Zahvaljujem se svojoj obitelji i svim prijateljima za potporu koju su mi pružali tijekom studiranja. Također se zahvaljujem doc. sc. Stjepanu Grošu, mag. ing. Ivanu Kovačeviću i bacc. ing. comp. Bruni Skendroviću na stručnom vodstvu i pomoći pri izradi ovog rada.

Sadržaj

1. Uvod

World Wide Web, ili skraćeno Web, je kolekcija web stranica koje su spremljene na poslužiteljima i spojene na računala preko interneta [2]. Stranice sadrže tekst, digitalne slike, audio snimke, video isječke i druge multimedijske sadržaje. Korisnici mogu pristupiti sadržaju tih stranica sa bilo koje lokacije na svijetu preko interneta koristeći uređaje poput računala, laptopa i mobitela. Jedan od programski jezika koji omogućava stvaranje više kompleksnih značajki na web stranicama je JavaScript. Ako stranice sadrže složenije značajke od samog prikazivanja multimedijskih sadržaja, velika je mogućnost da se u pozadini koristi JavaScript. Ukupno 94% web stranica koristi JavaScript u nekom obliku [4]. Javascript je jedan od najkorištenijih programskih jezika u svijetu, kojeg koristi oko 10 milijuna programera. Kako se popularnost JavaScripta povećavala i više ljudi ga je koristilo za razvoj web stranica i aplikacija, tako je JavaScript zajednica uočila određene dijelove koda koji se ponavljaju kako bi se obavili isti zadatci. Ponavljanje, odnosno prepisivanje koda i prepoznavanje određenih JavaScript funkcija koje su korištene više puta dovelo je do razvijanja JavaScript biblioteka. Prilikom razvoja web stranica programeri učestalo koriste JavaScript biblioteke kako bi ubrzali i olakšali svoj posao.

Upravo zbog svoje popularnosti JavaScript je postao popularna meta nad kojom se izvode kibernetički napadi. Zloćudan kod je pojam kojim opisujemo kod koji se nalazi u bilo kojem dijelu softverskog sistema ili skripti, čija je namjena da proizvede neželjene učinke, kršenje sigurnosti ili nanošenje ikakve štete sustavu [3]. Napadači koriste JavaScript biblioteke kako bi proširili zloćudan kod na druga računala te ga prikrivaju obfuskacijom. Obfusckacija je jedna od tehnika koja se koristi za prikrivanje originalnog koda koji je programer napisao. Sličnu funkcionalnost obfuskaciji posjeduje i minifikacija koja također pretvara originalni kod u oblik koje je teži za razumjeti, no za razliku od obfuskacije, minifikacija smanjuje veličinu koda čime ga ubrzava te joj je to primarna funkcionalnost. Taj proces se provodi brisanjem svakog nepotrebnog djela koda i skraćivanjem imena varijabli te daljnjom optimizacijom koda.

Cilj ovog rada je razviti programski alat za detekciju i raspoznavanje obfuskacije i minifikacije nad JavaScript bibliotekama u hrvatskoj domeni koje su spremljene u MongoDB bazi podataka, opisati obfuskatore i minifikatore te metode koje oni koriste.

Drugo poglavlje opisuje obfuskaciju i obfuskatore te metode koje se koriste pri obfusciranju JavaScript koda. Poglavlje također opisuje deobfuskatore. Programske alate koji uljepšavaju kod i preoblikuju ga u stanje koje sliči izvornom stanju te metode koje koriste.

U trećem poglavlju opisan je proces minifkacije te često korišteni minifikatori poput *jscompress* i toptal *javascript minifier* za JavaScript kod.

Četvrto poglavlje sadrži opis arhitekture programskog alata za detekciju minificiranih i obfusciranih JavaScript biblioteka. Razrađeni su parametri koji se koriste za detekciju poput regexa i entropije i opisano je implementirano programsko rješenje te dobiveni rezultati.

2. Obfuskacija

Obfusciranje je namjerni čin kojim se nešto preoblikuje kako bi se teže razumjelo [5]. Specifično u programiranju se obfuskacijom kod pretvara u verziju s istim funkcionalnostima, ali vrlo ga je teško ili nemoguće razumjeti, kopirati ili mijenjati bez pomoći dodatnih alata. Programski kod često je obfusciran kako bi se sačuvalo intelektualno vlasništvo ili poslovne tajne. Obfusciranje pomaže kod zaštite od reverznog inženjerstva te se često koristi pri prikrivanju malicioznog koda zbog svoje karakteristike težeg razumijevanja. Obfuskacija se može pronaći kod web predložaka koji ne žele lako kopiranje stranica koju servisi pružaju te u igrama koje se nalaze u web pregledniku koje bi se također mogle lako kopirati i izmijeniti jer je cijeli kod korisniku lako dostupan [6]. Moguća korist obfuskacije je pronalazak izvora podijeljenog prodanog koda. Pretpostavljajući da je obfuskacija jeftina operacija nad kodom, autor može svakom kupcu predati različitu obfusciranu verziju te time pronaći izvor otkud dolaze ilegalne kopije.

Kvaliteta obfuskacije mjeri se po četiri stavke [5]:

- Jačina
- Izmijenjenost
- Složenost
- Trošak

Jačina označava razinu truda i vremena koja je potreba za uložiti za deobfuskaciju koda. Izmijenjenost predstavlja razinu po kojoj se dobiveni kod razlikuje od originalnoga. Načini za mjerenje izmijenjenosti su po broju predikata, koji obfuscirani kod sadrži i dubini stabla nasljedstva, gdje veći stupanj označuje veću kompleksnost. Složenost se odnosi na količinu slojeva obfuskacije, što ih više ima to je bolje sakriveno. Zadnja značajka je trošak koja opisuje koliko je teško obfuscirati izvorni kod u njegovu krajnju obfusciranu verziju. Što je manji trošak, to je obfuskacija bolja.

Jedan od najvećih nedostataka obfuskacije je to što se koristi za izbjegavanje detekcije zloćudnih programa preko antivirusnih programa [1]. Antivirusni programi pregledavaju

kod za često korištene programske funkcije i ostale uzorke u kodu koje su učestale u zloćudnim programima. Jedna od često viđenih obfuskacija je XOR ili „ekskluzivno ili“ obfuskacija. „Ekskluzivni ili“ logički je operator koji rezultira istinom kada je jedan od operanda istinit ali oba nisu istinita i oba nisu lažna. Koristeći taj logički operator programer može sakriti važne dijelove koda. XOR obufskacija je popularna zbog jednostavnosti implementacije te velike mogućnosti izbjegavanje detekcije. S obfuskacijom, umjesto razvijanja novog zloćudnog koda, programeri često postave drugu obfuskaciju na krucijalne funkcije kako bi sakrili od detekcije. Jedan dodatan nedostatak obfuskatora je što, osim u slučaju minifikatora, dodaju veliku količinu koda koji nema funkcionalnost nego isključivo služi težem reverziranju programa te kao bolja obrana od automatske detekcije. Takav kod se pojavljuje u literaturi često pod nazivom mrtvi kod(engl. dead code). Zbog navedenog mrtvog koda te ostalih metoda obfuskacije poput obfuskacije podataka, koje će biti objašnjene u poglavlju 2.2. Metode obfuskatora. Takvi kodovi su veći od izvornih kodova i samim time sporiji.

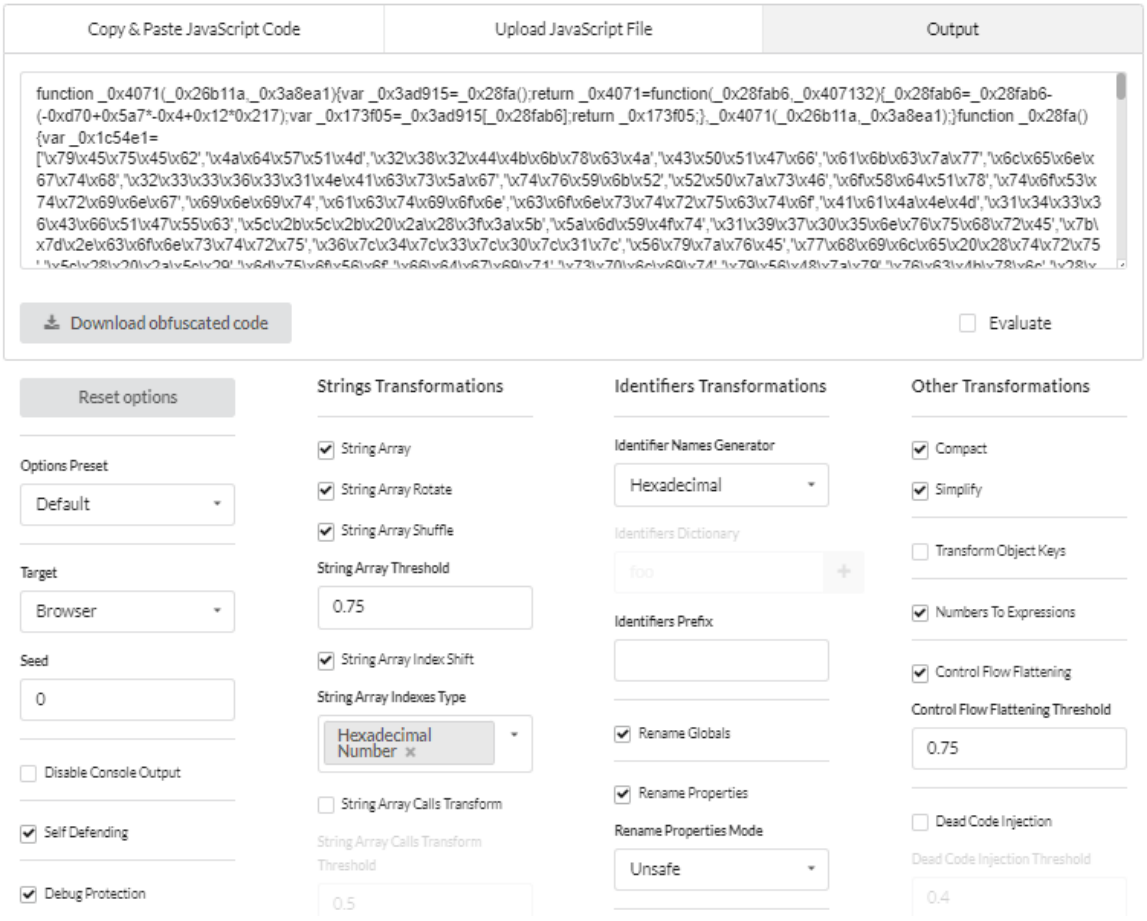
2.1. Obfuskatori

Obfuskatori su programski alati koji omogućavaju laku obfuskaciju izvornog koda. Skoro za svaki programski jezik postoji obfuskator, no ovaj rad je usredotočen na JavaScript obfuskatore. Velika količina obfuskatora pruža web sučelje koje omogućava ukrcavanje izvornog koda te pretvaranje u obfusciranu verziju koda. Jedan od tih alata je obfuscator.io [7]. Na slici 2.1 prikazano je navedeno sučelje obfuscator.io s rezultatom obfusciranja koda čija je funkcija ispisati „Hello World“ na ekran korisnika. Na slici se također mogu vidjeti neke od mnogih postavki koje mijenjaju krajnje obfuscirani kod. „Debug protection“, „Control Flow Flattening“ i „Dead Code Injection“ su objašnjeni u 2.2. odjeljku rada „Metode Obfuskatora“. Obfuscator.io također pruža parametar „Seed“ koji prima cijeli broj i koristi se kao inicijalizator nasumičnosti i osigurava da obfuskator s istim postavkama i istim izvornim kodom producira različite obfuscirane verzije. Navedena funkcionalnost olakšava zlonamjernim programerima da svoj zloćudan kod promijene bez ikakvih izmjena funkcionalnosti koda. Obfuscator.io također pruža opciju korištenja preko komandne linije (end. /command line interface/, CLI). Zbog te opcije je obfuscator.io vrlo lak za implementirati

u produkciju koda te je često korišten kada je potrebno cijeli kod ili neki dio koda u produkciji obfuscirati. U kodu 2.1 prikazan je unos u komandnoj liniji kako bi se obfuscirao ulazna JavaScript datoteka. U primjeru će izlazna obfuscirana datoteka biti naziva izlaz.js, a „source-map-base-url“ označuje mrežnu adresu gdje će „source-map“ biti ukrcan. Source map je datoteka koja pomaže pri transformiranju koda nazad u svoju izvornu verziju, omogućavajući prikazivanje originalnog koda direktno u debuggeru web preglednika [8]. Takva transformacija iznimno je korisna tijekom razvijanja web sadržaja.

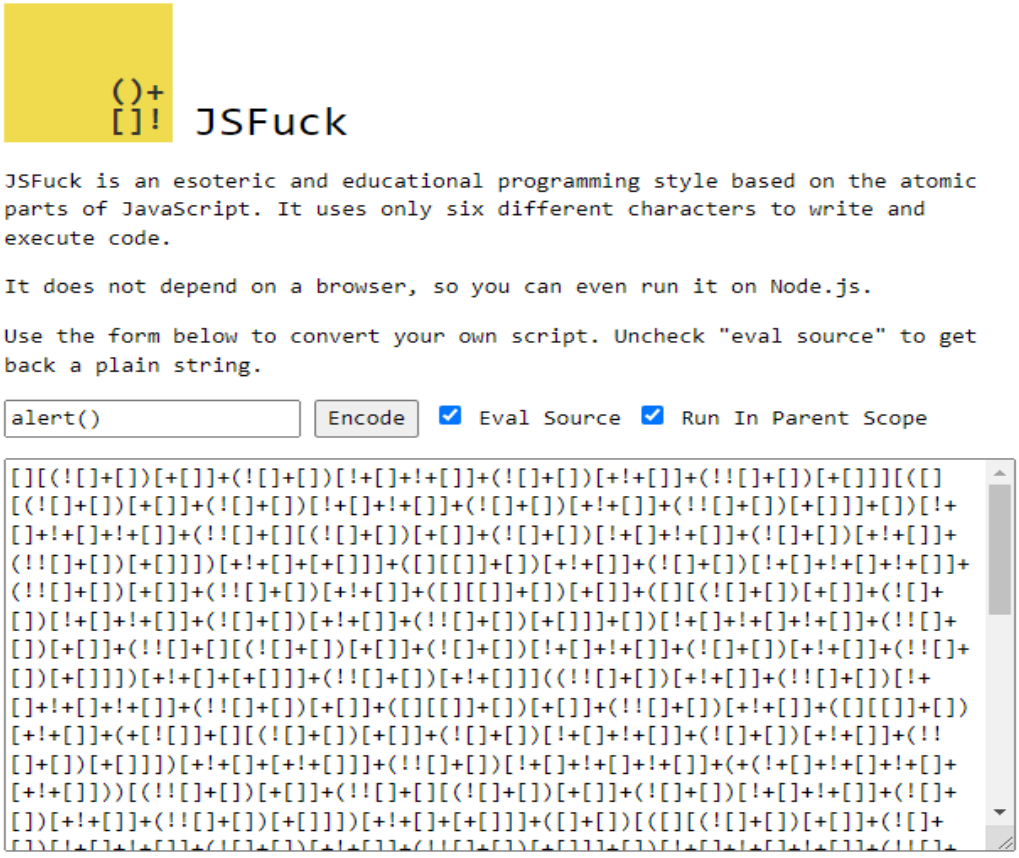
```
javascript-obfuscator ulaz.js --output izlat.js --source-map true --
source-map-base-url 'http://localhost:9000'
```

Kod 2.1: Primjer korištenja obfuskatora preko komandne linije



Slika 2.1: obfuscator.io

Postoje mnogi drugi obfuskatori koji na drugačije načine obfusciraju kod poput *freejsobfuscator* [9] i *JSFuck* [10]. Na slici 2.2 prikazan je ispis *JSFuck* obfuskatora koji izvorni kod pretvara u kombinaciju zagrada, uglatih zagrada, uskličnika i pluseva koje služe kao logički izrazi preko kojih se cijeli kod generira.



Slika 2.2: JSFuck obfuskator

2.2. Metode obfuskatora

Iako postoje mnogi obfuskatori, većina koristi iste metode obfuskacije te zbog toga većina obfuskatora s istim ili sličnim postavljenim postavkama na kraju producira na pogled sličan kod. Ovo poglavlje je usredotočeno na *Self Defending*, *Debug Protection*, *String array*, *Control Flow Flattening* i *Dead Code Injection* metode.

2.2.1. Self Defending

Self Defending je tip funkcionalnosti koji sprječava ikakvu manipulaciju koda [11]. Pri pokušaju mijenjanja koda, kod prepoznaje da se promjena dogodila te prekida izvršavanje koda ili postavlja kod u petlju neograničenog vremena trajanja bez pristupa ikakvih funkcionalnosti.

Prvi korak pri pokušaju razumijevanja obfusciranog koda je korištenje nekih od alata za deobfusciranje ili uljepšavanje koda. Kroz funkcionalnost self defendinga kod prepoznaje da se mijenja te prestaje raditi. Jedna od tehnika self defendinga je postavljane regexa nad nizom znakova koji prouzrokuje katastrofalno nazadovanje. Regexi i katastrofalno nazadovanje objašnjeni su u potpoglavlju 4.1 Regex. Kako bi se implementiralo Self defending koristi se i SHA-1 vrijednost neke varijable ili cijelog dijela koda. SHA-1 ili *Secure Hash Algorithm* 1 je kriptografski sigurnosni algoritam koji za različite unose vraća različite vrijednosti. Koristi se u kriptografskim aplikacijama, i područjima gdje je integritet podataka važan. Ukoliko kod detektira da je SHA-1 vrijednost drugačija nego što bi trebala biti, prekida se izvršavanje koda.

2.2.2. Debug protection

Otklanjanje pogrešaka (engl. debugging) je proces detektiranja i brisanja postojećih ili potencijalnih pogrešaka u kodu koji prouzrokuju neočekivano ponašanje ili moguće prekidanje rada programa. Kako bi se program debuggirao, korisnik prvo mora započeti s problemom te izolirati dio koda koji prouzročuje problem i popraviti ga. Alati za debugganje, također poznati kao debuggeri, služe za identifikaciju pogrešaka u raznim dijelovima razvoja programa. Koriste se kako bi se reproducirala specifična okolina u kojoj se pogreška dogodila te u proučavanju stanja programa u tom trenutku i identificiranju izvora problema. Debuggeri pružaju opciju prelaska koda naredbu po naredbu, gdje se u svakom koraku može provjeravati vrijednosti varijabli i stanja objekata. *Debug protection* pokušava spriječiti debugging nad obfusciranim kodom koristeći funkcionalnosti programskog jezika u kojemu je kod napisan.

Metode korištene za debug protection u JavaScriptu:

- Preimenovanje funkcija
- Lomna točka
- Razlika u vremenu
- Promjena veličine
- Mapa izvora
- Ne implementirana sintaksa

Preimenovanje funkcija prikazan je u kodu 2.2. Ta metoda je najpoznatija i najčešća pri sprječavanju debugginga u kodu. U JavaScriptu moguće je redefinirati funkcije koje se koriste za povrat informacija. U primjeru je korištena funkcija `console.log` koja se koristi kako bi se ispisao neki tekst ili vrijednost varijable. Ako redefiniramo tu funkciju možemo promijeniti njeno ponašanje, sakriti određene informacije ili lažirati cijelu funkciju. Ispis primjera bit će samo „Hello World“ zbog toga što smo funkciju console.log zamijenili za praznu funkciju te zbog toga drugi poziv console.log funkcije neće ništa ispisati.

```
console.log("Hello World");
var prazna_funkcija = function() {};
window['console']['log'] = prazna_funkcija;
console.log("Neću se ispisati");
```

Kod 2.2: Preimenovanje funkcije u praznu funkciju

Ako se umjesto prazne funkcije napiše nova i nju zamijeni sa console.logom ili nekom drugom funkcijom korisnom tijekom debugganja onda je moguće napraviti veću štetu ili potrošiti više vremena osobe koja pokušava debuggirati kod.

U razvoju programa, lomna točka predstavlja mjesto namjernog zaustavljanja ili pauziranja u programu tijekom debugiranja. Kako bi se dio koda preskočio tijekom debugganja, umjesto prelaženja svih naredbi po redu postavlja se lomna točka. Postavljanjem lomne točke prethodni dio se sav, bez prekida, izvodi do linije gdje je lomna točka postavljena. JavaScript sadrži naredbu debugger koja zaustavlja izvođenje programa u debugging načinu

rada, jednako kao što je i postavljanje lomne točke. Ako se u debuggeru pokrene kod 2.3. Lomne točke ispisat će se jedino „Tu sam“ dok se ne pritisne „Continue“ gumb u modu debugganja.

```
console.log("Tu sam");
debugger;
console.log("Nisam tu");
```

Koda 2.3: Lomne točke

Ova funkcionalnost se može ubaciti u neograničenu petlju tako da kada korisnik pritisne „Continue“ gumb odmah mu se prikaze novi „Continue“ gumb i tako sve dok ne uspije na neki način prekinuti petlju. Neki web preglednici ne dopuštaju neograničene petlje, ali u onim u kojima ova funkcionalnost nije detektirana, ona stvara velike vremenske troškove osobi koja pokušava debuggirati takav program.

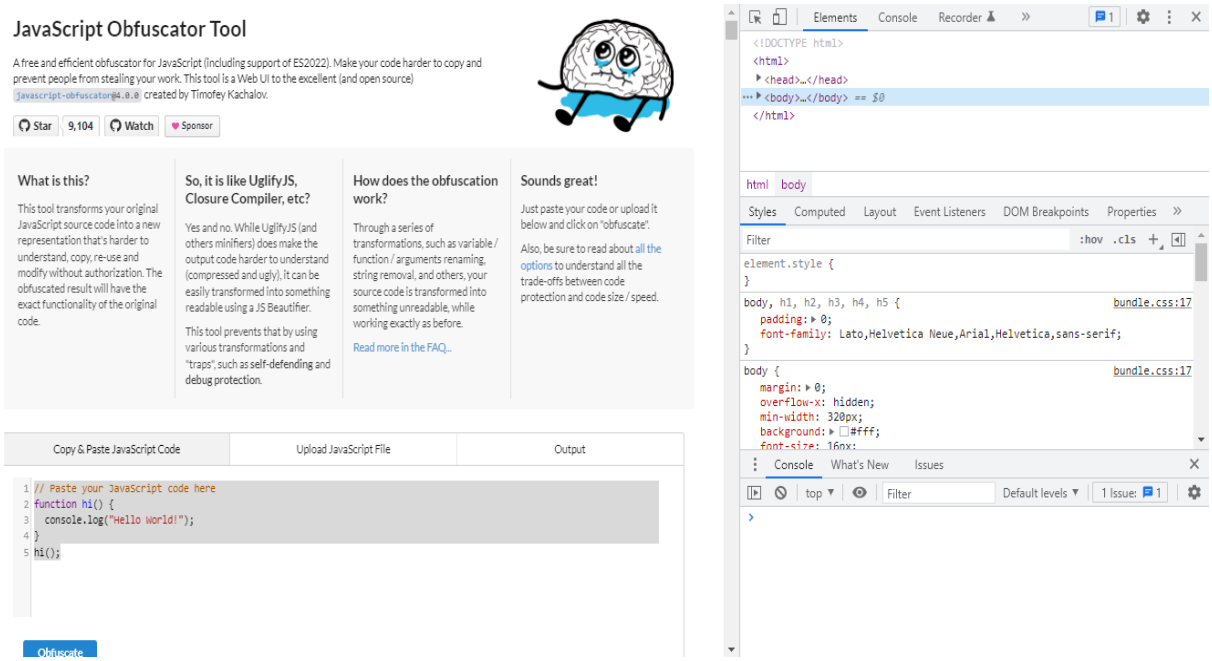
Daljnja mogućnost *Debugg protection-a* je vezana za vrijeme izvršavanja programa. Kada se program izvodi u debuggeru njegovo vrijeme izvršavanje je značajno usporeno. Ova činjenica može biti iskorištena na više načina kako bi se prepoznalo korištenje debuggera. Jedan od načina je mjerenje vremena između izvođenja dvije naredbe, što je prikazano u kodu 2.4. Ukoliko je poznato prosječno vrijeme između izvođenja te dvije varijable, a izvođenje je trajalo značajno dulje od očekivanog, program zaključuje da se radi o debuggiranju i zaustavlja se rad programa.

```
 var startTime = performance.now();
debugger;
var stopTime = performance.now();
if ((stopTime - startTime) > 1000) {
    alert("Debugger detected!")
}
```

Koda 2.4: Vrijeme izvršavanja programa

Drugi način je koristiti funkcije za koje je poznato vrijeme izvršavanja unutar petlje. Ako je nakon petlje prošlo više nego određeno vrijeme detektira se prisutnost obfuskatora.

Alati za razvoj (engl. DevTools) je skupina programa koji omogućavaju programerima stvaranje, testiranje i debuggiranje programa. Web preglednici pružaju integrirani DevTools koji omogućava direktnu analizu web stranica. Kada se DevTools otvori on dijeli ekran sa stranicom kao što je prikazano u slici 2.3 *DevTools*. Upravo ta funkcionalnost omogućava praćenje veličine ekrana kako bi se detektiralo debugiranje. Ako je ekran manji od neke određene veličine detektira se pokušaj debuggiranja i prekida se rad programa.



Slika 2.3: *DevTools*

Metoda mapa izvora je bazirana na HTTP curenju preko JavaScript mapa izvora. HTTP curenje predstavlja situaciju u kojoj određena kombinacija HTML elemenata i atributa prouzročuje slanje zahtjeva za određen resurs [12]. Mape izvora su datoteke koje sadrže informacije o originalnoj JavaScript datoteci prije nego što je bila obfuscirana kako bi DevTool mogao olakšati proces debuggiranja. Datoteka mape izvora povezana je linijom u JavaScript skripti koristeći svoju URL adresu na drugom serveru koja se kroz kod dohvaća. Pri otvaranju DevTools-a, datoteka se dohvaća. Ovo curenje HTTP-a omogućava

obavještavanje činjenice da se DevTools otvara. Ova metoda, također se koristi kako bi se odredio tok koji program mora pratiti. Ograničenja se mogu dodati u kod kako bi se odredilo kojim tokom program mora ići. Navedena ograničenja mogu promijeniti tok programa prema lažnim regijama ili zaustaviti dekriptiranje koda te se mogu spremati u obliku kolačića. Kolačići su datoteke koje su kreirane preko web stranica. Koriste se pri pamćenju povijesti pregleda web stranica, omogućavanju zadržavanja prijavljenosti na web stranicama, pamćenju preferenca stranica i pružanju lokalno relevantnog sadržaja [13]. Ako se vrijednost kolačića postavi kada se od servera zatraži mapa izvora, vrijednost kolačića može se provjeriti kako bi se detektirao pokušaj otvoranja DevTools-a.

Metoda neimplementirane sintakse bazirana je na činjenici da neki alati koji služe za debuggiranje poput *JSDetox* [14] koriste zastarjele verzije JavaScripta čiji parseri ne podržavaju novo implementirane metode. Primjer takve metode je eksponencijalni operator „**“ koji *JSDetox* ne podržava te debuggiranje nije moguće tim alatom ako se u kodu nalazi taj operator.

2.2.3. String Array

String array je metoda obfuskatora koja uzima sve funkcije, vrijednost varijabli i konstanti iz izvornog koda i postavlja ih u niz. Prvotno zamjenjuje sve vrijednosti i konstante s novim pozicijama u nizu gdje se nalaze. Nadalje se može koristiti metoda *String Array rotate* kako bi se pozicije stringova rotirale za određenu vrijednost kako bi se otežalo razumijevanje programa. Nalik *String array rotate* jest *String array shuffle* koja nasumično postavlja vrijednosti na pozicije u niz. Izvršavanje funkcija omogućava se pomoću eval ili document.write funkcija koje služe pretvaranju niza znakova u funkciju koja se izvodi. Navedene metode prikazane su u kodu 2.5 String Array.

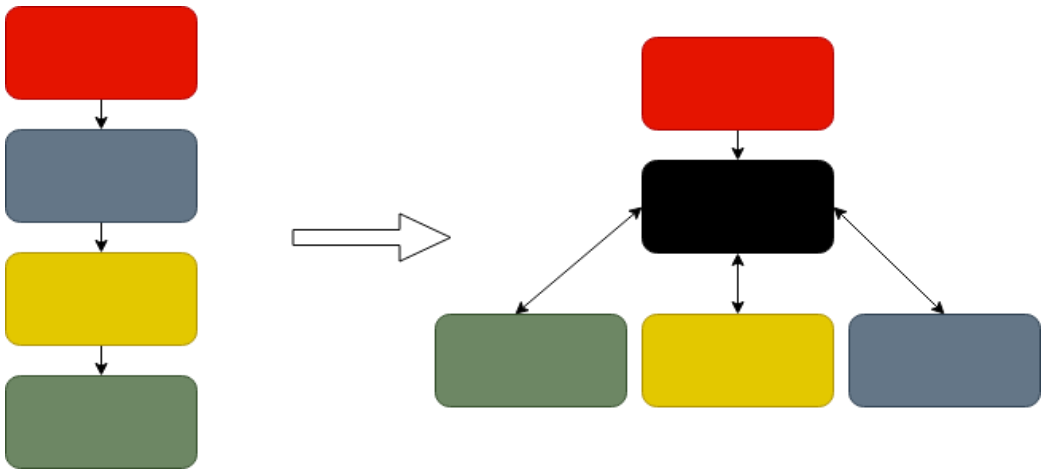
```
console.log(“Hello World”)
----------------
let niz = ["con", "sol", "e.l", "og(", "\"He", "llo", " Wo", "rld", "\")"]
```

```
eval(niz[0] + niz[1] + niz[2] + niz[3] + niz[4] + niz[5] + niz[6] + niz[7] + niz[8])
----------------------
let niz = ["\"He", "llo", " Wo", "rld", "\")", "con", "sol", "e.l", "og("]
eval(niz[4] + niz[5] + niz[6] + niz[7] + niz[8] + niz[0] + niz[1] + niz[2] + niz[3])
------------------
let niz = ["e.l", "llo", "con", "\")", " Wo", "sol", "\"He", "rld", "og("]
eval(niz[2] + niz[5] + niz[0] + niz[8] + niz[6] + niz[1] + niz[4] + niz[7] + niz[3])
```

Kod 2.5: String Array

2.2.4. Control Flow Flattening

Kontroliranje toka izravnavanjem je metoda koja se koristi kako bi se izravnao tok koda. Kako bi se izravnavanje toka koda postiglo, razdvajaju se svi bazični blokovi koda poput tijela funkcije, petlji i uvjetnih grana. Ti blokovi se sastavljaju se u beskonačnu petlju u kojoj switch naredba odabire koji dio koda nastavlja izvršavanje [15]. Takva struktura značajno otežava praćenje izvođenja programa zbog toga što više ne postoje običajne strukture koje čine kod lakšim za čitati. Slika 2.4. „Kontroliranje toka izravnavanjem“ predstavlja apstraktnu verziju kako se jednostavna struktura koda pretvori u izravnatu verziju kontroliranjem toka, gdje obojeni blokovi predstavljaju razdvojene blokove koda, a crni blok glavnu switch naredbu koja kontrolira tok.



Slika 2.4: Kotnroliranje toka izravnavanjem

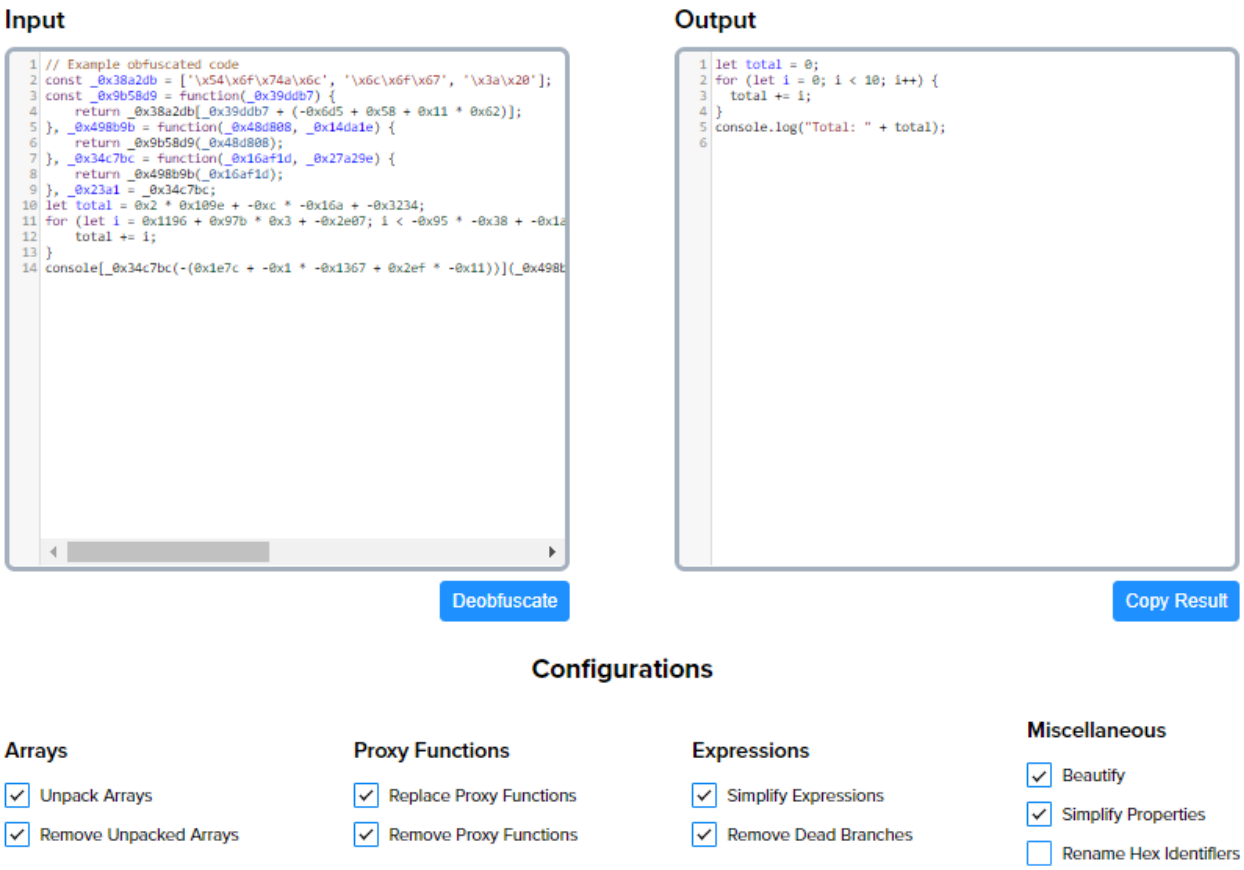
2.2.5. Umetanje mrtvog koda

Pojam mrtvog koda definiran je na dva različita načina. Prvi je kod koji se nikada neće izvesti. Primjer takvog koda je funkcija koja se nalazi unutar „if“ uvjeta u koju kod nikada neće ući. Drugo značenje mrtvog koda je kod čiji rezultat ne utječe nikako na daljnje izvršavanje programa. Mrtvi kod usporava vrijeme izvršavanja i puni memoriju sistema te otežava razumijevanje programa.

2.3. Deobfuskatori

Prevođenje obfusciranog koda nazad u kod koji je lako čitljiv i razumljiv je težak i dug proces koji je često nemoguć bez pomoćnih alata poput deobfuskatora. Deobfuskatori su programski alati koji uljepšavaju kod i preoblikuju ga u stanje koje sliči izvornom stanju. Veliki je problem što se svakim obfuskatorom može dobiti obfucsirani kod, ali ne može se svakim deobfuskatorom dobiti izvorni kod. Također, ne može se vratiti izvorno ime varijabli i funkcija, što je krucijalan dio razumijevanja koda. Alati uljepšavanja koda, poput Js-beautify [16], često ne uspiju deobfuscirati cijelu datoteku, no dobri su pri deobfusciranju minificiranih datoteka. Određene primjere obfusciranog koda nekada je potrebno nekoliko puta deobfuscirati deobfuskatorom kako bi se dostiglo stanje koje se može ručno proučavati. Deobfuskatori često imaju metode koje se odabiru te koriste u deobfuskaciji kao što je prikazanu na slici 2.5 Deobfuscate.io. Slika prikazuje sučelje Deobfuscate.io deobfuskatora koji je napravljen kao besplatni web alat. Alat pruža opcije poput *Unpack Arrays* i *Remove Unpack Arrays* koja uklanja metode poput *String Array* metode objašnjene u potpoglavlju 2.2.3. String Array. *Remove Proxy Functions* uklanja sve nepotrebne pozive dodatnih funkcija kako bi se pozvala izvorna funkcija. Primjer takve funkcije prikazan je u kodu 2.5 „Proxy funkcije“. Opcija *Beautify* uljepšava kod tako da postavlja razmake, tabove i prelaske u nove redove na konvencionalan način na koji su programeri naučeni čitati i pisati kod. *Rename Hex Identifiers* zamjenjuje heksadecimalne vrijednosti s uobičajenim imenima varijabli kako bi se čitane koda olakšalo programeru. Preostale metode *Simplify Expressions* i *Simplify Properties* služe kako bi se izrazi smanjili na najmanju moguću logičku cjelinu

bez mijenjanja sadržaja. Primjer tog slučaja je konkateniranje nizova koji se koriste u naredbi.



Slika 2.5: Deobfuscate.io

```
Proxy funkcije
----------------
function a() {b()}
function b() {c()}
function c() {console.log("Hello World");}
a()
Uklonjene proxy funckije
----------
console.log("Hello World");
```

Isječak koda 2.5: Proxy funkcije

3.0. Minifikacija

Minifikacija proces je smanjivanja količine koda koji se često koristi pri implementaciji web stranica i skriptnih datoteka. Jedna je od glavnih metoda koja se koristi kako bi se ubrzalo prikazivanje stranice korisniku, a također pomaže korisnicima koji imaju ograničen internet jer troši manje interneta od stranice koje nije minficirana. Razlika između originalne i minificirane jQuery JavaScript biblioteke je 60% ili 176kb [17]. Minifikacija se od ostalih metoda kompresije razlikuje po tome što kod zadržava svoju funkcionalnost. Nakon procesa minifikacije kod se pretvori u teže čitljivu verziju koja ima iste funkcionalnosti. Zbog toga se može reći da je minifikacija vrsta obfuskacije. Obfuscirane i minificirane datoteke lako je raspoznati od onih datoteka nad kojima nisu obavljene takve operacije, no u nekim slučajima teško je razlikovati minifikaciju od tipične obfuskacije. Ciljevi minifikacije različiti su od ciljeva obfuskacije, ali kako bi se programski kod minificirao koriste se metode koje obfuskatori koriste. Neke od tih metoda su skraćivanje imena varijabli i refaktoriranje koda. Kada minifikacija koristi takve metode, uljepšivač može u potpunosti vratiti izvoran kod samo ako mu se pružaju detalji oko transformacije koje su te tehnike napravile preko datoteka mape izvora. Ako takav izvor nije predan uljepšivaću, dobiveni kod sadržavat će drugačiji tok programa i imena varijabli.

Prednosti minificiranja JavaScript koda uključuju [18]:

- Smanjeno vrijeme učitavanja stranice
- Manji trošak podataka učitavanja stranice
- Skraćeno vrijeme izvršavanja skripti
- Umanjeni broj HTTP zahtjeva

3.1. Minifikatori

Gotovo je nemoguće mificirati cijeli JavaScript kod u velikim datotekama bez pomoći dodatnih programskih alata. Ručna minifikacija moguća je samo na datotekama male

veličine zbog količine vremena koje je potrebno za obavljanje zadatka. Za lakše rješenje minifikacije dostupni su minifikatori. Minifikatori su programski alati koji pretvaraju izvorni programski kod u njihovu minificiranu verziju. Mnoge integrirane okoline za razvoj poput *Visual Studio Code* omogućavaju minifikaciju u mnogim programskim jezicima u svojim implementacijama. Popularni JavaScript minifikatori dostupni na internetu kao web alati su *jscompress* [19] i *toptal javascript minifier* [20]. Jscompress pruža opciju direktnog pisanja i kopiranja teksta u polje za pisanje ili ukrcavanje cijele JavaScript datoteke izvornog koda. Alat minficira i ispisuje minificirani kod na ekran i pruža opciju skidanja teksta na osobno računalo. Jscompress, također ispisuje razliku u veličini izvornog i minificiranog koda.

4. Opis rješenja

Program je implementiran u programskom jeziku Python koristeći verziju 3.9.6. Fakultet elektrotehnike i računarstva razvio je web pauka koji je sakupio sve naslovne stranice koje se nalaze u hrvatskoj domeni. Te stranice i sav njihov dodatan sadržaj poput biblioteka koje koriste nalazi se u kolekciji crawled_data_pages_v0 u MongoDB *websecradar* bazi podataka. Navedena baza nije javno dostupna te je potrebno prvotno uspostaviti SSH konekciju na bazu. Sav sadržaj baze podatka zapisan je u JSON formatu koji strukturira svaku vrijednost. Biblioteke korištene na stranicama nalaze se u crawled_data_pages_v0 unutar Checks Objects Crawled_links, Scripts polja.
Detekcija obfusciranih i minificiranih biblioteka nije problem za ljudsko oko, ali automatizacija tog procesa nije trivijalna. U nekim slučajevima teško je i čovjeku detektirati razliku između obfusciranih i minificiranih biblioteka, no razlučiti između normalnih biblioteka i minificiranih i obfusciranih lak je zadatak za čovjeka.

Programski alat zamišljen je kao servis koji s MongoDB baze dohvaća URL adrese biblioteka te preko python requests biblioteke dohvaća njihov sadržaj. Nakon što je sadržaj dohvaćen, nad njim se provodi analiza preko pet parametara.

Navedeni parametri su:

- Regexi
- Entropija
- Veličina najveće riječi
- Prosječna veličina riječi
- Omjer najveće riječi i broj znakova cijele datoteke

Svaki parametar povećava vrijednost varijable sigurnosnog indeksa za jedan ili dva. Varijabla sigurnosnog indeksa odlučuje o tipu biblioteke bazirano na krajnjoj vrijednosti indexa.

4.1. Regex

Regularni izraz skraćeno regex je niz znakova koji označuje uzorak pretrage u tekstu [21]. Ti uzorci koriste se u algoritmima koji pretražuju nizove znakova kako se izvele pronađi ili pronađi i zamijeni operacije nad tim nizovima. Koriste se u web tražilicama, programima za procesiranje teksta i leksičkoj analizi. Većina programskih jezika opće namjene poput Pythona, Jave i Javascripta podržavaju regexe izvorno ili preko biblioteka. Regex specificira niz znakova koji su potrebni kako bi se pogodio ciljani niz. Primjer regexa koji odgovara riječima lopta, lopov, lopar prikazan je u kodu 4.1 „Jednostavan primjer regexa“. Prikazan primjer pogađa prva 3 slova svake riječi te nakon, dolazi do „[a-z]“ koji pogađa sva slova koja se po engleskoj abecedi nalaze između slova a i z. Navedi regex ne prepoznaje riječi poput lopoč i Lopar. U prvom primjeru se slovo č ne nalazi između slova a i z u engleskoj abecedi, a u drugom primjeru nije pogođeno veliko slovo L na početku riječi.

```
lop[a-z][a-z]
```

Isječak koda 4.1: Jednostavan primjer regexa.

Korisna funkcionalnost regexa je mogućnost upotrebe zamjenskog znaka koji služi da pogađanje bilo kojeg znaka. Zamjenski znak koji se označuje „.“, kombinira se sa znakovima

„+“ i „*“ koji označuju da se znak prije njih treba ponoviti neodređeni broj puta. Simbol „+“ predstavlja da se znak mora barem jednom ponoviti. Upravo ta kombinacija zamjenskih znakova te znakova za neodređeni broj ponavljanja uzrokuje katastrofalno nazadovanje[22]. Regularni izrazi funkcioniraju tako da počinju pretragu niza znakova od početka koristeći prvu dio uzorka pretrage. Nakon što je pronađen prvi znak koji ne odgovara nizu regularni izraz počinje pretraživati unazad pokušavajući pronaći novi niz koji odgovara. Katastrofalno nazadovanje uzrokuje dugo čekanje izvršavanja programa te nekada i prekidanje samog rada programa zbog trajanja i velikog napora na CPU koje to pretraživanje iskorištava.

4.2. Entropija

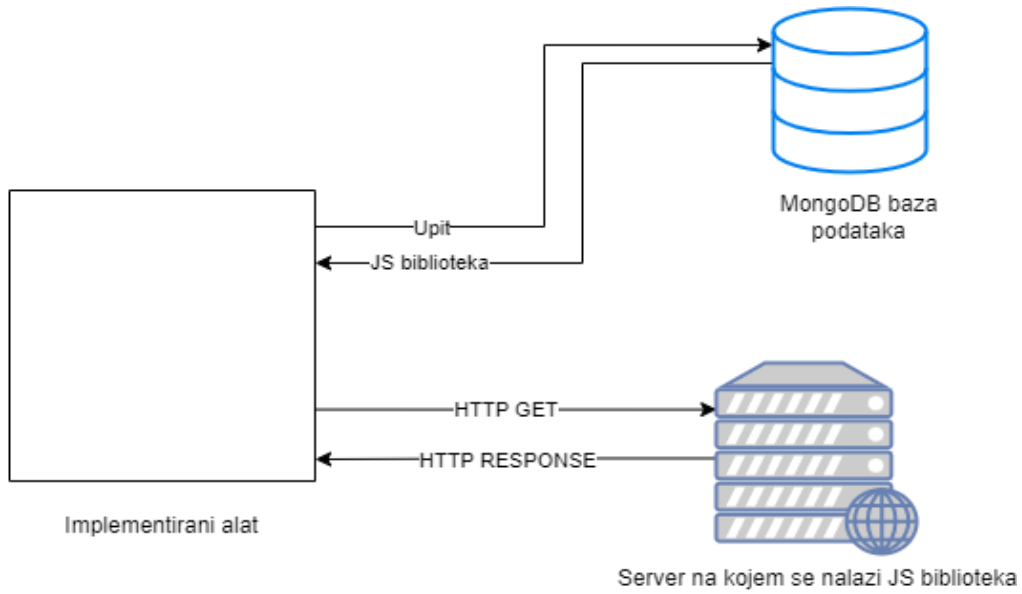
Teorija informacije je polje matematike koje se bavi prijenosom podataka kroz kanale te kompresijom informacija. Jedna od glavnih značajki teorije informacije je entropija. Entropija kvantificira koliko se informacije nalazi u nekom nizu [23]. Entropija se može izračunati za više od same informacije nekog teksta, gdje se takvo računanje obavlja koristeći vjerojatnost. Računanje entropije koristan je alat u području strojnog učenja gdje se koristi kao baza za tehnike poput izgrade stabala odluke, izabira buduće odluke i za korigiranje klasifikacijskih modela. Kod zloćudnog koda entropija se koristi kao razina nasumičnosti u relaciji s dijelovima koda koji zahtijevaju nasumične podatke, poput obfuskacije i kompresije podataka. Obfuskatori koriste metode koje se oslanjaju na mnoge metode poput kompresije tektsa i korištenje raznih enkripcijskih algoritama kako bi se originalan kod promijenio u nešto što je teško za razumjeti. Taj kod koji se s tim procesom stvori ima prosječno manju entropiju nego kodovi koji nisu prošli proces obfuskacije, ali znatno manju nego kodovi koju su prošli proces minifikacije ili kompresije. Najčešće korišten način za računanje entropije u kodu baziran je na Shannonovoj formuli(1). Koristeći navedenu formulu svakoj datoteci pridodana je vrijednost od nula do osam. Gdje manja vrijednost označuje veću vjerojatnost da je datoteka obfuscirana, a veća vrijednost veću vjerojatnost da je datoteka minficiana ili enkriptirana nekim algoritmom. Uobičajene tekstualne datoteke oslanjaju se na lingvistička pravila, poput činjenice da nakon „q“ generalno slijedi „u“, i generiraju vrijednost oko 4.5 koristeći Shanonovu formulu.

Programski kodovi prosječno imaju vrijednost entropije oko 5. Analiziranje nivoa entropije nužan je korak u detektiranju obfusciranih, minificiranih i enkriptiranih kodova.

$$\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{1}$$

4.3. Implementacija

Skica arhitekture sustava vidljiva je na slici 4.2. Arhitektura sustava. Alat prvotno šalje upit websecradar bazi podataka kako bi dohvatio URL-ove biblioteka. Alat parsira dobivene biblioteke i uklanja sve dodatne nepotrebne znakove na početku ili kraju URL-ova poput znaka navodnika. Nakon što su svi dodatni znakovi uklonjeni skripta šalje HTTP GET upite, pomoću python request biblioteke, na dobivene URL-ove kako bi se dobio kod biblioteke koji se šalje na analizu. Ukoliko status kod od HTTP upita nije 200 biblioteka je označena kao nedostupna i analiza prelazi na sljedeću datoteku.



Slika 1.1: Arhitektura sustava

Analiza započinje prolaskom biblioteke kroz listu regexa za prepoznavanje minificiranih i obfusciranih biblioteke od normalnih. Lista je sastavljena od 20 ručno napisanih regexa koji

pogađaju uzorke koji su učestali u minificiranim i obfusciranim datotekama. Kod 4.1 sadrži 3 regexa koja se nalaze u prvoj skupini regexa. Prvi regex podudara se s obfusciranim nizom JSFuck obfuskatora sa praznim uglatim zagradama koje se zbrajaju. Drugi regex podudara se s funkcijom koja prima četiri argumenta i priprema povratnu vrijednost bez znaka novog reda ili nepotrebnih razmaka. Treći regex služi za podudaranje s uzastopnom inicijalizacijom varijabli čija su imena i vrijednosti duljine jednog znaka. Treći regex također ne sadrži nepotrebne razmake ili druge metode uljepšavanja izraza koje se ne koriste u minifikatorima ili obfuskatorima.

```
\[\]\+\[\]
function\([a-zA-Z],[a-zA-Z],[a-zA-Z],[a-zA-Z]\)\{return
[a-zA-Z]=.,[a-zA-Z]=.,[a-zA-Z]=.,[a-zA-Z]=.,[a-zA-Z]=.,[a-zA-Z]=.,[a-zA-
Z]=.,[a-zA-Z]=.,[a-zA-Z]=.
```

Kod 4.1: Regexi minifikacije i obfuskacije

Za svaki regex koji se podudario s bibliotekom varijabla indeksa sigurnosti povećava se za jedan. Drugi dio analize je provjera vrijednosti entropije. Za računanje entropije korištena je python kolekcija „counter“ koja broji pojave znakova u nizu. Pomoću brojača izračunata je frekvencija pojave znaka te se s tom vrijednosti i pomoću formule za Shannonovu entropiju dobije vrijednost. Ako je dobivena vrijednost veća od 5.05 sigurnosni index poveća se za jedan ako je veći od 5.3 poveća se za 2. Biblioteka dalje prolazi kroz analizu veličine najveće riječi i ukoliko se uspostavi da je vrijednost veća od 200 indeks sigurnosti se povećava za jedan, a ako je duljina veća od 350 povećava se za 2. Daljnja analiza uključuje računanje prosječene duljine riječi i omjer najveće riječi i ukupne duljine biblioteke. Ukoliko je 10% ukupne veličine datoteke najdulja riječ vrijednost sigurnosnog indeksa povećava se za 1. Prosječna duljina riječi u biblioteci povećava sigurnosni indeks za 1 ukoliko je veća od 11 ili za 2 ukoliko je veća od 45. Sve vrijednosti povećanja sigurnosnog indeksa zapisane su u tablici 4.1: „Povećanje sigurnosnog indeksa“. Varijabla indeksa sigurnosti služi kao završna odluka o tipu biblioteke. Ukoliko finalna vrijednost varijable iznosi četiri ili više datoteka se označuje kao obfuscirana ili minificirana i šalje se na daljnju analizu.

Tablica 4.1: Povećanje sigurnosnog indeksa

	Povećanje za jedan	Povećanje za dva
Regex	Svaki pogođeni regex	
Entropija	Veće od 5.05	Veće od 5.3
Veličina najveće riječi	Veće od 200	Veće od 350
Prosječna veličina riječi	Veće od 11	Veće od 45
Omjer najveće riječi i broj znakova cijele datoteke	Veći od 10%	
Većina cijele datoteke	Veća od 1000000	

Nakon što su se pomoću sigurnosnog indeksa razdvojile minificirane i obfuscirane biblioteke od onih koje sadrže izvorni kod slijedi detekcija minificirane i obfuscirane biblioteke. Nalik prvom krugu detekcije drugi krug započinje listom regexa za prepoznavanje obfuskacije. U kodu 4.2. prikazana su tri regexa koja služe razlikovanju obfuskacije od minifkacije. Prvi regex prepoznaje često korištenu funkciju čiji argumenti ispisuju riječ packer ili packed. Drugi regex prepoznaje beskonačnu while petlju koja se koristi mnogim metodama poput zaštite od debugganja. Treća metoda podudara s oktalnom vrijednosti koja se nalazi unutar zagrada. Oktalne i heksadecimalne znamenke učestalo se koriste u obfuskatorima.

```
eval\(function\(p,a,c,k,e,d\)
"while\(\!\!\[\]\)
\(0x[1-9][0-9]+\)
```

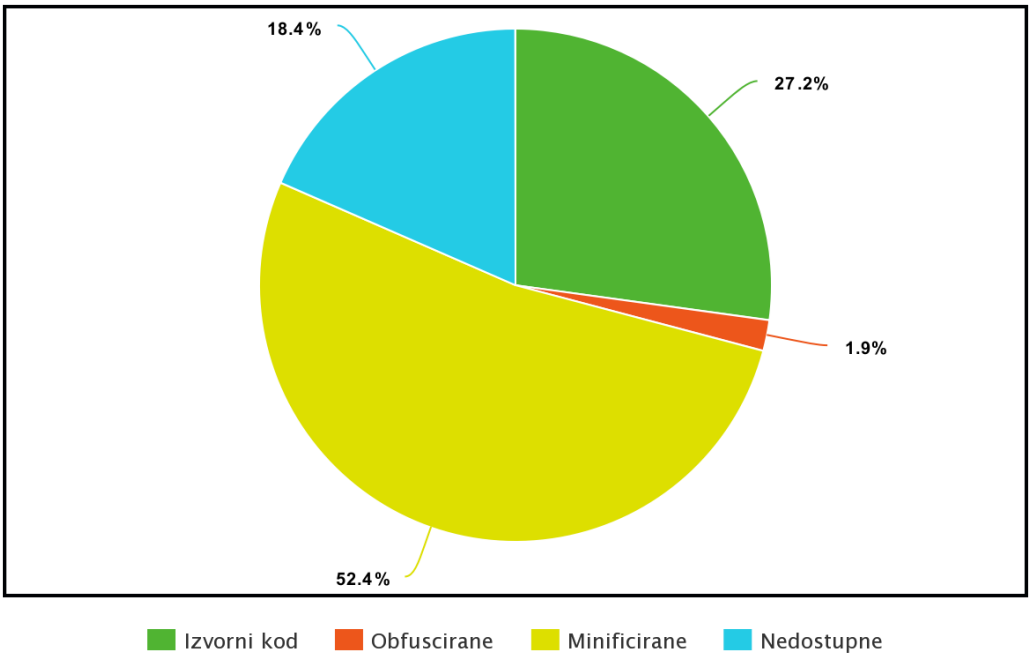
Kod 4.2: Regexi za obfuskaciju

Indeks obfuskacije, koji služi za detekciju obfuskacije, uvećan je za jedan za svaki regex koji se podudara sa bibliotekom i uveća je za jedan ukoliko je entropija biblioteke manja od 3. Biblioteka je označena kao obfuscirana ako je indeks obfuskacije jedan ili veći. Zadnji korak alata je zapisivanje tipa biblioteke i svih parametara preko kojih se provela analiza u bazu podataka.

4.4. Rezultati

Programski alat testiran je na ručno klasificiranoj bazi od 2000 uzoraka iz prije navedene *websecradar* MongoDB baze podataka. Vrijednost sigurnosnog ideksa značajan je utjecaj imao na broj lažnih pozitiva i negativa. Prve iteracije alata koristile su vrijednost 3 kao granicu klasifikacije. Koristeći tu vrijednost program je sadržavao veliki broj lažnih pozitiva za skup minificiranih i obfusciranih datoteka zbog parametara poput veličine najveće riječi i omjera najveće riječi i veličine datoteke. Ukoliko kod biblioteke sadrži iznimno dugu funkciju ili drugi ukrcani alat poput regexa ti dijelovi koda sadržali su jako duge količine teksta bez razmaka što bi povećavalo sigurnosni indeks za dva u slučajevima lažnih pozitiva. Unutar testiranih biblioteka nalazile su se također one koje su dovoljno male da je njihova najveća riječ duljine 10% ili više ukupne veličine biblioteke što povećava sigurnosni indeks za jedan. Kombinirajući takva dva slučaja datoteka je klasificirana kao obfuscirana ili minificirana iako je biblioteka izvornog koda

Uspostavilo se da je alat u nekim slučajevima bolji nego ručna klasifikacija. U određenim slučajevima kada je kod dovoljno dug da je ručna klasifikacija teška, alat pomoću regexa prepoznaje i male funkcije koje su obfuscirane ili minificirane. Te funckije najčešće sadrže know-how tajne ili maliciozne funckije koje programeri pokušavaju sakriti. Prosječna brzina klasifikacije pojedinačne biblioteke je pola sekunde. Većina vremena potrošena je na prolaženju regexa gdje se nalazi i najveći potencijal napredka brzine alata. Bolje napisani regexi koji koriste manje zamjenskih znakova uveliko povećavaju brzinu pretrage. Rezultati korištenja alata nad 10000 nasumično odabranih biblioteka prikazan je u slici 5.1: Biblioteke u MongoDB bazi podataka. Tortni grafikon prikazuje da većina testiranih biblioteka pripada skupu minificiranih biblioteka. Većina minificiranih biblioteka verzija je jQuery biblioteke koja je često implementirana kao minificirana verzija. 18.4% biblioteka nisu dostupne pri pokušaju dohvaćanja njihovog koda. Ta statistika opisuje kako su mnoge stranice neodržavane te se na njima također potencijalno nalazi ubačeni zloćudan kod. Ostatak grafikona podijeljen je na biblioteke izvornog koda i one koje su obfuscirane. Mali udio obfusciranih biblioteka detektiranih implementiranim alatom pokazuje kako obfuskacija trenutno nije često korištena tehnika.



Slika 5.1: Biblioteke u MongoDB bazi podataka

5. Zaključak

Obfusciranje je namjerni čin kojim se nešto preoblikuje kako bi se teže razumjelo. Koristi se kako bi se sačuvale tajne poput važnih funkcionalnosti koda i kako bi se izbjegla detekcija zloćudnog koda. Minifikacija proces je smanjivanja količine koda koji se često koristi pri implementaciji web stranica i skriptnih datoteka. Obfuskacija i minifikacija vrlo su slična dva procesa koji koriste veliku količinu istih metoda u svojim implementacijama. Ručno klasificarnje biblitoeka obfusciranog, minificiranog i izvornog koda lak je, no težak i dug posao. Zbog činjenice da su obufscirane biblioteke prosječno složenije pa samim time i veće, pokazalo se da korištenje metoda za proučavanje veličine riječi i sličnih parametara uz regex i entropiju proizvodi povoljan ishod. Razvijeni alat ubrzava i u nekim slučajevima korektnije klasificira biblioteke i potvrđuje uspješnost detekcije koristeći odabranu kombinaciju parametara. Daljnje razvijanje alata omogućava korektniju detekciju obfusciranih i minificiranih biblioteka uvođenjem novih parametara za detekciju te boljom manipulacijom sigurnosnog indeksa. Pokazalo se da program JavaScript biblioteke detektira pouzdano i značajno ubrzava klasifikaciju u odnosu na ručno klasificiranje.

6. Literatura

[1] N. Malviya, Simple malware obfuscation techniques, InfoSec Institute (2020. srpanj). Poveznica: https://resources.infosecinstitute.com/topic/simple-malware-obfuscation-techniques/; pristupljeno 20. svibanja 2022.

[2] What is World Wide Web?, Javatpoint. Poveznica: https://www.javatpoint.com/what-is-world-wide-web; pristupljeno 17. svibanja 2022.

[3] Malicious code, VeraCode. Available: https://www.veracode.com/security/malicious-code; pristupljeno 18. svibanja 2022].

[4] How much javascript you need for web development, Conquercoding. Poveznica: https://conquercoding.com/how-much-javascript-you-need-for-web-development/https://conquercoding.com/how-much-javascript-you-need-for-web-development/; pristupljeno 17. svibanja 2022.

[5] B. Lutkevich, obfuscation, Techtarget. Poveznica: https://www.techtarget.com/searchsecurity/definition/obfuscation; pristupljeno 19. svibanja 2022.

[6] C. Delgado, Why and when should you obfuscate your javascript code, Ourcodeworld, (2021 siječanj). Poveznica: https://ourcodeworld.com/articles/read/1439/why-and-when-should-you-obfuscate-your-javascript-code; pristupljeno 22. svibanja 2022.

[7] Poveznica: https://obfuscator.io/. Pristupljeno 25. svibanja 2022.

[8] Use a source map, Firefox mozilla. Poveznica: https://firefox-source-docs.mozilla.org/devtools-user/debugger/how_to/use_a_source_map/index.html; pristupljeno 3. lipanja 2022.

[9] freejsobfuscator. Poveznica: http://www.freejsobfuscator.com/; pristupljeno 28. svibnja 2022.

[10] JSFuck. Poveznica: http://www.jsfuck.com/. pristupljeno: 28. svibnja 2022.

[11] Self defending capabilities with no effort, Jscrambler (2014. ožujak). Poveznica: https://blog.jscrambler.com/self-defending-capabilities-with-no-effort; pristupljeno: 29. svibnja 2022.

[12] G. Weizman, Javascript Anti Debugging - Some Next Level Stuff (Part 1 - Abusing SourceMappingURL), Perimeterx, (2019. prosinac). Poveznica: https://www.perimeterx.com/tech-blog/2019/javascript-anti-debugging-1/; pristupljeno 29. svibnja 2022.

[13] What are Cookies?, Kaspersky. Poveznica: https://www.kaspersky.com/resource-center/definitions/cookies; pristupljeno 23. svibnja 2022.

[14] S. Taute, JsDetox. Poveznica: https://github.com/svent/jsdetox; pristupljeno 30. svibnja 2022.

[15] Jscrambler 101 — Control Flow Flattening, Jscrambler. Poveznica: https://blog.jscrambler.com/jscrambler-101-control-flow-flattening; pristupljeno 1. lipnja 2022.

[16] E. Lielmanis, js-beautify. Poveznica: https://beautifier.io/; pristupljeno 30. svibnja 2022.

[17] Minification, imperva. Poveznica: https://www.imperva.com/learn/performance/minification/; pristupljeno 1. lipnja 2022.

[18] Z. Powell, How to Minify JavaScript — Recommended Tools and Methods, Kinsta, (2022. svibanj). Poveznica: https://kinsta.com/blog/minify-javascript/; pristupljeno 28. svibnja 2022.

[19] JSCompress, Mrežno. Available: https://jscompress.com/; pristupljeno 29. svibnja 2022.

[20] Online JavaScript Minifier Tool and Compressor, with Fast and Simple API Access, Toptal. Poveznica: https://www.toptal.com/developers/javascript-minifier. pristupljeno 29. svibnja 2022.

[21] Regex, Computer Hope, (2020 prosinac). Poveznica: https://www.computerhope.com/jargon/r/regex.htm. pristupljeno 1. lipnja 2022.

[22] B. Crawley, Checking Strings: Avoiding Catastrophic Backtracking, Appway (2017. studeni). Poveznica: https://community.appway.com/screen/kb/article/checking-strings-avoiding-catastrophic-backtracking-1482810891360; pristupljeo 2. lipnja 2022.

[23] J. Brownlee, A Gentle Introduction to Information Entropy, Machine Learning Mastery, (2020. srpanj). Poveznica: https://machinelearningmastery.com/what-is-information-entropy/. pristupljeno 2. lipnja 2022.

Detekcija i analiza obfusciranih i minificiranih JavaScript biblioteka

Sažetak

JavaScript biblioteke omogućavaju veću brzinu i jednostavnost programiranja Web sadržaja. Kako bi se prikrile know how tajne i maliciozan kod koristi se obfuskacija. Obfusciranje je namjerni čin kojim se nešto preoblikuje kako bi se teže razumjelo. Često korišteni obfuskatori poput obfuscate.io pružaju mnogo metoda koje proizvode različite obfuscirane verzije istog izvornog koda. Sličnu funkcionalnost obfuskaciji posjeduje i minifikacija koja također pretvara originalni kod u oblik koje je teži za razumjeti, no za razliku od obfuskacije, minifikacija smanjuje veličinu koda čime ga ubrzava te joj je to primarna funkcionalnost.

U ovom radu dan je pregled obfuskacije i minifikacije te metoda koji alati za obfusciranje i minificiranje koriste. Razvijen programski alat pomoću parametara poput regexa, entropije i veličine riječi detektira i razlikuje minificirane i obfuscirane JavaScript biblioteke. Prikazan je rezultat izvođenja programskog alata nad bazom podataka koja sadrži sve naslovne stranice koje su u hrvatskoj domeni. Rezultati prikazuju veliku prisutnost minificiranih te malu količinu obufsciranih JavaScript biblioteka. Ovakvo automatizirano detektiranje pokazalo se iznimno bržim i u nekim slučajevima boljom od ručnog detektiranja obfuskacije i minifikacije.

Ključne riječi: obfuskacija, minifikacija, maliciozan kod, JavaScript, kibernetička sigurnost

Detection and analisys of obfuscated i minified JavaScript libraries

Abstract

JavaScript libraries allow for faster and easier programming of Web content. In order to conceal the know-how secrets and malicious code, obfuscation is used. Obfuscation is a deliberate act of reshaping something to make it harder to understand. Commonly used obfuscators such as obfuscate.io provide many methods that produce different obfuscated versions of the same source code. Minification has similar functionality to obfuscation, which also turns the original code into a form that is harder to understand, but unlike obfuscation, minification reduces the size of the code, which speeds it up, and is its primary functionality.

This paper provides an overview of obfuscation and minification and the methods used by obfuscation and minification tools. The developed software tool uses parameters such as regex, entropy and word size to detect and distinguish minified and obfuscated JavaScript libraries. The result of running the software tool on a database containing all title pages that are in the Croatian domain is presented. The results show a high presence of minified and a small amount of obfuscated JavaScript libraries. This automated detection has proven to be extremely faster and in some cases better than manual detection of obfuscation and minification.

Keywords: obfuscation, minification, malicious code, JavaScript, cyber security