**Weblogs to Detect attacks**

Securitatea site-urilor web este un criteriu foarte important in navigarea pe Internet, de aceea detectarea unor atacuri cibernetice ne ajuta sa navigam mult mai sigur pe Internet. In proiectul nostru am incercat sa detectam diferite atacuri pe baza exclusiva a log-urilor dintr-un server Apache.

**LOGS**

**Log-**urile sunt inregistrari ale evenimentelor unei aplicatii in timp ce ea este rulata. Acestea pot contine informatii despre erori interne sau ofera informatii despre cum a fost utilizata aplicatia si in ce scop. Serverul pe care noi il folosim este un server Apache HTTP si acesta are doua tipuri de log-uri, Access.log si Error.log.

**Error.log** ofera cele mai utile si importante informatii, iar acesta este locul in care ar trebui sa ne uitam cand este semnalata o problema. Acest tip de log poate contine mai multe informatii precum: timestamp-ul, in ce mod a aparut eroarea, gradul de severitate al erorii, clientul care a trimis cererea catre server si descrierea erorii.

Un exemplu de un astfel de log este prezentat mai jos:

[Fri Feb 25 19:06:39 2022] [error] [client 192.168.56.1] PHP Warning: SimpleXMLElement::xpath(): xmlXPathEval: evaluation failed in /owaspbwa/bwapp-git/bWAPP/xmli\_1.php on line 78, referer: <http://192.168.56.101/bWAPP/xmli_1.php>

Cu toate ca acest tip de log-uri contine foarte multe informatii utile, in studiul nostru ne-am axat mai multe pe celalalt tip de log, anume **Access.log**. Aceste tipuri de log-uri contin informatii doar despre erori, acest fapt ne-ar putea ajuta in cazul in care vrem sa si rezolvam eroarea, nu doar sa o detectam.

**Access.log** stocheaza informatii despre evenimentele ce au avut loc pe serverul web Apache. De exemplu, atunci cand cineva viziteaza site-ul web, un log este inregistrat si stocat pentru a furniza administratorului serverului web informatii precum: IP-ul utilizatorului, ce pagini a vizualizat, status-urile, browser-ul utilizat, etc.. .

Pentru utilizatorii de Apache, access.log este localizat in aceeasi locatie. De obicei putem gasi fisierul folosind urmatoarele comenzi in terminal :

1. /var/log/apache/access.log
2. /var/log/apache2/access.log

Exista 2 tipuri de access.log, Common log si Combined log, acestea se diferentiaza prin format.

Acum sa incepem sa intelegem informatiile din *access.log*. La prima vedere, pentru un necunoscator totul pare foarte complicat, dar lucrurile stau in felul urmator:

* Formatul fiecarui log, in functie de tipul acestuia, este:

In cazul Common log: *LogFormat "%h %l %u %t \"%r\" %>s %O*. Un exemplu bun este:

127.0.0.1 - Scott [10/Dec/2019:13:55:36 -0700] "GET /server-status HTTP/1.1" 200 2326

* + **127.0.0.1** - adresa IP a clientului care a facut cererea
  + **-** - identitatea clientului. Acest camp este adesea returnat cu cratima
  + **Scott** – userid-ul persoanei care a facut cererea
  + **[10/Dec/2019:13:55:36 –0700]** - data si ora la care a fost facuta cererea
  + **“GET/server-status HTTP/1.1”** - tipul request-ului si resursa solicitata
  + **200** – statusul intors de server
  + **2326** – marimea obiectului returnat

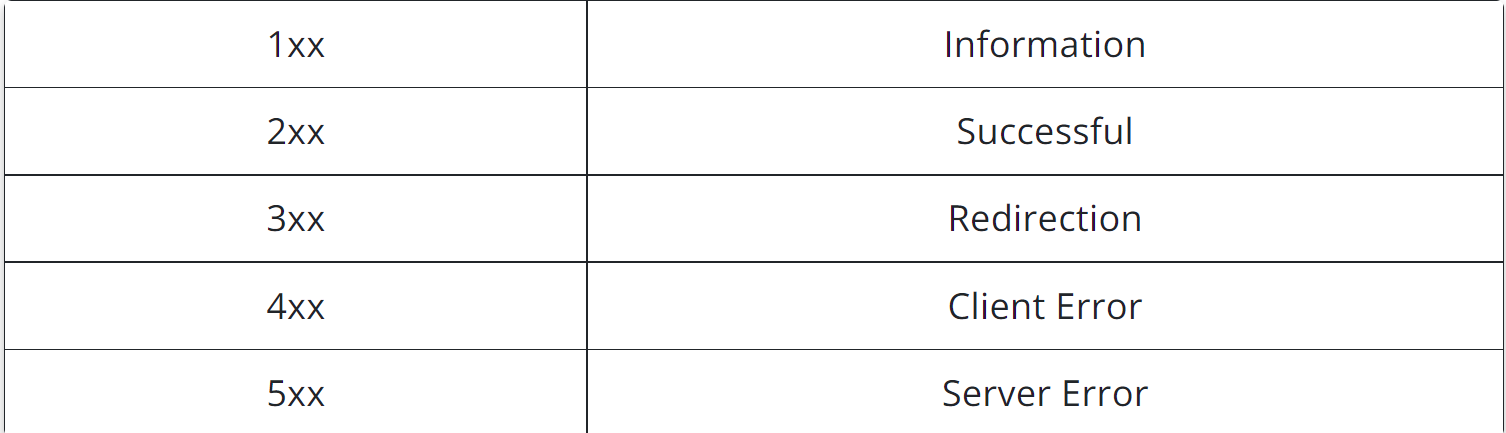
In cazul Combined Log: *LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\" .* Un exemplu bun este:

127.0.0.1 - Scott [10/Dec/2019:13:55:36 -0700] "GET /server-status HTTP/1.1" 200 2326 "<http://localhost/>" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36"

* + [**http://localhost/**](http://localhost/) **-** HTTP referer, care reprezinta adresa de la care provine cererea
  + **"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36" -** User Agent, care contine informatii despre browser-ul pe care clientul il foloseste.

Inainte ca un hacker să poată pătrunde cu succes site-ul web, el/ea ar fi făcut probabil câteva încercări nereușite. Aceste încercări, dacă nu sunt blocate de WAF, ar fi disponibile ca intrări neobișnuite în log-urile serverului web. De asemenea, în funcționarea normală a aplicațiilor web, utilizatorii obișnuiți ar folosi anumite adrese URL, fac un anumit tip de solicitări, etc.. . Acest comportament normal ar duce la anumite log-uri în access.log. Administratorii de securitate care operează site-ul ar trebui să fie familiarizați cu log-uri normale ale serverului web, corespunzătoare utilizării normale a aplicațiilor lor web. Astfel, atunci când apar intrări neobișnuite în access logs la serverul web, acestea reprezintă anomalii.

Pe lângă aceste cuvinte cheie, este foarte important să aveți cunoștințe de bază despre codurile de stare HTTP în timpul unei analize.Mai jos este tabelul care arată informații de nivel înalt despre codurile de stare HTTP:



Când există o cantitate mare log-uri, este dificil să se efectueze inspecția manuală. În astfel de scenarii, putem alege tool-uri automate împreună cu unele inspecții manuale. Un astfel de tool am incercat si noi sa construim pentru a detecta anumite injections.

Deși există multe instrumente comerciale eficiente, cel mai bun tool gratuit este cunoscut sub numele de Scalp. Potrivit link-ului lor oficial, „Scalp este un analizor de jurnal pentru serverul web Apache, care își propune să caute probleme de securitate. Ideea principală este să căutați fișiere log uriașe și să extrageți posibilele atacuri care au fost trimise prin HTTP/GET.”

Log-urile conțin doar un set parțial din traficul complet trecând prin rețea. În funcție de aplicația care scrie log-urile, aceasta poate fi o pistă de audit completă sau doar câteva date. Va vom arată cele mai semnificative diferențe între analiza fișierelor jurnal și a traficului complet:

|  |  |  |
| --- | --- | --- |
|  | Avantaje | Dezavantaje |
| Logs | datele sunt ușor disponibile de analizat (în fisiere) | jurnalele conțin de obicei doar o fracțiune din informatii |
| Full Traffic | toate informatiile pot fi analizate | ar putea fi dificil să captam datele (criptate trafic, trafic mare încărcătură etc.) |

Cel mai mare beneficiu al log-urilor este disponibilitatea rapida și analiza conținutului acestora mult mai usor.

**ATACKS**

**Injection Attacks** se referă la o clasă largă de vectori de atac. Într-un atac prin injecție, un atacator furnizează intrări nesigure unui program. Această intrare este procesată de un interpret ca parte a unei comenzi sau a unei interogări. La rândul său, acest lucru modifică execuția acelui program.

Injecțiile sunt printre cele mai vechi și mai periculoase atacuri care vizează aplicațiile web. Acestea pot duce la furtul de date, la pierderea datelor, la pierderea integrității datelor, la refuzul serviciului, precum și la compromiterea completă a sistemului. Motivul principal pentru vulnerabilitățile de injectare este, de obicei, validarea insuficientă a intrărilor utilizatorului.

Acest tip de atac este considerat o problemă majoră în securitatea web. Este listat ca riscul numărul unu pentru securitatea aplicațiilor web în Top 10 OWASP – și pentru un motiv întemeiat. Atacurile prin injectare, în special injecțiile SQL (atacuri SQLi) și Cross-site Scripting (XSS), sunt nu numai foarte periculoase, ci și răspândite, mai ales în aplicațiile vechi.

Ceea ce face vulnerabilitățile de injectare deosebit de înfricoșătoare este faptul că suprafața de atac este enormă (în special pentru vulnerabilitățile XSS și SQL Injection). În plus, atacurile prin injecție sunt o clasă de vulnerabilitate foarte bine înțeleasă. Aceasta înseamnă că există multe instrumente disponibile gratuit și de încredere care permit chiar și atacatorilor fără experiență să abuzeze de aceste vulnerabilități în mod automat.

**O mica lista cu tipuri de Injection:**

* ***Code Injection*** - Atacatorul injectează codul aplicației scris în limba aplicației. Acest cod poate fi folosit pentru a executa comenzi ale sistemului de operare cu privilegiile utilizatorului care rulează aplicația web. În cazuri avansate, atacatorul poate exploata vulnerabilități suplimentare de escaladare a privilegiilor, ceea ce poate duce la compromiterea completă a serverului web.
* ***CRLF Injection*** - Atacatorul injectează o secvență de caractere CRLF (Carriage Return și Line Feed) neașteptată. Această secvență este folosită pentru a împărți un antet de răspuns HTTP și pentru a scrie conținut arbitrar în corpul răspunsului. Acest atac poate fi combinat cu Cross-site Scripting (XSS).
* ***Cross-site Scripting (XSS)*** - Atacatorul injectează un script arbitrar (de obicei în JavaScript) într-un site web sau aplicație web legitimă. Acest script este apoi executat în browserul victimei.
* ***Email Header Injection*** - Acest atac este foarte asemănător cu injecțiile CRLF. Atacatorul trimite comenzi IMAP/SMTP către un server de e-mail care nu este direct disponibil printr-o aplicație web.
* ***Host Header Injection*** - Atacatorul abuzează de încrederea implicită a antetului HTTP Host pentru a otrăvi funcționalitatea de resetare a parolei și cache-urile web.
* ***LDAP Injection*** - Atacatorul injectează instrucțiuni LDAP (Lightweight Directory Access Protocol) pentru a executa comenzi LDAP arbitrare. Ei pot obține permisiuni și pot modifica conținutul arborelui LDAP.
* ***OS Command Injection*** - Atacatorul injectează comenzile sistemului de operare cu privilegiile utilizatorului care rulează aplicația web. În cazuri avansate, atacatorul poate exploata vulnerabilități suplimentare de escaladare a privilegiilor, ceea ce poate duce la compromiterea completă a sistemului.
* ***SQL Injection*** - Atacatorul injectează instrucțiuni SQL care pot citi sau modifica datele bazei de date. În cazul atacurilor avansate cu injecție SQL, atacatorul poate folosi comenzi SQL pentru a scrie fișiere arbitrare pe server și chiar pentru a executa comenzi ale sistemului de operare. Acest lucru poate duce la compromiterea completă a sistemului.
* ***XPATH Injection*** - Atacatorul injectează date într-o aplicație pentru a executa interogări XPath create. Le pot folosi pentru a accesa date neautorizate și pentru a ocoli autentificarea.

In proiectul nostru am inspectat o parte din aceste tipuri de Injection, dar si alte tipuri.

**SQL INJECTION**

Un atac de tip SQL injection constă în inserarea sau „injectarea” unei interogări SQL prin datele de intrare de la client în aplicație. O exploatare de succes a injecției SQL poate citi date sensibile din baza de date, poate modifica datele bazei de date (Inserare/Actualizare/Ștergere), executa operațiuni de administrare în baza de date (cum ar fi închiderea DBMS), recupera conținutul unui anumit fișier prezent în fișierul DBMS file system și, în unele cazuri, emite comenzi către sistemul de operare. Atacurile de tip SQL injection sunt un tip de atac de injecție, în care comenzile SQL sunt injectate pentru a afecta execuția comenzilor SQL predefinite.

SQL Injection a devenit o problemă comună cu site-urile web bazate pe baze de date. Defectul este ușor de detectat și exploatat cu ușurință și, ca atare, orice site sau pachet de software, chiar și cu o bază minimă de utilizatori, este susceptibil de a fi supus unei tentative de atac de acest fel. În esență, atacul este realizat prin plasarea unui meta caracter în intrarea datelor pentru a plasa apoi comenzi SQL în planul de control, care nu existau acolo înainte. Acest defect depinde de faptul că SQL nu face o distincție reală între planul de control și cel de date.

SQL Injection are loc de obicei atunci când cereți intrarea unui utilizator, cum ar fi numele de utilizator/id-ul de utilizator, iar în loc de nume/id, utilizatorul vă oferă o instrucțiune SQL pe care o veți rula fără să știți în baza de date.

**Exemplu de SQL Injection:**

Urmatorul exemplu este foarte simplu. Acesta arată cum un atacator poate folosi o vulnerabilitate SQL Injection pentru a ocoli securitatea aplicației și a se autentifica ca administrator.

Următorul script este un pseudocod executat pe un server web. Este un exemplu simplu de autentificare cu un *username* și un *password*. Exemplul de bază de date are un tabel numit *users* cu următoarele coloane: username si password.

*# Define POST variables*  
*uname = request.POST['username']*  
*passwd = request.POST['password']*  
  
*# SQL query vulnerable to SQLi*  
*sql = “SELECT id FROM users WHERE username=’” + uname + “’ AND password=’” + passwd + “’”*  
*# Execute the SQL statement*  
*database.execute(sql)*

Aceste câmpuri de intrare sunt vulnerabile la injecția SQL. Un atacator ar putea folosi comenzi SQL în intrare într-un mod care ar modifica instrucțiunea SQL executată de serverul bazei de date. De exemplu, ar putea folosi un truc și ar putea seta câmpul passwd la: *password' OR 1=1*

Ca rezultat, serverul bazei de date rulează următoarea interogare SQL:

***SELECT*** *id* ***FROM*** *users* ***WHERE*** *username='username'* ***AND******password****=****'password' OR 1=1****'*

Din cauza instrucțiunii *OR 1=1*, clauza *WHERE* returnează primul ID din tabelul utilizatorilor, indiferent de numele de utilizator și parola. Primul ID de utilizator dintr-o bază de date este de obicei administratorul. În acest fel, atacatorul nu numai că ocolește autentificarea, dar câștigă și privilegii de administrator.

**Detectarea SQL Injection in aceess.log:**

Un exemplu de SQL Injection este urmatorul:

*192*.*168*.*56*.*1* - - [*24*/Feb/*2022*:15:48:57 -*0500*] "GET /dvwa/vulnerabilities/sqli/?id=select&Submit=Submit HTTP/1.1" *200 1343* "<http://192.168.56.101/dvwa/vulnerabilities/sqli/?id=%27+union+select+if%281%3D1%2C+%27true%27%2C+%27false%27%29%2C+2+%23&Submit=Submit>" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36"

Putem observa cu usurinta ca request-ul are ceva diferit de un access.log obisnuit. Vom detecta un posibil SQL Injection cautand exclusiv prin request.

Metoda prin care incercam sa detectam un posibil SQL Injection este urmatoarea:

* Ideea functiei SQLi() este sa trecem logul nostru "suspect" prin mai multe teste. Testele sunt in ordinea importantei, incepand cu cel mai simplu caz prin care putem detecta un SQL injection si ajungand la ultimul test unde, pana in acel punct, suspectul deja a fost transformat din format URL la un text simplu ce contine doar cuvinte separate printr-un singur spatiu. Practic, primele teste sunt cazurile cele mai simple si rapide, ultimul test cel decisiv.
* Prima data, inainte de a trece prin orice test, vom decoda URL-ul obtinut din lista cu toate log-urile astfel incat sa avem un text ce nu are coduri unicode in loc de caractere ASCII.
* Dupa asta, obtinem partea din URL ce contine va contine posibilul injection, realizam acest lucru printr-un split in functie de "?". (injection-ul este bucata din dreapta lui "?").
* Testul "0" ar fi cazul cel mai de baza, anume atunci cand nu avem niciun input si deci nu putem avea niciun injection.
* Primul test adevarat este detectarea unui numar impar de caractere malitioase. Alegem un numar impar, intrucat daca avem un numar par se va genera cel mult un SQL error, nu un injection.
* Al doilea test este in cazul in care avem un "Comment Injection", pentru asta incercam sa detectam daca avem un "or" combinat cu un comentariu, daca da, atunci am detectat un posibil injection.
* Al treilea test va fi practic sa scanam si sa stergem toate caracterele specifice unui URL din suspectul nostru.
* Al patrulea test este sa detectam "Inline Comments Injection", pentru asta cautam simbolurile "/\*" si "\*/" care marcheaza un comentariu, dupa ce le detectam, stergem tot ce e intre ele. La acest tip de injection nu putem stii cu siguranta daca a fost doar o coincidenta ca cineva sa fi pus fix combinatia asta la parola, asa ca nu il adaugam la lista injection-urilor detectate, in schimb il lasam sa treaca la ultimul test care va determina cu adevarat daca suspectul este sau nu un injection.
* Ultimul test primeste suspectul ca fiind doar un text format doar din litere mici ale alfabetului englez si spatiu. Ultimul test va verifica ceea ce parea intuitiv sa verificam la primul test, anume existenta clauzelor din SQL. Daca acestea exista, trebuie sa existe in mod direct, adica(spre exemplu) trebuie sa avem literalmente cuvantul "select" pentru un injection, nu "selector", "selecting", sau alta forma a cuvantului "select".
* Pe langa cele 5 teste, mai exista multe alte cazuri de injection-uri, precum If statement, String Quote, Integer Codes, toate acestea putand fii depistate chiar din aceste 5 teste.

**Informatii cod:**

Pentru a putea pregati testele, ne folosim de doua liste:

- sqlWords = lista tuturor clauzelor SQL ce pot fi folosite pentru un injection

OBSERVATIE: toate clauzele sunt scrise cu litere mici, intrucat pana cand vom ajunge sa le folosim, suspectul nostru deja va fi transformat intr-un text ce contine cuvinte cu litere mici separate printr-un singur spatiu.

- harmfulCharacters = lista tuturor caracterelor ce pot provoca un injection

**XSS**

Atacurile **Cross-Site Scripting (XSS)** sunt un tip de injecție, în care scripturile rău intenționate sunt injectate în site-uri web altfel benigne și de încredere. Atacurile XSS apar atunci când un atacator folosește o aplicație web pentru a trimite cod rău intenționat, în general sub forma unui script din partea browserului, către un alt utilizator final. Defectele care permit acestor atacuri să reușească sunt destul de răspândite și apar oriunde o aplicație web folosește intrarea de la un utilizator în ieșirea pe care o generează fără a o valida sau codifica.

Un atacator poate folosi XSS pentru a trimite un script rău intenționat unui utilizator care nu bănuiește. Browserul utilizatorului final nu are de unde să știe că scriptul nu ar trebui să fie de încredere și va executa scriptul. Deoarece consideră că scriptul provine dintr-o sursă de încredere, scriptul rău intenționat poate accesa orice cookie-uri, simboluri de sesiune sau alte informații sensibile păstrate de browser și utilizate cu site-ul respectiv. Aceste scripturi pot chiar rescrie conținutul paginii HTML.

Pot apărea atacuri de scripturi între site-uri oriunde în care utilizatorii posibil rău intenționați au voie să posteze materiale nereglementate pe un site web de încredere pentru consumul altor utilizatori. Cel mai obișnuit exemplu poate fi găsit în site-urile web de buletin-board care oferă funcționalități bazate pe web în stil listă de corespondență. Acest atac poate fi efectuat în diferite moduri. În funcție de tipul de atac XSS, scriptul rău intenționat poate fi reflectat în browserul victimei sau stocat în baza de date și executat de fiecare dată când utilizatorul apelează funcția corespunzătoare.

Conform analizei *Positive Technologies*, XSS se numără printre cele mai comune trei atacuri de aplicații web. Procentul relativ de XSS în comparație cu alte tipuri de atacuri a scăzut în anii precedenți. Cu toate acestea, nu există niciun semn că XSS își va pierde popularitatea.

Atacurile XSS pot fi efectuate fără a utiliza etichetele <script>...</script>. Alte etichete vor face exact același lucru, de exemplu: <body onload=alert('test1')> sau alte atribute precum: onmouseover, onerror.

Exista mai multe tool-uri care ne ajuta sa ne protejam de astfel de atacuri, cele mai cunoscute fiind *Acunetix* si *Netsparker.*

**Principalele forme de Cross Site Scripting sunt următoarele:**

* *Cross Site Scripting poate apărea pe scriptul rău intenționat executat la nivelul clientului.*
* *Pagina sau formularul fals afișat utilizatorului (unde victima tasta acreditări sau dă clic pe un link rău intenționat).*
* Pe site-urile cu reclame afișate.
* E-mailuri rău intenționate trimise victimei.

Acest atac are loc atunci când utilizatorul rău intenționat găsește părțile vulnerabile ale site-ului web și le trimite ca intrare rău intenționată. Scriptul rău intenționat este injectat în cod și apoi trimis ca rezultat utilizatorului final.

**Tipuri de XSS Injection:**

1. **Reflected XSS -** *Acest atac are loc atunci când un script rău intenționat nu este salvat pe serverul web, ci se reflectă în rezultatele site-ului web.*
2. **Stored XSS -** *Acest atac are loc atunci când un script rău intenționat este salvat permanent pe serverul web.*
3. **DOM -***Acest lucru se întâmplă atunci când DOM-ul este schimbat, dar codul rămâne același.*

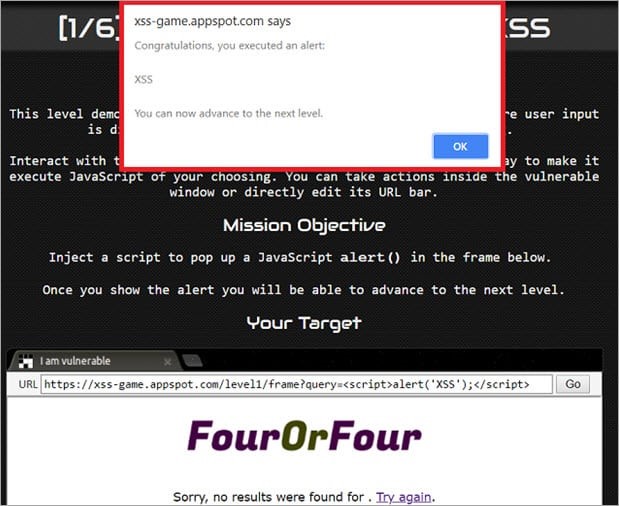
**Exemple de XSS Injection:**

* Luați în considerare că avem un site web cu un câmp de căutare.



* Dacă câmpul de căutare este vulnerabil, atunci când utilizatorul introduce orice script, atunci acesta va fi executat. Consideram urmatorul script : <script>alert(‘XSS’)</script> .



* Apoi, după ce faceți clic pe butonul „Search”, scriptul introdus va fi executat.   
  

După cum vedem în exemplu, scriptul introdus în câmpul de căutare este executat. Aceasta arată doar vulnerabilitatea atacului XSS. Cu toate acestea, poate fi introdus și un script mai dăunător. Mulți testeri combină atacul Cross Site Scripting cu Javascript Injection, care este, de asemenea, efectuat pe partea clientului. În ambele atacuri, scriptul rău intenționat este injectat. Cu toate acestea, în cazul atacului XSS, etichetele <script> nu sunt necesare pentru a executa scriptul.

**Detectarea XSS Injection in access.log :**

Un exemplu de access.log care are un posibil XSS Injection este urmatorul:

*192*.*168*.*100*.*12* - - [*24*/Feb/*2022*:17:05:33 -*0500*] "GET /bWAPP/xss\_href-2.php?name=%3E%3Cscript%3Ealert%281%29%3C%2Fscript%3E%3C&action=vote HTTP/1.1" *200 3554* "<http://192.168.100.51/bWAPP/xss_href-1.php>" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36"

Si in acest caz, request-ul pare sa fie locul unde se intampla anomalii, asadar o sa cautam anomalii in aceasta parte a log-ului. Cautam diferite taguri de JavaScript pentru a detecta un posibil XSS Injection.

**XPATH INJECTION**

Similar cu SQL Injection, atacurile **Xpath Injection** apar atunci cand un site web utilizeaza informatii furnizate de utilizator pentru a construi o interogare Xpath pentru date XML. Trimițând informații intenționate malformate pe site-ul web, un atacator poate afla cum sunt structurate datele XML sau poate accesa date la care ar putea să nu aibă acces în mod normal. Ei pot chiar să-și ridice privilegiile pe site-ul web dacă datele XML sunt utilizate pentru autentificare (cum ar fi un fișier de login bazat pe XML).

Interogarea XML se face cu XPath, un tip de declarație descriptivă simplă care permite interogării XML să localizeze o informație. La fel ca SQL, puteți specifica anumite atribute de găsit și modele de potrivire. Când se utilizează XML pentru un site web, este obișnuit să se accepte o anumită formă de intrare pe șirul de interogare pentru a identifica conținutul de localizat și afișat pe pagină.

XPath este un limbaj standard, sintaxa sa este întotdeauna independentă de implementare, ceea ce înseamnă că atacul poate fi automatizat. Nu există dialecte diferite, deoarece are loc în solicitările către bazele de date SQL.

**Exemplu de XPATH Injection:**

Vom folosi acest fragment XML pentru exemple:

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

Să presupunem că avem un sistem de autentificare a utilizatorilor pe o pagină web care a folosit un fișier de date de acest fel pentru a conecta utilizatorii. Odată ce au fost furnizate un nume de utilizator și o parolă, software-ul poate folosi XPath pentru a căuta utilizatorul:

*FindUserXPath = "//Employee[UserName/text()='" & Request("Username") & "' And Password/text()='" & Request("Password") & "']";*

Cu un nume de utilizator și o parolă normale, acest XPath ar funcționa, dar un atacator poate trimite un nume de utilizator și o parolă proaste și poate obține un nod XML selectat fără a cunoaște numele de utilizator sau parola, astfel:

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

Password: blahFindUserXPath devine

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

Care este echivalent cu:

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

În acest caz, doar prima parte a XPath-ului trebuie să fie adevărată. Partea cu parolă devine irelevantă, iar partea UserName se va potrivi cu toti angajații din cauza părții „1=1”.

**Detectarea Xpath Injection in access.log**

Un exemplu de Xpath Injection pentru un login form:

*192*.*168*.*56*.*1* - - [*25*/Feb/*2022*:18:42:17 -*0500*] "GET /bWAPP/xmli\_1.php?login=teck%27or+1%3D1+or+%271%27%3D%271&password=mara&form=submit HTTP/1.1" *200 3209* "<http://192.168.56.101/bWAPP/xmli_1.php>" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.81 Safari/537.36"

Observam ca singurul lucru iesit din comun este in request. Vom analiza un posibil injection cautand exclusiv in request.

**Etapele detectarii unui Xpath Injection sunt urmatoarele:**

1. Decodam request-ul astfel incat sa nu mai avem text unicode, ci doar caractere ASCII. Nu luam in considerare metoda(GET sau POST). Dupa decodare request-ul arata ce a introdus clientul in formularul de login.

*login=teck'or 1=1 or '1'='1&password=mara&form=submit\_http/1.1*

Putem observa ce a introdus in campul de login (*teck’or 1=1 or ‘1’=’1*) si in campul password (*mara*) . In campul de login se poate observa clar un posibil injection. Campul password devine irelevant datorita Xpath Injection-ului introdus la campul anterior.

1. Acum incepem sa cautam posibilul injection in dreapta fiecarui camp.
2. Dupa ce am preprocesat datele cum am stabilit la pasii anteriori, tot ce mai avem de facut este sa cautam posibile expresii Xpath.

Aici s-a dovedit partea dificila, fiindca neavand codul XML nu putem sa stim daca chiar s-a produs un injection corect. Aici vom detecta posibile Xpath Injections dupa cuvinte sau expresii cheie Xpath sau dupa anumite tipare de sintaxa.

Primul tipar de sintaxa este ca requestul sa inceapa cu caracterele “//” care indica selectarea unui anumit nod din document. Un alt tipar ar fi cand gasim caracterul “@” precedat de “[“. Aceasta configuratie poate sa indice selectarea unui anumit atribut. Alt posibil injection se poate intampla cand avem urmatoarea configuratie “ ‘something ” sau “ something’ “. Urmatoarea metoda prin care am incercat sa identificam posibile injections a fost sa cautam cuvinte sau expresii specifice Xpath Injection-urilor si anume : or, and, contain, node, div, class, child, find, “1=1”, “ ‘1’=’1 “ , etc.. .

Sunt posibile unele confunzii cu SQL Injection datorita sintaxei foarte similare ale celor doua injections.

**TO DO:** cautarea unor tipare cu o precizie mai ridicata de a gasi injection-urile in request. Cautarea unor diferente clare intre request-urile cu different injections pentru o acuratete mai mare in detectarea unui anumit tip de injection.

**LFI**

**Local File Inclusion (LFI)** permite unui atacator să includă fișiere pe un server prin intermediul browserului web. Această vulnerabilitate există atunci când o aplicație web include un fișier fără a igieniza corect intrarea, permițând atacatorului să manipuleze intrarea și să injecteze caractere de traversare a căii și să includă alte fișiere de pe serverul web.

Local File Inclusion (LFI) este foarte asemănătoare cu Remote File Inclusion (RFI). Cu toate acestea, un atacator care utilizează LFI poate include numai fișiere locale (nu fișiere la distanță, cum ar fi în cazul RFI).

**Exemplu de LFI:**

Următorul este un exemplu de cod PHP care este vulnerabil la LFI:

*/\*\* \* Get the filename from a GET input \**

*Example -* [*http://example.com/?file=filename.php*](http://example.com/?file=filename.php)

*\*/*

$file = $\_GET['file'];

*/\*\**

*\* Unsafely include the file*

*\* Example - filename.php*

*\*/*

**include**('directory/' . $file);

În exemplul de mai sus, un atacator ar putea face următoarea solicitare. Păcălește aplicația să execute un script PHP, cum ar fi un shell web, pe care atacatorul a reușit să îl încarce pe serverul web.

[*http://example.com/?file=../../uploads/evil.php*](http://example.com/?file=../../uploads/evil.php)

În acest exemplu, fișierul încărcat de atacator va fi inclus și executat de utilizatorul care rulează aplicația web. Acest lucru ar permite unui atacator să ruleze orice cod rău intenționat pe partea de server pe care îl dorește.

Acesta este cel mai rău scenariu. Un atacator nu are întotdeauna capacitatea de a încărca un fișier rău intenționat în aplicație. Chiar dacă au făcut-o, nu există nicio garanție că aplicația va salva fișierul pe același server unde există vulnerabilitatea LFI. Chiar și atunci, atacatorul ar trebui să cunoască calea discului către fișierul încărcat.

**CSRF**

**Cross-Site Request Forgery (CSRF)** este un atac care forțează un utilizator final să execute acțiuni nedorite pe o aplicație web în care este autentificat în prezent. Cu puțin ajutor de inginerie socială (cum ar fi trimiterea unui link prin e-mail sau chat), un atacator poate păcăli utilizatorii unei aplicații web să execute acțiuni alese de atacator. Dacă victima este un utilizator normal, un atac CSRF de succes îl poate forța pe utilizator să efectueze solicitări de schimbare a stării, cum ar fi transferul de fonduri, schimbarea adresei de e-mail și așa mai departe. Dacă victima este un cont administrativ, CSRF poate compromite întreaga aplicație web.

CSRF este un atac care păcălește victima să trimită o solicitare rău intenționată. Moștenește identitatea și privilegiile victimei pentru a îndeplini o funcție nedorită în numele victimei. Pentru majoritatea site-urilor, solicitările browserului includ automat orice acreditări asociate site-ului, cum ar fi cookie-ul de sesiune al utilizatorului, adresa IP, acreditările de domeniu Windows și așa mai departe. Prin urmare, dacă utilizatorul este în prezent autentificat pe site, site-ul nu va avea nicio modalitate de a distinge între cererea falsificată trimisă de victimă și o cerere legitimă trimisă de victimă.

CSRF atacă funcționalitatea țintă care provoacă o schimbare a stării pe server, cum ar fi schimbarea adresei de e-mail sau a parolei victimei sau cumpărarea a ceva. Forțarea victimei să recupereze date nu aduce beneficii unui atacator, deoarece atacatorul nu primește răspuns, victima primește. Ca atare, atacurile CSRF vizează cereri de schimbare a stării.

Această problemă este ușor de detectat, dar pot exista controale compensatoare în jurul funcționalității aplicației, care pot alerta utilizatorul cu privire la o încercare CSRF. Atâta timp cât aplicația acceptă o solicitare HTTP bine formată și cererea aderă la o logică de afaceri a aplicației, CSRF va funcționa (De acum presupunem că utilizatorul țintă este conectat în sistem pentru a fi atacat).

Prin verificarea redării paginii, trebuie să vedem dacă sunt atașați identificatori unici la linkurile redate de aplicație în browserul utilizatorului. Dacă nu există un identificator unic legat de fiecare cerere HTTP pentru a lega o solicitare HTTP de utilizator, suntem vulnerabili. ID-ul sesiunii nu este suficient, deoarece ID-ul sesiunii va fi trimis oricum dacă un utilizator face click pe un link rau intentionat, deoarece utilizatorul este deja autentificat.

**Exemple ale unui CSRF Attack:**

Următorul exemplu arată cum ar putea arăta o solicitare tipică GET pentru un transfer bancar de 5.000 USD:

*GET* [*https://abank.com/transfer.do?account=RandPerson&amount=$5000*](https://abank.com/transfer.do?account=RandPerson&amount=$5000) *HTTP/1.1*

Un atacator poate modifica scriptul, astfel încât să rezulte un transfer de 5.000 USD în contul personal. Solicitarea rău intenționată ar putea arăta astfel:

*GET* [*https://abank.com/transfer.do?account=SomeAttacker&amount=$5000*](https://abank.com/transfer.do?account=SomeAttacker&amount=$5000) *HTTP/1.1*

Ulterior, atacatorul poate încorpora cererea într-un hyperlink cu aspect inofensiv:

*<a href="https://abank.com/transfer.do?account=SomeAttacker&amount=$5000">Click for more information</a>*

Următorul pas este distribuirea hyperlinkului prin e-mail unui număr masiv de clienți bănci. Cei care sunt autentificati în contul lor bancar și dau clic pe acest link vor iniția neintenționat transferul de 5.000 USD.

**Metoda de detectare a unui CSRF:**

* Acest tip de atac este detectat pe baza diferentelor dintre request si referer link.
* Pentru a face acest lucru am folosit un dictionar care are ca si chei un string format din IP + request, iar elementele dictionarului sunt seturi in care adaugam mereu toate referer link-urile care se afla in logul care contine una din cheile dictionarului.
* In caz ca la o cheie exista mai mult de un referer link, verificam ca nu cumva acele linkuri sa fie de fapt subpagini ale aceluiasi site si asta am verificat comparand prefixurile link-urilor doua cate doua, iar in caz ca prefixurile difera in primele 10 caractere atunci detectam un CSRF attack.

**Brute-force Attack**

În criptografie, un **Brute-force Attack** constă în faptul că un atacator trimite multe parole sau fraze de acces cu speranța de a ghici în cele din urmă corect. Atacatorul verifică sistematic toate parolele și frazele de acces posibile până când este găsită cea corectă. Alternativ, atacatorul poate încerca să ghicească cheia care este de obicei creată din parolă folosind o funcție de derivare a cheii. Aceasta este cunoscută ca o căutare exhaustivă a cheilor.

Brute-force Attacks sunt adesea folosite pentru a ataca autentificarea și pentru a descoperi conținut/pagini ascunse într-o aplicație web. Aceste atacuri sunt de obicei trimise prin solicitări GET și POST către server. În ceea ce privește autentificarea, atacurile de forță brută sunt adesea lansate atunci când nu există o politică de blocare a contului.

Atacurile cu forță brută funcționează calculând fiecare combinație posibilă care ar putea alcătui o parolă și testând-o pentru a vedea dacă este parola corectă. Pe măsură ce lungimea parolei crește, timpul, în medie, pentru a găsi parola corectă crește exponențial.

**Metoda prin care detectam un Brute-force Attack:**

**1.** Aceasta detectie se bazeaza pe o analizare a frecventei cu care apar loguri de login de la un anumit IP si pe numarul de incercari de login ale respectivului IP. Pentru a putea avea o precizie cat mai mare in detectarea atacului am incercat sa incadram cat mai bine atacul in access.log intre un log initial si unul final. Am creat o functie care calculeaza timpul mediu de asteptare intre doua loguri ale unui anumit IP intre doua pozitii date din access.log.

**2.** Prima data am calculat timpul mediu de asteptare al IP-ului in tot fisierul access.log. Acum incercam sa stabilim incadrarea atacului despre care vorbeam, in felul urmator:

Initializez *first* cu 0, iar last cu len(*dates*)

Parcurg toate IP-urile distincte, folosesc o variabila ajutatoare *ok\_first* care ajuta sa decidem daca *first* a fost ales sau nu, pentru fiecare IP parcurgem access.log si distingem urmatoarele cazuri:

- daca IP-ul ales este IP-ul curent (dintre cele distincte) si cuvantul "login" se afla in request si *ok\_first* este 0, atunci *ok* primeste indexul din lista de loguri pe care ne aflam si *ok\_first* devine 1

- daca IP-ul ales este IP-ul curent (dintre cele distincte) si cuvantul "login" se afla in request si *ok\_first* = 1 si timpul de asteptare dintre logul actual si urmatorul cu acest IP este mai mic decat timpul mediu de asteptare calculat initial atunci setam si variabila *last*

- daca IP-ul ales este IP-ul curent si *ok\_first* = 1 si status-ul curent este mai mare sau egal cu 300 sau timpul mediu de asteptare pana la urmatorul request este mai mare decat timpul mediu calculat initial si totodata cu acest timp mediu de asteptare ne aflam in primele 15% loguri cu IP-ul respectiv, atunci trebuie sa spunem ca *first* a fost ales gresit si sa il initializam din nou 0, iar pe *ok\_first* sa il initializam cu 0

Daca nu foloseam aceasta conditie cu 15% algoritmul putea sa gaseasca la sfarsitul fisierului un log cu timp mare de asteptare inaintea lui si il selecta ca si *first*, ceea ce este o greseala.

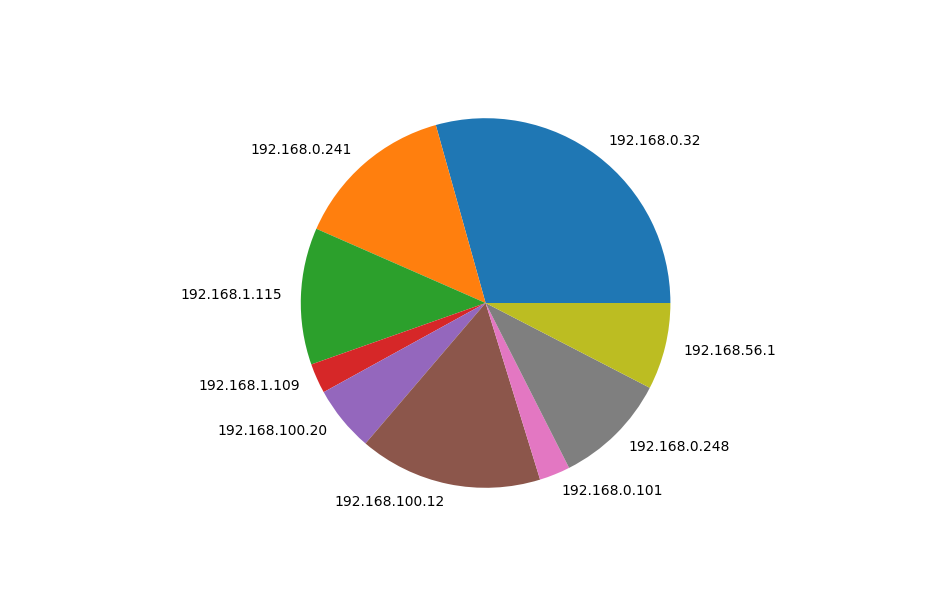
Am tratat si cazul in care cuvantul "login" nu exista in request cu singura modificare la al doilea if am pus acel sau sa se afle, de data asta in ultimele 25% dintre logurile cu IP-ul respectiv pentru a evita pe cat posibil ca *first* > *last* sau *last* = len(*logs\_list*).

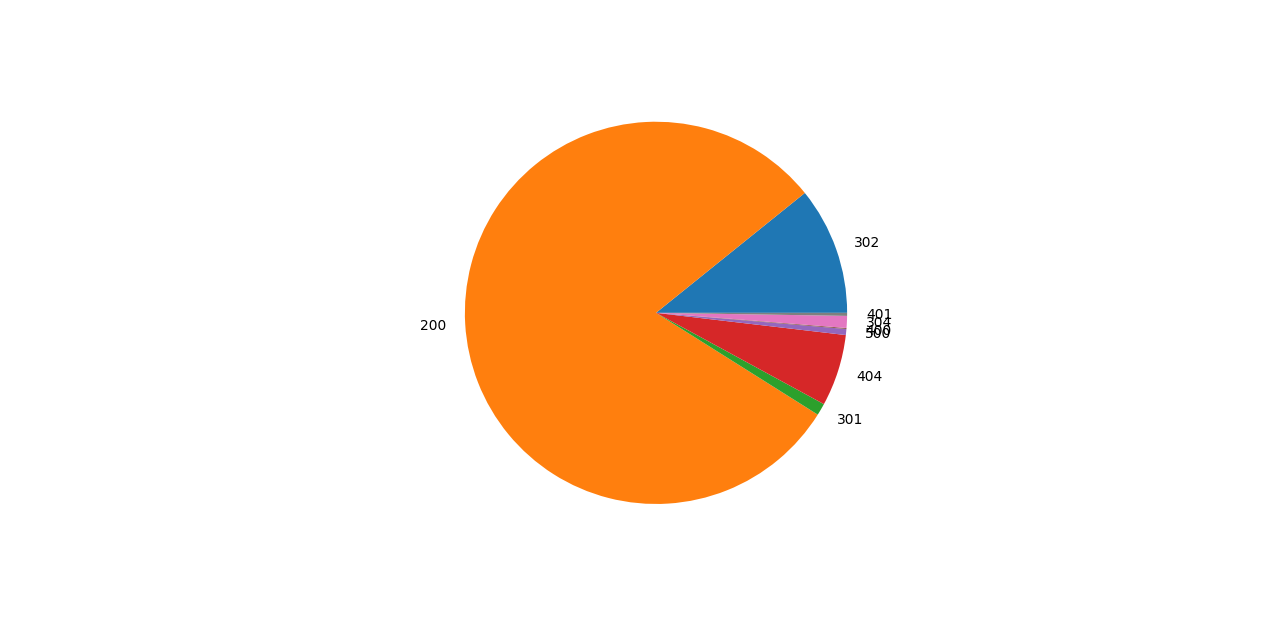
Dupa ce am stabilit *first* si *last,* calculam timpul mediu de asteptare intre *first* si *last,* numarul de requesturi ale IP-ului curent si daca timpul mediu este mai mic de 3 secunde (pentru a detecta si cel mai lent Brute-force Attack, adica cel configurat de noi) si daca numarul de request-uri este mai mare de 100 si cu timpul mediu pozitiv (in caz de orice, spre examplu un access.log facut pe bucati care nu are logurile sortate din data si ora atunci inseamna ca am gasit un Brute-force Attack).

Cel mai probabil acest algoritm nu va detecta mai multe atacuri cu pauza intre ele si cu mai putin de 100 de requests/atac.

**CONCLUZII**

Avand o idee clara despre ce inseamna un log, cateva informatii despre fiecare injection in parte, putem sa detectam un posibil atac si manual. In anumite cazuri, unele tipuri de Injection pot sa fie detectate prin simpla analizare manuala a logurilor. Am realizat diverse diagrame pentru a putea urmari anumite tipare care apar. Diagramele au fost realizate cu ajutorul unui program Python folosind libraria matplotlib.





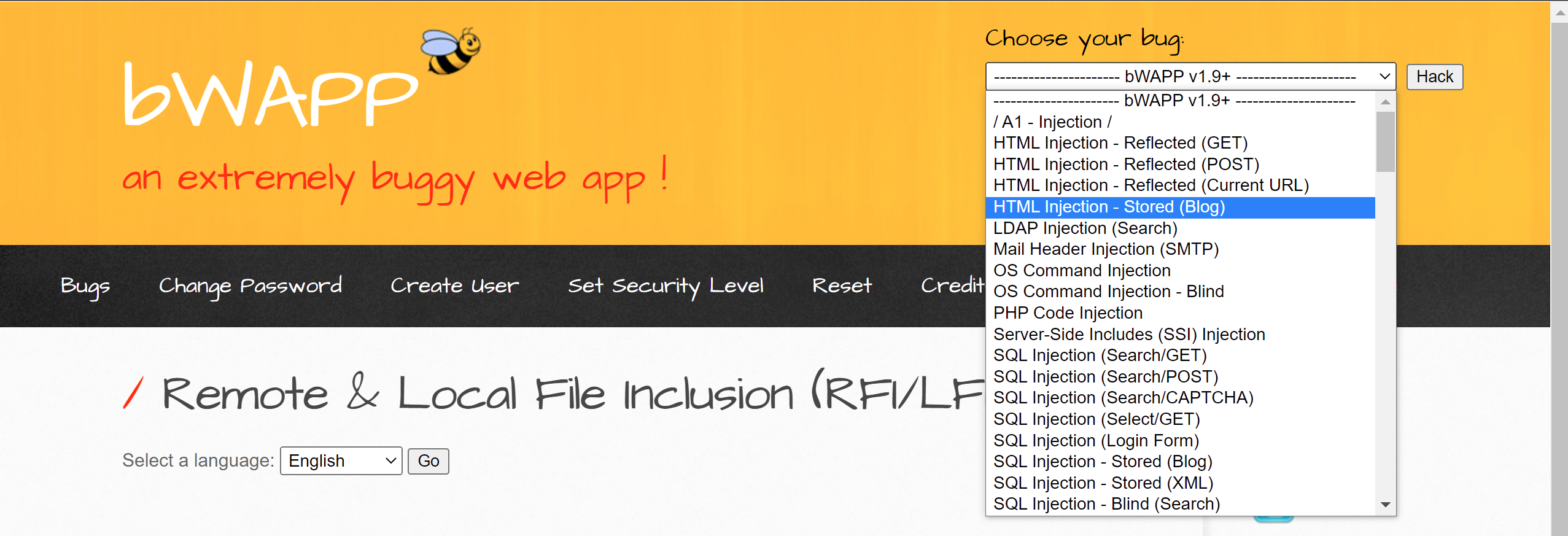
In prima figura avem o diagrama cu densitatea IP-urilor. Acest lucru ne poate ajuta in cautarea unui anumit atacator pe baza IP-ul. IP-ul ne poate furniza mai multe informatii, cea mai importanta fiind localizarea clientului. Putem sa analizam cu mai mare atentie IP-urile care vin din tari cu antecedente de atacuri cibernetice.

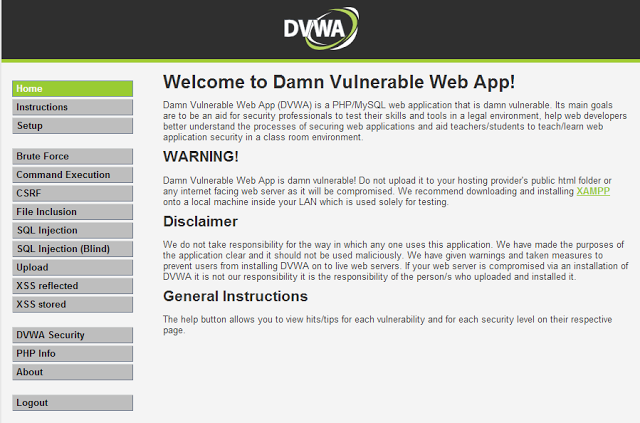
Putem sa cream o diagrama cu numarul de intrari intr-o anumita zi sau intr-o anumita ora pentru a ne da seama daca avem un posibil atac ( frecventa intrarilor in server poate semnala un posibil atac ).

Ca utilizator de rand, presupunem ca nu cunoastem toate aceste informatii si avem nevoie de un program care sa faca toata aceasta munca pentru noi. Acest lucru am incercat noi sa facem, un program care primeste un fisier cu log-uri si intoarce toate posibilele atacuri.

**TESTARE**

Pentru testare am folosit OWASP. Proiectul Open Web Application Security este o comunitate online care produce articole, metodologii, documentație, instrumente și tehnologii disponibile gratuit în domeniul securității aplicațiilor web.





Aplicatiile simuleaza o varietate foarte mare de injections si noi ne-am folosit de aceasta pentru a extrage fisierele de log-uri. Avem atasate in arhiva toate fisierele de log-uri extrase. Rezultatele au fost destul de bune, majoritatea atacurilor au fost detectate.

**REFERENCES**

1. <https://owasp.org/www-community/attacks/Brute_force_attack>
2. <https://portswigger.net/web-security/csrf>
3. <https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/>
4. <https://owasp.org/www-community/attacks/SQL_Injection>
5. <https://hackbotone.medium.com/cross-site-scripting-reflected-ajax-json-b280c1777e88>
6. <https://www.acunetix.com/websitesecurity/sql-injection/>
7. <http://projects.webappsec.org/w/page/13247005/XPath%20Injection>
8. <https://www.sciencedirect.com/science/article/pii/S2210832717300169>

*Proiectul a fost realizat de:*

* *Adelina Cristina Hincu*
* *Amar Kayed*
* *Contor Flavius-Andrei*
* *Serban Filote*
* *Dragos Polifronie*