SECURITATEA PRODUSELOR PROGRAM (SPP)

T5. Auditing și Testarea Securității Software. Metode de evaluare a securității software-ului. Testarea penetrantă (penetration testing) și evaluarea codului

Această prelegere abordează metodele și tehnicile utilizate pentru evaluarea securității unui produs software. Vom discuta despre auditing, testarea penetrantă și evaluarea codului, oferind exemple practice și instrumente folosite în industrie.

1. Introducere

- **Conținutul prelegerii**
  - Înțelegerea importanței evaluării continue a securității software.
  - Familiarizarea cu metodele de auditing și testare a securității.
  - Prezentarea unor exemple practice pentru testarea și evaluarea codului.

---

2. Auditing-ul în Securitatea Software

2.1 Ce este auditing-ul?

- **Definiție:**
  Auditing-ul este procesul de **colectare, analiză și evaluare a informațiilor** despre activitățile dintr-un sistem pentru a detecta și investiga incidente de securitate.
- **Obiective:**
  - Verificarea conformității cu politicile și standardele de securitate.
  - Monitorizarea activităților pentru a identifica comportamente anormale sau neautorizate.

2.2 Componentele unui sistem de auditing

- **Logare și monitorizare:**
  - Înregistrarea activităților critice (acces, modificări, autentificări).
  - Utilizarea unor sisteme SIEM (Security Information and Event Management) pentru analiză în timp real.
- **Revizuirea auditului:**
  - Audituri periodice ale logurilor și raportarea incidentelor.
- **Instrumente practice:**
  - **Splunk**, **ELK Stack** (Elasticsearch, Logstash, Kibana), **Graylog**.

2.3 Exemplu practic: Auditarea accesului la un sistem

- **Scenariu:** O aplicație web trebuie să monitorizeze autentificările reușite și nereușite.
- **Implementare simplificată în C:**
  // logarea autentificărilor

```
void logEvent(const char *user, const char *eventType) {
    FILE *logFile = fopen("audit.log", "a");
    if (logFile != NULL) {
fprintf(logFile, "User: %s, Eveniment: %s, Timestamp: %ld\n", user, eventType, time(NULL));
        fclose(logFile);
    }
}

// Exemplu de utilizare:
if (authenticate(user, password)) {
    logEvent(user, "Autentificare reușită");
} else {
    logEvent(user, "Autentificare eșuată");
}
```

- **Beneficiu:** Monitorizarea permite detectarea tentativelor de acces neautorizat și identificarea breșelor.

---

3. Metode de Evaluare a Securității Software

3.1 Analiza Statică a Codului

- **Definiție:**
  Analiza statică implică **examinarea codului sursă** fără a-l executa, identificând vulnerabilitățile, erorile de programare și practicile nesigure.
- **Instrumente:**
  - **SonarQube**, **Coverity**, **Fortify Static Code Analyzer**.

  Coverity Scan

  Coverity Scan găsește și remediază gratuit defectele din proiectul open source Java, C/C++, C#, JavaScript, Ruby sau Python (https://scan.coverity.com). Testează fiecare linie de cod și calea potențială de execuție, iar cauza principală a fiecărui defect este explicată în mod clar, facilitând remedierea erorilor.

  SonarQube este un instrument de analiză statică care examinează codul sursă pentru a detecta defecte, vulnerabilități, probleme de stil și alte aspecte legate de calitatea codului. Există o ediție Community (open source) care oferă funcționalități de bază și ediții comerciale (Developer, Enterprise, Data Center) cu funcționalități extinse (analiză a codului în timp real, integrare CI/CD avansată etc.).

- **Exemplu practic:**
    - Analiza unui cod C/C++ pentru detectarea posibilelor vulnerabilități de tip buffer overflow sau folosirea funcțiilor nesigure (ex: strcpy în loc de strncpy).
- Considerați următorul cod C nesigur care conține o vulnerabilitate de tip buffer overflow, prin folosirea funcției nesigure `strcpy`:

```
c
#include <stdio.h>
#include <string.h>

void vulnerableFunction(char *userInput) {
    char buffer[20];
    // Vulnerabil: dacă userInput este mai mare de 19 caractere, va
//apărea buffer overflow
    strcpy(buffer, userInput);
    printf("Buffer content: %s\n", buffer);
}

int main() {
    char userInput[50];
    printf("Introduceti un text: ");
    fgets(userInput, sizeof(userInput), stdin);
    // Elimină newline-ul de la sfârșit, dacă există
    userInput[strcspn(userInput, "\n")] = '\0';
    vulnerableFunction(userInput);
    return 0;
}
```

Detectarea vulnerabilității cu instrumente de analiză statică
Cu SonarQube:

1. **Configurare:**
    - Se configurează SonarQube în cadrul unui proiect C.
    - Se utilizează plugin-ul C/C++ sau un scanner compatibil (ex.: SonarScanner for C/C++).
2. **Scanare:**
    - Se rulează analiza statică prin comanda:

```
bash

sonar-scanner \
  -Dsonar.projectKey=exemplu_c_project \Definește cheia unică a proiectului.
  -Dsonar.sources=. \Specifică că sursele se află în directorul curent.
  -Dsonar.host.url=http://localhost:9000 \URL-ul la care este accesibil
SonarQube.
  -Dsonar.login=<token> Tokenul de autentificare generat în SonarQube.
```

Bash (Bourne Again Shell) este un **interpret de comenzi și un limbaj de scriptare** folosit în mod obișnuit în sistemele Unix și Linux. Bash este utilizat pentru a automatiza sarcini repetitive, gestiona procese și integra scripturi în pipeline-uri CI/CD pentru a executa

comenzi automat, cum ar fi compilarea codului, rularea testelor și implementarea aplicațiilor.

Se poate utiliza Bash pe Windows pentru a rula comenzi precum cele pentru SonarScanner. Iată câteva modalități și pași pentru a face acest lucru:

Exemplu Practic: Configurare și Rulare SonarScanner folosind Bash pe Windows

Presupunem că ai configurat SonarQube pe mașina ta (de exemplu, accesibil la http://localhost:9000) și ai instalat SonarScanner.

Configurarea SonarQube în cadrul proiectului C

- Instalează plugin-ul C/C++ sau utilizează SonarScanner for C/C++ (documentația oficială poate fi consultată pentru detalii suplimentare).

Rularea analizei cu SonarScanner în Git Bash sau WSL

Deschide Git Bash sau terminalul WSL și navighează în directorul proiectului tău. Apoi rulează analiza statică a codului prezentată mai sus.

Folosind Git Bash sau WSL, putem beneficia de mediul Bash pe Windows pentru a rula comenzi de analiză statică precum sonar-scanner. Această abordare permite de a integra SonarQube și de a analiza proiectele C/C++ la fel ca într-un mediu Linux.

3. **Raport:**
    - SonarQube va identifica utilizarea nesigură a funcției strcpy și va sugera înlocuirea cu strncpy sau alte măsuri de validare a lungimii datelor.

Remedierea vulnerabilității

Modificăm funcția vulnerabilă pentru a utiliza o funcție sigură, strncpy:

```
c

#include <stdio.h>
#include <string.h>

void secureFunction(char *userInput) {
    char buffer[20];
    // Se folosește strncpy pentru a limita numărul de caractere copiate
    strncpy(buffer, userInput, sizeof(buffer) - 1);
    buffer[sizeof(buffer) - 1] = '\0';  // Asigurarea terminatorului null
    printf("Buffer content: %s\n", buffer);
}

int main() {
    char userInput[50];
    printf("Introduceti un text: ");
    fgets(userInput, sizeof(userInput), stdin);
    userInput[strcspn(userInput, "\n")] = '\0';
    secureFunction(userInput);
    return 0;
}
```

După remediere, o nouă analiză statică cu oricare dintre aceste instrumente ar trebui să nu mai semnaleze vulnerabilitatea legată de buffer overflow.

3.2 Analiza Dinamică a Codului (Dynamic Analysis)

- **Definiție:**
  Testarea și evaluarea unui program în timpul execuției, pentru a identifica vulnerabilități care apar în runtime.
- **Metode:**
  - **Testare de penetrare (Penetration Testing)**
  - **Fuzzing:** Injectarea de date aleatoare pentru a vedea cum reacționează aplicația.
- **Instrumente:**
  - **Valgrind** (pentru detectarea erorilor de memorie în C/C++), **American Fuzzy Lop (AFL)**.

- **Exemplu practic:**
  - Utilizarea AFL pentru a genera intrări neașteptate la o aplicație C, detectând eventualele erori sau blocări.

Descrierea Instrumentelor

Valgrind

- **Ce este:**
  Un framework open-source pentru instrumentare dinamică a programelor.
- **Scop:**
  Detectează erori de memorie în programe C/C++ (ex.: buffer overflow, acces invalid, use-after-free, memory leaks).
- **Utilizare:**
  Rulează programul într-un mediu controlat și monitorizează accesul la memorie.
- **Disponibilitate:**
  Open-source.

American Fuzzy Lop (AFL)

- **Ce este:**
  Un fuzzer open-source care testează aplicațiile prin generarea de intrări mutate aleatoriu, pentru a descoperi vulnerabilități.
- **Scop:**
  Descoperirea de bug-uri și vulnerabilități (ex.: buffer overflow, crash-uri) prin testare automată intensă.

- **Utilizare:**
  Compilarea programului cu un compiler special (ex.: afl-gcc sau afl-clang) și rularea fuzzer-ului asupra binarului rezultat.
- **Disponibilitate:**
  Open-source.

Metasploit

- **Ce este:**
  Un framework de penetrare care permite simularea de atacuri și exploatarea vulnerabilităților.
- **Scop:**
  Testarea securității sistemelor prin exploatarea vulnerabilităților cunoscute.
- **Utilizare:**
  Există versiunea Metasploit Framework (open-source) și versiuni comerciale (Metasploit Pro).
- **Disponibilitate:**
  Versiunea de bază este open-source.

Burp Suite

- **Ce este:**
  Un set de instrumente pentru testarea securității aplicațiilor web.
- **Scop:**
  Interceptarea, modificarea și analizarea traficului web; detectarea vulnerabilităților precum XSS, SQL Injection etc.
- **Utilizare:**
  Se utilizează interfața grafică pentru a intercepta cererile HTTP/HTTPS.
- **Disponibilitate:**
  Există o ediție Community (gratuită, cu funcționalități limitate) și ediții comerciale (Developer, Enterprise).

Nmap

- **Ce este:**
  Un instrument open-source de scanare a rețelelor.

- **Scop:**
  Descoperirea hosturilor active, porturilor deschise și a serviciilor asociate, evaluarea politicilor de securitate a rețelei.
- **Utilizare:**
  Se rulează din linia de comandă cu diverse opțiuni pentru scanare.
- **Disponibilitate:**
  Open-source.

Wireshark

- **Ce este:**
  Un analizor de protocoale de rețea open-source.
- **Scop:**
  Capturarea și analiza traficului de rețea pentru identificarea unor eventuale activități anormale sau atacuri.
- **Utilizare:**
  Interfață grafică pentru filtrarea și vizualizarea pachetelor de rețea.
- **Disponibilitate:**
  Open-source.

Exemplu Practic de Analiză Dinamică folosind Instrumente Open-Source

Pentru acest exemplu vom folosi **American Fuzzy Lop (AFL)** pentru a descoperi vulnerabilități într-un program C vulnerabil la buffer overflow. De asemenea, vom menționa cum poate fi folosit Valgrind pentru a monitoriza erorile de memorie în timpul execuției.

Codul Vulnerabil

Considerați următorul program C care utilizează funcția nesigură strcpy și este predispus la buffer overflow:

c

```
#include <stdio.h>
#include <string.h>
```

```
void vulnerableFunction(char *input) {
   char buffer[20];
   // Vulnerabil: dacă input-ul este mai lung de 19 caractere, se produce overflow
   strcpy(buffer, input);
   printf("Buffer: %s\n", buffer);
}

int main(int argc, char *argv[]) {
   if (argc != 2) {
      printf("Utilizare: %s <text>\n", argv[0]);
      return 1;
   }
   vulnerableFunction(argv[1]);
   return 0;
}
```

Pregătirea pentru Fuzzing cu AFL

Pasul 1: Instalare AFL

- Descărcați și instalați AFL (de exemplu, de pe site-ul oficial AFL).
- Asigurați-vă că AFL este în PATH.

Pasul 2: Compilarea Codului cu afl-gcc/afl-clang

Utilizați compilatorul oferit de AFL pentru a genera un binar instrumentat:

bash

```
afl-gcc -o vulnerable vulnerable.c
```

Acest lucru va insera cod suplimentar pentru monitorizarea execuției și va facilita detectarea crash-urilor.

Pasul 3: Crearea unui Director cu Date de Intrare (Seeds)

Creați un director (de exemplu, in) cu cel puțin un fișier mic de intrare care să servească drept seed.

Exemplu:

bash

```
mkdir in
echo "test" > in/seed.txt
```

Pasul 4: Rularea AFL

Porniți AFL pentru a fuzz-ui programul:

bash

afl-fuzz -i in -o out -- ./vulnerable @@

- **-i in:** Directorul cu intrări inițiale (seed-uri).
- **-o out:** Directorul în care AFL va stoca rezultatele (crash-uri, cazuri interesante).
- **@@:** Markerul care va fi înlocuit cu numele fișierului de intrare.

AFL va începe să genereze variante mutate ale input-urilor și va monitoriza execuția programului pentru crash-uri sau comportamente neașteptate (de exemplu, buffer overflow).

2.3 Monitorizarea cu Valgrind

Pentru a verifica problemele de memorie detectate în timpul rulării, puteți utiliza Valgrind. De exemplu, pentru a rula programul cu o anumită intrare:

bash

valgrind --leak-check=full ./vulnerable "input test"

Valgrind va afișa eventualele erori de acces la memorie, cum ar fi buffer overflow sau acces după eliberare.

2.4 Interpretarea Rezultatelor

- **AFL:**
  AFL va salva în directorul out/crashes orice input care a cauzat un crash sau comportament neașteptat. Analiza acestor fișiere vă va ajuta să identificați vulnerabilitățile.
- **Valgrind:**
  Raportul Valgrind va indica liniile de cod unde au apărut erorile de memorie, permițându-vă să corectați codul.

Concluzii

- **Instrumentele de analiză dinamică și de penetrare** (AFL, Valgrind) sunt foarte utile pentru descoperirea vulnerabilităților care nu pot fi detectate prin analiză statică.
- **AFL** este un fuzzer open-source extrem de eficient pentru a genera intrări neașteptate și a detecta erori precum buffer overflow.
- **Valgrind** ajută la detectarea erorilor de memorie, completând analiza realizată de AFL.
- Prin integrarea acestor instrumente în pipeline-ul de dezvoltare, puteți îmbunătăți semnificativ calitatea și securitatea software-ului.

Acest exemplu practic arată cum puteți utiliza instrumente open-source pentru analiza dinamică a codului C, cu scopul detectării vulnerabilităților.

3.3 Testarea penetrantă (Penetration Testing)

- **Definiție:**

  Testarea penetrantă este o **simulare a unui atac real** asupra sistemului, prin identificarea vulnerabilităților prin tehnici și instrumente specifice.
- **Etapele unui test penetrant:**
  1. **Recunoaștere (Reconnaissance):** Colectarea de informații despre țintă.
  2. **Scanare:** Identificarea porturilor deschise, serviciilor și posibilelor vulnerabilități.
  3. **Exploatare:** Încercarea de a exploata vulnerabilitățile identificate.
  4. **Post-exploatare:** Evaluarea impactului și accesul la sistem.
  5. **Raportare:** Documentarea descoperirilor și recomandările de remediere.
- **Instrumente:**

  Metasploit, **Burp Suite**, **Nmap**, **Wireshark**.
- **Exemplu practic:**
  1. **Scenariu:** Testarea unei aplicații web pentru vulnerabilități SQL Injection.
  2. **Procedură:**
     1. Se utilizează **Nmap** pentru scanarea porturilor și identificarea serviciilor.
     2. Se folosește **Burp Suite** pentru interceptarea cererilor HTTP și injectarea de payload-uri SQL:

        sql

        ' OR '1'='1
     3. Se monitorizează răspunsurile serverului pentru a identifica dacă atacul a reușit.
  3. **Rezultat:** Dacă serverul returnează date neautorizate sau se comportă anormal, se identifică vulnerabilitatea SQL Injection.

4. Evaluarea Codului – Best Practices

4.1 Manual Code Review

- **Definiție:**

  Evaluarea manuală a codului sursă de către experți în securitate pentru a identifica practici nesigure.
- **Metodologie:**
  - Verificarea conformității cu standardele de securitate (ex.: OWASP Secure Coding Guidelines).
  - Utilizarea checklist-urilor de securitate.
- **Exemplu practic:**
  - Revizuirea codului C/C++ pentru identificarea vulnerabilităților de tip buffer overflow, folosirea corectă a funcțiilor sigure, gestionarea erorilor etc.

4.2 Automatizarea Analizei

- **Instrumente:**
  - **Static Application Security Testing (SAST):** SonarQube, Fortify.
  - **Dynamic Application Security Testing (DAST):** OWASP ZAP, Burp Suite.
- **Beneficii:**
  - Identificarea rapidă a erorilor și vulnerabilităților.
  - Integrarea în pipeline-ul de CI/CD pentru testare continuă.
- **Exemplu practic:**
  - Configurarea unui pipeline CI/CD care rulează SonarQube la fiecare commit pentru a detecta vulnerabilități în codul sursă.

---

5. Concluzii și Recomandări

- **Auditul și testarea securității software sunt procese continue:**
  - Este esențial ca evaluarea securității să fie integrată în întregul ciclu de dezvoltare (Secure SDLC).
  - Implementarea unor instrumente automate împreună cu revizuirile manuale oferă o acoperire cât mai completă.
- **Colaborarea între echipele de dezvoltare și securitate:**
  - Comunicarea și instruirea continuă a dezvoltatorilor în privința practicilor de codare securizată contribuie la reducerea vulnerabilităților.
- **Documentarea și raportarea:**
  - Rezultatele testelor de penetrare și audit trebuie documentate clar pentru a putea fi remediate vulnerabilitățile identificate și pentru a evalua progresul.

Exerciții practice sugerate pentru studenți:

1. **Test de penetrare pe o aplicație web demo:**
   - Utilizați Burp Suite pentru a intercepta cererile HTTP și injectați payload-uri pentru a detecta vulnerabilități SQL Injection și XSS.
2. **Analiză statică a unui proiect C/C++:**
   - Configurați SonarQube pentru un proiect simplu și identificați vulnerabilități legate de managementul memoriei și funcțiile nesigure.
3. **Simulare de audit de securitate:**
   - Realizați o revizuire manuală a codului unui modul de autentificare și propuneți îmbunătățiri pentru a elimina eventuale breșe de securitate.

Analiza statică și dinamică a soft-ului (material adăugător)

Cea mai bună modalitate de a asigura codarea securizată este utilizarea unui analizor de cod static. Analiza statică ajută echipele de dezvoltare care sunt sub presiune. Versiunile de calitate trebuiau livrate la timp. Standardele de codificare și conformitate trebuie îndeplinite. Și greșelile nu sunt o opțiune.

De aceea, echipele de dezvoltare folosesc cele mai bune instrumente de analiză statică a codului Aici, discutăm despre analiza statică și beneficiile utilizării analizoarelor de cod, precum și limitările analizei statice.

Ce este analiza statică

Analiza statică este cel mai bine descrisă ca o metodă de depanare care se face prin examinarea automată a codului sursă fără a fi nevoie să executați programul. Acest lucru oferă dezvoltatorilor o înțelegere a bazei lor de cod și îi ajută să se asigure că acesta este conform, sigur și securizat.

Analiza statică a codului sursă se referă la operația efectuată de un instrument de analiză a codului sursă, care este analiza unui set de cod față de un set (sau mai multe seturi) de reguli de codare.

Analiza statică a codului abordează punctele slabe ale codului sursă care ar putea duce la vulnerabilități. Desigur, acest lucru poate fi realizat și prin recenzii manuale ale codului sursă. Dar folosirea instrumentelor automate este mult mai eficientă.

Analiza statică este folosită în mod obișnuit pentru a respecta liniile directoare de codificare, cum ar fi MISRA. Și este adesea folosit pentru a respecta standardele din industrie, cum ar fi ISO 26262.

Cum funcționează Analiza codului static pentru a asigura un cod de înaltă calitate, sigur și fiabil.

Analiza statică în testarea software-ului

Analiza statică joacă un rol cheie înainte de a începe testarea software-ului. Se asigură că codul pe care îl transmiteți testării este de cea mai înaltă calitate posibilă. Și, dacă alegeți analizorul static potrivit, acesta accelerează procesul de dezvoltare.

Ce poate găsi analiza statică

Analiza statică găsește potențiale probleme de calitate în codul dvs. înainte de a rula programul. Aceasta include:

• Erori de programare

• Codarea cu încălcările standardului

• Deficiențe de securitate

Analizoarele statice sunt deosebit de bune în găsirea problemelor de codare, cum ar fi depășirea tamponului, scurgerile de memorie și pointerii nuli.

Și analiza statică educă dezvoltatorii cu privire la cele mai bune practici de codare, ceea ce vă ajută să îmbunătățiți calitatea pe termen lung.

Ce nu poate fi identificat în analiza statică

Există lucruri pe care analiza statică nu le poate identifica. De exemplu, analiza statică nu poate detecta dacă cerințele software au fost îndeplinite sau cum se va executa o funcție. Veți avea nevoie de testare dinamică pentru asta.

De aceea, analiza statică și analiza (testarea) dinamică sunt complementare. Analiza statică detectează erori în cod de la început. Acest lucru asigură că un produs de calitate superioară ajunge în faza de testare. Și accelerează dezvoltarea, asigurându-se că procesele de testare sunt mai eficiente.

Analiză statică vs analiză dinamică

Deci, care este diferența dintre analiza statică și analiza dinamică?

Ambele tipuri de analiză de cod detectează defecte. Marea diferență este unde găsesc defecte în ciclul de viață al dezvoltării.

Analiza statică identifică defectele înainte de a rula un program (de exemplu, între codare și testare).

Analiza statică a codului: este cunoscută și sub numele de Testare statică de securitate a aplicațiilor (SAST). Este o metodă de depanare care implică examinarea codului sursă înainte de a rula un program. Se realizează prin compararea unui set de cod cu un set de reguli de codare

Analiza dinamică a codului identifică defectele după ce rulați un program (de exemplu, în timpul testării). Cu toate acestea, este posibil ca unele erori de codare să nu apară în timpul testării. Deci, există defecte pe care testarea dinamică le poate scăpa și pe care analiza codului static le poate găsi.

Analiza dinamică a codului: este cunoscută și sub numele de Testare dinamică de securitate a aplicațiilor (DAST) și este folosit pentru a testa o aplicație care rulează pentru defecte potențial exploatabile. Instrumentele DAST pot detecta vulnerabilități de compilare și de rulare, cum ar fi erorile de configurare care apar numai într-un mediu de execuție realist

Care sunt limitările instrumentelor de analiză statică și instrumentelor de analiză a codului sursă statică?

Analiza statică a codului este utilizată într-un anumit scop într-o anumită fază de dezvoltare. Dar există unele limitări ale unui instrument de analiză statică a codului.

Fără înțelegere a intenției dezvoltatorului

```
int calculateArea(int length, int width)
{
    return (length + width);
}
```

Un instrument de analiză statică poate detecta o eroare posibilă în acest calcul. Dar nu poate determina că funcția în mod fundamental nu face ceea ce se așteaptă!

Posibilele defecte duc la false pozitive și false negative

În unele situații, un instrument poate raporta doar că există un posibil defect.

```
int divide(void)
{
 int x;
 if(foo()) {
 x = 0;}
else {
 x = 5; }
 return (10/x);
}
```

Dacă nu știm nimic despre foo(), nu știm ce valoare va avea x.

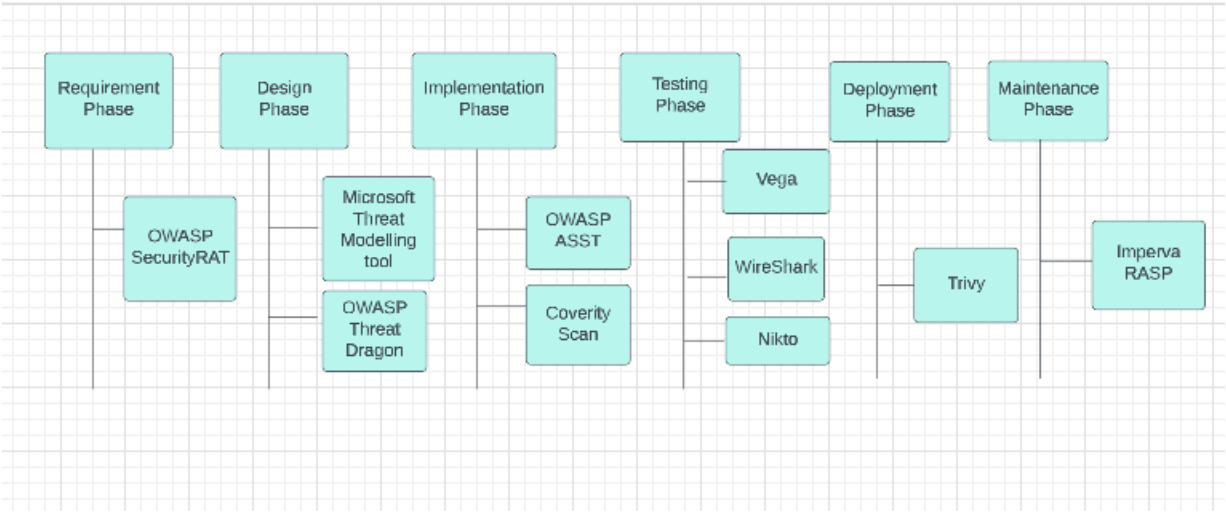
Asta înseamnă că instrumentele pot raporta defecte care nu există efectiv (**false pozitive**). Sau pot nu raporta defecte reale (**negative false**).

Instrumente open source securizate în ciclul de viață produsului program SDLC

Figura 1 prezintă un aspect al instrumentelor open source utile în dezvoltarea de aplicații securizate pentru dezvoltatori și persoane fizice pe care le pot utiliza pe parcursul ciclului de viață al dezvoltării software.

Figura 1

Instrumente open source securizate în ciclul de viață SDLC



Requirement Phase **Faza de cerință**

OWASP SecurityRAT

SecurityRAT (Security Requirement Automation Tool) este un instrument care ajută la gestionarea cerințelor de securitate în proiecte de dezvoltare agilă (OWASP SecurityRAT,2022). Ideea de bază este simplă: specificați proprietățile unei aplicații (sau „artefact") pe care o dezvoltați. Pe baza acestor proprietăți, instrumentul generează o listă de cerințe de securitate pe care trebuie să le îndepliniți. Pentru fiecare cerință, puteți decide dacă ar trebui/va fi implementată și puteți adăuga propriul comentariu. Când ați terminat, puteți salva cerința specifică într-un bilet JIRA pentru referință ulterioară ca fișier YAML (SecurityRAT, 2021). După aceea, puteți crea bilete JIRA în bloc pentru cerințe specifice și le puteți urmări cu SecurityRAT.

Design Phase **Faza de proiectare**

Microsoft Threat Modelling Tool Instrumentul de modelare a amenințărilor Microsoft

Le permite arhitecților software să identifice și să atenueze potențialele probleme de securitate devreme, atunci când acestea sunt relativ simple și ieftin de rezolvat (jegeib, 2022). Instrumentul a fost creat având în vedere experți care nu fac parte din securitate, cu scopul de a face modelarea amenințărilor mai ușoară pentru toți dezvoltatorii, oferind îndrumări clare cu privire la crearea și analiza modelelor de amenințări (jegeib, 2022).

Oricine poate folosi instrumentul pentru a:

● Comunicați despre arhitectura de securitate a sistemelor lor

● Utilizați o metodologie testată pentru a examina proiectele pentru eventualele defecte de securitate.

● Gestionați atenuările pentru problemele de securitate făcând sugestii.

OWASP Threat Dragon Dragonul de amenințare OWASP

OWASP Threat Dragon este un instrument de modelare folosit pentru a crea diagrame de modele de amenințări ca parte a unui ciclu de viață securizat (OWASP Threat Dragon, 2023). Poate fi folosit pentru a înregistra potențialele amenințări și pentru a decide strategiile de atenuare, precum și pentru a oferi o reprezentare vizuală a componentelor modelului de amenințare și a suprafețelor amenințărilor. Threat Dragon este disponibil ca aplicație web sau aplicație desktop.

Implementation Phase **Faza de implementare**

OWASP ASST

ASST este un instrument de scanare a codului open source cu o interfață de linie de comandă scrisă în JavaScript. Când ASST scanează pentru un proiect, examinează fiecare fișier linie cu linie pentru defecte de securitate. Dacă a fost descoperită o vulnerabilitate, raportul va indica în ce linie

în ce fișier a fost descoperită vulnerabilitatea, precum și un link „Click Here” pentru a explica atacul și cum să vă protejați împotriva acestuia (OWASP ASST, 2020). În prezent, se concentrează pe limbajele de programare PHP și MySQL, dar pentru că funcționalitățile sale de bază sunt gata și disponibile pentru toată lumea, programatorii pot contribui și adăuga plugin-uri sau extensii la acesta pentru a adăuga caracteristici și a-l face să scaneze pentru alte limbaje de programare și cadre precum Java, C#, Python și așa mai departe. Drept urmare, infrastructura sa este destinată să fie partajată cu alți programatori pentru a se îmbunătăți și a inova.

Coverity Scan

Coverity Scan găsește și remediază gratuit defectele din proiectul open source Java, C/C++, C#, JavaScript, Ruby sau Python (Coverity Scan - Static Analysis, n.d.). Testează fiecare linie de cod și calea potențială de execuție, iar cauza principală a fiecărui defect este explicată în mod clar, facilitând remedierea erorilor.

***Testing Phase* Faza de Testare**

Vega

Vega este un scanner de securitate și o platformă de testare pentru aplicații web gratuit și open source. Este scris în Java, are o interfață grafică cu utilizatorul și este compatibil cu Linux, OS X și Windows. Vulnerabilitățile cum ar fi scripturile reflectate între site-uri, scripturile stocate între site-uri, injecția oarbă SQL, includerea fișierelor la distanță, injectarea shell-ului și altele pot fi detectate folosind Vega. (Vega Vulnerability Scanner, 2014). De asemenea, verifică setările de securitate TLS / SSL și identifică modalități de îmbunătățire a securității serverelor dvs. TLS.
Vega include un scanner de testare rapidă și un proxy de interceptare pentru inspecția tactică.

- Scanerul automat Vega este alimentat de un crawler pentru site-uri web. Când i se oferă acreditări de utilizator, Vega se poate conecta automat la site-uri web.
- Proxy-ul Vega poate fi, de asemenea, configurat pentru a rula module de atac în timp ce utilizatorul navighează la site-ul țintă prin intermediul acestuia. Pentru a asigura o acoperire maximă a codului, este posibilă testarea de securitate semi-automatizată, condusă de utilizator.

Wireshark

Wireshark are un set de funcții bogat și puternic și rulează pe majoritatea platformelor de calcul, inclusiv Windows, OS X, Linux și UNIX (Banerjee et al., 2010). Este folosit frecvent de profesioniști în rețea, experți în securitate, dezvoltatori și educatori din întreaga lume. Este open source și distribuit sub Licența Publică Generală GNU versiunea 2. A fost creat și este întreținut de o echipă globală de experți în protocol și este un exemplu de tehnologie disruptivă. Wireshark este o aplicație gratuită de calculator pentru sniffer de pachete. Este folosit pentru depanarea rețelei, analiză, dezvoltare de software, dezvoltare de protocol de comunicare pment, și educație. Wireshark poate capta traficul „din aer” și decodați-l într-un format care îi ajută pe administratorii

să urmărească problemele care cauzează performanțe slabe, conectivitate intermitentă și alte probleme comune cu suportul adecvat pentru driver. Wireshark le permite utilizatorilor să capteze pachete care călătoresc prin întreaga rețea pe o anumită interfață la un anumit moment.
(Instrument de monitorizare și analiză a rețelei Wireshark https://www.wireshark.org .

Nikto

Nikto este un scaner de server web Open Source (GPL) care rulează teste cuprinzătoare împotriva serverelor web pentru o varietate de elemente, inclusiv peste 6700 de fișiere/programe potențial periculoase, versiuni învechite de peste 1250 de servere și probleme specifice versiunii pe peste 270 de servere. De asemenea, caută elemente de configurare a serverului, cum ar fi prezența mai multor fișiere index și opțiuni de server HTTP, precum și încercarea de a identifica serverele web și software-ul instalat. Elementele de scanare și pluginurile sunt actualizate frecvent și pot fi actualizate automat (Nikto2 | CIRT.Net, 2023).

***Deployment Phase* Faza de desfășurare**

Trivy

Trivy este un scaner de securitate ușor de utilizat, open source și versatil. Trivy are scanere care caută probleme de securitate și ținte unde pot fi găsite acele probleme. Este ideal pentru conductele DevSecOps, deoarece se integrează cu instrumente CI precum Travis, CircleCI, Jenkins și GitLab. Trivy poate scana - Container Images, Filesystems, Git Repository (la distanță), Virtual Machine Image, Kubernetes și AWS. Poate descoperi- pachete de sistem de operare și dependențe de software în uz (SBOM), vulnerabilități cunoscute (CVE), probleme IaC și configurări greșite, informații și secrete sensibile, licențe software (Prezentare generală - Trivy, n.d.).

***Maintenance Phase* Faza de intretinere**

Imperva RASP

RASP, care este încorporat în mediul de rulare al aplicației, poate detecta și preveni atacurile în timp real. Din cauza provocărilor de securitate de astăzi, aplicațiile dvs. native din cloud necesită mai multă confidențialitate decât firewall-urile de rețea, motiv pentru care Imperva oferă protecție din interior și rulează împreună cu aplicațiile dvs. (RASP Market Leader, n.d.). Poate proteja aplicația folosind:

● Reducerea riscului aplicației: RASP protejează aplicațiile de vulnerabilități, permițând echipelor să se concentreze pe logica de afaceri în loc să lase aplicația dumneavoastră vulnerabilă la exploatare.

● Securitatea pe măsură ce afacerea evoluează: aplicațiile native cloud necesită mai mult decât securitatea perimetrului din cauza controalelor decolorate și a sarcinilor de lucru efemere. RASP oferă securitate internă și urmărește aplicația dvs. oriunde ar merge.

Alte instrumente de testare

Instrumentele de testare sunt esențiale pentru ca profesioniștii în securitate să identifice și să evalueze vulnerabilitățile.

Instrumentele sunt clasificate în funcție de funcționalitățile lor:

Colectarea de informații și recunoaștere

Colectarea informațiilor și recunoașterea sunt faze critice ale securității cibernetice și ale testării de penetrare. Aceste faze implică colectarea de date despre sistemul țintă, rețea sau organizație pentru a obține informații în potențiale vulnerabilități, puncte slabe și vectori de atac:

• **Nmap** (network mapper): Nmap este un instrument puternic de scanare open source pentru descoperirea rețelei și audit de securitate. Permite utilizatorilor să descopere gazde și servicii într-o rețea de computere, creând astfel o hartă a rețelei. Nmap utilizează pachetele IP brute pentru a determina care gazde sunt vizibile în rețea, ce servicii oferă acele gazde, ce sisteme de operare rulează, ce tip de firewall-uri sunt utilizate și alte atribute.

Figura 14.1 prezintă o captură de ecran a utilizarea liniei de comandă a Nmap:

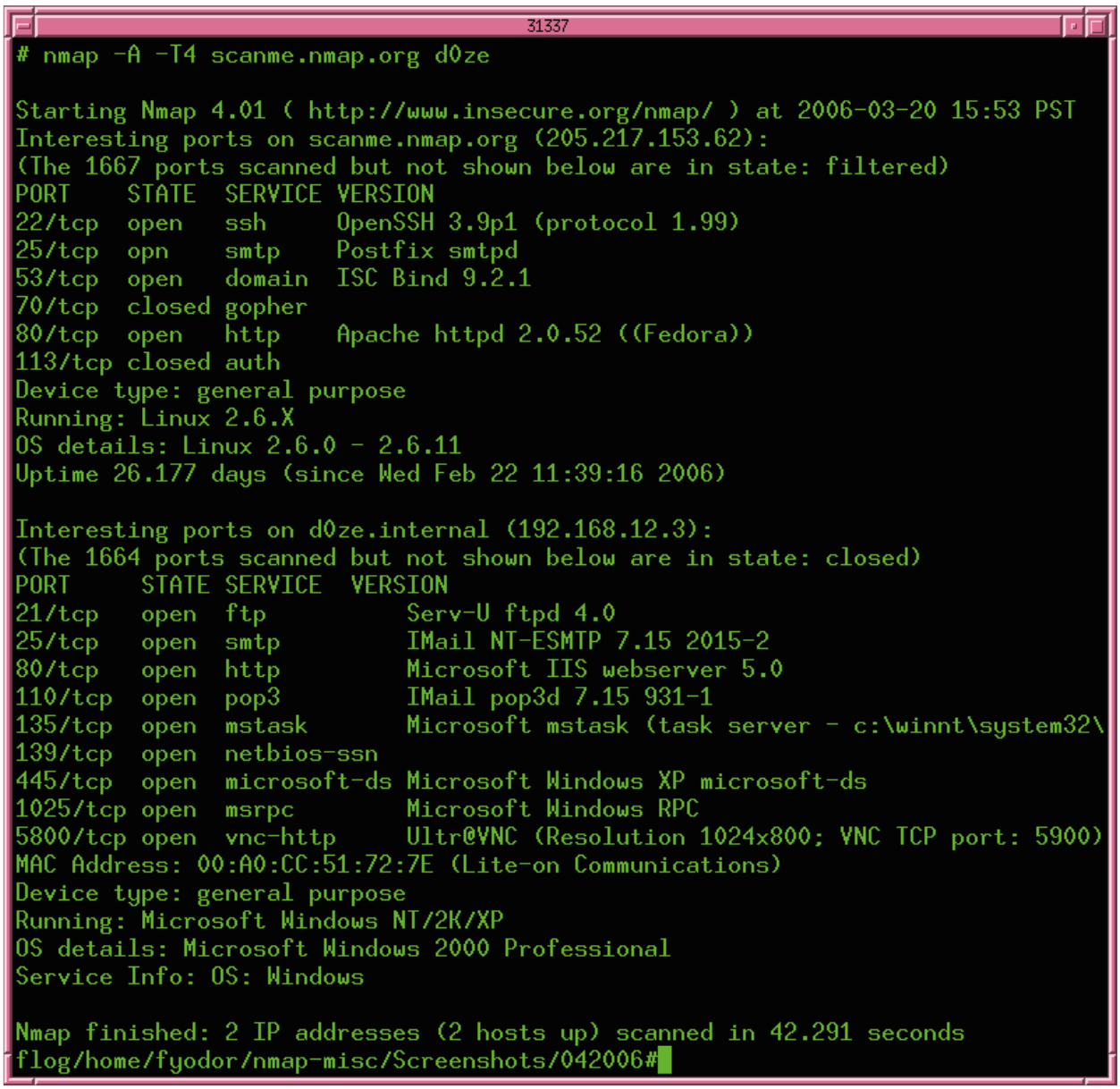


Figura 14.1 – Utilizarea liniei de comandă pentru NMap

• **Maltego:** Maltego este un instrument puternic de vizualizare și analiză a datelor pentru colectarea și corelarea informații despre oameni, organizații și relații din diverse surse online. Ea permite utilizatorilor să efectueze analize de legătură, recunoaștere a rețelei și colectare OSINT pentru a facilita investigații și colectare de informații. Figura 14.2 prezintă o captură de ecran a informațiilor Maltego a putut obține pentru domeniul gnu.org:

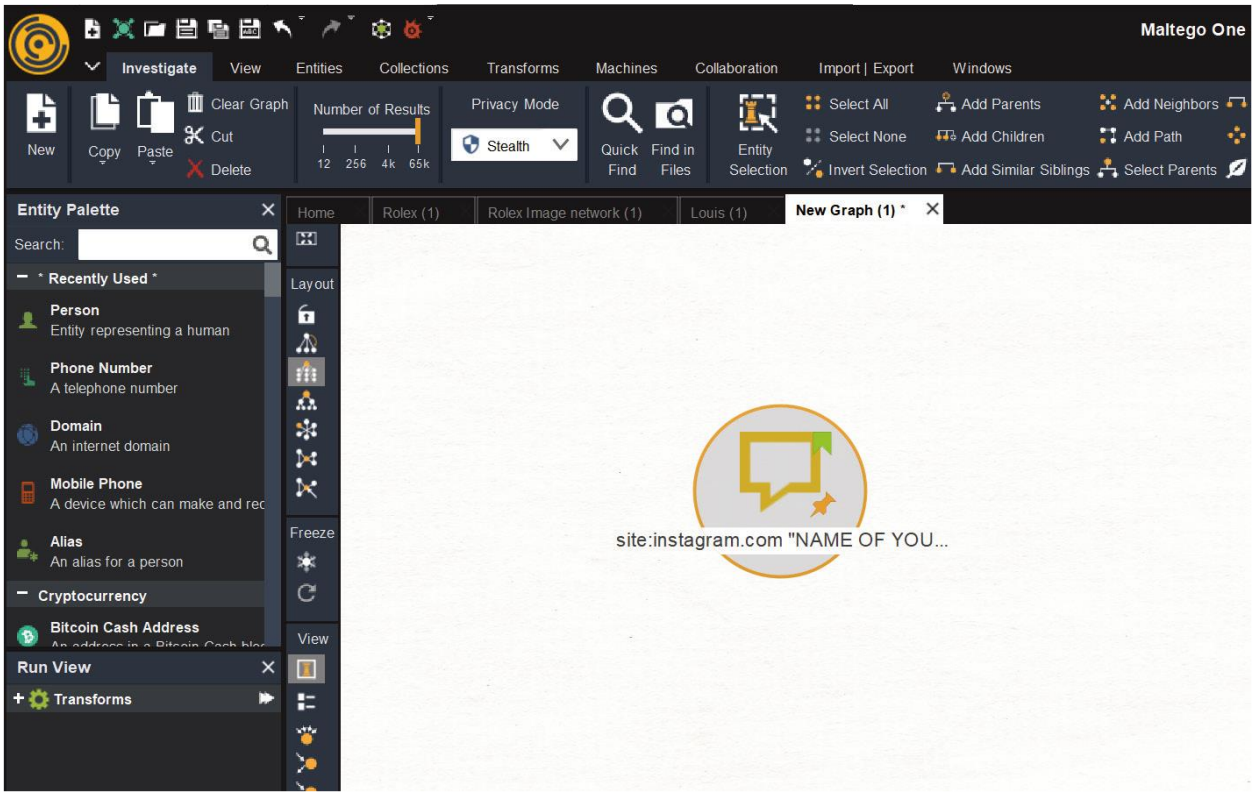


Figura 14.2 – Afișarea Maltego a inteligenței open source

• **theHarvester:** theHarvester este un instrument OSINT folosit pentru a culege informații despre e-mail adrese, nume de domenii și subdomenii din diverse surse publice de pe internet utilizat în mod obișnuit de profesioniștii în securitate, testerii de penetrare și anchetatori pentru a conduce recunoaștere și adună informații în timpul evaluărilor și investigațiilor de securitate.
Figura 14.3 arată o captură de ecran a interfeței liniei de comandă Harvester:

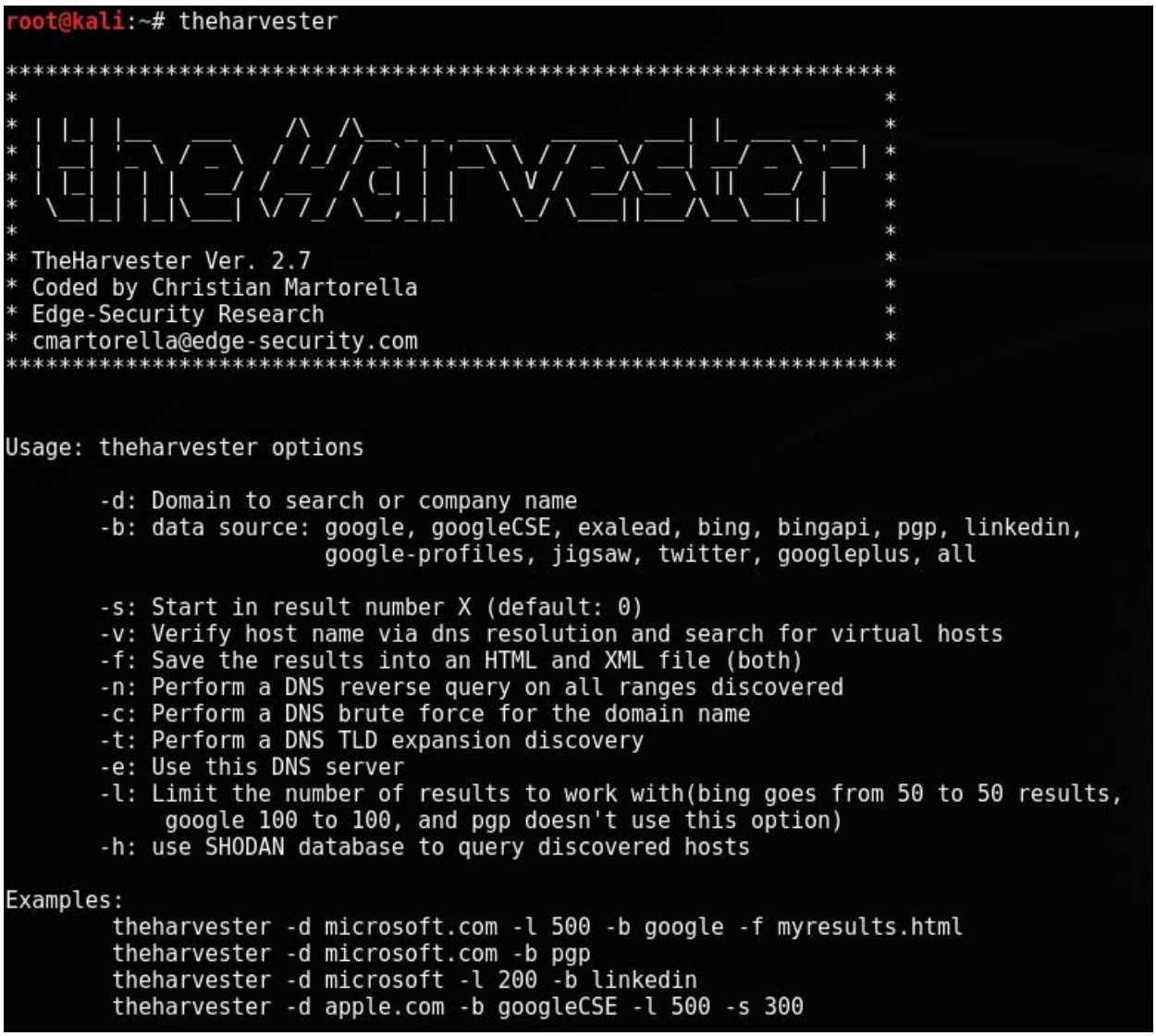


Figura 14.3 – linia de comandă Harvester

Analiza și exploatarea vulnerabilităților

Analiza și exploatarea vulnerabilităților sunt faze critice ale securității cibernetice și ale testării de penetrare.

Aceste faze implică identificarea și exploatarea vulnerabilităților și punctelor slabe de securitate în țintă sisteme, rețele sau aplicații:

• **Metasploit:** Metasploit este un cadru popular de testare de penetrare care permite profesioniștilor în securitate a evalua și exploata vulnerabilitățile sistemelor informatice, rețelelor și aplicațiilor. Ea oferă o suită cuprinzătoare de instrumente pentru testarea de penetrare, inclusiv dezvoltarea exploit-urilor, generarea sarcinii utile, modulele post-exploatare și atacurile de inginerie socială. **Metasploit** este utilizat pe scară largă de către cercetătorii de securitate, testerii de penetrare și hackerii etici pentru a identifica și să remedieze deficiențele de securitate din sistemele țintă. Figura 14.4 prezintă o captură de ecran a Liniei de comandă Metasploit:

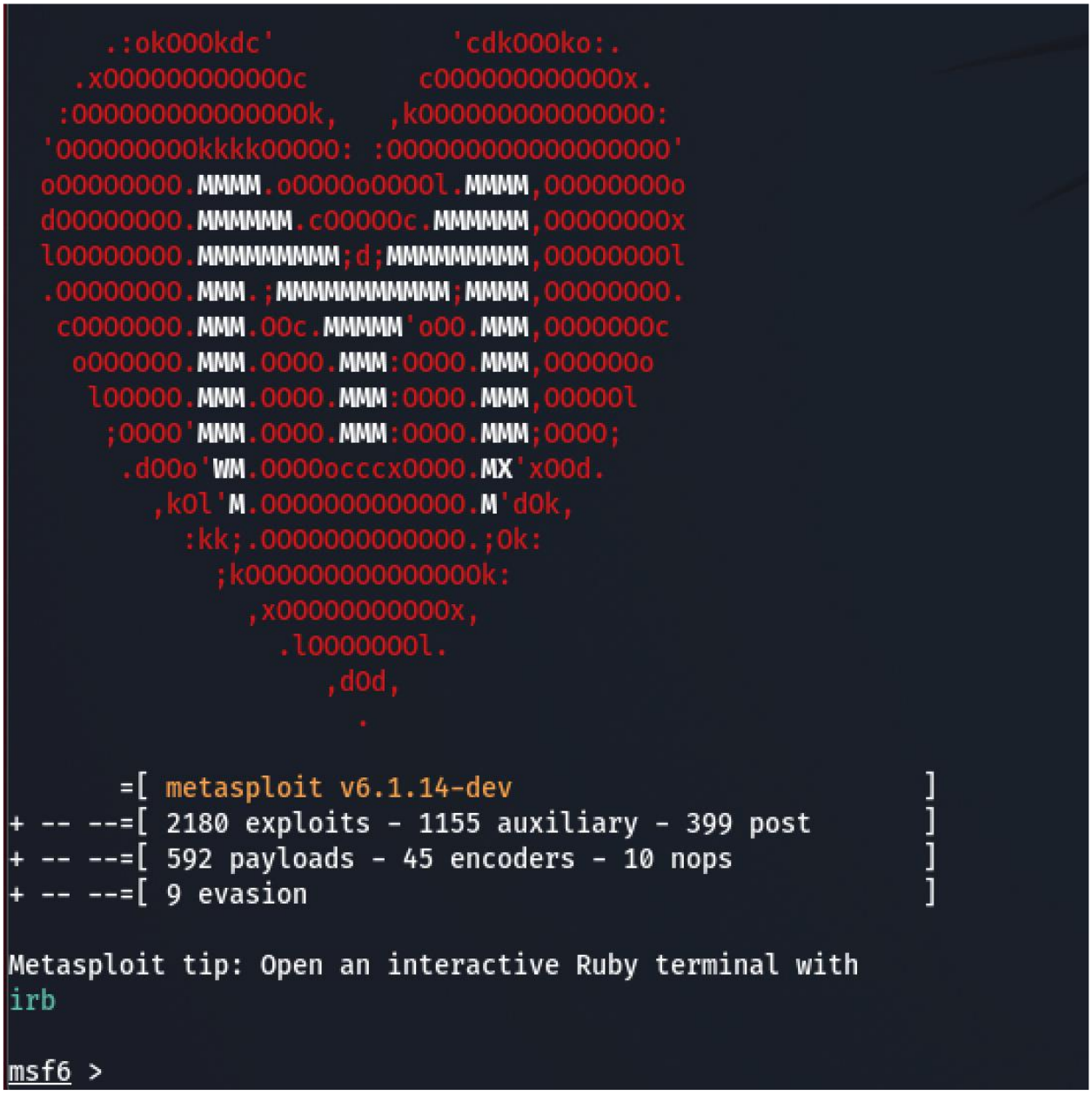


Figura 14.4 – Linia de comandă Metasploit

• **Nessus:** Nessus este un instrument de scanare a vulnerabilităților utilizat pe scară largă, dezvoltat de rețeaua Tenable. Este conceput pentru a scana rețelele de computere pentru vulnerabilități, configurații greșite și punctele slabe de securitate, oferind rapoarte detaliate care ajută organizațiile să identifice și să prioritizeze probleme de securitate.
Figura 14.5 prezintă o captură de ecran a aplicației web scaner Nessus:

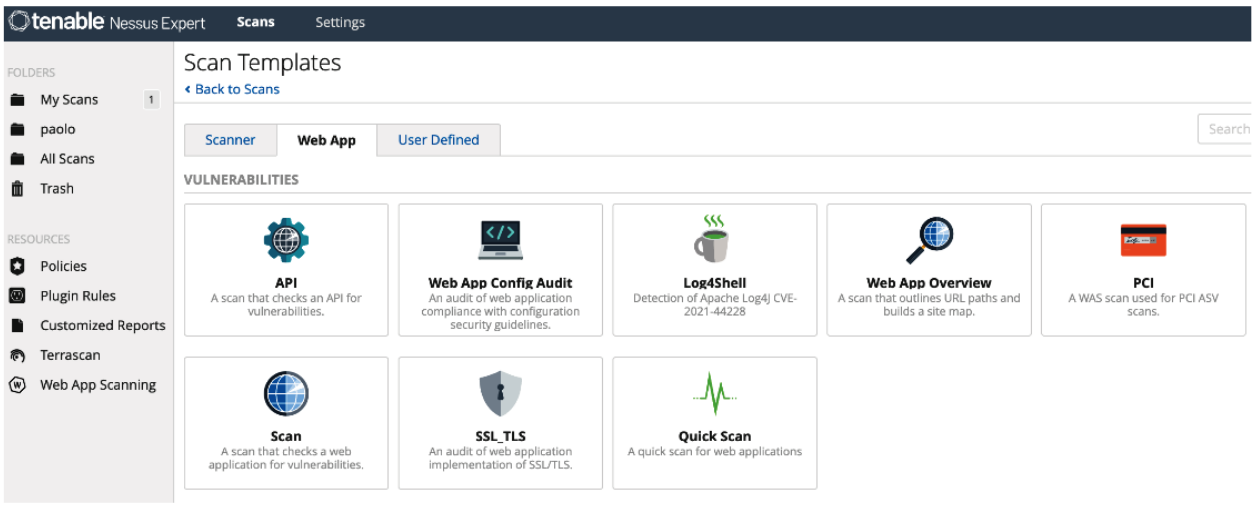


Figura 14.5 – Șabloanele de scanare a aplicației web Nessus

• **Burp** Suite: Burp Suite este o platformă de testare a aplicațiilor web. PortSwigger dezvoltat Burp Suite, care este utilizat pe scară largă de profesioniștii în securitate, testerii de penetrare și dezvoltatorii web pentru a găsi și a atenua vulnerabilitățile de securitate în aplicațiile web. Figura 14.6 prezintă o captură de ecran din aplicația Burp Suite:

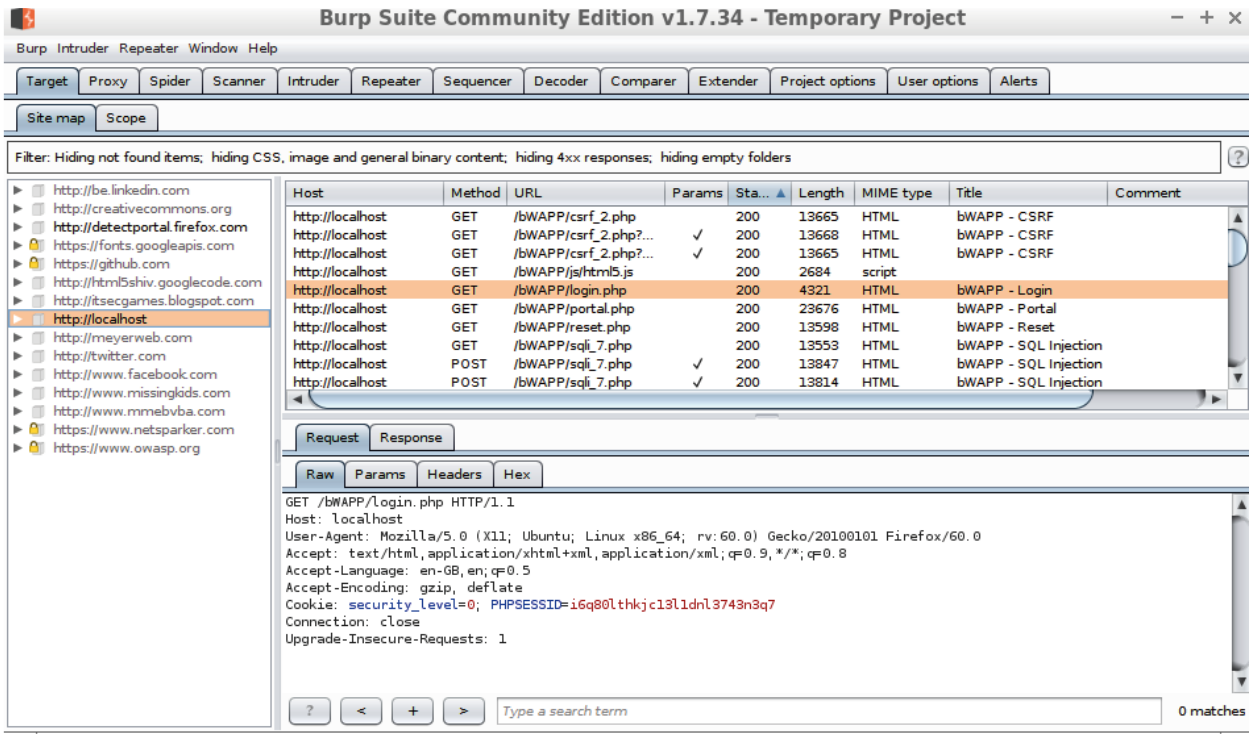


Figura 14.6 – Captură proxy Burp Suite

• **VEGA:** Vega este o platformă de testare și scaner de vulnerabilități web open source utilizată de securitate de profesioniști, testeri de penetrare și dezvoltatori web pentru a identifica și remedia securitatea vulnerabilități în aplicațiile web. Oferă o interfață grafică ușor de utilizat și instrumente pentru scanare automată, testare manuală și analiza vulnerabilităților.
Figura 14.7 prezintă o captură de ecran al scanerului Vega:

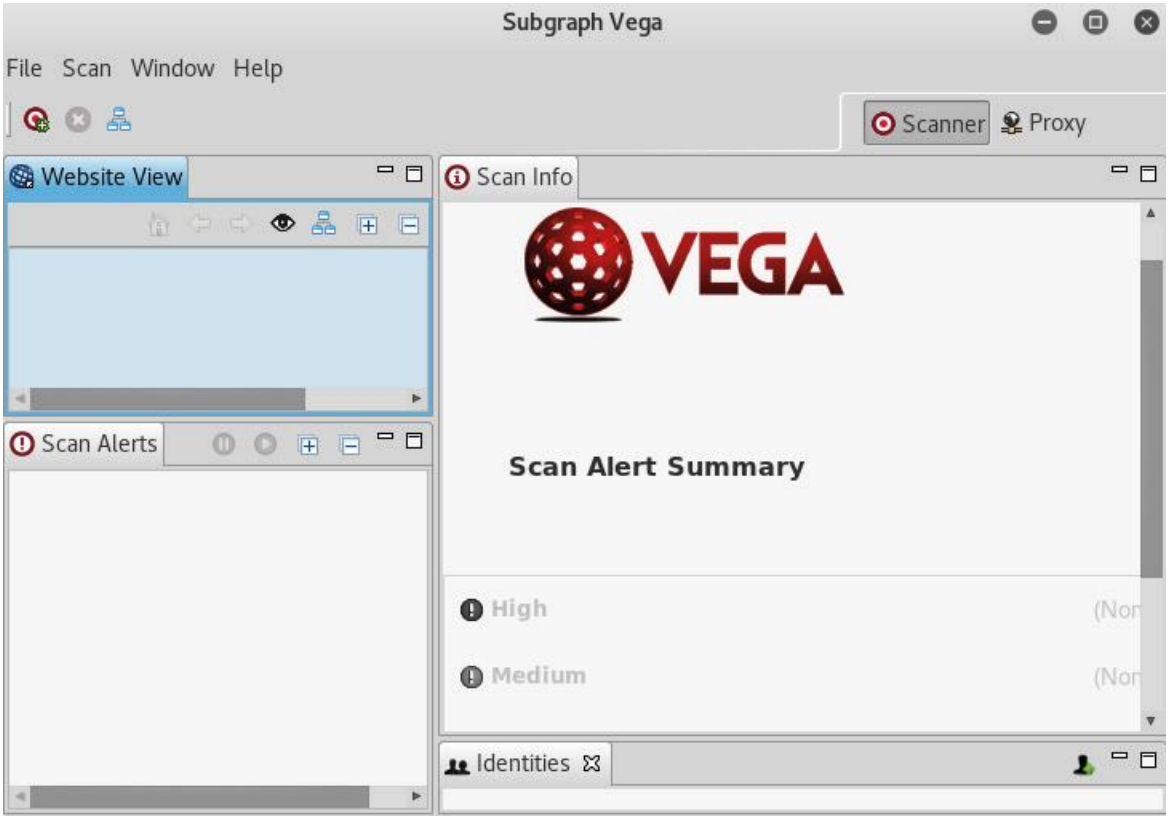


Figura 14.7 - Scanerul Vega

• **OWASP ZAP** (Zed Attack Proxy): OWASP ZAP este un instrument de testare a securității aplicațiilor web dezvoltat de Open Web Application Security Project (OWASP). Este conceput pentru a ajuta utilizatorii să identifice și să atenueze vulnerabilitățile de securitate din aplicațiile web. OWASP ZAP este open source și gratuit pentru utilizare de oricine.
Figura 14.8 arată o captură de ecran a OWASP Zap:

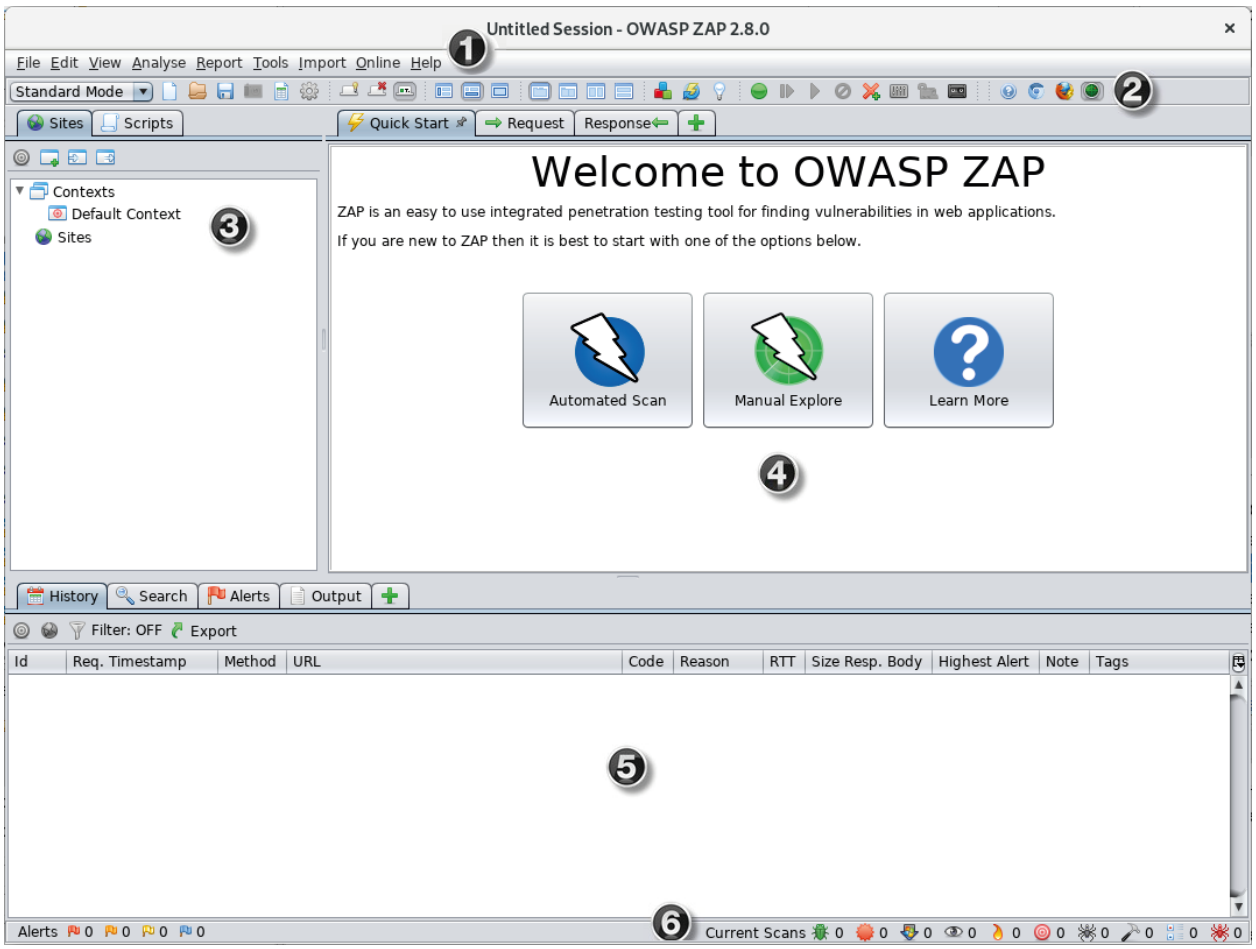


Figura 14.8 – Ecranul de start OWASP Zap

• **SQLmap:** SQLmap este un instrument de testare a pătrunderii bazei de date care automatizează detectarea și exploatarea vulnerabilități de injectare SQL în aplicațiile web. Acesta permite utilizatorului să identifice și să exploateze SQL defecte de injectare în backend-ul bazei de date a unei aplicații țintă. Figura 14.9 prezintă rezultatele an Scanării SQLMap împotriva unei baze de date:

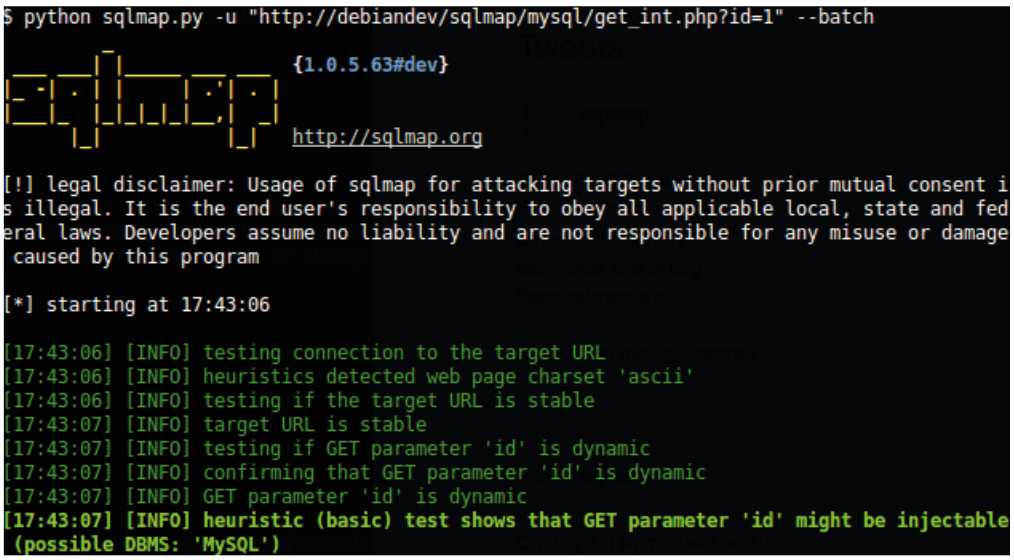


Figura 14.9 – Scanare SQLMap

Post-exploatare și escaladare a privilegiilor

Post-exploatare și escaladarea privilegiilor sunt faze critice ale securității cibernetice și testării de penetrare, procese, care apar de obicei după ce accesul inițial a fost obținut la un sistem sau o rețea țintă. În timpul în aceste faze, atacatorii urmăresc să-și extindă controlul asupra mediului compromis, să-și escaladeze privilegii și menținerea accesului persistent pentru exploatarea viitoare.

• **PowerShell Empire:** PowerShell Empire este un cadru post-exploatare conceput pentru echipa roșie de operațiuni și teste de penetrare. Oferă instrumente și module puternice pentru întreținere persistență, escaladarea privilegiilor și exfiltrarea datelor pe sisteme Windows compromise. PowerShell Empire este adesea folosit de profesioniștii în securitate, testerii de penetrare și membrii echipei roșii pentru a simula atacuri avansate de amenințări persistente (APT) și pentru a evalua securitatea unei organizații. Figura 14.10 arată o captură de ecran a Power Shell Empire:

Figura 14.10 – Powershell Empire

• **Mimikatz:** Mimikatz este un instrument puternic de post-exploatare care este folosit în mod obișnuit de profesioniști securitate, testeri de penetrare și atacatori pentru a extrage parole în text simplu, hashuri și Bilete Kerberos din memoria computerului pe sisteme Windows. Dezvoltat de Benjamin Delpy Mimikatz a devenit cunoscut pe scară largă pentru capacitatea sa de a efectua furtul de acreditări și de a transmite hash atacuri, printre alte activități post-exploatare. Figura 14.11 prezintă o captură de ecran a lui Mimkatz extragerea parolei:

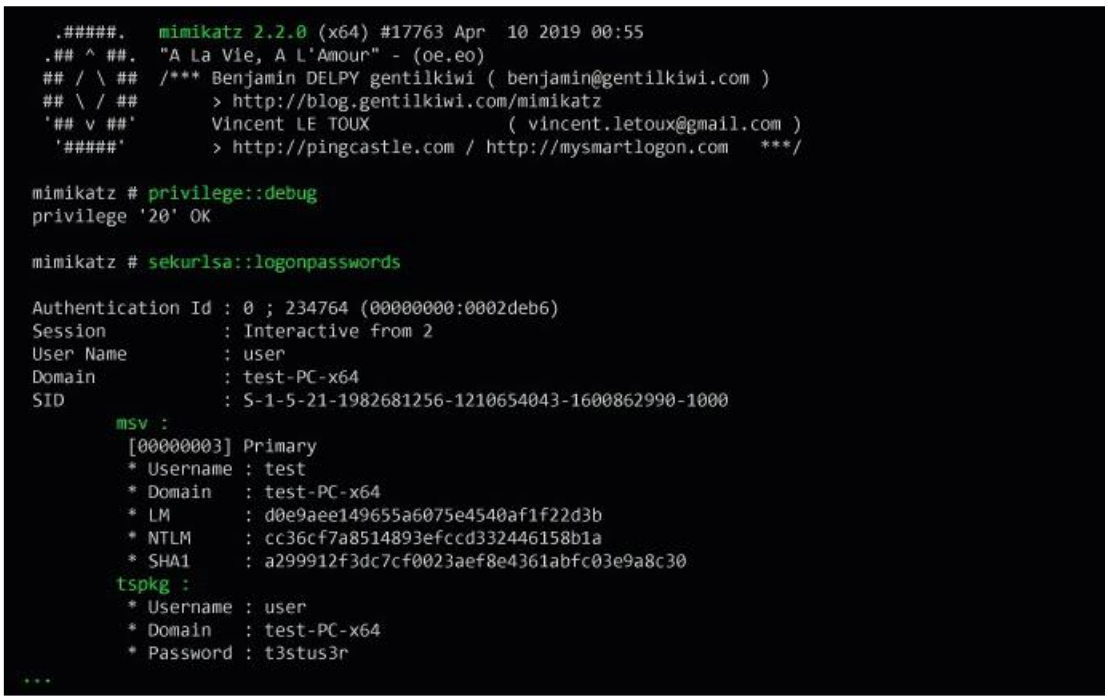


Figura 14.11 – Interfața de linie de comandă Mimikatz

Network sniffing

Instrumentele Network Sniffing sunt de obicei folosite pentru testarea de penetrare a infrastructurii, nu pentru penetrarea software-ului, dar poate fi util în înțelegerea vulnerabilităților din datele pe care software-ul utilizatorului le transmite prin rețea.

• **Aircrack-ng:** Aircrack-ng este o suită populară de instrumente utilizate pentru testarea și analiza securității a rețelelor fără fir. Este utilizat pe scară largă de profesioniștii în securitate, testerii de penetrare și rețele administratorilor să evalueze vulnerabilitățile rețelelor Wi-Fi și să conducă audituri de securitatea a rețelelor fără fir. Figura 14.12 arată o captură de ecran cu Aircrack-ng spargerea parolelor WiFi:

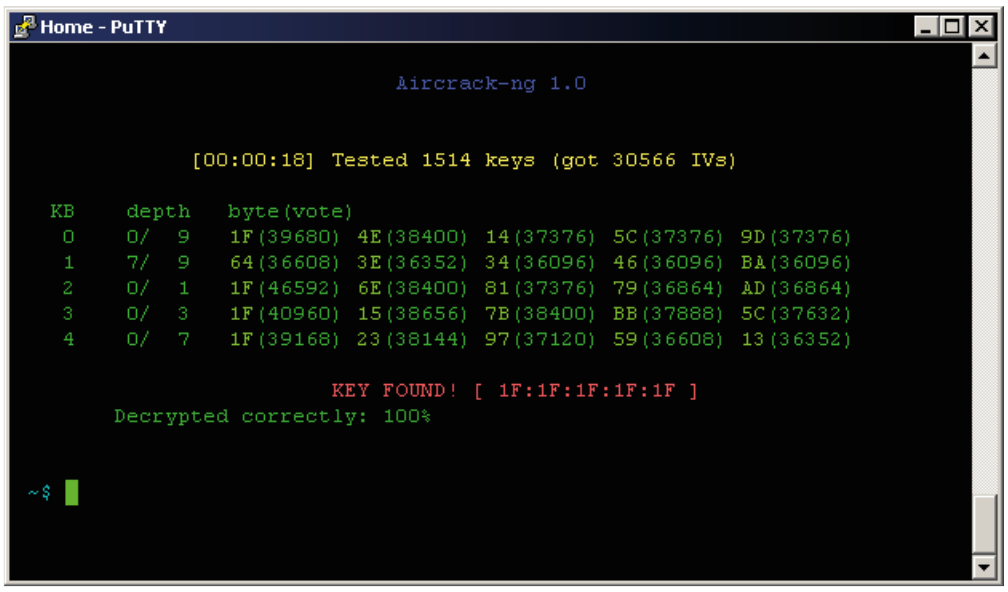


Figura 14.12 – Schimbarea parolei Aircrack-ng

• **Wireshark:** Wireshark este un puternic analizor de protocol de rețea și un instrument de captare a pachetelor folosit deadministratori de rețea, profesioniști în securitate, dezvoltatori și educatori pentru a captura, analiza,și depanați traficul de rețea în timp real. Permite utilizatorilor să inspecteze pachetele de date care circulă printr-o interfață de rețea sau salvat într-un fișier de captură de pachete. Wireshark este un instrument foarte bun de văzut ce date trimite aplicația dvs. și dacă pot fi descifrate. Figura 14.13 prezintă o capturăa traficului de rețea în Wireshark:

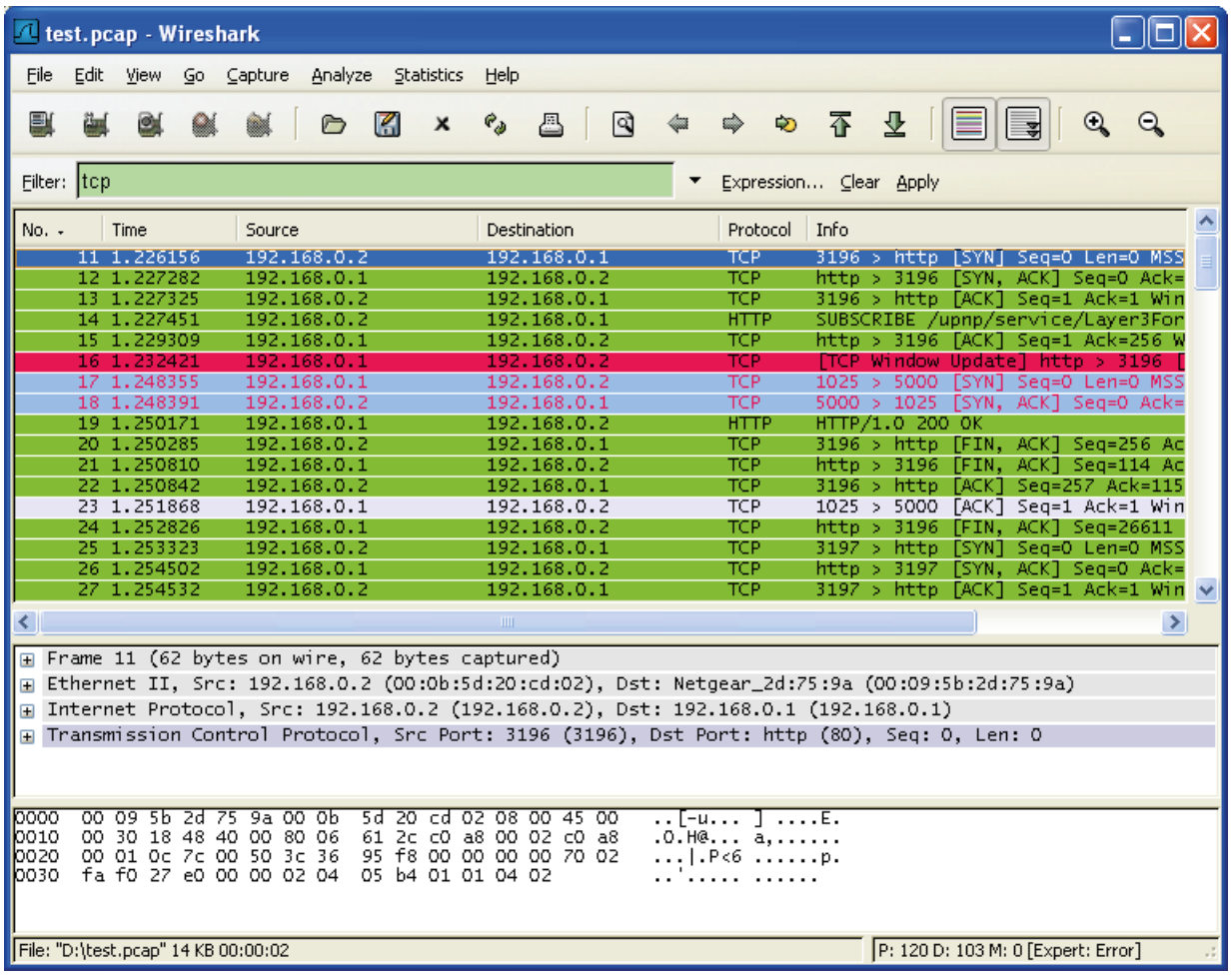


Figura 14.13 – Ecran de captură Wireshark

Forensics and monitoring *(Criminalistica si monitorizare)*

În testele de penetrare, criminalistica și monitorizarea joacă un rol esențial în înțelegerea impactului evaluări de securitate, identificarea riscurilor potențiale și asigurarea faptului că activitățile de testare a pătrunderii sunt condus în mod responsabil și etic:

• **Autopsie:** Autopsie este o platformă open source de criminalistică digitală utilizată de examinatorii criminaliști, agențiile de aplicare a legii și profesioniștii în securitate cibernetică să analizeze și să investigheze dovezile digitale colectate de pe computere, dispozitive de stocare și dispozitive mobile. Oferă un set cuprinzător de instrumente pentru efectuarea de analize criminalistice, inclusiv analiza sistemului de fișiere, căutarea prin cuvinte cheie, cronologie, analiza și examinarea artefactelor. Figura 14.14 prezintă o captură de ecran a managementului de caz ecran în Autopsie:

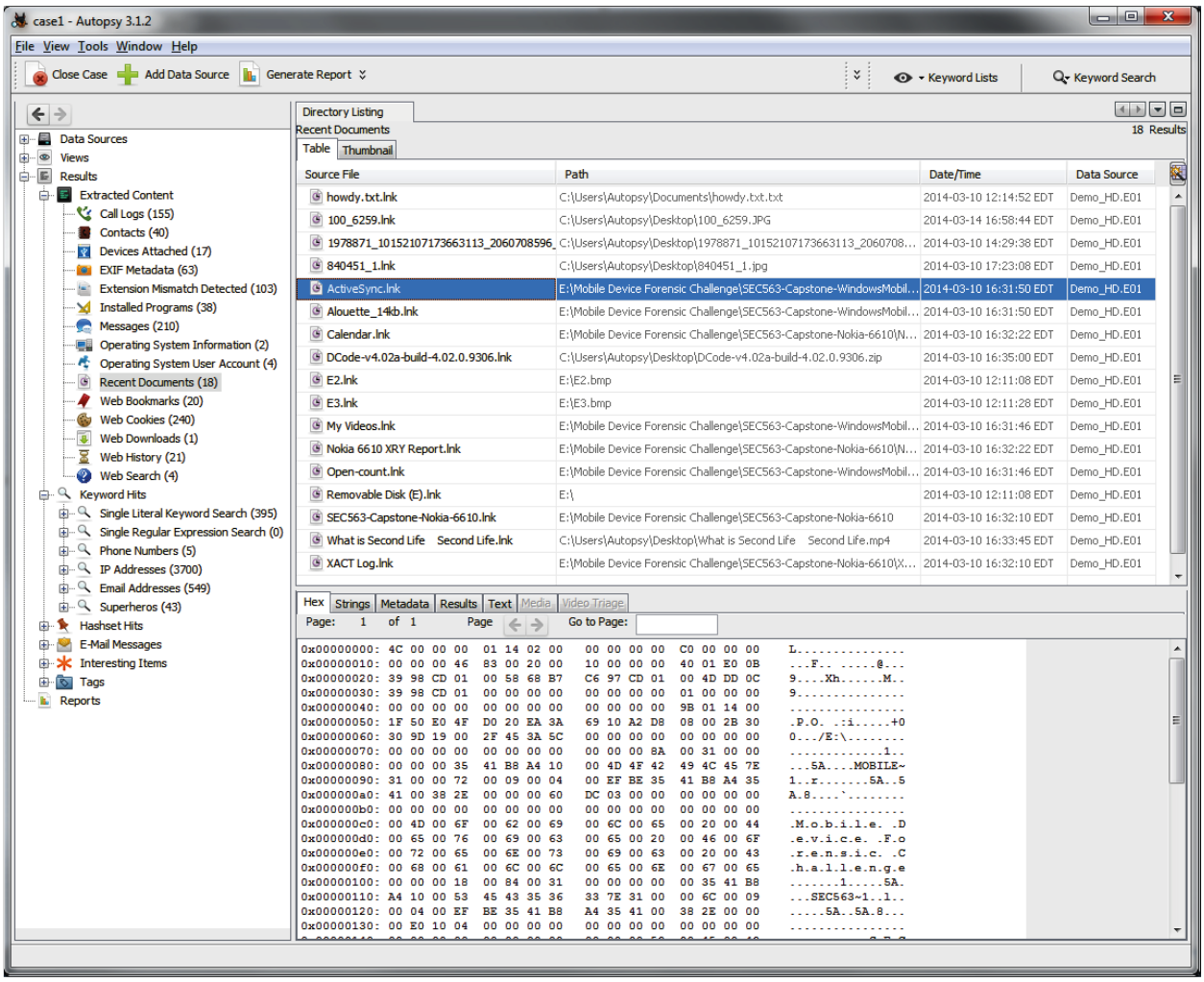


Figura 14.14 – Ecranul cazului de Autopsie

• **Snort:** Snort este un sistem open source de detectare și prevenire a intruziunilor în rețea (NIDS/NIPS) dezvoltat de Sourcefire, care este acum deținut de Cisco. Este utilizat pe scară largă de către profesioniștii în securitate, administratorii de rețea și organizațiile pentru a detecta și a preveni traficul de rețea rău intenționat și atacuri în timp real. Figura 14.15 arată o captură de ecran a unei alerte în Snort IDS:

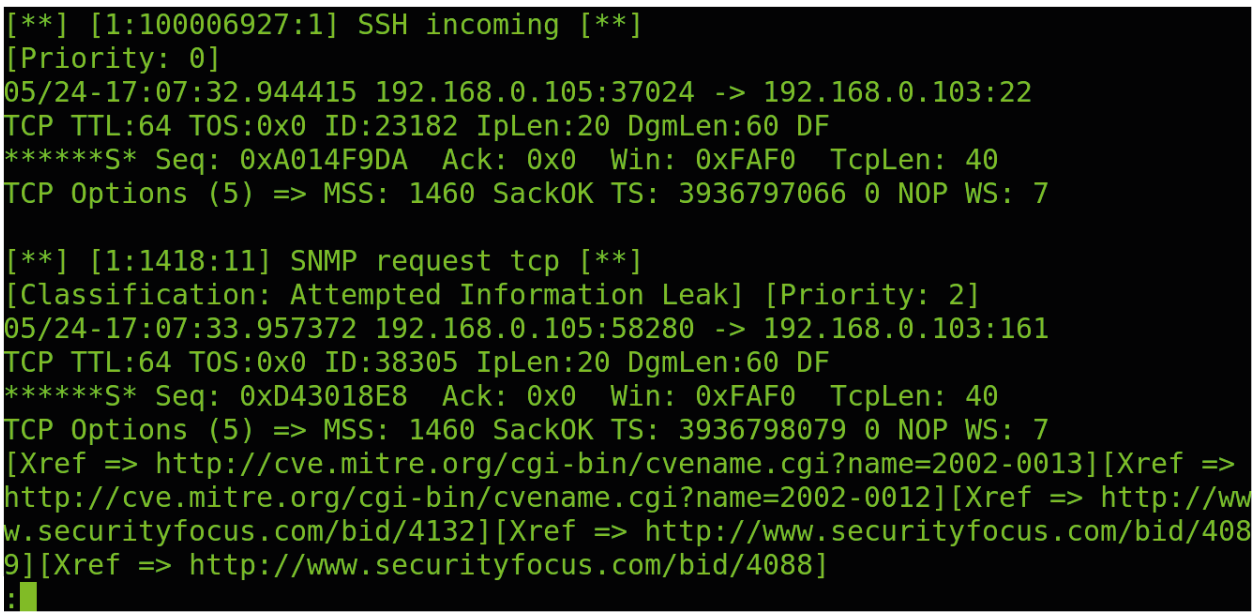


Figura 14.15 – Notificare de alertă Snort

Raportare și documentare

Raportarea și documentarea sunt aspecte cruciale ale testării de penetrare, deoarece comunică eficient constatările, vulnerabilitățile și recomandările către părțile interesate. Un raport bine scris și cuprinzător ajută organizațiile să își înțeleagă postura de securitate, să prioritizeze eforturile de remediere și să-și îmbunătățească rezistența la securitate.

• **Dradis framework**: este o colaborare și raportare open source platformă concepută pentru a facilita partajarea, integrarea și gestionarea informațiilor generate în timpul evaluărilor de securitate și a misiunilor de testare de penetrare. Ea eficientizează documentarea constatări, organizarea dovezilor și generarea de rapoarte cu aspect profesional, contribuind la securitate profesioniștii și echipele oferă rezultate mai eficiente și mai consistente. Figura 14.16 prezintă tabloul de bord al unui proiect în cadrul Dradis:

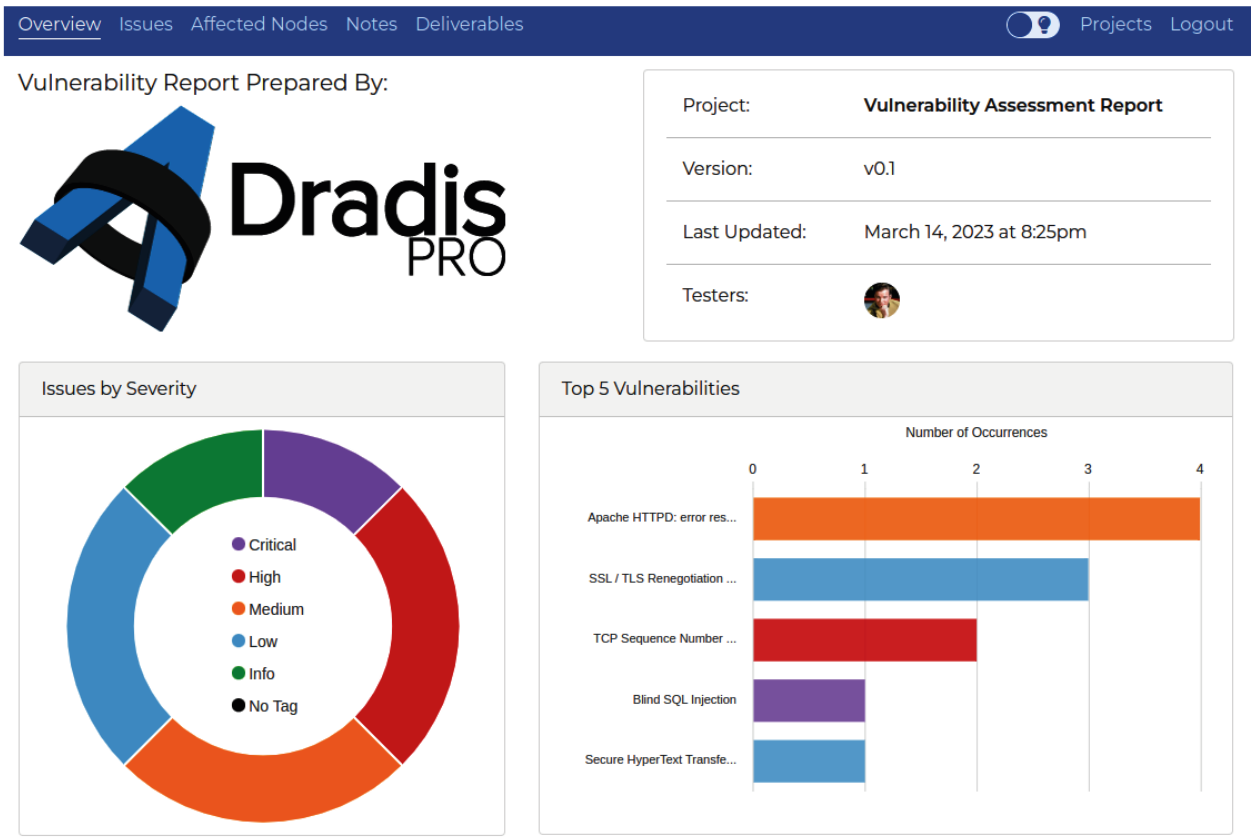


Figura 14.16 – Vedere tablou de bord Dradis

• **Faraday:** Faraday este o platformă de securitate colaborativă open source concepută pentru a facilita gestionarea informațiilor și a fluxurilor de lucru în timpul testelor de penetrare evaluarea vulnerabilității și procesele de răspuns la incident. Oferă un mediu centralizat pentru echipele de securitate, organizați constatările, urmăriți progresul și generați rapoarte în mod eficient. Figura 14.17 arată tabloul de bord al unui proiect în Faraday:

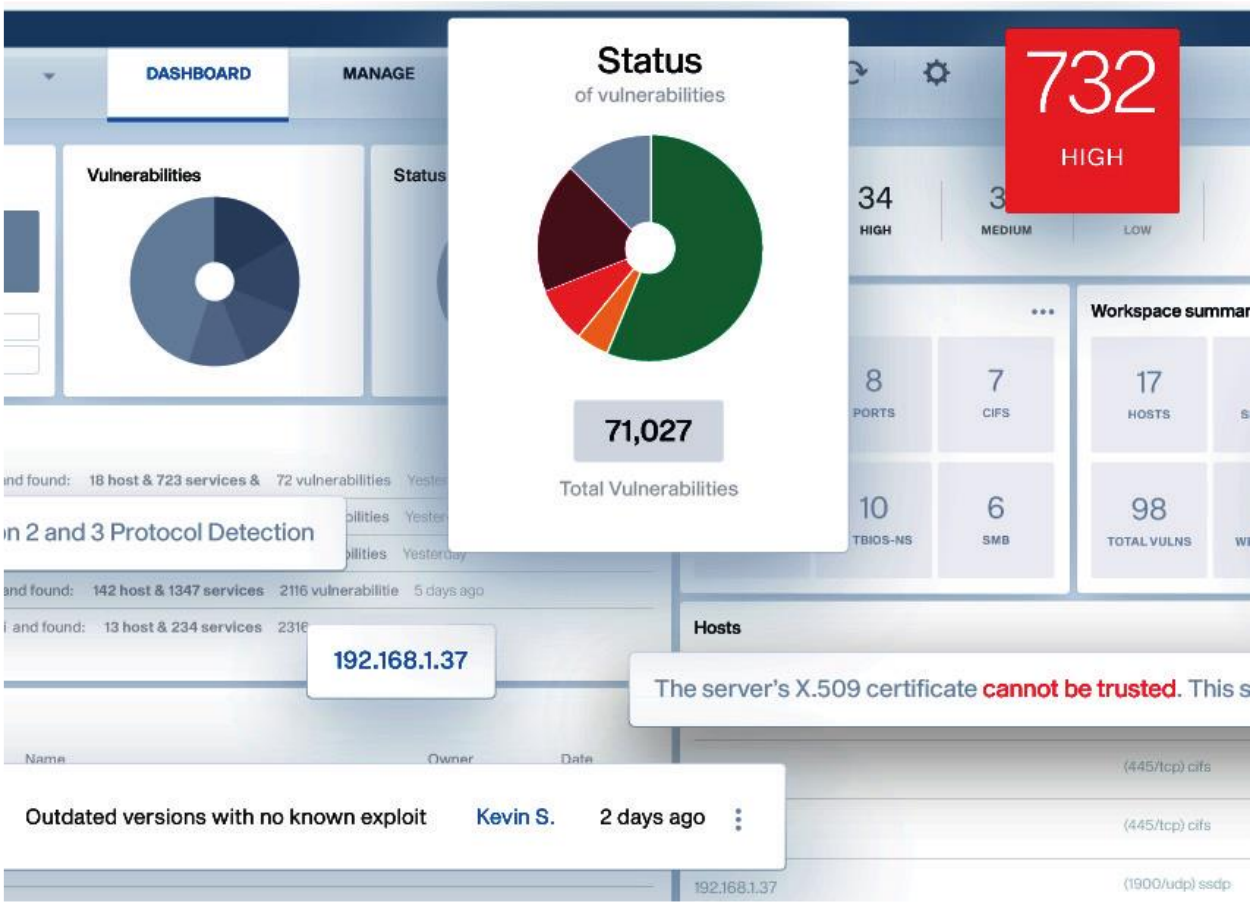


Figura 14.17 – Tabloul de bord Faraday

Aceste instrumente trebuie utilizate în mod responsabil și etic. Asigurați-vă întotdeauna că aveți autorizația corespunzătoare înainte de a desfășura activități de testare de penetrare și fiți conștienți de considerentele legale și etice.

În plus, eficacitatea testării de penetrare se bazează adesea pe o combinație de instrumente automate și testare manuală de către profesioniști calificați în securitate. Există distribuții Linux, cum ar fi Kali Linux, care includ multe instrumente de care are nevoie profesioniștii în securitate și pe care le-am menționat aici.

Un exemplu de raport de testare de penetrare a întreprinderii

Pe parcursul acestei cărți, vom construi un design sigur pentru un sistem de bilete pentru evenimente. Imaginează-ți un software sistem care permite unei casete de bilete sau unui site web să vândă bilete la un concert muzical celebru sau un eveniment de teatru.

Un raport simplificat de testare a probei de penetrare este detaliat în următoarele:

Rezumat la nivel înalt

Au fost analizate o stație de lucru, un server web și un server de baze de date. S-au constatat că stațiile de lucru sunt vulnerabil dacă a fost instalat malware. Programul malware poate modifica și citi apelurile API.

Serverul web a avut câteva vulnerabilități web comune, care sunt incluse în raportul Vega. The Recomandările sunt incluse în următoarea secțiune de analiză individuală a gazdei. Scanarea periodică a serverelor web și a bazelor de date este inclusă în recomandări.

Analiza gazdei

Această secțiune a raportului va documenta recunoașterea, analiza vulnerabilităților, exploatarea și recomandări pentru mașinile implicate în testarea de penetrare:

server web

Recunoaştere

• Instrument utilizat: NMap

• Adresă IP: 192.168.1.1

• Subrețea: 255.255.255.0

• Adresă MAC: 00-A5-B1-65-C2-32

• Sistem de operare: RHEL 9

• Alt software descoperit: Apache, PHP

Analiza vulnerabilitatii

• Unelte folosite: Vega

• Multe vulnerabilități au fost descoperite la utilizarea Vega; vezi lista atasata. Exemplele includ următoarele:

-XSS în ticketbasket.php

-Injectare SQL în ticketpayment.php

Exploatare

• Un atac XSS a fost dezvoltat și livrat împotriva paginii ticketbasket.php

Recomandări

• Remediați toate vulnerabilitățile atașate și executați o scanare Vega în fiecare noapte.

Server de baze de date

Recunoaştere

• Instrument utilizat: NMap

• Adresă IP: 192.168.1.121

• Subrețea: 255.255.255.0

• Adresă MAC: 00-E5-B1-23-A2-54

• Sistem de operare: RHEL 9

• Conectivitate API: API MySQL

• Alt software descoperit: MySQL 8.2

Analiza vulnerabilitatii

• Instrumente utilizate: SQLMap

• Potențială injecție SQL în procedura stocată: spLockSeats

Exploatare

• SQLMap a fost folosit pentru a încerca exploatarea fără succes.

Recomandări

• Remediați procedura stocată și executați o scanare Nmap pe un server de rezervă săptămânal.

Stație de lucru de birou

Recunoaştere

• Instrument utilizat: NMap

• Adresă IP: 192.168.1.120

• Subrețea: 255.255.255.0

• Adresă MAC: 00-F5-A1-63-D2-52

• Sistem de operare: Windows 11

• Conectivitate API: REST

• Alte software descoperite: MS Teams, MS RDP

Analiza vulnerabilității

• Instrumente folosite: Burp Suite

• Programele malware pot modifica solicitările HTTPS către API-ul REST

Exploatare

• A fost dezvoltat și implementat un malware care a mutat apelurile REST pentru a provoca în aplicație un refuz de serviciu

Recomandări

• Utilizați certificate client X.509

Rezumat

Acest capitol prezintă testarea de penetrare. Am analizat tipurile de teste de penetrare și fazele de testarea de penetrare. În continuare, ne-am uitat la instrumentele care sunt utile în diferitele faze de penetrare. Am încheiat cu un exemplu simplu de test de penetrare pentru aplicația noastră de exemplu pentru întreaga carte.

Întrebări de autoevaluare

1. Care dintre următoarele descrie cel mai bine testarea de penetrare a software-ului?

A. Verificarea conformității software-ului cu standardele și reglementările din industrie

B. Testarea software-ului pentru probleme de performanță și scalabilitate

C. Evaluarea interfeței de utilizator și a experienței utilizatorului unei piese software
D. Evaluarea securității unui software prin încercarea de a exploata vulnerabilitățile

2. Care este scopul principal al testării de penetrare?
A. Pentru a ne asigura că software-ul îndeplinește cerințele utilizatorului
B. Pentru a descoperi punctele slabe de securitate și vulnerabilități în software
C. Pentru a verifica funcționalitatea software-ului pe diferite platforme
D. Pentru a identifica toate erorile și defectele software-ului

3. Care dintre următoarele tipuri de teste de penetrare simulează un atac cibernetic în lumea reală încercând să exploateze vulnerabilitățile fără cunoștințe prealabile despre sistem?
A. Testarea cutiei gri
B. Fuzz testarea
C. Testarea cutiei albe
D. Testarea cutiei negre

4. Ce fază a testării de penetrare implică planificarea, culegerea de informații despre țintă sistem și definirea domeniului și obiectivelor testului?
A. Raportare
B. Recunoaștere
C. Analiza
D. Exploatarea

5. Care este diferența dintre scanarea vulnerabilităților și testarea de penetrare?
A. Scanarea vulnerabilităților identifică punctele slabe de securitate, în timp ce testarea de penetrare le exploatează
B. Scanarea vulnerabilităților încearcă să ocolească controalele de securitate, în timp ce testarea de penetrare identifică configurațiile greșite
C. Scanarea vulnerabilităților este automatizată, în timp ce testarea de penetrare necesită intervenție manuală
D. Scanarea vulnerabilităților se concentrează pe funcționalitatea software-ului, în timp ce testarea de penetrare se concentrează pe experiența utilizatorului

Răspunsuri
1. D 2. B 3. D 4. B 5. A

T6. Managementul Vulnerabilităților. Identificarea, clasificarea și tratamentul vulnerabilităților. Instrumente și tehnici de management al patch-urilor

Această prelegere abordează procesele și tehnicile de **identificare, clasificare și tratament al vulnerabilităților**, precum și instrumentele și metodele folosite în managementul patch-urilor. Vom prezenta concepte teoretice și exemple practice, pentru a ilustra cum se poate implementa un program de management al vulnerabilităților într-o organizație.

---

1. Introducere

1.1 Ce este managementul vulnerabilităților?

- **Definiție:**
  Managementul vulnerabilităților este un proces continuu de identificare, evaluare, clasificare și remediere a vulnerabilităților din sistemele IT. Scopul său este de a reduce riscurile și de a preveni atacurile cibernetice.
- **Importanță:**
  - Reducerea riscului de exploatare a vulnerabilităților.
  - Protejarea datelor și infrastructurii.
  - Menținerea conformității cu standarde și reglementări (ex.: ISO 27001, GDPR).
  - Creșterea rezilienței sistemelor în fața amenințărilor.

1.2 Etapele procesului de management al vulnerabilităților

1. **Identificarea vulnerabilităților:** Scanări periodice, monitorizare continuă, raportare de la utilizatori.
2. **Clasificarea și prioritizarea:** Evaluarea severității, impactului și probabilității exploatării.
3. **Tratamentul vulnerabilităților:** Aplicarea patch-urilor, remedierea codului, compensarea riscurilor.
4. **Monitorizarea și raportarea:** Verificarea remediilor aplicate, audituri periodice.

---

2. Identificarea și Clasificarea Vulnerabilităților

2.1 Identificarea Vulnerabilităților

- **Scanare automată:**
  Folosirea instrumentelor de scanare a vulnerabilităților (ex.: **Nessus**, **OpenVAS**, **Qualys**) pentru a evalua rețeaua, aplicațiile și sistemele.

- **Evaluări manuale:**
  Revizuiri de cod (code reviews) și teste de penetrare (penetration testing) pentru a identifica vulnerabilități ce pot fi omise de scanerele automate.
- **Surse de informații:**
  Baze de date de vulnerabilități precum **CVE (Common Vulnerabilities and Exposures)** și rapoarte de la CERT-uri.

2.2 Clasificarea Vulnerabilităților

- **Sistem de clasificare:**
  Se utilizează standarde precum **CVSS (Common Vulnerability Scoring System)** care evaluează severitatea pe baza:
  - **Impact:** Confidențialitate, integritate, disponibilitate.
  - **Exploitabilitate:** Complexitatea exploatării, accesul necesar.
- **Prioritizarea:**
  Vulnerabilitățile sunt clasificate în funcție de scorul CVSS și de impactul asupra afacerii. Vulnerabilitățile critice trebuie tratate imediat, în timp ce cele de nivel scăzut pot fi programate în ciclul de mentenanță.

Exemplu practic:

- Un scaner de vulnerabilități descoperă o vulnerabilitate CVE-2021-44228 (Log4Shell) într-o aplicație web.
- Utilizând CVSS, se determină un scor de 10.0, indicând o vulnerabilitate critică.
- Se prioritizează tratamentul imediat și se emite o alertă către echipa de dezvoltare și securitate.

---

3. Tratamentul Vulnerabilităților și Managementul Patch-urilor

3.1 Tratamentul Vulnerabilităților

- **Patch Management:**
  Aplicarea patch-urilor oferite de furnizori sau dezvoltatori pentru a remedia vulnerabilitățile identificate.
- **Remediere prin configurare:**
  Schimbarea configurațiilor sistemelor pentru a reduce expunerea (de exemplu, dezactivarea serviciilor nefolosite).
- **Implementare de măsuri compensatorii:**
  Utilizarea de firewall-uri, IDS/IPS, sisteme de monitorizare pentru a detecta și preveni exploatarea vulnerabilităților.

- **Dezvoltare de cod sigur:**
  Revizuirea și refactorizarea codului vulnerabil pentru a elimina erorile de programare (ex.: înlocuirea funcțiilor nesigure precum strcpy cu variante sigure precum strncpy).

3.2 Managementul Patch-urilor

- **Ce este managementul patch-urilor?**
  Procesul de identificare, testare, implementare și verificare a patch-urilor pentru sisteme și aplicații.
- **Pași în managementul patch-urilor:**
  1. **Identificarea:** Colectarea informațiilor despre patch-uri disponibile din surse oficiale.
  2. **Evaluarea:** Testarea patch-urilor într-un mediu de testare pentru a evalua impactul.
  3. **Implementarea:** Distribuirea și instalarea patch-urilor în sistemele de producție.
  4. **Verificarea:** Auditarea post-implementare pentru a confirma că vulnerabilitățile sunt remediate.
- **Instrumente de management al patch-urilor:**
  1. **WSUS (Windows Server Update Services):** Pentru sistemele Windows.
  2. **SCCM (System Center Configuration Manager):** Pentru medii enterprise.
  3. **Red Hat Satellite:** Pentru sistemele Linux.
  4. **Automatizarea prin scripturi Bash sau PowerShell** – Pentru patch-uri custom și actualizări automate.

Exemplu practic:

- **Scenariu:** O companie folosește o aplicație internă dezvoltată în-house, iar echipa de securitate identifică o vulnerabilitate critică în modulul de autentificare.
- **Acțiune:**
  - Echipa de dezvoltare pregătește un patch care remediază vulnerabilitatea.
  - Patch-ul este testat într-un mediu de staging pentru a se asigura că nu afectează alte funcționalități.
  - Patch-ul este apoi implementat utilizând un sistem de management al patch-urilor (de exemplu, SCCM).
  - Un audit post-implementare confirmă că vulnerabilitatea a fost eliminată.

---

4. Instrumente și Tehnici Practice

4.1 Instrumente de Scanare și Audit

- **Nessus/OpenVAS/Qualys:**

    - Scanează rețele, servere și aplicații pentru a identifica vulnerabilități.
    - **Exemplu:** O scanare Nessus identifică vulnerabilități de tip outdated software și configurări incorecte.
- **CVE Database:**
    - Se consultă pentru a obține detalii despre vulnerabilități și patch-uri disponibile.
- **SIEM:**
    - Monitorizează evenimentele de securitate și generează rapoarte pentru management.

Ce este un SIEM?

- **Security Information and Event Management (SIEM):**
  Un SIEM colectează, agregă și corelează date din diverse surse (loguri, alerte de la scanere, evenimente de rețea etc.) pentru a oferi o vedere centralizată asupra stării de securitate a unei organizații.
- **Analiză și alertare:** SIEM-ul analizează datele în timp real, detectează comportamente anormale și generează alerte pentru echipele de securitate.

Integrarea Scannerului de Vulnerabilități cu SIEM

Prin integrarea rezultatelor unui scanner de vulnerabilități cu un sistem SIEM se obține următoarea funcționalitate:

- **Centralizare:** Rapoartele de vulnerabilități sunt colectate și stocate împreună cu alte date de securitate (ex.: loguri de acces, evenimente de rețea).
- **Corelare:** SIEM-ul poate corela vulnerabilitățile descoperite cu evenimentele de securitate, permițând identificarea mai rapidă a atacurilor sau a riscurilor critice.
- **Alertare și raportare:** Dacă un scanner raportează o vulnerabilitate critică, SIEM-ul poate genera alerte imediate, facilitând răspunsul prompt din partea echipei de securitate.

Exemplu practic de integrare

1. **Scenariu:**
    - O organizație rulează un scanner de vulnerabilități (ex.: Nessus) care detectează o vulnerabilitate critică într-un server web.
2. **Colectarea datelor:**
    - Rezultatele scanării sunt exportate (de obicei în format XML, CSV sau JSON).
3. **Integrarea cu SIEM:**

- Platforma SIEM (de exemplu, Splunk, ELK Stack sau QRadar) este configurată să preia și să indexeze aceste rezultate.
- Prin corelare, SIEM-ul poate asocia datele din scanare cu loguri de acces neobișnuite sau alte evenimente de securitate.

4. **Alertare:**
   - Dacă vulnerabilitatea are un scor CVSS foarte mare și, de exemplu, se observă trafic suspect în loguri, SIEM-ul generează o alertă pentru investigație.
5. **Raportare:**
   - Echipa de securitate primește un raport detaliat, care include atât rezultatele scanării, cât și evenimentele corelate din rețea.

Concluzie

Integrarea rezultatelor unui scanner de vulnerabilități într-o platformă SIEM permite:

- O monitorizare centralizată a stării de securitate.
- Corelarea informațiilor despre vulnerabilități cu alte evenimente de securitate.
- Răspuns rapid la riscurile identificate prin alerte și rapoarte detaliate.

4.2 Tehnici de Testare și Validare

- **Testare de penetrare (Penetration Testing):**
  - Se efectuează simulări de atac pentru a verifica eficiența măsurilor de securitate.
  - **Exemplu:** Folosirea Metasploit și Burp Suite pentru a testa vulnerabilități SQL Injection.
- **Audit de cod (Code Review):**
  - Se revizuiește manual codul sursă pentru a identifica erori de programare și vulnerabilități.
- **Automatizarea în pipeline-uri CI/CD:**
  - Integrarea unor instrumente de analiză statică (SAST) și dinamică (DAST) pentru a detecta vulnerabilitățile în mod continuu.

---

5. Concluzii și Recomandări

- **Managementul vulnerabilităților** este un proces continuu ce implică identificare, clasificare, tratament și verificare.
- **Patch managementul** eficient asigură că vulnerabilitățile cunoscute sunt remediate rapid, reducând riscul de exploatare.

- Implementarea unui **program robust de management al vulnerabilităților** necesită colaborarea între echipele de dezvoltare, operațiuni și securitate.
- **Instrumentele automatizate** (scanere de vulnerabilități, SIEM, SAST/DAST) ajută la detectarea și remedierea rapidă a vulnerabilităților, dar trebuie completate cu evaluări manuale și teste de penetrare periodice.

Exerciții practice sugerate pentru studenți

1. **Scanare de vulnerabilități:**
    - Utilizați un instrument de scanare (ex.: OpenVAS) pe un mediu de test pentru a identifica vulnerabilități ale sistemului și clasificați-le folosind CVSS.
2. **Test de penetrare:**
    - Efectuați un test de penetrare asupra unei aplicații demo, identificând punctele slabe, apoi documentați procesul de remediere.
3. **Implementarea patch-urilor:**
    - Creați un script simplu (de exemplu, cu PowerShell sau Bash) pentru a automatiza verificarea și instalarea patch-urilor pe un sistem de test, apoi evaluați rezultatele.