SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 221

Tehnike otkrivanja ranjivosti web sjedišta s klijentske strane

Luka Srdarev

Zagreb, srpanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

ZAVRŠNI ZADATAK br.

Pristupnik:	**Luka Srdarev (0036517080)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Stjepan Groš

Zadatak: **Tehnike otkrivanja ranjivosti web sjedišta s klijentske strane**

Opis zadatka:

Web sjedišta uslijed loših praksi u razvoju, konfiguraciji i održavanju mogu sadržavati brojne ranjivosti. Iskorištavanjem ranjivosti napadači mogu ostvariti pristup poslužiteljima na kojima su ta web sjedišta smještena i/ili računalima i Web preglednicima posjetitelja. Osim što je takve ranjivosti poželjno prijavljivati i otklanjati, može se očekivati i da će web sjedišta koja obiluju poznatim težim ranjivostima biti češće kompromitirana nego usporediva web sjedišta na kojima su iste odsutne, zbog čega se nameće potreba za učestalijim nadgledanjem ranjivih web sjedišta i povećanjem stupnja opreza njihovih posjetitelja. U sklopu završnoga rada potrebno je istražiti tehnike kojima se s klijentske strane može otkrivati ranjivosti web sjedišta, razviti rješenje koje na temelju preuzetih web stranica i popratnih sadržaja pronalazi što veći broj njihovih ranjivosti, te provesti evaluaciju razvijenog rješenja nad skupom podataka i prokomentirati dobivene rezultate. Rješenje mora uključivati testove i biti robusno. Radu priložiti izvorni kôd razvijenih i korištenih programa. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 11. lipnja 2021.

Zahvaljujem se mentoru doc. dr. sc. Stjepanu Grošu i mag. ing. Ivanu Kovačeviću na stručnom vodstvu i pomoći pri izradi ovoga rada.

SADRŽAJ

1. Uvod **1**

2. Ranjivosti weba **3**

2.1. Cross site scripting 4

2.1.1. Reflektirani XSS 5

2.1.2. Pohranjeni XSS 6

2.1.3. XSS temeljen na DOM-u 7

2.1.4. Prevencija XSS-a 8

2.2. SQL ubacivanje 9

2.3. Korištenje zastarjelih JavaScript biblioteka 12

2.4. Korištenja ranjivih verzija CMS-ova 13

2.5. Korištenje potencijalno nesigurnih JavaScript funckija 14

3. Tehnike detekcije ranjivosti weba **16**

3.1. Detekcija ranjivosti weba s klijentske strane 16

4. Razvijeno programsko rješenje **18**

4.1. Opis rješenja, korištenih alata i tehnologija 20

4.2. Implementacija 24

4.3. Primjeri korištenja i rezultati 30

4.4. Ograničenja implementiranog rješenja 33

5. Zaključak **34**

Literatura **35**

1. Uvod

Internet je javno dostupna globalna podatkovna mreža. *World wide web* ili kraće *Web* je sustav međusobno povezanih web stranica koji se temelji na internetu. Web se zasniva na HTTP protokolu i omogućuje korisnicima brz i jednostavan pregled sadržaja web stranica putem web preglednika. Upravo zbog te brzine i jednostavnosti broj korisnika Weba od njegovog razvoja devedesetih godina prošlog stoljeća do danas je porastao čak do 4.66 milijardi što čini skoro 60% svjetske populacije [13]. Samim time i velik broj tvrtka orijentiralo je svoje usluge i rješenja na web. Također tokom godina razvija se sve više naprednijih alata koji olakšavaju razvoj web aplikacija pa zbog toga i njihov broj raste.

U tom smislu sigurnost weba postala je temelj modernog poslovanja i komunikacije. Pod pojmom sigurnosti weba smatra se proces zaštite od prijetnji koje iskorištavaju ranjivosti u web aplikacijama nastale uslijed manjkavosti u implementaciji, dizajnu upotrebi ili upravljanju. Kako bi se osigurala adekvatna zaštita web aplikacija potrebno je poznavati greške do kojih može doći prilikom razvoja, ranjivosti koje one uzrokuju, odgovarajuće napade i kako se od njih zaštititi. Isto tako od ključne važnosti je moći pravovremeno i pouzdano detektirati i ukloniti slabosti prije nego što ih napadač odnosno zlonamjerni korisnik nađe.

Cilj ovoga rada je opisati neke od učestalijih ranjivosti i njihovu klasifikaciju, opisati tehnike detekcije ranjivosti Weba i razmotriti programsko rješenje za automatizaciju tog postupka. Rad se sastoji od pet poglavlja.

Nakon uvodnog poglavlja, u drugom poglavlju je defiinirano što su ranjivosti Weba, kako ih se klasificira i koji su poznatiji repozitoriji ranjivosti i kako ih popisuju. Zatim su detaljnije opisane ranjivosti XSS i SQL ubacivanje. Za te ranjivosti je objašnjeno je kako nastaju, kako ih napadači iskorištavaju te kako ih preventirati.

U trećem poglavlju opisana je podjela tehnika detekcija ranjivosti Weba. Nakon toga objašnjene su neke metode detkecije koje se provode s klijentske strane, a implementirane su u praktičnom dijelu rada.

U četvrtom poglavlju opisuje se program razvijen u sklopu praktičnog dijela ovoga rada. Navedene su tehnologije i alati korišteni pri razvoju programa. Nakon toga prikazani su i objašnjeni su neki odsječci koda koji su potrebni za razumijevanje rada programa. Zatim su prikazani primjeri korištenja programa i odgovarajući rezultati. Također prokomentirana su ograničenja implementiranog rješenja i kako ga potencijalno poboljšati. Rad završava sa zaključkom u petom poglavlju, nakon kojeg slijedi pregled korištene literature.

2. Ranjivosti weba

Ranjivosti weba su sve slabosti odnosno pogreške u implementaciji, dizajnu, upotrebi ili upravljanju Web aplikacijama čijim se iskorištavanjem može narušiti njihova sigurnost [32]. Iako promjena broja otkrivenih ranjivosti Weba prati silazni trend zadnjih par godina, brojke su i dalje alarmantne [30]. Istraživanja pokazuju da nešto manje od 35% ranjivosti Web aplikacija otkrivenih u 2020. godini upada u kategoriju kritične ili visoke ozbiljnosti [1] . U nastavku su objašnjeni neki pojmovi vezani uz klasifikaciju ranjivosti koji se koriste u ostatku rada.

OWASP (engl. *Open Web Application Security Project*) [19] je međunarodna neprofitna organizacija posvećena sigurnosti web aplikacija. Jedno od temeljnih načela OWASP-a je da su svi njihovi materijali slobodno dostupni na njihovoj web stranici, što omogućava svakome da poboljša sigurnost vlastitih web aplikacija. Materijali koje nude uključuju dokumentaciju, alate, videozapise i forume. Njihov najutjecajniji projekt je OWASP Top 10 [20]. To je popis deset najučestalijih kritičnih sigurnosnih propusta u Web aplikacijama te smjernica za njihovo popravljanje. Popis se temelji na konsenzusu sigurnosnih stručnjaka iz cijelog svijeta.

U tom kontekstu bitno je spomenuti i CVE. CVE (engl. *Common Vulnerabilities and Exposures*) je javno dostupan popis ranjivosti kojeg je definirala organizacija MITRE [16]. Svakom zapisu se pridjeljuje jedinstveni identifikator neovisan o proizvođaču programske podrške u obliku *CVE-yyyy-nnnnnn* gdje *yyyy* označava znamenke godine u kojoj je ranjivost otkrivena, a *nnnnnn* označava jedinstveni identifikator te ranjivosti. Također u daljnjem tekstu će se spominjati CWE. CWE (engl. *Common Weakness Enumeration*) je popis odnosno sustav kategorizacije učestalih tipova slabosti u programskim i sklopovskim sustavima [5].

2.1. Cross site scripting

Cross site scripting ili kraće XSS [37] je ranjivost Web aplikacija koja omogućuje napadaču da u stranicu ubaci maliciozni kod koji se potom izvršava u pregledniku žrtve koja pregledava tu stranicu. Zbog nepravilne provjere, odnosno validacije korisničkog unosa, stranica interpretira zlonamjerni kod kao legitimni dio stranice te ga potom Web preglednik žrtve izvršava. U tom smislu na XSS se ponekad referira kao JavaScript *injection*. XSS je jedna od najučestalijih ranjivosti weba i nalazi se na sedmom mjestu zadnjeg OWASP top 10 popisa iz 2017. godine [20]. Po provedenim istraživanjima XSS čini oko 40% svih napada na Web u 2019. godini[41].

Da bi razumjeli punu snagu XSS-a potrebno je razumjeti politiku istog izvorišta. Izvor nekog sadržaja na stranici je definiran kao protokol, domena i port u URL-u kojim se pristupa tom sadržaju. Politika istog izvorišta (engl. *Same origin policy, SOP*) [24] je sigurnosni mehanizam koji ograničava način na koji dokument ili skripta učitani iz jednog izvora mogu komunicirati s resursom drugog izvora. Pomaže u izolaciji potencijalno zlonamjernih dokumenata, smanjujući moguće vektore napada. Dva izvora su smatrana jednaka ako imaju jednak protokol, domenu i port. Zbog toga su povjerljivi podatci poput kolačića (engl. *cookies*) koji sadržavaju identifikatore sjednice (engl. *session ID*) izolirani na stranicu za koju su namijenjeni odnosno druge stranice im nemaju pristup.

Koristeći XSS napadač ima pristup svim povjerljivim podatcima korištenim u interakciji korisnika i ranjive stranice. Ubačenim malicioznim kodom napadač te podatke može proslijediti negdje gdje će im imati pristup npr. na vlastit web server i tako efektivno zavarati politiku istog izvorišta i ukrasti povjerljive podatke.

Ovisno o načinu umetanja zlonamjernog koda XSS se dijeli na tri vrste [40] :

1. Pohranjeni XSS
2. Reflektirani XSS
3. Lokalni XSS

U nastavku su opisani navedeni tipovi XSS-a i dani primjeri.

2.1.1. Reflektirani XSS

Reflektirani XSS [21], još poznat kao neustrajni ili privremeni XSS, je vrsta XSS ranjivosti u kojoj se maliciozni kod unosi kao dio URL-a. Napadač kreira poveznicu u kojoj se nalazi maliciozni kod. Jednom kada žrtva klikne na poveznicu taj maliciozni kod se prosljeđuje ranjivoj aplikaciji koja ga tretira kao legitimni kod stranice i vraća ga u HTTP odgovoru korisniku te se kod izvršava u njegovom Web pregledniku. Zbog toga se ova ranjivost zove "reflektirani" XSS jer se maliciozni kod izvršava nakon što ga Web poslužitelj ranjive aplikacije ubaci i vrati u HTTP odgovoru. Dakle, da bi iskoristio reflektirani XSS napadač mora nekako prisiliti žrtvu da klikne na poveznicu. Bitno je napomenuti da takvi URL-ovi imaju sličan početni dio legitimnim URL-ovima te ranjive stranice zbog čega su žrtve sklonije kliknuti na poveznicu . Takva poveznica može žrtvu preusmjeriti na stranicu kao što to i očekuje tako da žrtva neće ni biti svjesna da se izvršila zlonamjerna funkcionalnost.

Za primjer uzmimo ranjivi webshop. Recimo da na webshopu postoji *textbox* za unos koji služi za pretragu svih proizvoda na stranici. Jednom kada korisnik unese *query* preusmjeren je na stranicu s URL-om ovakvog oblika:

```
https://nesiguran-webshop.com/search?term=query
```

Stranica na temelju upita ispiše adekvatnu poruku i prikaže filtrirane proizvode:

```
<p>Rezultati pretrage za: query</p>
```

Iz ovoga je jasno da napadač ima kontrolu nad varijablom *query*. Ako ta varijabla nije pravilno provjerena napadač može iskoristiti ranjivost ako kreira ovakav URL:

```
https://nesiguran-webshop.com/
search?term=<script> zloćudni kod...</script>
```

Takav unos rezultira izvršavanjem ovakvog koda:

```
<p>Rezultati pretrage za: <script>zloćudni kod...</script></p>
```

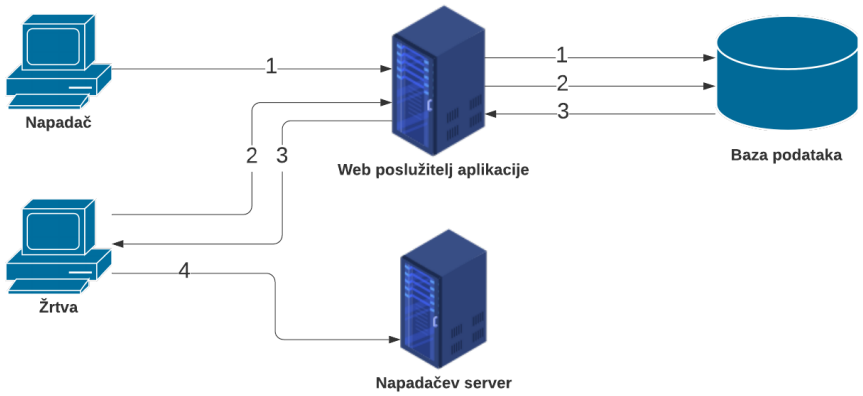
Stranice se često od ovakvih napada pokušavaju obraniti tako da profiltriraju po `<script>` tagu no to je naivan i površan način. Napadač može maliciozni kod ubaciti na mnoštvo načina bez da koristi `<script>` tag npr:

```
https://nesiguran-webshop.com/
search?term=<img src='#' onerror=zloćudni kod...>
```

U primjeru iznad za `src` vrijednost `img` taga postavljen je nespotojeći izvor `#`, što uzrokuje grešku i izvodi se kod naveden u `onerror` što omogućuje napadaču da tamo postavi zloćudni kod.

2.1.2. Pohranjeni XSS

Pohranjeni (engl. *Stored*) XSS [28] je vrsta XSS ranjivosti u kojoj slično kao kod reflektiranog, napadač unosi proizvoljni maliciozni kod, no razlika je u tome što se kod pohranjenog XSS-a taj maliciozni negdje sprema npr. u bazu podataka koja je povezana s Web aplikacijom. Nakon što je takav maliciozan input pohranjen, Web aplikacija ga vraća unutar HTTP odgovora na svaki GET zahtjev što dovodi do izvršavanja tog koda u žrtvinom Web pregledniku. Iz ovoga proizlazi da kod pohranjenog XSS-a nije potrebno žrtvu natjerati da klikne na poveznicu kao kod reflektiranog XSS-a, nego se maliciozni kod izvršava prilikom svakog pristupa Web stranici. Pohranjeni XSS još se naziva i ustrajni ili stalni XSS. Pohranjeni XSS dodatno je opisan kroz sliku ispod na primjeru foruma.



Slika 2.1: Opis pohranjenog XSS-a

1. Napadač unosi maliciozni kod na ranjivu stranicu foruma koja ne provodi validaciju i prosljeđuje nefiltrirani input bazi podataka.

2. Korisnik posjećuje forum odnosno stvara GET zahtjev.

3. Da bi Web aplikacija prikazala objave na forumu radi upit bazi podataka koja nazad vraća sve objave na forumu među kojima je i ona koja sadrži napadačev maliciozni kod.

4. Maliciozni kod se izvršava u žrtvinom pregledniku, preuzima povjerljive podatke poput kolačića i šalje ih na napadačev Web server.

2.1.3. XSS temeljen na DOM-u

Da bi se razumio XSS temeljen na DOM-u prvo je potrebno spomenuti što je to DOM. DOM (engl. *Document Object Model*) [8] je API (engl. *Application Programming Interface*) za HTML i XML dokumente koji na temelju organizacije tagova u dokumentu taj dokument tretira kao stablastu strukturu.

XSS baziran na DOM-u [9] ili lokalni XSS je vrsta XSS ranjivosti koja se za razliku od druge dvije odvija na klijentskoj strani. Moguć je ako web aplikacija upisuje podatke u objektni model dokumenta bez odgovarajuće provjere. Napadač može manipulirati tim podacima kako bi u DOM ubacio XSS sadržaj, na primjer zlonamjerni JavaScript kod. Dakle za razliku od reflektiranog XSS-a kod kojega se kod ubacuje na poslužiteljskoj strani, kod DOM XSS-a maliciozni kod se ubacuje koristeći JavaScript funkcije u pregledniku s klijentske strane bez ikakve komunikacije sa poslužiteljem. Samim time ni filteri s poslužiteljske strane ne pomažu kod DOM XSS-a. Uzmimo za primjer stranicu koja generira poruku dobrodošlice prilagođenu za korisnika.

Izvorni kod 2.1: Isječak koda stranice ranjive na DOM XSS

```
<html>
   <head>
      <title>Stranica dobrodošlice </title>
      ...
   </head>
   Pozdrav i dobrodošli na naše stranice,
   <script>
      var pos=document.URL.indexOf("username=")+9;
      document.write(
      document.URL.substring(pos,document.URL.length));
   </script>
   ...
</html>
```

Stranica 2.1 iz URL-a čita vrijednost parametra `username` i stvara statičku stranicu koja ispisuje odgovarajuću poruku. Za validan unos poput:

```
http://www.ranjivastranica.com/dobrodoslica.html?username=DennisRitchie
```

na stranici se pojavljuje ispis *Pozdrav i dobrodošli na naše stranice DennisRitchie*.
Da bi iskoristio ranjivost napadač ugrađuje zlonamjernu skriptu u URL:

```
http://www.ranjivastranica.com/
dobrodoslica.html#username= <script> maliciozan kod ... </script>.
```

Žrtva klikne na ovako formiranu poveznicu, stvara GET zahtjev na stranici i nazad dobiva sadržaj stranice.
Tek nakon toga, prilikom parsiranja dobivene HTML datoteke i gradnje DOM-a web preglednik pokušavajući dohvatiti vrijednost varijable `username` zapravo dohvaća cijelu skriptu koju ugrađuje u DOM na mjesto na koje je pokušavao upisati ime.

2.1.4. Prevencija XSS-a

Sve vrste XSS ranjivosti temelje se na tome da stranica ne provodi potrebnu kontrolu i validaciju korisničkog unosa. Postoji nekoliko smjernica kojih se Web developeri trebaju pridržavati kako bi spriječili unošenje XSS ranjivosti u svoj kod [38].

Potrebno je provesti odgovarajuću validaciju nad svim dijelovima stranice čiji sadržaj je pod kontrolom korisnika ili neke vanjske aplikacije. Ovisno o programskom jeziku postoje brojni paketi s preddefiniranim XSS filterima koje se može primijeniti na *user input*. Zatim potrebno je HTML šifrirati dio outputa koji se pojavljuje u HTTP odgovorima, a korisnik ima kontrolu nad njim. Primjer funkcije koja HTML šifrira nepovjerljiv korisnički input:

```
function htmlEncode(str){
  return String(str).replace(/[^\w. ]/gi, function(c){
     return '&#'+c.charCodeAt(0)+';';
  });
}

<script>
    document.body.innerHTML = htmlEncode(untrustedValue)
</script>
```

Također pametno je postaviti i `HTTPOnly` zastavicu u zaglavljima HTTP odgovora kojom se zabranjuje skriptama s klijentske strane da pristupaju tom kolačiću.

Postoji još velik popis pravila i dobrih praksi [39]te, no većina upada u neku od iznad navedenih smjernica. Ovisno o jeziku u kojemu se razvija stranica postoji mnoštvo radnih okvira (engl. *framework*) koji neke od opisanih provjera imaju ugrađeno i nije potrebna dodatna kontrola.

2.2. SQL ubacivanje

SQL ubacivanje (engl. *SQL injection, SQLi*) je ranjivost koja napadaču omogućuje da modificira upite koje aplikacija prosljeđuje svojoj bazi podataka i tako ostvari pristup podatcima kojima prvotno nije bio ovlašten pristupiti. To se odnosi na sve podatke kojima sama aplikacija može pristupiti što uključuje i podatke koji pripadaju drugim korisnicima. U mnogim slučajevima napadač može izmijeniti ili izbrisati ove podatke, uzrokujući trajne promjene u sadržaju ili ponašanju aplikacije. SQL ubacivanje spada u općenitiju kategoriju ubacivanje koja se nalazi prva na popisu OWASP top 10 ranjivosti iz 2017 godine [20].
U sklopu iskorištavanja ranjivosti SQL ubacivanja razlikuju se četiri tipa ubacivanja:

1. ubacivanje kojim se pristupa skrivenim podatcima
2. ubacivanje kojim se saznaju informacije o bazi podataka
3. ubacivanje UNION upita
4. ubacivanje na slijepo

1) Razmotrimo primjer u kojemu Web aplikacija korisnika traži da upiše ID te na temelju tog unosa kreira upit za bazu podataka i ispisuje ime i prezime odgovarajućeg korisnika. Aplikacija ne radi nikakve provjere na unesenom stringu i upit formira ovako:

```
$query =
"SELECT first_name, last_name
FROM users WHERE user_id = '$user_input';";
```

Napadač može napraviti upit koji će se uvijek evaluirati kao `TRUE` u `WHERE` dijelu upita i tako efektivno ispisati sve korisnike koji se nalaze u toj bazi. Za unos `' OR '1'='1` upit se evaluira u:

```
$query =
"SELECT first_name, last_name
FROM users WHERE user_id = '' OR '1'='1';";
```

2) Pomoću upita koji spadaju u ovu kategoriju napadač prikuplja informacije o bazi podataka koje mu mogu koristiti kod drugih upita. Na primjer za MySql bazu podataka informacije o tablicama u bazi mogao bi dobiti ako uspije formirati ovakav upit:

```
SELECT * FROM information_schema.tables
```

Isto tako napadač može saznati više informacija o predpostavljenom (engl. *default*) dijelu upita kojeg kreira sama stranica koristeći ubacivanje poput `' ORDER BY 1-`. Tako može postepeno inkrementirati redni broj atributa po kojem sortira rezultat, a jednom kada premaši broj atributa iz upita baza ispisuje pogrešku:
`The ORDER BY position number 3 is out of range of the number of items in the select list.`

3) Ubacivanjem UNION upita napadač može u potpunosti kreirati svoj upit kako želi. To znači da može dohvaćati i podatke iz drugih tablica. U kontekstu prošlog primjera napadač bi mogao zatvoriti početni upit jednim jednostrukim navodnikom i pomoću ključne riječi UNION kreirati bilo kakav drugi upit koji ima jedan broj atributa kao početni.

```
$query =
"SELECT first_name, last_name
FROM users WHERE user_id = '' OR '1'='1';
UNION
SELECT first_name, password FROM users";
```

Opisani postupak se može automatizirati i postoji mnoštvo alata koji to rade poput alata `sqlmap` [25].

4) Do slijepog SQL ubacivanja dolazi kad je Web aplikacija ranjiva na SQL, ali HTTP odgovori koje vraća ne sadržavaju rezultate relevantnog SQL upita ili detalje o pogreškama baze podataka. Sa slijepim SQL ubacivanjima mnoge tehnike poput UNION napada nisu učinkovite jer se oslanjaju na to da mogu vidjeti rezultate ubrizganog upita u odgovorima aplikacije. Još je uvijek moguće iskoristiti slijepo ubrizgavanje SQL-a za pristup neovlaštenim podacima, ali moraju se koristiti drugačije tehnike. Na primjer napadač može konstruirati neki upit za kojeg je siguran da se evaluira kao `FALSE` poput:

```
$query =
"SELECT first_name, last_name
FROM users WHERE user_id = '' OR '0'='1';";
```

Napadač zatim konstruira novi upit kojim pokuša saznati neku informaciju o shemi. Na primjer može provjeriti započinje li ime prve sheme u bazi sa slovom "a" kao što se to vidi u 2.2. To radi tako da na kraj naredbe koja je istinita doda logički operator `AND`

i upiše drugi dio naredbe čiju točnost želi provjeriti. Ako je dobiveni odgovor isti kao i za upit za kojega zna da se evaluira kao neistinit, onda zna da je i ovaj upit neistinit odnosno zna da ime prve sheme u bazi ne započinje slovom "a". Tako postepeno skuplja informacije i radi sve složenije upite.

Izvorni kod 2.2: Primjer slijepog SQL ubacivanja

```
$query =
"SELECT first_name, last_name
FROM users WHERE user_id = '1' and (
    SELECT TOP 1 substring(name, 1, 1) FROM sysobjects
    WHERE id=(
        SELECT TOP 1 id FROM (
            SELECT TOP 1 id FROM sysobjects ORDER BY id)
        AS subq ORDER BY id DESC)
    ) = 'a'
```

Kao i kod XSS-a prevencija napada ubacivanjem SQL-a temelji se na validaciji korisničkog unosa. Primjerice svi iznad navedeni primjeri ne bi funkcionirali ako bi aplikacija zabranila unos navodnika u polje u kojemu očekuje broj ID-a. U tom smislu korisno je popisati očekivane validne unose, a sve ostale odbiti (engl. *whitelisting*). Primjer *whitelistinga* u aplikaciji u kojoj je jedan od korisničkih unosa država SAD-a. Konstruiran je regularan izraz sa svim očekivanim vrijednostima za ime države, a bilokakv drugačiji unos se odbacuje.

```
^(AA|AE|AP|AL|AK|AS|AZ|AR|CA|CO|CT|DE|DC|FM|FL|GA|GU|
HI|ID|IL|IN|IA|KS|KY|LA|ME|MH|MD|MA|MI|MN|MS|MO|MT|NE|
NV|NH|NJ|NM|NY|NC|ND|MP|OH|OK|OR|PW|PA|PR|RI|SC|SD|TN|
TX|UT|VT|VI|VA|WA|WV|WI|WY)$
```

Dobra praksa je i Web aplikaciju konstruirati tako da se upiti koje su kreirali korisnici izvršavaju sa najmanjim mogućim privilegijama nad bazom za koje mogu obaviti sve potrebne radnje. Također u većini radnih okvira za razvoj poslužiteljske strane Web aplikacija postoje već ugrađene provjere.

2.3. Korištenje zastarjelih JavaScript biblioteka

JavaScript je skriptni programski jezik dizajniran za web, pretežito namijenjen izvršavanju na klijentskoj strani u pregledniku, no koristi se i za razvoj poslužiteljske strane. JavaScript je jedan od najkorištenijih programskih jezika te se većina modernih web usluga bazira upravo na njemu.

Prema podatcima iz 2019. godine postoji preko 1.6 milijarda web stranica od čega ih preko 95% koristi JavaScript [43]. Ti podatci se očituju i u broju paketa u npm-u, standardnom upravitelju paketa i biblioteka za JavaScript. Trenutno se u npm-u nalazi preko 1.3 milijuna JavaScript paketa [17].

Biblioteke same po sebi nisu ništa drugo nego JavaScript kod koji pruža određen set funkcionalnosti i samim time podložne su svim ranjivostima kojima je podložan i svaki drugi JavaScript kod. Čak štoviše funkcije iz tih biblioteka imaju potpun pristup DOM-u. S obzirom na to da kod iz biblioteka može utjecati na DOM, jasno je da putem potencijalno ranjivih biblioteka i napadači mogu kontrolirati DOM, mijenjati njegovu strukturu, izvršavati proizvoljni kod i slično.

Većina biblioteka su projekti otvorenog koda (engl. *open-source*) što znači da je stručnjacima za sigurnost, ali isto tako i napadačima omogućen pristup izvornom kodu što im olakšava traženje slabosti i ranjivosti. Upravo zbog toga nije rijetkost vidjeti da se u repozitorij paketa dodaju nove verzije biblioteka sa sigurnosnim zakrpama (engl. *patch*).

Iz ovoga je jasno da web developeri korištenjem neažurnih odnosno zastarjelih biblioteka mogu potencijalno unijeti ranjivosti u svoj kod. Koliko je to često slučaj i koliko je zapravo malo nepažnje potrebno da zbog toga kod postane nesiguran, govori činjenica da taj tip slabosti ima svoj službeni CWE zapis.
Korištenje zastarjelih JavaScript biblioteka spada pod slabost korištenja neodržavanih komponenta treće strane (engl. *Use of Unmaintained Third Party Components*) i nalazi se pod zapisom **CWE-1104** [6]. Slična stavka nalazi se i na OWASP top 10 popisu sigurnosnih rizika iz 2017. godine [20]. Na devetom mjestu je ***A9:2017**-Using Components with Known Vulnerabilities* [18] u što spadaju i ranjive biblioteke.

2.4. Korištenja ranjivih verzija CMS-ova

U početku weba pretežno su se koristile statične web stranice. Web stranice sastojale su se samo od html datoteka i nisu mijenjale svoj sadržaj. Kako je popularnost weba rasla, tako se pokazala potreba za uvođenjem responzivnih dinamičkih stranica koje mogu mjenjati svoj sadržaj ovisno o interakciji s korisnicima. Uvedene su nove mogućnosti poput dodavanja slika, spremanja podataka, prijenosa i preuzimanja datoteka. Zbog ubrzanog razvoja weba porasla je i potražnja za razvojem, održavanjem i upravljanjem web stranicama.

Danas gotovo svaka firma, akademska ustanova, pa čak i pojedinci imaju vlastite Web stranice. Problem je u tome što stranice nije lako napraviti. Često je to posao cijelog tima programera, dizajnera, arhitekta sustava itd. Upravo zbog toga su razvijeni CMS-ovi (engl. *Content Management System*). CMS je u širem smislu sustav za upravljanje sadržajima čija je uloga olakšati i organizirati procese vezane uz kreiranje i prijenos sadržaja.

Konkretno vezano za web CMS-ovi su programi odnosno alati koji programerima pomažu da brže i jednostavnije rade stranice tako da većinu komponenata stranice ne trebaju programirati od nule, nego im CMS nudi neka gotova rješenja. Neki od poznatijih i češće korištenih CMS-ova su: Wordpress [34], Joomla [14], Wix [33], Squarespace [26], Drupal [10], Bolt [3], Magento [15] itd.

U praksi CMS-ovi se vrlo često koriste baš zbog jednostavnosti koju pružaju. Naravno kao i sa svim drugim programima, prilikom izrade CMS-ova može doći do propusta koji uzrokuju ranjivosti.

Budući da su vrhunski CMS-ovi toliko popularni, ove sigurnosne ranjivosti se aktivno traže - kako istraživači sigurnosti, tako i napadači. Isto tako većina CMS-ova su programi otvorenog koda što znači da napadači imaju pristup izvornom kodu što im olakšava traženje slabosti te su im baš zbog toga CMS-ovi primamljive mete.
Jednom kad se identificiraju, ove se mane mogu pretvoriti u virtualni rudnik zlata za zlonamjerne hakere zbog toga što sve web stranice koje ne koriste ažurnu verziju CMS-a ostaju ranjive. Ovdje ostaje velika odgovornost na administratorima stranica da prate i primjenjuju aktualne zakrpe za sustave koje koriste, a pokazalo se da često to ne rade [29].

Najkorišteniji je Wordpress za kojega prema službenoj stranici WordPressa, trenutno postoji preko 55 000 dodataka (engl. *plugin*) [35]. *Pluginovi* za WordPress su programi koji sadrže skup dodatnih funkcionalnosti koje se može uključiti u WordPress stranicu. Zbog toga nije začuđujuće da je upravo u Wordpresu nađeno najviše ranji-

vosti u odnosu na ostale CMS-ove [31]. Do sada je pronađeno preko 22 000 ranjivosti Wordpressa [36]. Velika većina ranjivosti vezana je upravo za te dodatke a ne za sam Wordpress. Prema CVE podatcima, XSS je najveća prijetnja u WordPressu, nakon čega slijedi SQL injection [42].

Zbog svih iznad navedenih razloga smisleno je izgraditi programsku podršku koja će automatizirati proces detekcije korištenog CMS-a i njegove verzije te otkriti radi li se o ranjivoj verziji. Upravo to je razmatrano u praktičnom djelu rada.
Detalji o implementaciji detekcije ranjivih CMS-ova opisani su u odjeljku 3. 2.

2.5. Korištenje potencijalno nesigurnih JavaScript funckija

U JavaScriptu postoji nekoliko funkcija koje vrlo lako mogu dovesti do ranjivosti poput XSS-a temeljenog na DOM-u [2] i brojnih drugih te ih zbog toga treba izbjegavati. Te funkcije nisu ranjive same po sebi, no neopreznim rukovanjem u aplikacijama vrlo je lako napraviti sigurnosne propuste. Kako do toga dolazi opisano je u odjeljku 2.2.3.
Funkcije koje najčešće uzrokuju takve slabosti su:

- `eval(...)`
- `.innerHTML = ...`
- `.outerHTML = ...`
- `document.write(...)`
- `document.writeln(...)`

Da bi se dodatno zaštitili web developeri mogu koristiti preddefinirane XSS filtere.
Primjer ranjivog korištenja funkcije `.innerHTML()`

```
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.getElementById("Welcome")
.innerHTML=" Hello, "+user;
```

Sličan primjer za `document.write()`

```
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.write("<h1>Hello, " + user +"</h1>");
```

Oba primjera ne provode validaciju nad korisničkim unosom i napadač može u DOM ugraditi svoj zloćudan kod. Na primjer može ih iskoristiti tako da se unese URL kojim u stablo DOM-a pokušava staviti sliku sa nepoznatim izvorom i prilikom greške izvodi napdačev kod.

```
http://vulnerable.site/
page.html?user=<img%20src='aaa'%20onerror=alert(1)>
```

Primjer korištenja XSS filtera:

```
document.getElementById("temp")
.innerHTML=xssFilters
.inHTMLData(untrusted_ajax_res);
```

3. Tehnike detekcije ranjivosti weba

Razlikuju se statička [27] i dinamička [7] analiza Web aplikacija. Kod statičke analize promatra se izvorni kod stranice bez da se on izvršava. S druge strane kod dinamičke analize kod se izvršava te se prati njegova interakcija s okolinom, kakve sve HTTP zahtjeve šalje, s kime sve pokušava komunicirati, kojim sve lokalno spremljenim podatcima pokušava pristupiti i slično. Kod traženja slabosti u Web aplikacijama često se još spominje i Fuzzing.

Fuzzing ili fuzz testiranje [12] je automatizirana tehnika testiranja softvera u kojoj se generiraju nevaljani, neočekivani ili nasumični unosi u program te se prati kako aplikacija na njih reagira. Ovakav pristup posebno je povoljan za traženje XSS ranjivosti zbog toga što se većina XSS ranjivosti(osim onih baziranih na DOM-u) ne može uočiti iz koda dostupnog s klijentske strane. Tako alati za detekciju XSS-a generiraju uobičajene maliciozne unose kod XSS-a, predaju ih aplikaciji i bilježe rezultat.

U odjeljku 3.1 razmatraju se tehnike detekcije ranjivosti weba sa klijentske strane. To je pristup traženju ranjivosti koji se oslanja isključivo na kod koji se izvršava u Web pregledniku korisnika Web stranice čiju ranjivost se ispituje. Takav pristup je nešto zahtjevniji zbog toga što je velik broj ranjivosti Web stranica nastao kao posljedica grešaka u implementaciji poslužiteljske strane ili u komunikaciji poslužitelja i klijenta, a te informacije nisu dostupne s klijentske strane i ovim pristupom se ne razmatraju. Također dodatna poteškoća je što se na klijentskoj strani može potencijalno izvršavati jako velika količina koda što znači da za otkrivanje ranjivosti treba moći efikasno pretražiti veliku količinu podataka.

3.1. Detekcija ranjivosti weba s klijentske strane

Detekcijom s klijentske strane pokušavaju se otkriti ranjivosti analiziranjem izvornog koda klijentske strane Web aplikacije. U nastavku ovoga rada i u praktičnom djelu razmatrane su isključivo metode detekcije s klijentske strane. Rad se bavi detekcijom ranjivosti koje se traže i detektiraju na temelju:

- Zastarjelih i ranjivih JavaScript biblioteka,
- Ranjivih verzija CMS-ova,
- Nepravilnog korištenja nekih JavaScript funkcija koje rukuju s korisničkim unosom

S klijentske strane moguće je detektirati korištene zastarjele JavaScript biblioteke. Problem sa prepoznavanjem ranjivih verzija JavaScript biblioteka je taj što ne postoji centralizirana baza podataka s popisom svih ranjivosti i odgovarajućih verzija. Zbog toga puno alata kreira svoje varijante takvih kataloga. Jedan od takvih kataloga razvijen je u sklopu alata Retire.js [22]. Retire.js je alat koji se koristi u obliku Chrome i Firefox plugina, a cilj mu je na stranici otvorenoj u pregledniku prepoznati potencijalno zastarjele dijelove koda. Detekcija korištenja ranjivih biblioteka mogla bi se raditi ručno, no to je vrlo neefikasan i zamoran posao. Jedan od ciljeva praktičnog rada bio je tu pretragu automatizirati.

Isto tako na temelju izvornog koda klijentske strane moguće je detektirati uporabu CMS-a i pronaći njegovu verziju. Jednom kada se detekrira korišteni CMS može se provjeriti postoje li ranjivosti opisane u 4.8 za specifičnu verziju detektiranog CMS-a. Iz izvornog koda klijenstske strane se također lako mogu pronaći korištene funkcije pa među njima i neke od potencijalno opsanih funkcija koje mogu prouzrokovati ranjivosti opisane u 2.5 ako nisu pravilno korištene. Detalji o implementaciji detekcije ranjivih biblioteka, CMS-ova i opsanih funkcija opisani su u odjeljku 4.2.

4. Razvijeno programsko rješenje

U okviru završnog rada razvijen je program za detekciju ranjivih web stranica. Razvijeni program pokreće se iz naredbenog retka i može mu se predati nekoliko različitih zastavica prilikom pokretanja. Duži ispis svih opcija može se dobiti zastavicom `-h` ili `--help` dok se skraćeni ispis može dobiti pokretanjem programa bez opcija kao što se to vidi u 4.1.

Izvorni kod 4.1: Ispis opcija razvijenog programa

```
./vuln_detect.py --help
usage:
vuln_detect.py [-h] [-f FILE | -u URL | -d DIR] [-l] [-w] [-r]

Detect vulnerabilities in websites source code.

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Specify the path to javascript file.
  -u URL, --url URL     Specify the url of the website.
  -d DIR, --dir DIR     Specify the path to
                        root dir with js and html files.
  -l, --log             Save the output of script in log.txt
  -w, --warn            Show additional warnings.
  -r, --db              Update/refresh database
                        used for vulnerabilites.

./vuln_detect.py
usage:
vuln_detect.py [-h] [-f FILE | -u URL | -d DIR] [-l] [-w] [-r]
```

U nastavku su ukratko objašnjene funkcionalnosti pojedinih zastavica.

Zastavicom `-f` ili `-- file` programu se zadaje putanja do Javascipt ili HTML datoteke spremljene lokalno na računalu koju se želi skenirati za ranjivosti. Ako korisnik koji pokreće program nema ovlasti za čitati navedenu datoteku dobije poruku poput: `Error: could not read file /etc/shadow`

Zastavicom `-u` ili `-- url` programu se predaje URL stranice koju se želi skenirati. Program na temelju url-a provjerava postoji li stranica u bazi podataka. Ukoliko se ne navede protokol, `http` ili `https` prije samoga url-a program redom pokušava dohvatiti stranicu kao https://url i https://url/ te ako ne uspije pokušava http://url i http://url/.
Ako ne postoji zapis za tu stranicu u bazi podataka, program istim redom radi HTTP GET zahtjev (engl. *request*) na navedeni url i iz odgovora (engl. *response*) čita sadržaj stranice. Ako program ne uspije dohvatiti traženi resurs ispisat će adekvatnu grešku. Isto tako prilikom svakog uspješnog preuzimanja stranice program ispisuje odakle je preuzet sadržaj stranice, sa weba ili iz baze podataka.

Zastavicom `-d` ili `-- dir` programu se može predati putanja do korijenskog (engl. *root*) direktorija sa JavaScript i HTML datotekama. Program će ranjivosti pretraživati rekurzivno u dubinu iz korijenskog direktorija po svim datotekama koje imaju `. js` ili `. html` ekstenziju. Ova zastavica primarno je namijenjena web developreima da na jednostavan način mogu skenirati cijeli svoj projekt za vrijeme izgradnje tako da program pokrenu iz root direktorija projekta na kojemu rade.

Zastavicom `-l` ili `-- log` specifira se da sav izgenerirani ouput program spremi u log datoteku. Ukoliko ne postoji, program će stvoriti direktorij logs u direktoriju u kojemu je pozvan. Tamo će spremati log datoteke u `.txt` formatu s imenom oblika `godina-dan-mjesec_XhrsYmins`, gdje su X i Y broj sati odnosno broj minuta, dakle npr. `2021-10-5_14hrs25mins. txt`

Zastavicom `-w` ili `-- warn` programu specificiramo da želimo da nas upozori za korištenje funkcija koje mogu dovesti do ranjivosti ako ne barataju sa korisničkim unosom pravilno (funkcije razmatrane u odjeljku 3. 1. 3.). Ova funkcionalnost također je razvijena imajući web developere na umu. Budući da program ne može sa sigurnošću utvrditi da se navedene funkcije koriste na nesiguran način, web developeri mogu ovako jednostavno provjeriti gdje su sve koristiili takve funkcije. Program ispisuje redak u datoteci u kojoj se pojavljuje ta funkcija i upozorenje da se provjeri

je li *input* saniran (engl. *sanitized*). Saniranje ulaza je postupak provjere, čišćenja i filtriranja unosa korisničkih podataka bilo kakvih neželjenih znakova i nizova kako bi se spriječilo ubacivanje zloćudnih kodova.

Zastavicom `-r` ili `-- db` da želimo osvježiti odnosno ažurirati bazu podataka koja se koristi za popis ranjivih biblioteka. Ako se navede ova zastavica program briše stari `jsrepository.json` file i s weba preuzima novi.

Također u popisu zastavica vidi se koje su međusobno isključive, no u slučaju nepravilnog pozivanja programa ispiše se adekvatna poruka poput:
`vuln_detect.py: error: argument -u/-url: not allowed with argument -f/-file`

4.1. Opis rješenja, korištenih alata i tehnologija

Program je razvijen u programskom jeziku Python. Korištena je verzija Pythona `3.8.5`.

Na FER-u je u istraživačke svrhe koristeći web pauka (engl. *web crawler*) razvijena baza podataka koja sadržava preko 2 milijuna zapisa o prikupljenim URl-ovima i odgovarajućim stranicama.
To je MongoDb baza i za pristup njoj unutar programa korišten je pymongo modul. Baza nije javno dostupna pa je prije pokretanja programa potrebno ostvariti `SSH` pristup. URL-ovi se nalaze u kolekciji `crawled_data_urls_v0` dok se stranice nalaze u `crawled_data_pages_v0` . Na slikama ispod prikazane su zapisi za stranicu FER-a.

Za svaki URL spremaju se još i podatci poput HTTP zaglavlja (engl. *headers*), HTTP statusa odgovora i kriptografski sažetak (engl. *hash*). Za stranice koje se posjete više puta podatci iz svakog posjeta spremaju se u array `checks`.
Na slici ispod vidi se zapis za istu stranicu, odnosno zaspis sa istom vrijednosti hash-a u `crawled_data_pages_v0`. Iz tog zapisa može se dohvatiti cijeli sadržaj stranice.

Za detekciju zastarjelih biblioteka korišten je dokument `jsrepository.json` preuzet iz repozitorija ranije spomenutog Retire.js-a.

```
_id: ObjectId("603a863739ec133a07afe968")
url: "https://www.fer.unizg.hr"
checks: Array
  0: Object
    timestamp: 1614448179.9576273
    headers: Object
      Date: "Sat, 27 Feb 2021 17:49:40 GMT"
      Server: "Apache/2.4.18 (Ubuntu)"
      X-MST3k: ":Mulder: Was it the same "they" who gave you those bikes? :Crow: No, I..."
      Expires: "Thu, 19 Nov 1981 08:52:00 GMT"
      Cache-Control: "no-store, no-cache, must-revalidate"
      Pragma: "no-cache"
      P3P: "CP="NOI CURa ADMa DEVa TAIa PSAa PSDa IVAa IVDa HISa OTPa OUR BUS IND ..."
      Content-Encoding: "gzip"
      Vary: "Accept-Encoding"
      Set-Cookie: "CMS=35eak4a6apt10sgi2ogs19fsa6; expires=Sun, 07-Mar-2021 05:49:40 GMT;..."
      Strict-Transport-Security: "max-age=360"
      Keep-Alive: "timeout=5, max=100"
      Connection: "Keep-Alive"
      Transfer-Encoding: "chunked"
      Content-Type: "text/html; charset=utf-8"
    hash: "f2987ce3d265d5084dc8c7a746789056a8b584b14e9a19d0d04e01f8d3e81d20"
    status: "ok"
    params: ""
    query: ""
    fragment: ""
```


Slika 4.1: Zapis za url `https://www.fer.unizg.hr` u `crawled_data_urls_v0`


```
_id: ObjectId("603a863739ec133a07afe964")
hash: "f2987ce3d265d5084dc8c7a746789056a8b584b14e9a19d0d04e01f8d3e81d20"
checks: Array
last_check: 1614448179.9576273
page: "<!DOCTYPE HTML>
<html lang="hr" class="htmlcms">
<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Language" content="hr" />
<meta name="generator" content="Quilt CMS 2.5, https://www.fer.unizg.hr/quilt-cms" />

<!--meta name="robots" content="noindex" /-->
<meta name="keywords" content="" />

<title>Fakultet elektrotehnike i računarstva</title>
```


Slika 4.2: Odgovarajući zaspis u `crawled_data_pages_v0`

U Json zapisu za svaku ranjivost neke biblioteke postoji zapis na koje verzije se odnosi ta ranjivost. Za neke ranjivosti zadan je samo `below` parametar, dakle ranjive su sve verzije manje od navedene, dok je za druge zadana i donja granica parametrom `atOrAbove` što indicira da je ranjivost zakrpana. Primjer jednog takvog zapisa vidi se u 4.2.

Izvorni kod 4.2: Oblik zapisa u `jsrepository.json`

```
{
"retire-example": {
"vulnerabilities" : [
{
        "below" : "0.0.2",
        "severity" : "low",
        "identifiers" : {
                "CVE" : [ "CVE-XXXX-XXXX" ],
                "bug" : "1234",
                "summary" : "bug summary"
        },
        "info" :
        [ "http://github.com/eoftedal/retire.js/" ]
}
],
"extractors" : {
        "func" :
        [ "retire.VERSION" ],
        "filename" :
        [ "retire-example-(§§version§§)(.min)?\\.js" ],
        "filecontent"        :
        [ "/\\*!? Retire-example v(§§version§§)" ],
        "hashes" :
        { "07f8b94c8d601a24a1914a1a92bec0e4fafda964" : "0.0.1" }
}
},
```

Za lakše parsiranje izvornog koda preuzetih stranica korištena je biblioteka `beautifulsoup4`.

Detektirati CMS korišten na stranici često nije problem za čovjeka, no automatizirati taj proces nije trivijalno. Nekad čak nije uopće moguće iz izvornog koda prepoznati o kojem CMS-u je riječ, a svatko tko pogleda stranicu vidjet će npr. veliku reklamu u obliku slike na dnu stranice. Prvi pristup bio je ime i verziju CMS-a tražiti u HTML tagovima oblika `meta name="generator" content="CmsName version"`, no tim pristupom program je uspješno raspoznao vrlo mali postotak CMS-ova pa je

taj pristup odbačen. Nekada se ime može raspoznati i iz oblika putanja do skriti koje stranica koristi, no ovdje postoji mnogo odstupanja i svaki CMS se drugačije ponaša. Iz ovoga je jasno da samo izvorni kod nije dovoljno za pouzdanu detekciju.

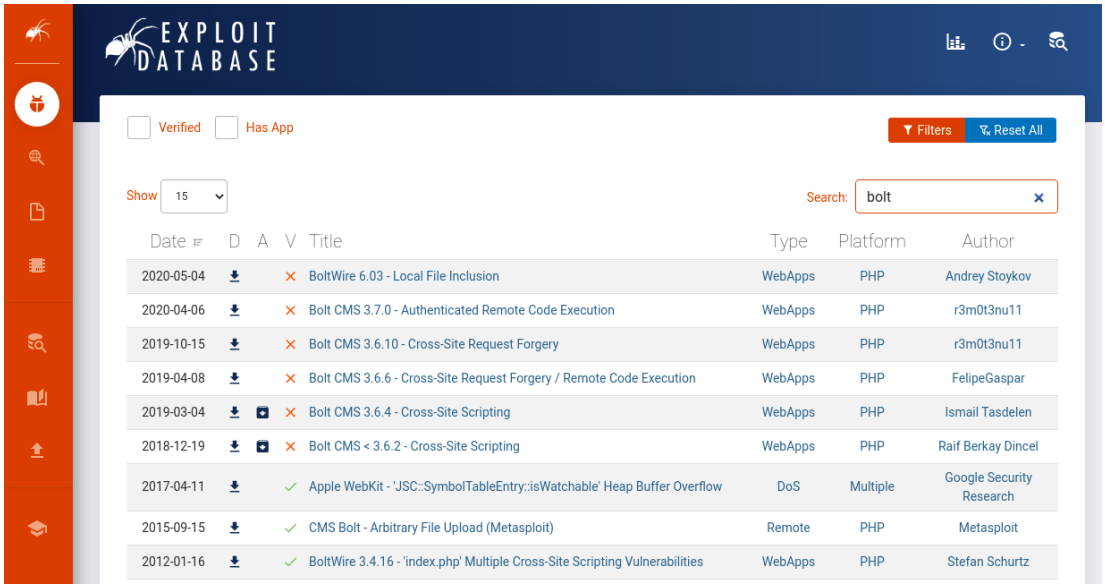
Problem je riješen koristeći Python modul `python-Wappalyzer` . Taj modul je

```
<!DOCTYPE HTML>
<html lang="hr" class="htmlcms">
<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Language" content="hr" />
<meta name="generator" content="Quilt CMS 2.5, https://www.fer.unizg.hr/quilt-cms" />

<!--meta name="robots" content="noindex" /-->
<meta name="keywords" content="" />

<title>Fakultet elektrotehnike i računarstva</title>
```

Slika 4.3: FER-ova stranica koristi `Quilt CMS` verzije `2.5`

zapravo Python implementacija popularnog alata Wappalyzer koji služi za detekciju korištenih tehnologija na stranici poput u kojem jeziku je stranica napisana, je li spojena na neku bazu podataka, koje fontove koristi itd. Biblioteka radi HTTP zahtjeve na zadani URL te na temelju HTTP zaglavlja odgovora, strukturi URL-a, meta tagova u izvornom kodu. U razvijenom programu ova biblioteka je korištena za detekciju CMS-a.

Slično kao i s bibliotekama, kod CMS-ova je problem što ne postoji službeni popis svih ranjivosti pojedinih CMS-ova kojega bi bilo jednostavno pretraživati.
Ovaj problem svladan je tako da, jednom kada program otkrije korišteni CMS i njegovu verziju, pretražuje bazu exploita. Logika iza toga je da ukoliko postoji exploit postoji i ranjivost. Obrat naravno ne vrijedi. Korištena je ExploitDB[11] baza podataka, najpoznatija baza exploita razvijena i održavana od tvrtke Offensive Security.
Osim putem stranice napravljen je i alat za pretragu po toj bazi iz naredbenog retka. Alat se zove `searchsploit` [23] i dolazi predinstaliran na Kali linux distribuciji.



Slika 4.4: ExploitDB baza podataka

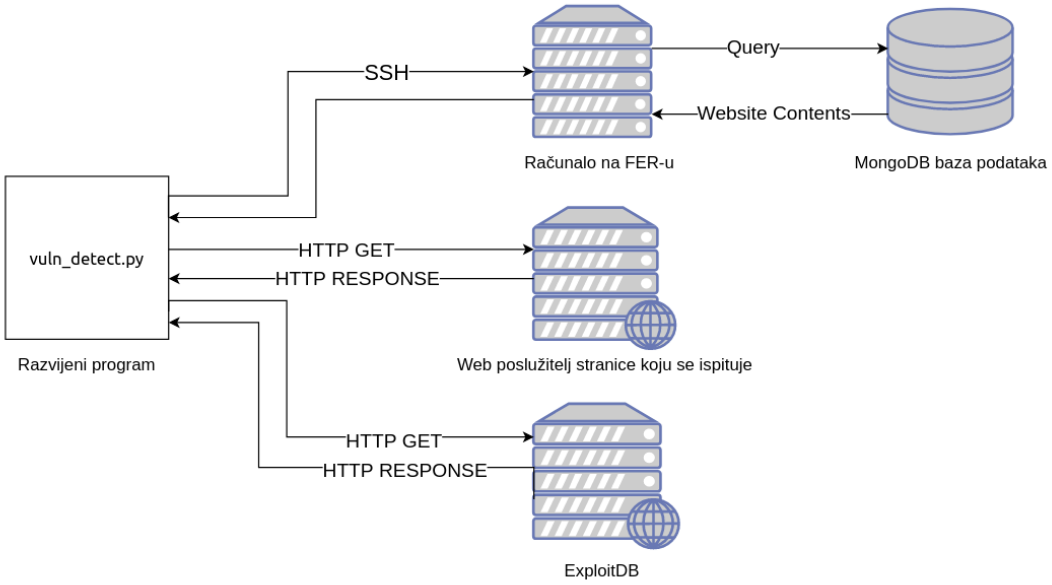
4.2. Implementacija

U ovom odjeljku opisani su detalji implementacije razvijenog programa. Skica arhitekture citavog sustava na višoj razini se vidi na slici 4.5.

Ako se programu navede `--url` pokušava dohvatiti sadržaj stranice kao što je ranije opisano. Prioritizira dohvaćanje iz baze podataka te samo ako nema zapisa za predani URL u bazi podataka dohvaća stranicu preko HTTP-a. Ako korisnik želi preuzimati stranice iz baze treba kreirati datoteku `database_login_data.txt` u koju treba staviti connection string s korisničkim imenom i lozinkom za spajanje na MongoDB bazu.

Jednom kad je preuzet sadržaj stranice s weba ili iz baze ili iz lokalno spremljene datoteke kreira se `BeautifulSoup` objekt koristeći taj sadržaj te se on dalje predaje funkcijama za detekciju ranjivosti.

Glavnina funkcionalnosti vezane za detekciju ranjivih biblioteka raspoređena je u tri funkcije: `find_vulnerable_libraries()`, `check_vulnerable_lib()` i `check_vulnerable_lib_version()`.
U funkciji `find_vulnerable_libraries(soup)` 4.3, program prolazi po pronađenim `<script>` tagovima iz izvornog koda i za svakog poziva `check_vulnerable_lib()` sa `src` vrijednošću tog taga.



Slika 4.5: Skica arhitekture sustava

Izvorni kod 4.3: funkcija `find_vulnerable_libraries`

```
1  def find_vulnerable_libraries(soup):
2      libs = []
3
4      script_tags = soup.findAll('script', {"src": True})
5      for script_tag in script_tags:
6          lib_name = check_vulnerable_lib(script_tag.get('src'))
7          if lib_name is not None:
8              libs.append(lib_name)
9      if not libs:
10         print(" [+]  No vulnerable libraries found.")
11         if args.log: log_data([" [+]  No vulnerable libraries found."])
```

Za shvatiti kako radi funkcija `check_vulnerable_lib()` prvo je potrebno razmotriti na koje se sve načine vanjske biblioteke i skripte mogu uključiti u JavaScript kod. Naime, biblioteke se mogu iz ranije navedenih upravitelja paketima preuzeti lokalno. Zatim da bi koristili određenu biblioteku potrebno je navesti relativnu putanju do nje unutar script taga. Primjer:

```
<script src="jquery-3.5.1.min.js"></script>
```

Drugi i ujedno češći način uključivanja biblioteka je koristeći CDN-ove.

CDN (Content distribution network) je zapravo skupina servera koja služi da bi osigurala visoku dostupnost i performanse, pružajući između ostalog i JavaScript biblioteke. Veće firme poput Googla i Microsofta imaju vlastite CDN-ove. Čak i neke popularnije biblioteke imaju svoje CDN-ove.(jquery, bootstrap...). Jedna od mogućnosti CDN-ova je „version alisaing", svojstvo CDN-ova koje omogućuje programeru da navede samo prefiks verzije biblioteke koju želi koristiti, a CDN će se pobrinuti da koristi najažurniju sa tim prefiksom. Ovo vrijedi samo za biblioteke koje su nakon zakrpe kompatibilne unatrag (engl. *backwards compatible*). Primjer uključivanja jquery biblioteke u kod putem Googleovog CDN-a:

```
<script src="https://ajax.googleapis.com/
ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

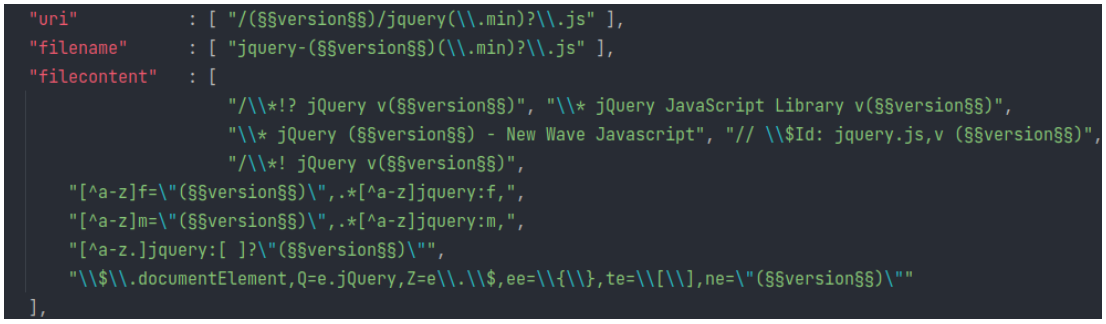
S obzirom da ne postoji neko univerzalno pravilo kako se zadaju biblioteka i njena verzija odnosno u kojem formatu je src vrijednost script taga dosta je teško automatizirati proces koji će izvući točno ime i verziju.
Generalno JavaScript biblioteke koriste sljedeću konvenciju za imenovanje verzija: `major.minor.patch` npr. `1.0.1`.
Prvi, naivni pristup bio je pokušati iz `src` vrijednosti script taga isparsirati ime biblioteke i njezinu verziju, no ubrzo sam uočio koliko odstupanja od pravila postoji pa je ta ideja odbačena.
Za taj problem poslužio je `jsrepository.json`.
U tom JSON-u se osim ranjivih bibliteka nalaze i zapisi koji opisuju kako izgleda URI korišten za uključivanje tih biblioteka putem CDN-ova i najčešći oblik imena datoteke kada se uključuje lokalno. Problem je što su zapisi koji se nalaze u toj datoteci u obliku specifičnom ta `Retire.js`. Npr sa `§§version§§` označeno je predviđeno mjesto za regex koji opisuje verziju biblioteke.

Da bi riješio taj problem u programu je korišten Pythonov `re` modul za rukovanje s regexima te su zapisi iz `Retire.js` pretvoreni u regexe s kojima Python rukovati.
Npr. zapis is `jsrepository.json`

```
"/(§§version§§)/jquery(\\.min)?\\.js"
```

koristeći funkciju `replace` prelazi u regex koji se može koristiti za prepoznavanje URI-ja korištenog za uključivanje `jQuery` biblioteke.

```
/([0-9][0-9.a-z_\\\\-]+)/jquery(\\.min)?\\.js
```



Slika 4.6: Primjer zapisa regularnih izraza za uključivanje jQuery biblioteke

Prije nego započne traženje ranjivih biblioteka, program preuzima zadnju verziju `jsrepository.json` ako je korisnik tako specificirao.
To se može vidjeti u ispisu 4.4. Program zatim iterira po svim popisanim ranjivim bibliotekama, i svim bibliotekama detektiranima na stranici te pokušava pronaći podudarnost. Jednom kad nađe podudarnost znači da se u zadanoj stranici koristi biblioteka za koju postoji barem jedna ranjiva verzija. Zatim program detektira korištenu verziju i poziva funkciju
`check_vulnerable_lib_version(lib_version, vuln)`.

Izvorni kod 4.4: Odsječak koda zadužen za traženje podudarnosti detektirane biblioteke i neke od ranjivih

```
for library in vuln_libs_json:

    # scan for libraries imported localy on website without CDN
    try:
        filename_pattern_list = \
        = vuln_libs_json[library]["extractors"]["filename"]
    except KeyError:
        continue

    for lib_filename_regex in filename_pattern_list:
        lib_filename_regex = lib_filename_regex.replace(
            u"§§version§§",
            "[0-9][0-9.a-z_\\\\-]+")
        match = re.search(lib_filename_regex, script_tag_src)
        if match:
            lib_name_ret = library
            lib_version = match.group(1)
            lib_version = re.search("([0-9.]+[0-9])",
                          lib_version).group(0)
            print(" >>  library detected:", lib_name_ret)
            print(" >>  version:", lib_version)
```

```
22
23              for vuln in vuln_libs_json[library]["vulnerabilities"]:
24                  if check_vulnerable_lib_version(lib_version, vuln):
25                      formatted_print_vuln_lib(
26                      vuln, lib_name_ret, lib_version)
27
28
29  return lib_name_ret
```

Funkcija `check_vulnerable_lib_version(lib_version, vuln)` 4.5, provjerava je li detektirana verzija neka od ranjivih odnosno je li manja od verzije biblioteke u kojoj je ranjivost zakrpana i veća od verzije biblioteke u koju je ranjivost prvo unesena.

Izvorni kod 4.5: Funkcija `check_vulnerable_lib_version`

```
def check_vulnerable_lib_version(lib_version, vuln):
    global args
    is_vulnerable_flag = False
    vulnerable_below, vulnerable_at_or_above = None, None

    # "below" argument is provided
    # for every vulnerability,
    #atOrAbove for some
    if "below" in vuln.keys():
        vulnerable_below = vuln["below"]
    if "atOrAbove" in vuln.keys():
        vulnerable_at_or_above = vuln["below"]
    # vulnerabily given as
    # range of versions
    # below_ver > ver >= at_or_above
    if vulnerable_at_or_above is not None:
        if version.parse(vulnerable_below) >
        version.parse(lib_version) >=
        version.parse(vulnerable_at_or_above):
            return True

    # all versions below ceratain version are vulnerable
    elif version.parse(lib_version) < version.parse(vulnerable_below):
        return True
    return False
```

Funckija `formatted_print_vuln_lib()` za pronađernu ranjivost ispisuje:

- Kratki sažetak o kakvoj je ranjivosti riječ
- `CVE` te ranjivosti ako postoji
- ozbiljnost (engl. *severity*) ranjivosti
- dodatne informacije poput linka na github push zakrpe te ranjivosti ili npr. blog istraživača koji je otkrio ranjivost

Funkcionalnost za detekciju ranjivih CMS-ova je implementirana u funkciji `find_vulnerable_cms(url)` 4.6. Prvo pomoću Wappalyzer biblioteke program pokušava identificirati je li stranica izgrađena pomoću nekog CMS-a i ako da koja verzija je korištena. Ako uspješno prepozna CMS i verziju program poziva alat `searchsploit` kojemu predaje te informacije.

Izvorni kod 4.6: funkcija `find_vulnerable_cms`

```
def find_vulnerable_cms(url):
    warnings.simplefilter("ignore")
    wappalyzer = Wappalyzer.latest()
    try:
        webpage = WebPage.new_from_url(url)
    except Exception:
        print("Error while detecting cms.")
        sys.exit(1)
    print("Retrieved website from: ", url)
    site_components = wappalyzer
    .analyze_with_versions_and_categories(webpage)
    CMS_name, CMS_version = None, None
    ...
    if CMS_name is not None:
        print("\n" * 4)
        print(" [+] CMS detected: " + CMS_name)
        print(" [+] version: " + "".join(CMS_version))

        if CMS_version != "unknown":
            print(" [+] Searching for
            " vulnerabilties and matching exploits...")
            subprocess_var = \
            subprocess.Popen(
                                "searchsploit "
                                + CMS_name + " " + CMS_version,
                                shell=True,
                                stdout=subprocess.PIPE)
            subprocess_return = subprocess_var.stdout.read()
```

4.3. Primjeri korištenja i rezultati

Zadavanjem URL-a program možemo pozvati npr. na Web stranicama Arheološkog muzeja u Zagrebu kao što se to može vidjeti u 4.7. Detektirana je ranjiva verzija bibliteke jquery, a stranica je preuzeta putem weba. Za tu verziju biblioteke detektiranje su tri ranjivosti i ispisani su detalji o svakoj od te tri. Npr. detektirana je ranjivost koja ima CVE zapis `CVE-2020-11023` [4]. To je ranjivost u kojoj je putem `.htmlPrefilter-a` moguće izvršiti XSS. Čak dvije od tri detektirane ranjivosti dovode do XSS-a. Zbog zastavice `--log` kreirana je i datoteka `2021-28-5_20hrs2mins.txt` u kojoj se nalazi isti ispis.

Izvorni kod 4.7: Primjer poziva programa na stranicama Arheološkog muzeja grada Zagreba.

```
./vuln_detect.py --url https://www.amz.hr/hr/naslovnica/ --log

No such entry in database,
retrieving website contents via http get...
-------------------------------------------------
Scan results for site:
https://www.amz.hr/hr/naslovnica/

 >>  library detected: jquery
 >>  version: 2.2.4
=================================================
 [+] Vulnerable library detected: jquery
 [+] Vulnerable version: 2.2.4
 [+] summary: jQuery before 3.4.0,
    as used in Drupal,
    Backdrop CMS, and other products,
    mishandles jQuery.extend(true, {}, ...)
    because of Object.prototype pollution
 [+] CVE:CVE-2019-11358
 [+] severity: medium
 [+] info:
 https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/
 https://nvd.nist.gov/vuln/detail/CVE-2019-11358
 https://github.com/jquery/jquery/
 commit/753d591aea698e57d6db58c9f722cd0808619b1b

=================================================
 [+] Vulnerable library detected: jquery
 [+] Vulnerable version: 2.2.4
 [+] summary: Regex in its jQuery
    .htmlPrefilter sometimes may introduce XSS
 [+] CVE:CVE-2020-11022
 [+] severity: medium
 [+] info:
 https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/

=================================================
 [+] Vulnerable library detected: jquery
 [+] Vulnerable version: 2.2.4
 [+] summary: Regex in its jQuery
    .htmlPrefilter sometimes may introduce XSS
 [+] CVE:CVE-2020-11023
```

```
 [+] severity: medium
 [+] info:
 https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/

Retrieved website from:  https://www.amz.hr/hr/naslovnica/
```

Sličan primjer pozivanja programa sa stranicom koja je izgrađena uz pomoć CMS-a može se vidjeti u 4.8. U ispisu se vidi da je program preuzeo stranicu putem Weba i detektirao korištenje WordPress-a i to verziju 5.7.2. Pretražio je bazu `exploitDB` za tu verziju WordPress-a i ispisao pronađene *exploite*.

Izvorni kod 4.8: Poziv programa za stranicu rađenu u WordPressu

```
./vuln_detect.py --url https://barmen.hr/

No such entry in database, retrieving website contents via http get...
------------------------------------------------
Scan results for site: https://barmen.hr/
Retrieved website from:  https://barmen.hr/

============================================================
| [+] CMS detected: WordPress                              |
| [+] version: 5.7.2                                       |
| [+] Searching for vulnerabilties and matching exploits... |
============================================================
           |              |             |
           |              |             |
           |              |             |
           V              V             V
-----------------------------------------------------------------------
 Exploit Title                                                         |
-----------------------------------------------------------------------|
WordPress Plugin DZS Videogallery < 8.60 - Multiple Vulnerabilities |
WordPress Plugin iThemes Security < 7.0.3 - SQL Injection           |
WordPress Plugin Rest Google Maps < 7.11.18 - SQL Injection         |
-----------------------------------------------------------------------


Shellcodes: No Results
```

Razvijeni program je pokrenut na 100 stranica kako bi se mjerile performanse. U prosjeku mu je za analizu jedne stranice potrebna oko jedna sekunda osim ako je stranica nedostupna, onda pokušava više puta i čeka *timeout* od 5 sekundi. Program

generalno vrlo pouzadno detektira sve biblioteke i dosta pouzdano pronalazi ranjive među njima. Od CMS-ova program uspješno prepoznaje samo one poznatije i često korištene dok one manje popularne često ne uspije detektirati.

4.4. Ograničenja implementiranog rješenja

Proizvedeni program provodi detekciju ranjivosti isključivo s klijentske strane i u tom smislu ne može otkriti sve ranjivosti koje se događaju na poslužiteljima ili u komunikaciji poslužiteljske i klijentske strane. Samim time program ne može direktno otkriti ranjivosti poput XSS-a i SQL ubacivanja, no može otkriti korištenje biblioteka i CMS-ova koji sadržavaju takve ranjivosti. S obzirom da ne postoji centralizirani službeni popis svih ranjivih JavaScript biblioteka program uvelike ovisi o bazi podataka koju koristi i o njenoj ažurnosti. Drugim riječima moguće je da program detektira biblioteku ali ju ne vidi kao ranjivu zato što neka novoobjavljena ranjivost još uvjek nema zapis u bazi podataka koju program koristi. Isto tako jedna od mana razvijenog programa jest da kad detektira ranjivu biblioteku program to odmah smatra ranjivošću bez da se uvjeri da je korišten baš ranjivi dio bibliotke, za to je potrebna daljnja ručna provjera. Također može se pojaviti problem ako Web stranica koristi modificirane verzije biblioteka koje lokalno sadrži. Takve modifikacije mogu unijeti nove ranjivosti ili ukloniti prijašnje. S obzirom na to da program koristi nazive JavaScript biblioteka za određivanje verzije, problem može uzrokovati i promijena imena biblioteka primjerice ako piše naziv biblioteke bez naznačene verzije (npr. "jquery.js").

5. Zaključak

Ranjivosti Weba su česte i vrlo lako ih je slučajno unijeti u kod prilikom programiranja. Upravo zbog toga je potrebno poznavati takve ranjivosti i pridržavati se dobrih praksi i pravila programiranja. Također velik broj aplikacija oslanja se na kod drugih programera koristeći razne biblioteke i alate. Tako programeri mogu nesvjesno unijeti ranjivosti u svoj kod putem komponenata koje oni sami nisu razvijali. Zbog toga potrebno je održavati razvijene aplikacije i redovito ažurirati sve korištene biblioteke i alate. Kako bi se taj proces olakšao i ubrzao mogu se koristiti programi poput onog razvijenog u sklopu ovoga rada. Razvijeni program traži ranjivosti na temelju zastarjelih biblioteka i CMS-ova. Program korištenje takvih ranjivih biblioteka i CMS-ova traži u izvornom kodu klijentske strane Web aplikacije odnosno u kodu koji se izvršava u pregledniku korisnika. Pokazalo se da program opisane ranjivosti detektira dosta pouzdano i značajno ubrzava takvu pretragu u odnosu na neautomatizirano traženje ranjivosti. Za još ispravniju detekciju ranjivosti poželjno je koristiti i alate koji koriste različite metode detekcije i ne oslanjaju se samo na izvorni kod s klijentske strane.

LITERATURA

[1] 2020 vulnerability statistics report. https://www.edgescan.com/edgescans-2020-vulnerability-stats-report-released/. 24.5.2021.

[2] Dom based xss prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheatq_Sheet.html". 24.5.2021.

[3] Bolt. https://boltcms.io/. 25.5.2021.

[4] Cve-2020-11023 detail. https://nvd.nist.gov/vuln/detail/CVE-2020-11023. 25.5.2021.

[5] Common weakness enumeration. https://cwe.mitre.org/, . 23.5.2021.

[6] Cwe-1104: Use of unmaintained third party components. https://cwe.mitre.org/data/definitions/1104.html, . 23.5.2021.

[7] Dynamic program analysis. https://en.wikipedia.org/wiki/Dynamic_program_analysis. 25.5.2021.

[8] Introduction to the dom. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction, . 25.5.2021.

[9] Dom-based xss. https://portswigger.net/web-security/cross-site-scripting/dom-based, . 25.5.2021.

[10] Drupal. https://www.drupal.org/. 25.5.2021.

[11] Exploit database. https://www.exploit-db.com/. 30.5.2021.

[12] Fuzzing. https://owasp.org/www-community/Fuzzing. 25.5.2021.

[13] Global digital population as of january 2021. https://www.statista.com/statistics/617136/digital-population-worldwide/. 22.5.2021.

[14] Joomla. https://www.joomla.org/. 25.5.2021.

[15] Magento. https://magento.com/. 25.5.2021.

[16] The mitre corporation. https://www.mitre.org/. 26.5.2021.

[17] So long, and thanks for all the packages! https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages.html. 22.5.2021.

[18] A9:2017 using components with known vulnerabilities. https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities, . 23.5.2021.

[19] Owasp. https://owasp.org/, . 26.5.2021.

[20] Owasp top 10, 2017. https://owasp.org/www-project-top-ten/, . 21.5.2021.

[21] Reflected xss. https://portswigger.net/web-security/cross-site-scripting/reflected. 25.5.2021.

[22] Retire.js - what you require you must also retire. https://retirejs.github.io/retire.js/. 30.5.2021.

[23] Searchsploit – the manual. https://www.exploit-db.com/searchsploit. 29.5.2021.

[24] Same-origin policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. 25.5.2021.

[25] Sqlmap. https://github.com/sqlmapproject/sqlmap. 25.5.2021.

[26] Squarespace. https://www.squarespace.com/. 25.5.2021.

[27] Static program analysis. https://en.wikipedia.org/wiki/Static_program_analysis. 25.5.2021.

[28] Stored xss. https://portswigger.net/web-security/cross-site-scripting/stored. 25.5.2021.

[29] 2019 website threat research report. https://sucuri.net/reports/2019-hacked-website-report/. 30.5.2021.

[30] 77% of 433,000 sites use vulnerable javascript libraries. https://snyk.io/blog/77-percent-of-sites-still-vulnerable/. 24.5.2021.

[31] The state of web application vulnerabilities in 2018. https://www.imperva.com/blog/the-state-of-web-application-vulnerabilities-in-2018/, . 23.5.2021.

[32] Web application vulnerabilities. https://www.rapid7.com/fundamentals/web-application-vulnerabilities/, . 21.5.2021.

[33] Wix. https://www.wix.com/. 25.5.2021.

[34] Wordpress. https://wordpress.com/. 25.5.2021.

[35] Wordpress plugins. https://hr.wordpress.org/plugins/. 30.5.2021.

[36] Wordpress vulnerabilities statistics. https://wpscan.com/statistics. 24.5.2021.

[37] Cross site scripting (xss). https://owasp.org/www-community/attacks/xss/, . 23.5.2021.

[38] How to prevent xss. https://portswigger.net/web-security/cross-site-scripting/preventing, . 25.5.2021.

[39] Cross site scripting prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html, . 25.5.2021.

[40] Types of xss. https://owasp.org/www-community/Types_of_Cross-Site_Scripting, . 24.5.2021.

[41] Study: Cross-site scripting nearly 40% of all online cyber attacks in 2019. https://www.msspalert.com/cybersecurity-research/study-cross-site-scripting-attacks/, . 23.5.2021.

[42] Robert Abela. Statistics highlight the biggest source of wordpress vulnerabilities. https://www.wpwhitesecurity.com/statistics-highlight-main-source-wordpress-vulnerabilities/. 24.5.2021.

[43] Eric Elliott. How popular is javascript, 2019. https://medium.com/javascript-scene/how-popular-is-javascript-in-2019-823712f7c4b1. 20.5.2021.

Tehnike otkrivanja ranjivosti web sjedišta s klijentske strane

Sažetak

U današnje vrijeme Web je jedan od najznačajnijih sredstava komunikacije i poslovanja. Zbog raznih greška i manjkavosti u implementaciji, dizajnu, upotrebi ili upravljanju web aplikacijama dolazi do ranjivosti. Iskorištavanjem takvih ranjivosti napadači mogu ostvariti pristup poslužiteljima na kojima su ta web sjedišta smještena i/ili računalima i Web preglednicima posjetitelja. Zbog toga je potrebno poznavati takve ranjivosti, kako do njih dolazi, kako ih spriječiti i kako ih napadači mogu iskoristiti. Zbog razmjera i količine današnjih Web apllikacije poželjno je i razviti program za automatsku detekciju takvih ranjivosti.

U ovom radu dan je pregled nekih učestalijih ranjivosti Weba i opisan je razvijeni program za automatsku detekciju ranjivosti Weba s klijentske strane. Razvijeni program traži ranjivosti na temelju zastarjelih biblioteka i CMS-ova. Prikazani su rezultati tog programa koji pokazuju da velik broj današnjih stranica koristi ne-ažurne odnosno zastarijele verzije nekih komponenata. Također pokazalo se da je ovakvo automatizirano traženje ranjivosti prilično efikasno zato što se u relativno kratkom vremenu može ispitati velik broj stranica, no potrebno je poznavati ograničenja takvih alata. Komentirana su ograničenja razvijenog programa i potencijalne promjene koje koje bi ga mogle napraviti učinkovitijim.

Ključne riječi: kibernetička sigurnost, ranjivosti weba, detekcija ranjivosti

Client-Side Website Vulnerability Detection Techniques

Abstract

Nowadays, the Web is one of the most important means of communication and business. Due to various errors and shortcomings in the implementation, design, use or management of web applications, vulnerabilities arise. By exploiting such vulnerabilities, attackers can gain access to the servers on which those websites are hosted and/or to the computers and Web browsers of visitors. It is therefore necessary to understand such vulnerabilities, how they occur, how to prevent them and how attackers can exploit them. Due to the scale and quantity of today's Web applications, it is desirable to develop a program for automatic detection of such vulnerabilities.
This paper provides an overview of some of the more common Web vulnerabilities and describes the developed program for automatic detection of Web vulnerabilities on the client side. The developed program searches for vulnerabilities based on outdated libraries and CMSs. The results of this program are shown, which indicate that a large number of today's sites use outdated versions of some components. This automated vulnerability search has also been shown to be quite effective because a large number of pages can be scanned in a relatively short time, but it is necessary to know the limitations of such tools. The limitations of the developed program and potential changes that could make it more effective are commented on.

Keywords: cybersecurity, web vulnerabilities, vulnerability detection