by glob expressions in SSH_FXP_STAT requests to an sftp daemon, a different vulnerability than CVE-2010-2632.

This vulnerability affects openbsd openssh 4.7, it has a CVSS score of **4** and is **remotely exploitable**.

Vulnerability Summary at NVD

Figure 25. LaTeX report generated after a scan session

7.3 Command Line Options

```
usage:        HostScanner [args] targets
arguments:

-t [ --target ] arg

  List of targets to scan:
    Each can be a hostname, IP address, IP range or CIDR.
    E.g. `192.168.1.1/24` is equivalent to `192.168.1.0-192.168.1.255`.

-p [ --port ] arg

  TCP ports to scan:
    Can be a single port (80), a list (22,80) or a range (1-1024).
    Range can be unbounded from either sides, simultaneously.
    E.g. `1024-` will scan ports 1024-65535. `-` will scan all ports.
    Specifying `top` or `t` will scan the top 100 most popular ports.

-u [ --udp-port ] arg

  UDP ports to scan:
    Supports the same values as --port, with the difference that
    specifying `top` will scan all of the ports with known payloads.

-s [ --scanner ] arg

  Scanner instance to use:
    internal - Uses the built-in scanner. (active)
    nmap     - Uses 3rd-party application Nmap. (active)
    shodan   - Uses data from Shodan. (passive; requires API key)
    censys   - Uses data from Censys. (passive; requires API key)
    looquer  - Uses data from Mr Looquer. (passive; requires API key)
    shosys   - Uses data from Shodan, Censys and Mr Looquer. (passive)

--shodan-key arg

  Specifies an API key for Shodan.

--shodan-uri arg

  Overrides the API endpoint used for Shodan. You may specify an URI
    starting with file:// pointing to a directory containing previously
    downloaded JSON responses.
      Default: https://api.shodan.io/shodan

--censys-key arg

  Specifies an API key for Censys in the `uid:secret` format.
```

```
--censys-uri arg

  Overrides the API endpoint used for Censys. You may specify an URI
    starting with file:// pointing to a directory containing previously
    downloaded JSON responses.
      Default: https://censys.io/api/v1

--looquer-key arg

  Specifies an API key for Mr Looquer.

--looquer-uri arg

  Overrides the API endpoint used for Mr Looquer. You may specify an URI
    starting with file:// pointing to a directory containing previously
    downloaded JSON responses.
      Default: https://mrlooquer.com/api/v1

-f [ --input-file ] arg

  Process an input file with the selected scanner.
    E.g. the nmap scanner can parse XML reports.

-d [ --delay ] arg

  Delay between packets sent to the same host. Default is 3 for 100ms.
    Possible values are 0..6, which have the same effect as nmap's -T:
      0 - 5m, 1 - 15s, 2 - 400ms, 3 - 100ms, 4 - 10ms, 5 - 5ms,
      6 - no delay

-r [ --resolve ]

  Resolves vulnerable CPE names to their actual package names depending
    on the automatically detected operating system of the host.

-w [ --validate ]

  Validate all vulnerabilities with the package manager of the distribution.

-e [ --estimate ]

  Estimate date range of the last system upgrade based on the installed
    package versions and security patches.

-o [ --output-latex ] arg

  Exports the scan results into a LaTeX file, with all the available
    information gathered during the scan.

-x [ --passive ]
```

```
    Globally disables active reconnaissance. Functionality using active
      scanning will break, but ensures no accidental active scans will be
      initiated, which might get construed as hostile.

  -l [ --logging ] arg

    Logging level to use:
      i, int - All messages.
      d, dbg - All debug messages and up.
      v, vrb - Enable verbosity, but don't overdo it.
      m, msg - Print only regular messages. (default)
      e, err - Print only error messages to stderr.

  -q [ --no-logo ]

    Suppresses the ASCII logo.

  -v [ --version ]

    Display version information.

  -h [ --help ]

    Displays this message.
```

Listing 35: Command line interface options

7.4 Configuration

The application developed within the scope of this thesis, supports reading configuration files. The available configuration options are the same as the options presented in section **7.3**. As such, this functionality allows for settings to persist between sessions.

The software has different code for handling the Linux, BSD, Mac platforms and Windows. The paths shown in listing **36** will be probed in the listed order, and the first found file will be read. This allows the user the have either a global configuration or a local one tied to the user account.

```
1 Linux, BSD, Mac:
2     %AppPath%/HostScanner.ini
3     ~/.HostScanner.conf
4     /etc/HostScanner/HostScanner.conf
5
6 Windows:
7     %AppPath%\HostScanner.ini
8     %AppData%\RoliSoft\Host Scanner\HostScanner.ini
```

Listing 36: Configuration file locations on each supported platform

A particularly useful use-case for this feature is the ability to store API keys of the supported online services (see section **6.2**) and lose the need to resupply them constantly as command line arguments. Listing **37** presents the contents of a configuration file, which stores Shodan, Censys, and Mr Looquer keys. All the user needs to do is to store one or more of these lines (with the proper key filled) into one of the paths listed in **36**.

```
1 shodan-key=abcdefghijklmnopqrstuvwxyz012345
2 censys-key=abcdefgh-ijkl-mnop-qrst-uvwxyz012345:abcdefghijklmnopqrstuvwxyz012345
3 looquer-key=abcdefghijklmnopqrstuvwxyz012345
```

Listing 37: Persist keys of online services in the configuration file

Another use-case is storing the `x` option in a global configuration file, as shown in listing **38**. Doing so results in the globally disallowed usage of any of the active scanners as a fail-safe. The user cannot unset or disable this options after it has been loaded from a configuration file.

```
1 passive
```

Listing 38: Disallow any active scanner usage globally from the configuration file

With the passive mode globally enabled, the user is only allowed to use passive scanners, such as Shodan, Censys, Mr Looquer, and to load and analyze earlier Nmap reports, but not to launch new Nmap scans.

8 Results

The application was tested against three higher education institutions with both active and passive scanning methods. After these tests have concluded, another one was conducted with only passive scanning against five randomly selected leading[13] banking institutions, which I would have expected to represent the "gold standard" in the test.

	Active Scan			**Passive Scan**							
Institution	u_1	u_2	u_3	u_1	u_2	u_3	b_1	b_2	b_3	b_4	b_5
Services	165	178	455	201	269	623	41	19	69	31	11
Identifiable	143	145	402	112	148	352	24	8	58	9	9
Identified	140	137	394	109	139	348	24	8	58	9	9

Table 8. Number of identifiable and identified CPE names by the application

[13]Based on the amount of assets held as per performance reports for the year 2015[**67**].

Table **8** presents the number of services discovered at the tested institutions for each scanning attempt. The "Services" row represents an *open port*. The "Identifiable" row represents the services which have replied with a service banner, and at the same time that service banner contains enough information for the service to be properly identified under normal conditions, such as its software name and version number. This number was determined through manual verification of the service banners for this test. The "Identified" row represents the number of services correctly identified by the software.

Out of the **1,410** identifiable service banners, **1,374** services were correctly identified by the application, resulting in a success rate of **97.45%**.

	Active Scan			**Passive Scan**							
Institution	u_1	u_2	u_3	u_1	u_2	u_3	b_1	b_2	b_3	b_4	b_5
Services	165	178	455	201	269	623	41	19	69	31	11
Critical	183	121	160	161	131	230	8	0	30	6	6
High	675	414	645	583	446	826	7	0	67	21	5
Medium	2545	1441	2775	2308	1589	3247	19	0	299	133	9
Low	231	151	275	195	170	335	7	0	26	13	4
AV:N	3296	1970	3493	2961	2173	4248	40	0	393	153	22

Table 9. Discovered vulnerabilities of the services identified by the application

Table **9** presents the number of vulnerabilities found on the services which have been discovered and successfully identified. The "Critical," "High," "Medium" and "Low" rows represent the number of vulnerabilities discovered for each severity level, while the "AV:N" row presents the number of vulnerabilities which can be remotely exploited over a network connection. (This information is available from the *CVSS* scoring system presented in section **2.5**, where the "Access Vector" field has the "Network" level associated to it.)

The number of vulnerabilities discovered in the IP ranges of the banking institutions is unfortunately not surprising, as with a related project[14] of mine, I measured the response time of various banks to patching newly discovered critical vulnerabilities (such as Heartbleed, Shellshock, POODLE) and I've come to the conclusion, that some banks require up to 3 months in order to apply a security update for a critical vulnerability.

[14] `https://github.com/RoliSoft/Bank-Security-Audit`

Software	CVE#
Host Scanner[15]	166
OpenVAS	107
Nessus	68
Nexpose	311

Table 10. Comparison of identified vulnerabilities by competing software

As a second test, the application was compared to the commercial tools discussed in section **4.1**. Since these applications make a lot of noise (as in, network traffic) plus might take too much time to complete besides potentially overloading the services, not to mention potential licensing issues (most of them only allow home use for the free/community editions, defined as scanning one private /24 IP range), rescanning the institutions in table **9** would not be feasible.

The scan targets were instead comprised of virtual machines on a virtual network, compiled of vulnerable machines which were made for the express purpose of CTF ("Capture the Flag") style wargames, such as *Metasploitable* and *VulnOS*. These VM images were gathered from VulnHub[16], as suggested in article [**68**] and paper [**69**], a website whose aim is to provide downloadable VM images which simulate a vulnerable real-world infrastructure in order to allow interested parties to practice and gain hands-on penetration testing experience without breaking the law.

The weak results of "OpenVAS" and "Nessus" can be explained with the fact that their extension-based architecture failed to identify various obscure services on these hosts, as they did not have an extension written for them, and failing to identify some crucial services resulted in a decreased CVE count.

On the other side of the spectrum, "Nexpose" exploited one of the discovered vulnerabilities, and using the acquired shell access, enumerated the list of installed packages using the operating system's package manager, gaining a clear advantage, as the other three testing applications only analyzed the vulnerability status of public-facing network-accessible servers, without having access to the full list of installed software on the host.

It should be noted, that exploitation of the discovered vulnerabilities is out of the scope of this application, as it may not be desirable in live production environments, and straight-up illegal for security researchers without prior agreement. In the application, vulnerability validation is instead performed without penetration, by gathering environmental information, identifying the operating system, and then using the operating system distribution's package manager to check whether the discovered package is affected by the vulnerability, as described in section **6.11**.

[15]Name of the application developed within the scope of this thesis.
[16]`https://www.vulnhub.com/`

9 Bibliography

[1] John Matherly. *Complete Guide to Shodan*. Lean Publishing. 2016. Retrieved from https://leanpub.com/shodan. → 6, 11, 20, 28, 59, 70

[2] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, J. Alex Halderman. *A Search Engine Backed by Internet-Wide Scanning*. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*. oct 2015. Retrieved from https://censys.io/static/censys.pdf. → 6, 11, 20, 28, 59, 71

[3] Rafa Sánchez, Fran Gomez. *Mr Looquer – IPv6 Intelligence*. Retrieved from http://seclists.org/fulldisclosure/2016/Mar/23. → 6, 11, 20, 28, 59, 72

[4] Nmap Project. *SecTools.Org: Top 125 Network Security Tools*. Retrieved from http://sectools.org/tag/vuln-scanners/. → 7, 23, 56

[5] Tenable Network Security. *Nessus Vulnerability Scanner*. Retrieved from https://www.tenable.com/products/nessus-vulnerability-scanner. → 7, 23, 56

[6] Greenbone Networks. *Open Vulnerability Assessment System (OpenVAS)*. Retrieved from http://www.openvas.org/. → 8, 23, 57

[7] Rapid7. *Nexpose Vulnerability and Risk Management*. Retrieved from https://www.rapid7.com/products/nexpose/. → 8, 24, 57

[8] Hannes Holm, Teodor Sommestad, Jonas Almroth, Mats Persson. *A quantitative evaluation of vulnerability scanning*. Information Management & Computer Security, vol. 19, no. 4, (2011), pp. 231–247. Retrieved from http://www.diva-portal.org/smash/get/diva2:545791/FULLTEXT01.pdf. → 8, 24, 57

[9] Jason Bau, Elie Bursztein, Divij Gupta, John Mitchell. *State of the art: Automated black-box web application vulnerability testing*. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010. pp. 332–345. Retrieved from https://crypto.stanford.edu/~jcm/papers/pci_oakland10.pdf. → 8, 24, 57

[10] Adam Doupé, Marco Cova, Giovanni Vigna. *Why Johnny can't pentest: An analysis of black-box web vulnerability scanners*. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 111–131. Springer. 2010. Retrieved from http://www.issp.ua/issp_system_images/resources_files/acunetix/Acunetix_Analytics_Analysis_of_Black-box_Web_Vulnerability_Scanners_rus.pdf. → 8, 24, 57

[11] Stefan Kals, Engin Kirda, Christopher Kruegel, Nenad Jovanovic. *Secubat: a web vulnerability scanner*. In *Proceedings of the 15th international conference on World Wide Web*. ACM. 2006. pp. 247–256. Retrieved from http://www.iseclab.net/papers/secubat.pdf. → 8, 24, 57

[12] Fanglu Guo, Yang Yu, Tzi cker Chiueh. *Automated and safe vulnerability assessment.* In *Computer Security Applications Conference, 21st Annual.* IEEE. 2005. pp. 10–pp. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1565243. → 8, 24, 58

[13] Béla Genge, Călin Enăchescu. *ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services.* Security and Communication Networks. Retrieved from http://www.ibs.ro/~bela/Papers/SCN2015.pdf. → 8, 13, 24, 29, 58, 82, 85

[14] Free Software Foundation. *GNU General Public License, version 3.* Retrieved from https://www.gnu.org/licenses/gpl-3.0.html. → 8, 24, 60, 104

[15] Massachusetts Institute of Technology. *The MIT License.* Retrieved from https://opensource.org/licenses/MIT. → 8, 24, 60

[16] Jon Erickson. *Hacking: The Art of Exploitation, 2nd Edition.* No Starch Press Series. No Starch Press. 2008. ISBN 9781593271442. Retrieved from https://books.google.com/books?id=0FW3DMNhl1EC. → 9, 25, 66

[17] Kris Katterjohn. *Port Scanning Techniques.* Retrieved from https://dl.packetstormsecurity.net/papers/attack/port-scanning-techniques.txt. → 9, 26, 66

[18] Andrew Buttner, Neal Ziring. *CPE 2.2 Specification.* Retrieved from https://cpe.mitre.org/files/cpe-specification_2.2.pdf. → 12, 29, 81

[19] Robert W. Shirey. *RFC 2828: Internet Security Glossary.* The Internet Society, p. 13. Retrieved from https://tools.ietf.org/html/rfc2828. → 21, 47

[20] Patrick D. Gallagher, Rebecca M. Blank. *NIST Special Publication 800-30: Guide for Conducting Risk Assessments.* NIST Publications. Retrieved from http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pdf. → 21, 47

[21] National Institute of Standards and Technology. *National Vulnerability Database.* Retrieved from https://nvd.nist.gov/. → 22, 50

[22] Dave Evans. *The Internet of Things.* How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG). Retrieved from https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. → 42

[23] Geoff Huston. *IPv4 Address Report, daily generated.* Retrieved from http://www.potaroo.net/tools/ipv4/index.html. → 42

[24] Gary McGraw. *Data supports need for security awareness training despite naysayers.* Retrieved from http://searchsecurity.techtarget.com/news/2240162630/Data-supports-need-for-awareness-training-despite-naysayers. → 42

[25] Aberdeen Group. *Business Context: The Biggest No-Brainer in Security.* Retrieved from http://www.enterpriseittools.com/sites/default/files/Aberdeen_SecureAppsGetStarted.pdf. → 42

[26] Charlie Miller, Chris Valasek. *Remote Exploitation of an Unaltered Passenger Vehicle.* Retrieved from http://illmatics.com/Remote%20Car%20Hacking.pdf. → 43

[27] Herb Stutter. *C++/CLI Rationale.* Retrieved from http://www.gotw.ca/publications/C++CLIRationale.pdf. → 43

[28] Mark Stanislav, Zach Lanier. *The Internet of Fails: Where IoT Has Gone Wrong and How We're Making It Right.* Retrieved from https://www.defcon.org/html/defcon-22/dc-22-speakers.html#Stanislav. → 44

[29] Jakob Nielsen. *User Satisfaction vs. Performance Metrics.* Retrieved from http://www.nngroup.com/articles/satisfaction-vs-performance-metrics/. → 44

[30] Adele E. Howe, Indrajit Ray, Mike Roberts, Malgorzata Urbanska, Zinta Byrne. *The psychology of security for the home computer user.* In *Security and Privacy (SP), 2012 IEEE Symposium on.* IEEE. 2012. pp. 209–223. Retrieved from http://www.ieee-security.org/TC/SP2012/papers/4681a209.pdf. → 44

[31] Steven Furnell, Peter Bryant, Andy Phippen. *Assessing the security perceptions of personal Internet users.* Computers & Security, vol. 26, no. 5, (2007), pp. 410–417. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167404807000363. → 44

[32] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, Greg Norcie. *A comparative usability study of two-factor authentication.* arXiv preprint arXiv:1309.5344. Retrieved from http://arxiv.org/pdf/1309.5344.pdf. → 44

[33] Gregg Kreizman, Bruce Robertson. *Incorporating Security into the Enterprise Architecture Process.* Gartner, Inc. Retrieved from https://www.gartner.com/doc/488575. → 45

[34] Jose Pagliery. *Heartbleed bug affects gadgets everywhere.* Retrieved from http://money.cnn.com/2014/04/11/technology/security/heartbleed-gear/. → 45

[35] Payment Card Industry Security Standards Council. *Payment Card Industry Data Security Standard, Version 3.1.* Retrieved from https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf. → 46

[36] Wells Fargo. *Data Security / PCI Mandatory Compliance Programs.* Retrieved from https://www.wellsfargo.com/biz/merchant/service/manage/risk/security. → 46

[37] Payment Card Industry Security Standards Council. *Information Supplement: Requirement 6.6 Code Reviews and Application Firewalls Clarified*. Retrieved from https://www.pcisecuritystandards.org/pdfs/infosupp_6_6_applicationfirewalls_codereviews.pdf. → 46

[38] Payment Card Industry Security Standards Council, Penetration Test Guidance Special Interest Group. *Information Supplement: Penetration Testing Guidance*. Retrieved from https://www.pcisecuritystandards.org/documents/Penetration_Testing_Guidance_March_2015.pdf. → 46

[39] State of Washington. *Engrossed Second Substitute House Bill 1149*. Public Law. Retrieved from http://apps.leg.wa.gov/documents/billdocs/2009-10/Pdf/Bills/Session%20Laws/House/1149-S2.SL.pdf. → 47

[40] United States Congress. *Health Insurance Portability and Accountability Act*. Public Law, vol. 104, (1996), p. 191. → 47

[41] Nicole Perlroth, David E. Sanger. *Nations Buying as Hackers Sell Flaws in Computer Code*. New York Times, vol. 13. Retrieved from http://www.nytimes.com/2013/07/14/world/europe/nations-buying-as-hackers-sell-computer-flaws.html. → 48

[42] Google Application Security. *Google Vulnerability Reward Program Rules*. Retrieved from https://www.google.com/about/appsecurity/reward-program/. → 49

[43] Carnegie Mellon University, CERT Division. *About Us*. Retrieved from https://cert.org/about/. → 49

[44] Romanian Computer Emergency Response Team. *RFC 2350 description for CERT-RO*. Retrieved from https://www.cert-ro.eu/files/doc/RFC2350-CERT-RO.pdf. → 49

[45] Government Incident Response Team of Hungary. *RFC 2350 description for GovCERT-Hungary*. Retrieved from http://www.cert-hungary.hu/en/node/17. → 49

[46] Computer Emergency Response Team European Union Task Force. *RFC 2350 description for CERT-EU*. Retrieved from https://cert.europa.eu/static/RFC2350/RFC2350.pdf. → 49

[47] Matteo Meucci, Andrew Muller. *Open Web Application Security Project Testing Guide 4.0*. Retrieved from https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf. → 51

[48] Open Web Application Security Project. *The Ten Most Critical Web Application Security Risks*. Retrieved from https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. → 51

[49] ControlScan. *PCI Compliance Guide.* Retrieved from https://www.pcicomplianceguide.org/pci-faqs-2/. → 52

[50] Peter Mell, Karen Scarfone, Sasha Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0.* Retrieved from https://www.first.org/cvss/v2/guide. → 52

[51] Michał Zalewski. *American Fuzzy Lop.* Retrieved from http://lcamtuf.coredump.cx/afl/. → 54, 103

[52] Susmit Panjwani, Stephanie Tan, Keith M. Jarrin, Michel Cukier. *An experimental evaluation to determine if port scans are precursors to an attack.* In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on.* IEEE. 2005. pp. 602–611. Retrieved from https://sm.asisonline.org/ASIS%20SM%20Documents/port_scans0306_0.pdf. → 57

[53] Wenjian Xing, Yunlan Zhao and Tonglei Li. *Research on the defense against ARP Spoofing Attacks based on Winpcap.* In *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*, vol. 1. IEEE. 2010. pp. 762–765. Retrieved from https://www.researchgate.net/publication/224136193_Research_on_the_Defense_Against_ARP_Spoofing_Attacks_Based_on_Winpcap. → 65

[54] Internet Assigned Numbers Authority. *Service Name and Transport Protocol Port Number Registry.* Retrieved from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt. → 67

[55] Cisco. *Snort Rule Documentation: Sid 1-2049.* Retrieved from https://www.snort.org/rule_docs/1-2049. → 68, 69

[56] Zakir Durumeric, Eric Wustrow, J. Alex Halderman. *ZMap: Fast Internet-Wide Scanning and its Security Applications.* In *Proceedings of the 22nd USENIX Security Symposium.* aug 2013. Retrieved from https://zmap.io/paper.pdf. → 71, 72

[57] Verisign. *TLD Zone File Access Program.* Retrieved from https://www.verisign.com/en_US/channel-resources/domain-registry-products/zone-file/index.xhtml. → 72

[58] National Vulnerability Database. *Vulnerabilities in Nginx 1.8.0.* Retrieved from https://web.nvd.nist.gov/view/vuln/search-results?adv_search=true&cpe=cpe%3a%2fa%3anginx%3anginx%3a1.8.0. → 74

[59] Sergey Shekyan. *How to Protect Against Slow HTTP Attacks.* Retrieved from https://blog.qualys.com/securitylabs/2011/11/02/how-to-protect-against-slow-http-attacks. → 74

[60] A.A.A. Donovan and B.W. Kernighan. *The Go Programming Language*. Addison-Wesley Professional Computing Series. Pearson Education. 2015. ISBN 9780134190563. Retrieved from https://books.google.com/books?id=SJHvCgAAQBAJ. → 75

[61] Network Working Group. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. The Internet Society. Retrieved from https://tools.ietf.org/html/rfc2616. → 84

[62] Sean Gallagher. *Kali NetHunter turns Android device into hacker Swiss Army knife*. Retrieved from http://arstechnica.com/information-technology/2014/09/kali-nethunter-turns-android-device-into-hacker-swiss-army-knife/. → 92

[63] James Shore, Shane Warden. *The Art of Agile Development*. Theory in practice. O'Reilly Media, Incorporated. 2008. ISBN 9780596527679. Retrieved from https://books.google.com/books?id=2q6bAgAAQBAJ. → 93

[64] Richard Hipp. *SQLite*. 2007. Retrieved from https://www.sqlite.org/. → 99

[65] Martin Reddy. *API Design for C++*. Elsevier Science. 2011. ISBN 9780123850041. Retrieved from https://books.google.com/books?id=IY29LylT85wC. → 101

[66] Paul Hamill. *Unit Test Frameworks: Tools for High-Quality Software Development*. O'Reilly Media. 2004. ISBN 9780596552817. Retrieved from https://books.google.com/books?id=2ksvdhhnWQsC. → 101

[67] Deloitte. *Central Europe Top 500*. 2015. Retrieved from http://www2.deloitte.com/content/dam/Deloitte/global/Documents/About-Deloitte/central-europe/CE_Top_500_2015.PDF. → 110

[68] Chris Carthern, William Wilson, Richard Bedwell, Noel Rivera. *Introduction to Network Penetration Testing*. In *Cisco Networks*, pp. 759–772. Springer. 2015. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4842-0859-5_22. → 112

[69] Cliffe Schreuders, Lewis Ardern. *Generating randomised virtualised scenarios for ethical hacking and computer security education: SecGen implementation and deployment*. Retrieved from http://eprints.leedsbeckett.ac.uk/1479/1/Vibrant%20Workshop%20Paper%20-%20SecGen.pdf. → 112

Universitatea Sapientia din Cluj-Napoca
Facultatea de Științe Tehnice și Umaniste, Târgu-Mureș
Specializarea Calculatoare

Declarație

Subsemnatul, *Bogosi Roland*, student la specializarea Calculatoare, Facultatea de Științe Tehnice și Umaniste din Târgu-Mureș, de la Universitatea Sapientia din Cluj-Napoca, certific că am luat la cunoștință de cele prezentate mai jos și îmi asum, în acest context, originalitatea lucrării mele de diplomă cu titlul *Sistem pentru Testarea Penetrabilității și Descoperirea Vulnerabilităților*, coordonator *dr. Vajda Tamás*, prezentată în sesiunea iunie 2016.

La elaborarea lucrării de diplomă, se consideră plagiat una dintre următoarele acțiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimelele și referința precisă;
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări dacă nu se indică sursa bibliografică;
- prezentarea unor date experimentale obținute sau a unor aplicații realizate de alți autori fără menționarea corectă a acestor surse;
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Data: 27.06.2016 Semnătura:

Notă: Se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării;
- indicarea în text a reformulării unei idei, opinii sau teorii și corespunzător în lista de referințe a sursei originale de la care s-a făcut preluarea;
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele etc.;
- precizarea referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim acceptată.

Universitatea Sapientia din Cluj-Napoca
Facultatea de Științe Tehnice și Umaniste, Târgu-Mureș
Specializarea Calculatoare

Vizat decan
Ș.l. Dr. ing. Kelemen András

Vizat șef catedră
Dr. ing. Domokos József