SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 765

DETEKCIJA JAVASCRIPT BIBLIOTEKA I NJIHOVIH DODATAKA KORIŠTENJEM IDENTIFIKATORA I KIRIPTOGRAFSKIH SAŽETAKA

Saša Lončarević

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 765

DETEKCIJA JAVASCRIPT BIBLIOTEKA I NJIHOVIH DODATAKA KORIŠTENJEM IDENTIFIKATORA I KIRIPTOGRAFSKIH SAŽETAKA

Saša Lončarević

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 765

Pristupnik:	**Saša Lončarević (0036526146)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Stjepan Groš

Zadatak: **Detekcija JavaScript biblioteka i njihovih dodataka korištenjem identifikatora i kiriptografskih sažetaka**

Opis zadatka:

Na Internetu su dostupne stotine milijuna web stranica koje olakšavaju razmjenu informacija i omogućuju korištenje brojnih usluga. Uslijed loših praksi u održavanju, brojne web stranice koriste zastarjele i ranjive komponente, poput JavaScript biblioteka ili sustava za upravljanje sadržajem (engl. Content Management System, CMS). Napadači mogu zloupotrijebiti ranjivosti takvih komponenti kako bi ostvarili svoje ciljeve, poput izmjene sadržaja Web-stranica, napadanja njihovih posjetitelja, širenja neželjeni poruka i širenja zloćudnog koda. Pronalaženje ranjivih i zastarjelih komponenti bilo bi korisno jer bi se na stranicama koje ih sadrže moglo aktivnije tražiti znakove kompromitacije i proaktivno obavijestiti njihove vlasnike o pronađenim problemima. Zbog brojnosti Web-stranica na Internetu nemoguće je to obaviti bez automatizacije te je za njihovo otkrivanje i prioritiziranje važan razvoj automatskih metoda detekcije. U završnom radu potrebno je razviti rješenje koje za zadanu JavaScript datoteku utvrditi radi li se o nekoj od popularnih JavaScript biblioteka, poput jQuery, Bootstrap ili AngularJS. Nadalje, potrebno je istražiti kako se JavaScript biblioteku može obraditi kako bi se pospješilo rezultate prepoznavanja nepoznate JavaScript datoteke. Rješenje je potrebno evaluirati nad zadanim skupom podataka s JavaScript bibliotekama i prokomentirati dobivene rezultate. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 10. lipnja 2022.

Sadržaj

1. Uvod

Internet je danas visoko zastupljena globalna mreža računala, a na njemu počiva web, veliki skup web stranica koje olakšavaju razmjenu informacija i tako pružaju važne usluge i poduzećima i pojedincima. Važnost i ovisnost o web-u prikazuje njegova veličina. Trenutno se na internetu nalazi 1.95 milijardi web stranica [1]. Zbog tako velike zastupljenosti, i zbog ograničenog broja web programera, u praksi dolazi do propusta u izgradnji i održavanju web stranica. Koriste se zastarjele i potencijalno nesigurne komponente, poput JavaScript biblioteka, njihovih dodataka ili sustava za upravljanje sadržajem (engl. *Content Management System*) koji ih implementiraju. To jesu moderna rješenja koja ubrzavaju i olakšavaju razvoj web stranica, ali nepažnjom mogu dati napadačima priliku da ih zloupotrijebe. Mogu se dogoditi napadi usmjereni na poslužitelje, ali i na krajnje korisnike, mogu se širiti neželjeni sadržaji i zlćudni kodovi. Sve navedeno predstavlja prijetnju bilo za poslovanje koje ovisi o svojoj web stranici ili o korisniku koji koristi uslugu koju mu web stranica pruža.

Prirodno se postavlja potreba za rješenjem koje će podići razinu sigurnosti i spriječiti napadače da čine neželjene radnje. Stoga je cilj ovoga rada ponuditi takvo rješenje, implementirano na način da automatiziranim metodama detekcije pronalazi korištene JavaScript biblioteke i daje informacije o njihovim verzijama, te posljedično i njihovim ranjivostima. Takav alat omogućuje proaktivno djelovanje, te bi se pronađene ranjivosti mogle ukloniti prije nego li ih potencijalni napadači iskoriste.

Nakon uvodnog poglavlja ovoga rada slijedi opis JavaScript biblioteka, objašnjenja načina na koji se koriste, metode pribavljanja, te primjeri ranjivosti koje se pojavljuju u praksi. Općenito razumijevanje navedenih stavki važno je kao polazišna točka za shvaćanje principa prepoznavanja ranjivih biblioteka.

U drugom poglavlju opisane su vrste detekcija biblioteka u ovisnosti o načinu njihova korištenja i samom formatu biblioteka. Navodi se generalna podjela metoda, i njihova teorijska podloga na kojoj počiva implementirano rješenje.

U trećem poglavlju je predočena tehnička strana razvijenog programa, korišteni alati, arhitektura, dizajn i konkretna programska implementacija. Navedeni su rezultati dobiveni testiranjem programa nad skupom web stranica iz hrvatskog mrežnog prostora, te je

objašnjeno njihovo značenje i važnost. Također su opisana ograničenja i mogući alternativni pristupi rješavanju ovoga zadatka.

Rad završava zaključivanjem teme o detekciji ranjivih JavaScript biblioteka, te novostečenim uvidima o rješavanju problema.

2. JavaScript biblioteke

JavaScript [2] je jedan od najpopularnijih programskih jezika za izradu web aplikacija. Istraživanje iz 2021. godine, provedeno nad više od 80 tisuća ispitanika, govori kako ga 64.96% programskih inženjera aktivno koristi [3]. JavaScript je objektno orijentiran jezik što ga čini korisnim za programiranje poslužiteljske strane (engl. *server-side*), ali također je i skriptni jezik koji se ne mora prethodno prevesti (engl. *compile*) kako bi se izvršio, što ga čini odličnim izborom za programiranje klijentske strane (engl. *client-side*) web aplikacija.

Želja za stvaranjem bogatih, interaktivnih web stranica je potaknula udaljavanje od ideje razvoja tankog klijenta (engl. *thin client*), što je direktno dovelo do učestalog korištenja JavaScript koda na klijentskoj strani. Kako bi se razvoj web aplikacija ubrzao prihvaćena je ideja JavaScript biblioteka, te je postala vrlo popularna u kratkome roku. Biblioteke predstavljaju skup gotovog koda koji pruža određenu funkcionalnost, pri čemu je dovoljno generaliziran kako bi se mogao primjenjivati na većem skupu problema. Biblioteke se rade tako da imaju puni pristup HTML [27] dokumentu i pritom ga mogu dinamički mijenjati i uređivati u konkretnim situacijama. Primjer jedne od najpopularnijih biblioteka je jQuery [4] koja na jednostavan i brz način omogućuje obilazak stabla i manipuliranje njime, rukovanje korisnički potaknutim događajima i CSS [5] animacijama. Uz jQuery izrazito popularne su i biblioteke Bootstrap [19], React [20], Vue [21]. Važan dio biblioteka su također i njihovi dodatci (engl. *plugin*). Dodaci predstavljaju manje programe koji se nadovezuju na biblioteku i pružaju dodatne funkcionalnosti, a da pritom nije potrebno izmijeniti originalnu biblioteku. Zbog fleksibilnosti koju nude široko su zastupljeni.

2.1. Prikupljanje JavaScript biblioteka

JavaScript biblioteke nisu organizirane na centraliziranom sjedištu gdje bi se sve mogle pronaći za preuzimanje. Razlog tomu je što su biblioteke uglavnom otvorenog izvora (engl. *open-source*) i vlasnici ih objavljuju na različitim platformama. No postoje CDN (engl. *Content Distribution Network*) poslužitelji koji nude uslugu preuzimanja velikog broja

biblioteka koje čuvaju. Često su u vlasništvu velikih kompanija, ali postoje i poslužitelji čije su usluge besplatne. Usprkos CDN poslužiteljima problem prikupljanja svih biblioteka i svih njihovih verzija, podvrsta i dodataka nije riješen. Niti jedan od popularnih CDN poslužitelja ne nudi kompletnu uslugu. Problem pribavljanja starih verzija je prisutan čak i na službenim stranicama proizvođača biblioteka.

Kao pojednostavljenje, za programere su stvorene platforme za upravljanje paketima (engl. *package manager*) koje olakšavaju i ubrzavaju preuzimanje biblioteka. Pritom nude opciju da se uvijek dohvaćaju najnovije dostupne verzije, što djelomično razrješava programera brige o ranjivostima iz starih verzija biblioteka. Popularan primjer takve platforme je npm (engl. *node package manager*) [11].

2.2. Ranjivosti JavaScript biblioteka

Unatoč svojoj popularnosti i raširenosti JavaScript biblioteke su podložne ranjivostima. Sam JavaScript jezik je tipski siguran (engl. *type-safe*), odnosno određene operacije se mogu izvoditi samo nad tipovima podataka koji ih podržavaju, no nije jako tipiziran (engl. *strongly-typed*) jezik pa ne ograničava programera da pravilno koristi apstraktne tipove podataka. Uz nepažljivo organiziranje koda i zanemarivanje dobrih sigurnosnih praksi može doći do propusta koje napadači mogu iskoristiti. Tome svjedoče rezultati istraživanja provedenog na skupu popularnih web stranica gdje je pokazano da ih više od 50% sadrži barem jednu poznatu ranjivost uslijed korištenja zastarjelih ili ranjivih JavaScript biblioteka [6].

Pokazatelj ozbiljnosti ovog problema je i organizacija OWASP (engl. *Open Web Application Security Project*) koja svake godine objavljuje najčešćih 10 ranjivosti web aplikacija [7]. Neke od njih su i zastupljene u JavaScript bibliotekama, kao primjer koji se često pojavljuje u CVE (engl. *Common Vulnerabilities and Exposures*) [8] bazama ranjivosti vezan za JavaScript biblioteke jest XSS ranjivost (engl. *Cross-Site Scripting*) [9].

Konkretan primjer se može pronaći u biblioteci jQuery do verzije 1.9.0 koja nudi funkciju naziva `load()` pri čemu se nepravilno provjerava unos i moguće je predati HTML oznaku `<script>` s JavaScript kodom koji će se posljedično izvršiti [10]. Primjera ranjivosti ima mnogo i to je razlog za stvaranjem automatiziranih alata koji će detektirati ranjive biblioteke na web sjedištima.

3. Tehnike detekcije JavaScript biblioteka

JavaScript biblioteke se mogu koristit na više načina, u više različitih oblika i samim time postoji više načina detekcije. Praksa je dodavati biblioteke u kod web stranice putem `<script>` oznake u HTML kodu, pri čemu se dodaje poveznica na biblioteku koja se može nalaziti na samom web poslužitelju ili na drugoj lokaciji poput CDN poslužitelja. To već omogućava jedan način detekcije gdje se pregledom `<script>` oznaka mogu tražiti podudaranja u poveznicama i tako detektirati korištene biblioteke.

Sama biblioteka koja se dodaje može biti u određenom obliku, ovisno o potrebi, pa tako osim razlika u verzijama postoje razlike u načinu pakiranja biblioteke, konkretno *uncompressed*, *slim* i *minified* vrste. Navedene vrste se temelje na ideji komprimiranja programskog koda kako bi se brže prenosio mrežom i skratio vrijeme učitavanja web stranice. U produkciji se preferira minificirani kod, odnosno kod iz kojeg se automatiziranim putem izbacuju komentari, prazni znakovi, nekorišten kod, te se skraćuju imena varijabli i funkcija [32].

Uz različite oblike prepoznati su i nepredvidivi aspekti korištenja biblioteka, situacije kada programeri vlastoručno mijenjaju izvorni kod biblioteke. Razlozi modificiranja su često brisanje dijelova koda koji se neće koristiti i dodavanje koda koji na neki način dopunjuje funkcionalnost biblioteke. Također spajanje više biblioteka u jednu datoteku te personaliziranije biblioteka.

Uočene su i modifikacije od strane poslužitelja JavaScript biblioteka, no to su najčešće samo izmjene u početnom komentaru kako bi se naznačilo mjesto preuzimanja biblioteke.

Uz sve navedene oblike biblioteka i vrste modifikacija koje mogu predstavljati problem tehnikama detekcije JavaScript biblioteka, postoji i problem količine biblioteka koje su aktivno u uporabi, a također i njihovih dodataka (engl. *plugin*) kojih ima mnogo više.

Navedene stavke su izvor ideja različitih metoda detektiranja biblioteka. Gruba podjela se odnosi na statičke i dinamičke metode detekcije.

3.1. Statička detekcija

Generalno, statička analiza se odnosi na proučavanja programskog koda bez njegova izvršavanja, tako se u ovom slučaju detekcija, odnosno prepoznavanje, odvija bez pokretanja koda. Metoda je često korištena u analizi malicioznog koda jer se smatra sigurnijim pristupom, međutim kod detekcije je prednost u ubrzanju koje nudi jer se kod biblioteke ne mora izvršavati putem relativno sporih parsera ili interpretera.

3.1.1. Detekcija kriptografskim sažetcima

Funkcija sažetka (engl. *hash function*) je svaka funkcija koja ulaznom nizu proizvoljne duljine pridjeljuje jedinstveni niz fiksne duljine [12]. Dakle, izlaz funkcije jedinstveno određuje sam ulaz. Ideja za korištenjem funkcija sažetaka dolazi zbog potrebe za skraćivanjem nizova koji se uspoređuju, konkretno u ovom slučaju umjesto uspoređivanja cijelog koda JavaScript biblioteka mogu se usporediti sažetci i saznati podudaraju li se.

Kriptografske funkcije sažetaka su podvrste funkcija sažetaka koje nude sigurnost od kolizija. Kolizija se odnosi na situaciju kada dva različita ulaza u funkciju daju identičan izlaz, odnosno sažetak. U radu je za izračunavanje sažetaka biblioteka korištena funkcija SHA-256 [13] koja se smatra sigurnom funkcijom sažetka.

Preduvjet za izvršavanje detekcije kriptografskim sažetcima jest napravljena kolekcija koja sadrži sve potrebne JavaScript biblioteke i njihove izračunate sažetke. Zatim se za nepoznatu biblioteku može izračunati sažetak i tražiti podudaranje u kolekciji. Pronađeni sažetak direktno daje informaciju o kojoj se biblioteci radi.

Pretpostavka za korištenje ove vrste detekcije jest da se biblioteke koriste u svom izvornom obliku (engl. *use-as-is*), dakle bez ikakvih modifikacija. Razlog tomu je svojstvo funkcija sažetaka da za najmanju promjenu u ulazu, za jedan izmijenjeni bit, daju potpuno različit izlaz, tako da dva jednaka sažetka garantiraju 100% podudarnost u izvornom ulazu, no za minimalnu promjenu detekcija nije uspješna. Kao rješenje problema u radu su provedeni postupci svođenja biblioteka na kanonski oblik, pri čemu se u obzir ne uzimaju dijelovi koda za koje se smatra da ih programeri i razvijatelji web stranica često modificiraju i tek onda se računa kriptografski sažetak koji se koristi za detekciju.

3.1.2. Detekcija identifikatorima

Alternativni pristup u statičkoj detekciji jest detekcija identifikatorima. Odnosi se na izvlačenje ključnih riječi iz koda JavaScript biblioteke, pri čemu se ključne riječi dobivaju leksičkom analizom koda. Kod se tokenizira i spremaju se identifikatori leksičke analize (engl. *Identifier Token*) pri čemu je potrebno uzeti u obzir samo one koji se ne pojavljuju u ostalim bibliotekama, jer onda ne bi bili jedinstveni za danu biblioteku, dakle ne bi bili identifikatori. Pretpostavka je da su biblioteke relativno duge skripte i da identifikatora ima mnogo, stoga će detekcija biti proporcionalno pouzdana. Međutim za dodatke bibliotekama (engl. *library plugin*) ta pretpostavka ne vrijedi, ti kodovi su često relativno kratki jer ispunjavaju manju, specifičnu svrhu. Iz tog razloga se kod detekcije identifikatorima u obzir uzimaju samo biblioteke i dodatci koji imaju značajan broj identifikatora kako ne bi došlo do lažno pozitivnih detekcija. Ova metoda, sama po sebi, nije otporna na situacije gdje proizvoljan JavaScript kod koji, slučajno ili namjerno, koristi velik broj identifikatora neke poznate biblioteke i zato bude pozitivno detektiran, kada zapravo nije trebao biti. No pokazuje se da takvih primjera u praksi nema, iako su teoretski mogući.

Mana ove metode jest računska zahtjevnost. Za veće kolekcije biblioteka s mnogo identifikatora koji se moraju uspoređivati s mnoštvom identifikatora nepoznatih biblioteka dolazi do kombinatorne eksplozije. Velik broj naizgled jednostavnih operacija može drastično usporiti detekciju zato je potrebno prilikom implementacije razmotriti naprednije strukture podataka i algoritme poput raspršenog adresiranja i algoritama za brzo pretraživanje.

3.2. Dinamička detekcija

Kao suprotnost statičkoj detekciji, dinamička se odnosi na prepoznavanje koda njegovim izvršavanjem. Određeni objekti koje program stvara i funkcije koje pokreće mogu se iskoristiti za prepoznavanje biblioteke. Ono što omogućava ovu vrstu detekcije je nepostojanje imenskog prostora (engl. *namespace*) u JavaScript-u. Posljedično objekti i funkcije su u kodu dostupni globalno i može ih se pozivati za vrijeme izvođenja te kombinacijom prepoznatih elemenata odrediti o kojoj biblioteci je riječ. Ovom načinu detekcije posebno u korist idu biblioteke koje pružaju atribute o samoj verziji poput biblioteke jQuery. Primjer implementacije ove vrste detekcije, korišten u vidu ekstenzije

web preglednika, na modernim web stranicama uspješno detektira više od 100 različitih biblioteka [14]. No alat ne uspijeva riješiti problem starijih verzija biblioteka, kao niti modificiranih, stoga takav pristup detekciji nije implementiran u završnom radu.

4. Razvijeno programsko rješenje

U sklopu završnog rada razvijeno je programsko rješenje za detekciju JavaScript biblioteka. Program prikuplja JavaScript biblioteke s CDN poslužitelja i gradi kolekciju koja će služiti kao osnova za detektiranje nepoznatih biblioteka. Ista kolekcija se nadopunjuje izračunatim vrijednostima sažetaka i dodatno informacijama o biblioteci poput naziva, verzije i ranjivosti. Program nudi tri mehanizma statičke detekcije koji su optimizirani i mogu se izvoditi na više procesorskih dretvi.

Program je ostvaren kao alat koji se pokreće iz komandne ljuske (engl. *Command Line Tool*). To znači da je za pokretanje alata u određenom načinu potrebno zadati prikladnu zastavicu. Pisan je u jeziku Python [16] verzije 3.8, a baza podataka je ostvarena alatom MongoDB [15] koji nudi nerelacijsku bazu orijentiranu na dokumente.

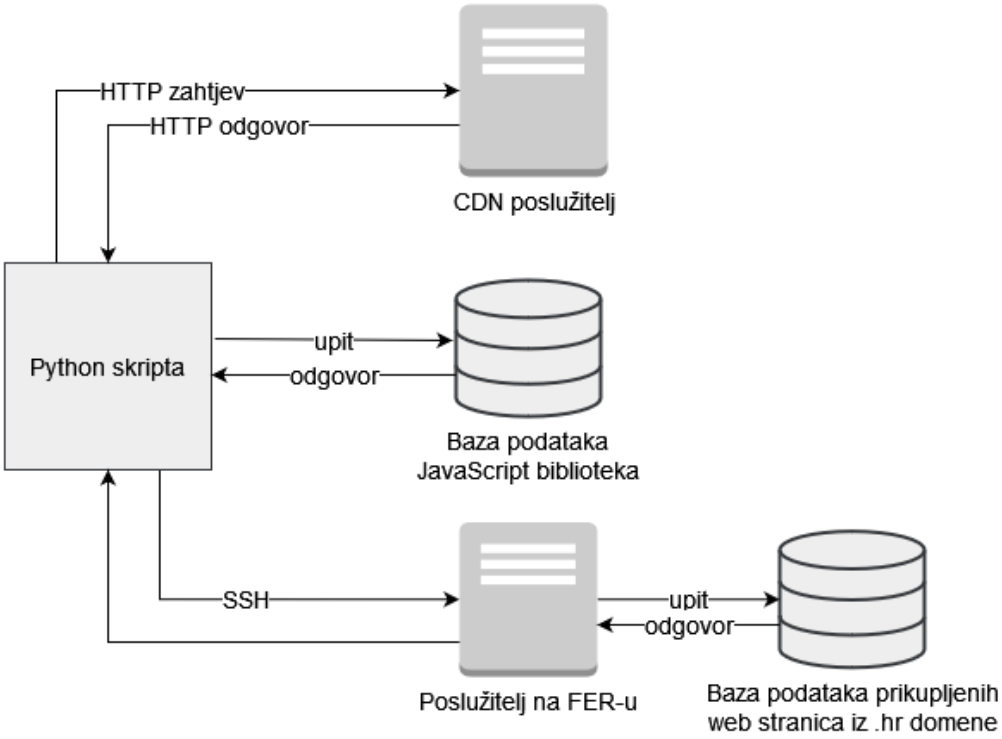
Za provođenje testiranja programa korištena je baza podataka stvorena na FER-u, koju popunjava web pauk (engl. *web crawler*) s podatcima web stranica iz hrvatskog mrežnog prostora.

4.1. Implementacija

Program je podijeljen u cjeline po funkcijama koje se obavljaju. Dvije vršne cjeline su namijenjene stvaranju baze i izvršavanju algoritama detekcije. Pri čemu dio za stvaranje baze obrađuje dohvaćene podatke i također služi za naknadno ažuriranje baze, a dio za izvršavanje algoritama nudi opcije za tri vrste statičke detekcije i organizirano zapisivanje rezultata u odabranom formatu.

Arhitektura sustava i dionici komunikacije su prikazani slikom 3.1. Program prilikom stvaranja ili ažuriranja baze JavaScript biblioteka komunicira s CDN poslužiteljem putem HTTP zahtjeva i odgovora, i tako pribavlja izvorne kodove biblioteka koje zatim pohranjuje u bazu podataka. Drugu cjelinu predstavlja komunikacija sa FER-ovim poslužiteljem, odnosno bazom podataka koja sadrži web stranice prikupljene iz hrvatskog mrežnog prostora putem web pauka. Svaki dokument iz baze se dohvaća i program ga

obrađuje kako bi detektirao radi li se o ranjivoj JavaScript biblioteci, te rezultat pohranjuje u svoju bazu.



Slika 4.1, "Skica arhitekture sustava"

4.1.1. Prikupljanje podataka i izgradnja baze

Kao osnova za bazu odabrano je 26 najkorištenijih JavaScript biblioteka koje pri tom imaju, ili su imale u prošlim verzijama, javno poznate ranjivosti [18]. Biblioteke su dostupne za preuzimanje na više servisa i poslužitelja, a u ovome projektu je izabran CDN pod nazivom „CDNPKG“ kao izvor biblioteka [17]. Ono što omogućuje automatizirano dohvaćanje biblioteka jest implementirani web scraper koji na web aplikaciji CDN poslužitelja za dani upit dohvaća sve rezultate pretraživanja, te nad novo dobivenim stranicama ponovno skuplja rezultate koji predstavljaju poveznice za preuzimanje instanci biblioteka. Web scraper je programiran tako da skuplja sve biblioteke i dodatke vezane za odabranih 26 biblioteka i također sve njihove verzije i podtipove. Njegova implementacija počiva na Python modulu `requests` namijenjenom slanju i primanju HTTP zahtjeva odnosno odgovora, te biblioteci `beautifulsoup4` [22] koja omogućava naprednu obradu HTML dokumenta.

Svi se dohvaćeni podatci slijedno obrađuju, za svaku dohvaćenu biblioteku je potrebno iz zahtjeva izdvojiti informacije o nazivu, tipu i verziji. Za verzioniranje biblioteka koristi se standardna semantika `major.minor.patch`, a tipovi su organizirani po razini kompresije, pa tako postoje *uncompressed*, *slim* i *minified* tipovi. Uz to, važan dio za izvršavanje detekcije je izračunavanje kriptografskog sažetka biblioteke. Taj dio postupka se obavlja unaprijed, prije izvršavanja detekcije, kako bi se algoritam ubrzao i kako bi se uklonilo redundantno računanje. Sažetci se izračunavaju uz pomoć modula `hashlib` algoritmom SHA256. Zatim se za obrađenu biblioteku u popisu ranjivosti traži povezana CVE oznaka, ako takva postoji. Sam popis ranjivosti je preuzet iz alata RetireJS [18]. Svi dobiveni podaci se dodaju u kolekciju u MongoDB bazi kao dokument JSON formata. Pri čemu se komunikacija s bazom odvija putem Python modula `pymongo`. Svaki dokument kolekcije je jedinstveno određen pripadnim kriptografskim sažetkom biblioteke, i u ostalim kolekcijama se koristi za referenciranje poput primarnog ključa. Navedeni postupak stvara dokumente poput prikazanog slikom 4.2.

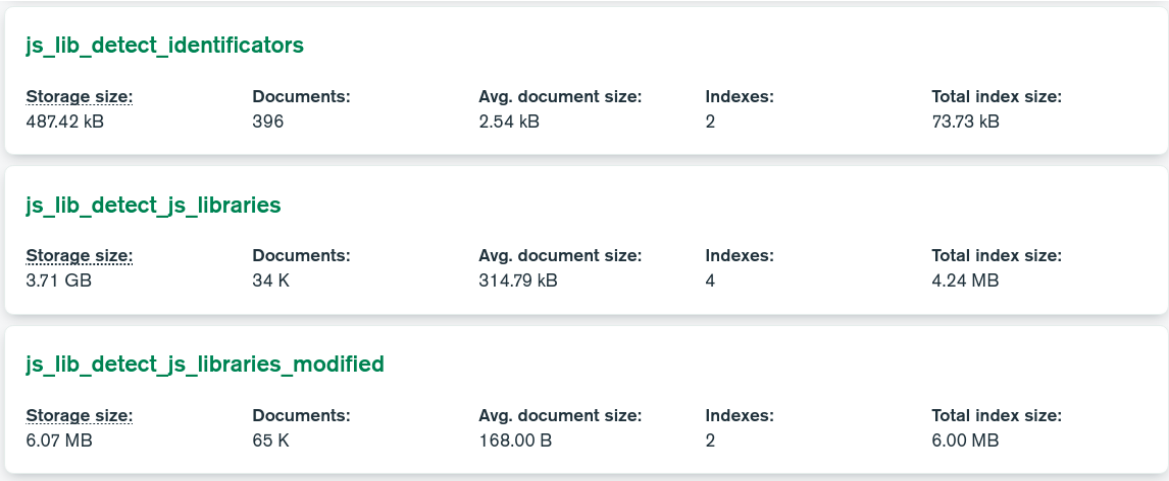
Slika 4.2, "Primjer dokumenta iz baze podataka"

Stvorena kolekcija je dovoljna za izvršavanje osnovnog tipa detekcije gdje se samo uspoređuju kriptografski sažetci biblioteka sa sažetkom nepoznate biblioteke, no za izvođenje naprednijih metoda potrebno je dalje obraditi podatke o bibliotekama.

Kako bi se riješio problem minimalnih izmjena u JavaScript bibliotekama potrebno ih je svesti na zajednički oblik, odnosno zanemariti dijelove koda koji nisu ključni za samu biblioteku, a da su pritom često mijenjani. Zato se iz izvornog koda biblioteka brišu svi prazni znakovi, točnije razmaci, tabulatori i novi redovi, te se brišu svi komentari bilo jednolinijski ili duži u više linija. Također se izbacuju pozivi funkcija poput `jQuery.noConflict()` jer oni ne pripadaju originalnom izvornom kodu već su naknadno dodani kako bi se razriješili konflikti i nepodudaranja među različitim verzijama. Iz tako obrađenog izvornog koda se izračunava novi kriptografski sažetak i sprema u novu kolekciju pri čemu se zapisuje i referenca na originalni dokument iz kojeg nastaje. Kasnije, prilikom detekcije, će se ovaj postupak ponavljati i nad nepoznatom bibliotekom kako bi obje strane bile svedene na kanonski oblik.

Nadalje, potrebno je pripremiti podatke i za detekciju identifikatorima. Postupak uključuje tokenizaciju izvornog koda biblioteke, gdje se ekstrahiraju identifikatori po leksičkoj analizi. To su svi nazivi varijabli, funkcija, klasa i sve ono što identificira određeni objekt, a da nije specificirano samom sintaksom jezika, već da je to programer morao sam zadati. Leksička analiza je ostvarena pomoću Python biblioteke `esprima` [23] koja nudi funkciju `tokenize()`. Dobiveni popis identifikatora za svaku biblioteku se mora usporediti s popisima identifikatora ostalih biblioteka u bazi kako bi se izbacili svi oni identifikatori koji se pojavljuju u više različitih biblioteka. Cilj je zadržati samo one identifikatore koji su uistinu jedinstveni za danu biblioteku, kako ne bi bilo dvojbe prilikom detekcije. Konačno, identifikatori se za svaku biblioteku zapisuju u novu kolekciju kako bi se prilikom detekcije mogli učitavati i tražiti u nepoznatim bibliotekama.

Cijeli navedeni proces prikupljanja i obrade podataka rezultira bazom podataka prikazanoj na slici 4.3. Za 26 odabranih biblioteka je dohvaćeno 34 tisuće dokumenata koji obuhvaćaju sve originalne biblioteke, dodatke, sve njihove podvrste, tipove, i naravno sve dostupne verzije iz više razvojnih faza. Također za kolekcije se kreiraju i odgovarajući indeksi po parametrima koji će se za vrijeme detekcije pozivati, kako bi se izvođenje ubrzalo.



Slika 4.3, "Prikaz kolekcija iz baze podataka"

Prikupljanje i obrada podataka, te izgradnja baze su automatizirani do mjere da se samo pozivom programa može stvoriti cijela baza. Iz tog razloga nije nužno odrađivati migracije baze prilikom puštanja u pogon, nije potrebno niti izrađivati i čuvati sigurnosne kopije baze jer ju program može lako nanovo stvoriti. Sam proces izgradnje baze traje nekoliko sati, pri čemu je vremenski najzahtjevniji dio uspostavljanje velikog broja konekcija prema CDN poslužitelju. Također, program omogućava i ažuriranje baze, kada nove verzije JavaScript biblioteka postanu dostupne.

4.1.2. Izvršavanje detekcije

Nakon izgradnje i popunjavanja baze, kada su pripremljeni i pohranjeni svi potrebni podaci, program može započeti s detekcijom. Implementirana su tri načina statičke detekcije pri čemu jedan drugoga proširuju i tako povećavaju broj detektiranih biblioteka.

Prvi, standardni način, od nepoznate biblioteke uzima kriptografski sažetak i isti traži u kolekciji JavaScript biblioteka. Pronalazak podudaranja znači da je biblioteka uspješno detektirana, i daje informacije da je nepoznata biblioteka korištena u originalnoj formi bez ikakvih izmjena. Također, ova metoda je egzaktna i garantira apsolutnu ispravnost detekcije, odnosno garantira da se neće dogoditi lažno pozitivne detekcije biblioteka. Za očekivati je da ova metoda neće davati mnoštvo rezultata ukoliko osnovna baza nije potpuna. Pod izrazom potpunost se misli na prikupljanje biblioteka iz svih izvora koje programeri koriste prilikom razvijanja web aplikacije i puštanja u pogon. Dakle, nije dovoljno da baza zadrži sve biblioteke, već treba sadržavati sve kopije ili instance

biblioteka sa svih izvora. Razlog tomu je da različiti servisi koji nude biblioteke rade izmjene pa se tako i kriptografski sažetak za zapravo istu biblioteku može razlikovati. Primjer minimalne izmjene je promjena početnog komentara biblioteke, no to je dovoljno da se sažetak izmjeni jer kriptografske funkcije sažetaka ne toleriraju ni minimalne razlike. Ova metoda ne može detektirati biblioteke koje su programeri mijenjali, jer je se takve biblioteke ne mogu pribaviti prijevremeno i spremiti u bazu.

Drugi način statičke detekcije se odnosi također na detekciju uporabom kriptografskih sažetaka, ali implementira svođenje biblioteka na kanonski oblik. To je pristup koji ublažava problem minimalnih izmjena u izvornom kodu biblioteka. Nad nepoznatim bibliotekama se provodi isti postupak kakav je opisan prilikom stvaranja kolekcije s kriptografskim sažetcima modificiranih biblioteka. Nepoznata biblioteka se čisti od komentara, praznih znakova i poziva funkcija za rješavanje konflikata, te se onda računa vrijednost kriptografske funkcije sažetka i ona se traži u bazi. Ovakva metoda pokriva sve rezultate detekcije koje daje prijašnja standardna detekcija i pritom detektira i dio minimalno izmijenjenih biblioteka. Naravno, implementacija i izvođenje ove metode dolazi s cijenom kompleksnosti i vremenske zahtjevnosti.

Treći način statičke detekcije jest pretraživanje nepoznatih biblioteka s ciljem pronalaska identifikatora koji određuju pojedine biblioteke. Nad nepoznatom bibliotekom se provodi leksička analiza, izvlače se identifikatori i zatim se traže podudaranja u bazi. Mjera kojom se određuje sigurnost detekcije je omjer broja detektiranih identifikatora i ukupnog broja identifikatora za danu biblioteku u bazi. Zbog toga ova metoda gubi svojstvo egzaktnosti za razliku od prošlih. Teoretski je moguće da se pronađe potpuno podudaranje identifikatora, a da se radi o različitim JavaScript kodovima, no prilikom testiranja takvi problemi nisu uočeni. Iz rezultata detekcije se očitava da metoda radi pravilno jer programeri poštuju dobre prakse i zadaju identifikatore koji su specifični i smisleni pa se lažno pozitivne detekcije te vrste ne pojavljuju. Opet, zbog povećanja kompleksnosti algoritma dolazi veće vremenske zahtjevnosti. U sklopu implementacije istraženi su i algoritmi za brzo pretraživanje teksta poput Boyer-Moore [24] i Rabin-Karp [25] algoritama, no pokazalo se da kombinacijom smanjivanja broja identifikatora i korištenjem struktura podataka s raspršenim adresiranjem bolje optimizira izvođenje detekcije.

Kako je ovaj program namijenjen izvršavanju nad velikim skupom podataka, organiziran je tako da paralelno koristi više procesorskih dretvi prilikom detekcije. Konkretna implementacija se oslanja na Python modul `concurrent.futures` koji pruža razred

`ProcessPoolExecutor` [26] i tako se program ubrzava sukladno broju procesorskih jezgri.

Svi rezultati detekcije se zapisuju u zasebnu kolekciju u bazi, pri čemu svaki dokument sadržava reference, u obliku kriptografskih sažetaka, potrebne za dohvaćanje detaljnijih informacija o detektiranim bibliotekama. Dodatan razlog za kolekcijom rezultata je i mogućnost korištenja agregacija u bazi. Primjer jednog takvog dokumenta je prikazan slikom 4.4. Biblioteka iz primjera je detektirana standardnom i modificiranom detekcijom, te detekcijom identifikatorima s 98.28% sigurnosti.


```
_id: ObjectId('627eaa53139301a6c83fc709')
hash_ref_cdpv0: "0925e8ad7bd971391a8b1e98be8e87a6971919eb5b60c196485941c3c1df089a"
file_type: "js"
hash_ref_std_det: "0925e8ad7bd971391a8b1e98be8e87a6971919eb5b60c196485941c3c1df089a"
timestamp_std_det: 1652469233.493032
hash_ref_mod_det: "0925e8ad7bd971391a8b1e98be8e87a6971919eb5b60c196485941c3c1df089a"
timestamp_mod_det: 1652469233.493032
hash_ref_ids_det: "f7f6a5894f1d19ddad6fa392b2ece2c5e578cbf7da4ea805b6885eb6985b6e3d"
confidence_ids_det: "98.28"
timestamp_ids_det: 1652469233.493032
```


Slika 4.4, "Primjer dokumenta iz kolekcije rezultata"

4.2. Rezultati detekcije

Testiranje programa je provedeno nad 5 milijuna dokumenata koje je prikupio web pauk iz hrvatskog mrežnog prostora. Analizom skupa podataka utvrđeno je da se sastoji od

- 93.03% HTML dokumenata,
- 3.41% JavaScript datoteka,
- 1.98% CSS datoteka,
- 1.58% ostalih vrsta datoteka.

Ova raspodjela ne reprezentira nužno i stvarnu raspodjelu tipova datoteka na web aplikacijama, već je ovakva raspodjela rezultat organizacije web pauka koji zbog iterativnog dohvaćanja izmijenjenih dokumenata dohvaća HTML više od ostalih tipova, jer se u ovom slučaju HTML dokumenti češće mijenjaju.

Daljnje testiranje provodi se nad JavaScript datotekama kojih je točno 170515 što čini navedenih 3.41% skupa podataka. Implementirani program je namijenjen detekciji

isključivo JavaScript biblioteka, naravno, onih koje su sadržane u bazi. No pokazuje se da u skupu JavaScript datoteka ne postoje samo biblioteke, već i znatan udio datoteka koje sadrže nepoznati JavaScript kod. To su skripte koje su ručno rađene za specifičnu web aplikaciju i njih program ne može detektirati pa tako ni otkriti potencijalne ranjivosti. Sama raspodjela koliko je u skupu biblioteka, a koliko nepoznatih JavaScript kodova, ostaje nepoznanica jer trenutno ne postoji razvijeno rješenje koje bi ih diferenciralo. Stoga će se detekcija provoditi nad cijelim skupom datoteka, kako nema načina da se taj skup ograniči samo na biblioteke.

Izvršavanjem triju načina detekcije detektirano je

- 966 biblioteka standardnom detekcijom usporedbom kriptografskih sažetaka,
- 2870 biblioteka svođenjem na kanonski oblik i usporedbom kriptografskih sažetaka,
- 16097 biblioteka korištenjem identifikatora.

Kao što je pretpostavljeno, standardna detekcija daje najmanje rezultata. Razlozi su već spomenuti, uvođenje minimalnih izmjena u kod i nepotpunost baze uslijed velikog broja izvora za preuzimanje biblioteka pri čemu različiti izvori uvode izmjene u originalne biblioteke.

S druge strane detekcija prije koje se izvršava svođenje na kanonski oblik daje gotovo tri puta više rezultata, što dokazuje da su pretpostavljene minimalne izmjene doista česte. No daljnja pretpostavka jest da postoji još više biblioteka s izmjenama koje se nisu mogle predvidjeti i ukloniti prije detekcije. Međutim, brojke koje bi to potvrdile su nepoznate jer ne postoji alat koji bi dao točne informacije koje bi se koristile kao heuristika za ovu vrstu detekcije.

Konačno, detekcija identifikatorima daje gotovo šest puta više rezultata od detekcije sa svođenjem na zajednički oblik. Prednost ove metode jest da izmjene u kodu biblioteke nisu bitne ukoliko nisu mijenjani identifikatori po kojoj se biblioteka prepoznaje. Tu se pokazuje još veći potencijal za rješavanje problema minimalnih izmjena u odnosu na detekciju svođenjem na zajednički oblik. No, mana je da se točne verzije biblioteka ne mogu pouzdano odrediti.

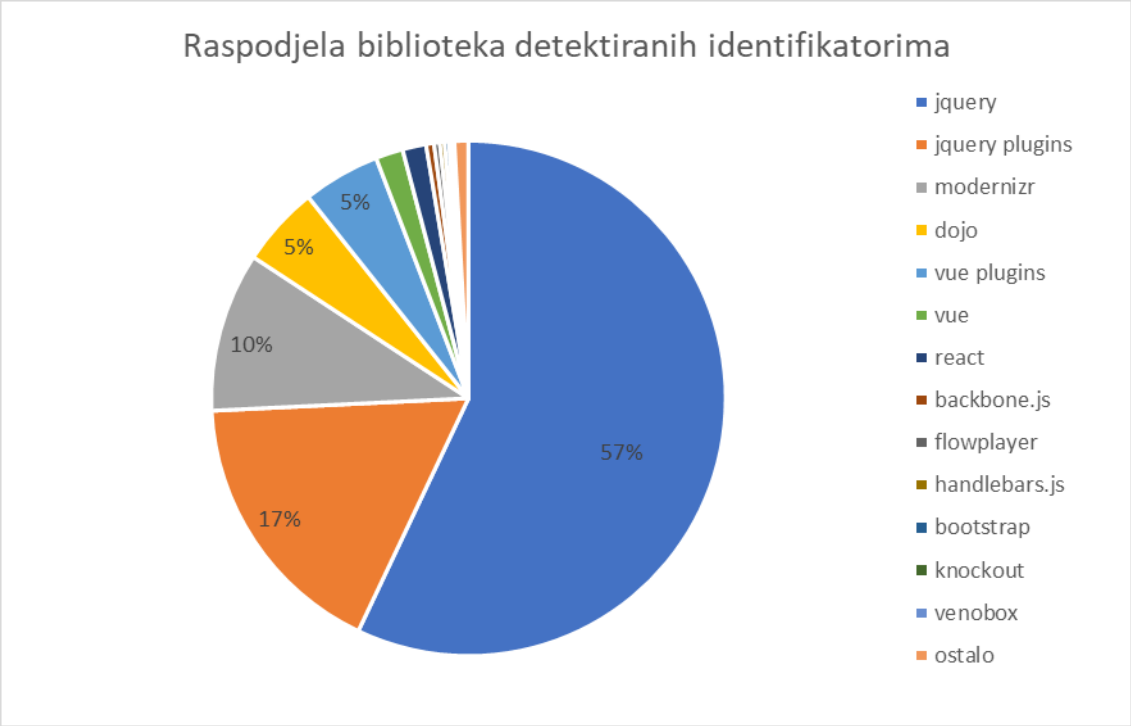
Rezultati detekcije pokazuju trend da su određene biblioteke više prisutne o odnosu na ostale. Graf 4.1 prikazuje raspodjelu 25 najzastupljenijih biblioteka koje su detektirane detekcijom sa svođenjem na zajednički oblik.

Graf 4.1, "Graf raspodjele biblioteka detektiranih kriptografskim sažetcima"

Važna stavka na koju rezultati ukazuju jest da se češće koriste minificirane biblioteke, na grafu zapisane s izrazom „min“. To potvrđuje pretpostavku iz rada slične tematike [28] gdje su upravo minificirane biblioteke kočile detekciju. Tu se također pojavljuje problem s izmjenama koda, no različit od prethodno navedenih. Minifikacija se može obavljati različitim alatima koji će dati za isti kod različite smanjene verzije, no čak i isti alat može prilikom skraćivanja imena varijabli na jednoslovčana, permutirati imena i tako kočiti detekciju usporedbom sažetaka.

Nadalje, detekcija identifikatorima daje nešto drugačiju raspodjelu detektiranih biblioteka, za što je glavni razlog ograničen skup podataka koji služe kao osnova za tu vrstu detekcije. Već navedena metoda izvlačenja identifikatora pogoduje samo određenim bibliotekama, potrebno da je imaju znatnu količinu samih identifikatora i da pritom nisu generičkih

naziva tako da uistinu mogu identificirati danu biblioteku. Graf 4.2 prikazuje saženu raspodjelu najučestalijih biblioteka.



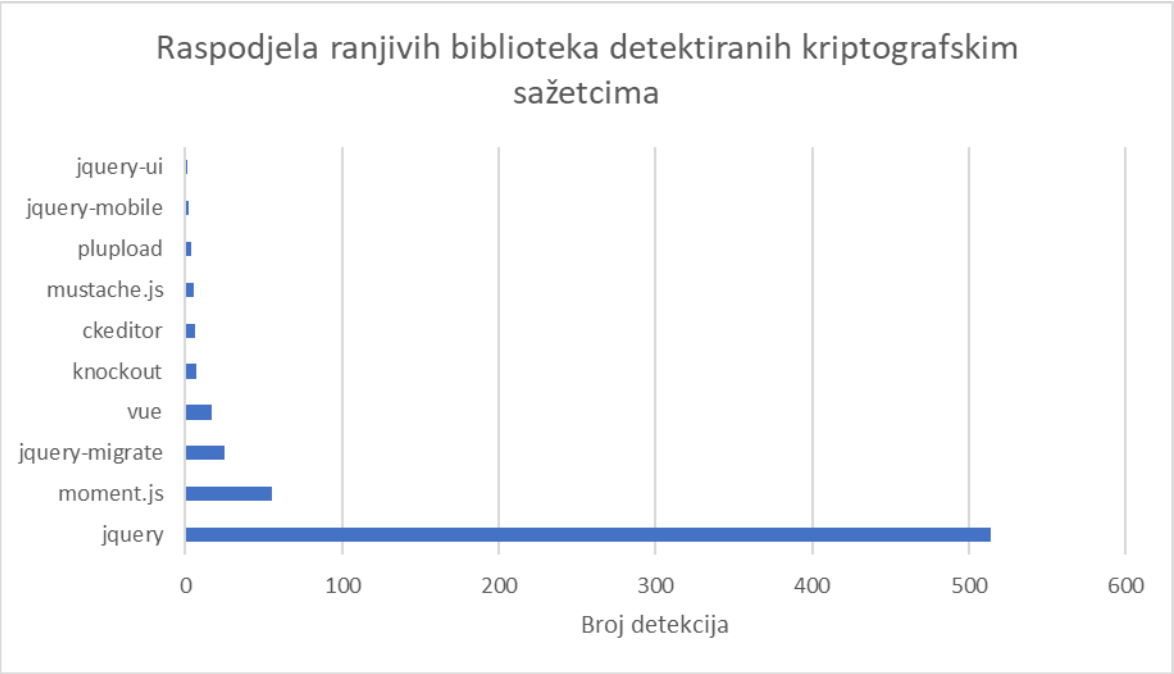
Graf 4.2, "Graf raspodjele biblioteka detektiranih identifikatorima"

Vidljivo najzastupljenija biblioteka prepoznata svim korištenim metodama detekcije jest jQuery i također jQuery dodatci. Ispravnost raspodjele biblioteka potvrđuje i statistika dobivena istraživanjem nad odabranim skupom svjetskih web stranica [29]. Potvrđeno je da je jQuery svakako najzastupljeniji, te da ga slijede biblioteke Bootstrap, Modernizr, React, Backbone. Nažalost, to su neke od biblioteka koje u prošlim verzijama imaju javno poznate ranjivosti.

Među detektiranim bibliotekama pojavljuje se i nezanemariv dio onih koje imaju povezane zapise o ranjivostima iz NVD [30] baze. Prepozanto je

- 165 ranjivih od ukupno 966 biblioteka korištenjem standardne detekcije (17.08%)
- 636 ranjivih od ukupno 2870 biblioteka korištenjem detekcije sa svođenjem na zajednički oblik (22.16%)

Detekcija identifikatorima ne daje apsolutne rezultate o ranjivostima jer ne daje pouzdanu informaciju o verziji detektirane biblioteke, stoga se rezultati oslanjaju na detekcije kriptografskim sažetcima. Raspodjelu ranjivih biblioteka prikazuje graf 4.3.



Graf 4.3, "Raspodjela ranjivih biblioteka detektiranih kriptografskim sažetcima"

Pokazuje se da je jQuery najzastupljenija ranjiva JavaScript biblioteka, predstavlja čak 80.82% skupa ranjivih biblioteka. Neki od razloga koji dovode do ovakvog rezultata su da sam jQuery ima 88 javno poznatih ranjivosti po NVD bazi [31], te da se tek zadnje dvije verzije biblioteke smatraju sigurnima dok su sve prijašnje ugrožene barem jednom ranjivošću.

Podaci o ranjivostima u ovom slučaju nisu potpuni jer se odnose samo na biblioteke, a ne i na JavaScript dodatke, kojih je često i više od samih biblioteka. Razlog tomu je ograničena dostupnost informacija o javno poznatim ranjivostima specifičnih za dodatke bibliotekama.

4.3. Ograničenja implementacije

Implementirano rješenje dokazuje ispravnost teorijske strane detektiranja JavaScript biblioteka, no u praktičnom smislu se nameću određena ograničenja. Program, za početak, ovisi o potpunosti baze koju koristi kao osnovu za detekciju, konkretna relacija jest da se mogu detektirati samo one nepoznate biblioteke koje su sadržane u bazi. Korijen ovoga ograničenja dolazi od prirode funkcija sažetaka koje zahtijevaju apsolutno identične ulaze za dobivanje identičnih sažetaka. To i ne bi predstavljalo problem kada bi JavaScript biblioteke bile sistematično organizirane na službenom, centraliziranom poslužitelju i da su

pritom sve korištene instance biblioteka identične. To nije slučaj, stoga bi baza morala sadržavati biblioteke prikupljene sa svih CDN poslužitelja, svih platformi za upravljanje paketima, svih službenih stranica biblioteka i njihovih repozitorija. Da bi se svi takvi podatci automatizirano prikupljali potreban je napredan web scraper i znatni računalni resursi.

Nadalje, problem minimalnih izmjena uvedenih s korisničke strane je zapravo algoritamski nerješiv. Iako neke izmjene prate određene uzorke, postoji mnoštvo nepredvidivih izmjena i također izmjena koje se ne može izbaciti bez narušavanja integriteta originalnog izvornog koda.

Također, ograničenje je dostupnost informacija o ranjivostima. Iako neprofitne organizacije ulažu velik trud da ranjivosti učine javno poznatima i dostupnima, kako bi podigli razinu sigurnosti, informacije nisu potpune koliko bi ovakav program zahtijevao. Mnoge ranjivosti se zasnivaju na neispravnom korištenju ili na samoj organizaciji i dizajnu programskog koda, stoga nisu konkretno zapisane pod CVE oznakom i pritom vezane uz određenu biblioteku.

Implementirano rješenje u određenoj mjeri rješava navedena ograničenja, no i dalje ostavlja prostora za napredak. Jedan od pristupa koji bi mogao predstavljati uspješno alternativno rješenje u detekciji JavaScript biblioteka jest računanje sličnosti. Konkretno, postoje algoritmi za računanje udaljenosti dvaju tekstualnih ulaza, pa bi na primjer, usprkos modifikacijama i izmjenama biblioteka, informacija o 90% sličnosti između biblioteka mnogo značila za detekciju. Mana ovog pristupa jest drastično veća računska zahtjevnost u odnosu na uspoređivanje kriptografskih sažetaka ili čak identifikatora, pretpostavka je nekoliko redova veličine dulje vrijeme izvođenja čak i sa većim računalnim resursima.

5. Zaključak

Web je danas od velike važnosti u svijetu i stoga postoji potreba za podizanjem razine sigurnosti. Ranjivosti su dokazano prisutne i uvode se na više načina, od kojih je jedan upravo korištenje ranjivih JavaScript biblioteka. Sam JavaScript nije nužno nesiguran jezik, no neispravni načini korištenja, i nepažnja prilikom dodavanja JavaScript biblioteka u projekte web aplikacija mogu rezultirati nepoželjnim događajima. Razvoj alata koji bi olakšao proaktivno pronalaženje ranjivih biblioteka je nužno potreban. Za ostvarenje takvog alata, jedan od pristupa zahtjeva kvalitetno pronalaženje i pribavljanje JavaScript biblioteka potrebnih kao osnova za detekciju, te implementaciju mehanizama same detekcije. U radu je obrađen statički pristup koji djeluje na osnovu usporedbe kriptografskih sažetaka i pretraživanja identifikatora biblioteka. Rezultati detekcije pokazuju veliku zastupljenost popularnih biblioteka, pri čemu se preferiraju minificirane verzije. Također prisutan je nezanemariv udio ranjivih biblioteka, koje su upravo glavni razlog razvijanja ovakvog programskog rješenja. Sama implementacija nailazi na određena ograničenja poput izmjena u kodu biblioteka te probleme pribavljanja svih potrebnih podataka, no u određenoj mjeri ih uspješno rješava. Kako bi rješavanje problema detekcije biblioteka bilo što potpunije i ispravnije nametnuta je potreba za kombiniranjem metoda detekcije, ostvarivanjem hibridnih pristupa, te naprednim prikupljanjem potrebnih podataka.

Literatura

[1] „Total number of websites,“ [Mrežno]. Dostupno: https://www.internetlivestats.com/total-number-of-websites/. [Pokušaj pristupa 15.5.2022.].

[2] „JavaScript,“ [Mrežno]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Pokušaj pristupa 15.5.2022.].

[3] „Most used programming languages,“ [Mrežno]. Dostupno: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. [Pokušaj pristupa 15.5.2022.].

[4] „jQuery,“ [Mrežno]. Dostupno: https://jquery.com/. [Pokušaj pristupa 16.5.2022.].

[5] „CSS,“ [Mrežno]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/CSS. [Pokušaj pristupa 16.5.2022.].

[6] „Vulnerable JavaScript Libraries,“ [Mrežno]. Dostupno: https://snyk.io/blog/77-percent-of-sites-still-vulnerable/. [Pokušaj pristupa 16.5.2022.].

[7] „OWASP Top Ten,“ [Mrežno]. Dostupno: https://owasp.org/www-project-top-ten/. [Pokušaj pristupa 16.5.2022.].

[8] „Attacks XSS,“ [Mrežno]. Dostupno: https://owasp.org/www-community/attacks/xss/. [Pokušaj pristupa 16.5.2022.].

[9] „CVE Mitre,“ [Mrežno]. Dostupno: https://cve.mitre.org/. [Pokušaj pristupa 16.5.2022.].

[10] „CVE-2020-7656,“ [Mrežno]. Dostupno: https://www.cvedetails.com/cve/CVE-2020-7656/. [Pokušaj pristupa 17.5.2022.].

[11] „npm,“ [Mrežno]. Dostupno: https://www.npmjs.com/. [Pokušaj pristupa 17.5.2022.].

[12] „Hash Function,“ [Mrežno]. Dostupno: https://en.wikipedia.org/wiki/Hash_function. [Pokušaj pristupa 17.5.2022.].

[13] „SHA 256,“ [Mrežno]. Dostupno: https://www.n-able.com/blog/sha-256-encryption. [Pokušaj pristupa 17.5.2022.].

[14] J. Michel, „Library Detector for Chrome,“ [Mrežno]. Dostupno: https://github.com/johnmichel/Library-Detector-for-Chrome. [Pokušaj pristupa 18.5.2022.].

[15] „MongoDB documentation,“ [Mrežno]. Dostupno: https://www.mongodb.com/docs/. [Pokušaj pristupa 18.5.2022.].

[16] „Python,“ [Mrežno]. Dostupno: https://docs.python.org/3/. [Pokušaj pristupa 19.5.2022.].

[17] „CDNPKG,“ [Mrežno]. Dostupno: https://www.cdnpkg.com/. [Pokušaj pristupa 20.5.2022.].

[18] „RetireJS repository,“ [Mrežno]. Dostupno: https://github.com/RetireJS/retire.js/blob/master/repository/jsrepository.json. [Pokušaj pristupa 20.5.2022.].

[19] „Bootstrap,“ [Mrežno]. Dostupno: https://getbootstrap.com/. [Pokušaj pristupa 20.5.2022.].

[20] „React,“ [Mrežno]. Dostupno: https://reactjs.org/. [Pokušaj pristupa 20.5.2022.].

[21] „Vue,“ [Mrežno]. Dostupno: https://vuejs.org/. [Pokušaj pristupa 20.5.2022.].

[22] „Beautifulsoup4 library,“ [Mrežno]. Dostupno: https://pypi.org/project/beautifulsoup4/. [Pokušaj pristupa 20.5.2022.].

[23] „Esprima,“ [Mrežno]. Dostupno: https://docs.esprima.org/en/4.0/. [Pokušaj pristupa 21.5.2022.].

[24] „Boyer–Moore string-search algorithm,“ [Mrežno]. Dostupno: https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm.

[Pokušaj pristupa 21.5.2022.].

[25] „Rabin–Karp algorithm,“ [Mrežno]. Dostupno: https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm. [Pokušaj pristupa 21.5.2022.].

[26] „Python concurrent.futures,“ [Mrežno]. Dostupno: https://docs.python.org/3/library/concurrent.futures.html. [Pokušaj pristupa 21.5.2022.].

[27] „HTML: HyperText Markup Language,“ [Mrežno]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/HTML. [Pokušaj pristupa 25.5.2022.].

[28] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, E. Kirda, „Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web,“ 2018.

[29] „W3Techs Usage statistics of JavaScript libraries for websites,“ [Mrežno]. Dostupno: https://w3techs.com/technologies/overview/javascript_library. [Pokušaj pristupa 25.5.2022.].

[30] „National Vulnerability Database,“ [Mrežno]. Dostupno: https://nvd.nist.gov/vuln/search. [Pokušaj pristupa 25.5.2022].

[31] „National Vulnerability Database jQuery,“ [Mrežno]. Dostupno: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=jquery&search_type=all&isCpeNameSearch=false. [Pokušaj pristupa 26.5.2022.].

[32] „Minification,“ [Mrežno]. Dostupno: https://developer.mozilla.org/en-US/docs/Glossary/minification. [Pokušaj pristupa 1.6.2022].

Detekcija JavaScript biblioteka i njihovih dodataka korištenjem identifikatora i kriptografskih sažetaka

Sažetak

Web je široko rasprostranjena platforma za razmjenu podataka i pružanje usluga o kojoj mnogi ovise. Zbog velike potražnje za kreiranjem web stranica nude se metode koje pružaju vremenski efikasnije dizajniranje i programiranje. Jedna od njih se svodi na korištenje JavaScript biblioteka koje, između ostaloga, mogu uvesti ranjivosti u web aplikacije. Zbog same količine web stranica, a tako i biblioteka pa i njihovih ranjivosti otežano je održavanje sigurnosti. Javlja se potreba za stvaranjem alata koji će automatiziranim putem, proaktivno tražiti ranjive biblioteke i omogućiti pravovremeno saniranje. Stoga se ovaj rad bavi danom problematikom i pruža uvid u implementirano rješenje. Opisane su JavaScript biblioteke, načini njihova korištenja i pribavljanja, te metode kojima ih se može detektirati. Prikazana je tehnička strana ostvarenja implementacije, te rezultati dobiveni provođenjem detekcije nad skupom web stranica iz hrvatskog mrežnog prostora. Također, prikazana je raspodjela zastupljenosti popularnih biblioteka i njihovih ranjivosti. Komentirana su i ograničenja uočena prilikom implementacije, te njihova potencijalna rješenja s ciljem unaprjeđenja programa.

Ključne riječi: ranjivosti weba, JavaScript biblioteke, detekcija ranjivosti

Detecting JS libraries and their plugins using identifiers and cryptographic hashes

Abstract

The web is a widespread platform for data exchange and service delivery on which many depend. Due to the high demand for website creation, methods are offered that provide more time-efficient design and programming. One of them comes down to using JavaScript libraries that, among other things, can introduce vulnerabilities into web applications. Due to the sheer number of websites, as well as libraries and their vulnerabilities, it is difficult to maintain security. There is a need to create tools that will automatically, proactively search for vulnerable libraries and enable timely remediation. Therefore, this paper deals with the given problem and provides an insight into the implemented solution. JavaScript libraries, how to use and obtain them, and methods for detecting them are described. The technical side of the implementation is presented, as well as the results obtained by detecting a set of web pages from the Croatian web space. Also, the usage distribution of popular libraries and their vulnerabilities is shown. The limitations observed during the implementation were commented on, as well as their potential solutions with the aim of improving the program.

Keywords: web vulnerabilities, JavaScript libraries, vulnerability detection