SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1174

Analiza i mjerenje izlaza obfuskatora JavaScript koda

Matija Jelavić

Zagreb, srpanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 1174

Pristupnik:	**Matija Jelavić (0036529517)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Stjepan Groš

Zadatak: **Analiza i mjerenje izlaza obfuskatora JavaScript koda**

Opis zadatka:

Obfusciranje JavaScript koda često se koristi za prikrivanje zloćudnih aktivnosti. Prema tome, detektiranjem obfusciranog JavaScript koda moguće je detektirati potencijalno zloćudan kod za koji se naknadnom analizom može utvrditi je li zloćudan ili nije. Problem je što nije jednostavno razlikovati obfuscirani kod od minificiranog, tj. od koda koji je mijenjan isključivo s ciljem smanjenja količine podataka koji je potrebno dohvaćati. Strojno učenje je potencijalni pristup detekciji obfusciranog koda, ali kako bi se strojno učenje primijenilo nužno je imati veliku količinu podataka nad kojima će se obavljati učenje. S obzirom da moguć broj alata i opcija na ulazu potrebno je utvrditi koji je minimalan smisleni skup podataka koji treba generirati. U sklopu završnog rada potrebno je pronaći slobodno raspoložive obfuskatore za programski jezik JavaScript te ih usporediti sa alatom obfuscator.io. Treba utvrditi jesu li njihove mogućnosti kao i izlaz dovoljno različiti od alata obfuscator.io kako bi se uključili u proces generiranja skupa podataka. Radi utvrđivanja različitosti potrebno je definirati mjeru ili nekakav kriterij koji će određivati različitost. Dodatno, potrebno je proučiti koje opcije alata obfuscator.io treba koristiti, a koje se mogu zanemariti na temelju kriterija što bi napač koji želi prikriti zloćudni kod koristio. Radu priložiti izvorni kôd. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 9. lipnja 2023.

SADRŽAJ

1. Uvod

JavaScript je skriptni jezik koji u području razvoja za web ima razne primjene, a prema istraživanjima stranice W3Techs, JavaScript se koristi kao programski jezik na strani klijenta u 98.6% web stranica [1]. Samim time što je JavaScript toliko upotrebljavan programski jezik, postaje meta koju napadači žele iskoristiti, a razne obrambene mehanizme pokušavaju zaobići korištenjem obfuskacije. Pojam obfuskacija, u kontekstu računalstva, odnosi se na tehniku otežavanja čitanja izvornog koda, ali zadržavajući istu funkcionalnost te mogućnost izvršavanja.

Prema tome, detektiranjem obfusciranog JavaScript koda moguće je detektirati potencijalno zloćudan kod te naknadnom analizom utvrditi je li zloćudan ili nije. Strojno učenje je potencijalni pristup detekciji obfusciranog koda, a s obzirom na to da postoje razni obfuskatori s različito definiranim opcijama, potrebno je utvrditi koji je minimalni smisleni skup podataka koji je potrebno generirati za strojno učenje.

Cilj ovog rada je dati osnovni uvid u obfuskaciju i tehnike korištene prilikom obfuskacije te predstaviti javno dostupne obfuskatore za programski jezik JavaScript i usporediti ih s alatom obfuscator.io [2] kako bi se utvrdilo je li ih potrebno uključiti u proces generiranja obfusciranih kodova za strojno učenje.

Rad se sastoji od šest poglavlja. Nakon uvodnog poglavlja, slijedi drugo poglavlje koje daje osnovni uvid u obfuskaciju i razne metode obfuskacije. Treće poglavlje opisuje alat obfuscator.io te ukratko predstavlja ostale obfuskatore, dok četvrto poglavlje opisuje alat za uspoređivanje obfuskatora, njegovu implementaciju i testiranje. Naposljetku slijede peto i šesto poglavlje, tj. zaključak i literatura.

2. Obfuskacija

Prvi spomeni obfuskacije koda pojavljuju se 1970-ih godina [3], a obfuskacija se koristi za razne programske jezike kao što su C, C++ ili Perl. Ipak, najveću popularnost stekla je za JavaScript, upravo zbog njegove velike uloge u web stranicama. Primjer izvornog koda vidi se na ispisu 2.1, a obfuscirana varijanta se vidi na ispisu 2.2.

```
console.log("Hello world!");
```

Kod 2.1: Izvorni kod

```
function _0x1ac0(){var _0x636740
   ['log','Hello\x20World!','1whVrDt','1564370DvKWLc','
12qlUXly','740128cCAjoS','60rhJTGs','157944DIjdEz',
'8442567IcmLZU','7606760ZiqPse','3792141VkSAFZ',
'37161140XnMPJW'];_0x1ac0=function(){return
   _0x636740;};return
   _0x1ac0();}(function(_0x4a30ff,_0x2c4865){function
   _0x5384c8(_0x127621,_0x18e076){return _0x4f73(_0x127621-
   -'0x12d',_0x18e076);}var _0x18e319=_0x4a30ff();
```

Kod 2.2: Isječak obfusciranog koda

Postoje tri ključne metrike koje definiraju obfuskaciju: jačina, elastičnost i trošak [4]. *Jačina* definira do koje mjere je obfuscirani kod nečitljiv čovjeku, a može se izračunavati pomoću Halsteadovih mjera složenosti [5]. Budući da određeni obfuscirani kod može biti velike jačine, a i dalje se uz pomoć automatiziranih alata lako transformirati u čovjeku razumljivu verziju, definira se *elastičnost* tj. otpornost obfusciranog koda na deobfuskatore, alate koji pretvaraju obfuscirani kod u oblik sličan izvornom kodu. Naposljetku, *trošak* definira koliko obfuscirani kod usporava vrijeme izvršavanja aplikacije te koliko povećava veličinu datoteke.

Osim skrivanja zloćudne namjere, obfuskacija se koristi i u druge svrhe. Jedan od

razloga je zaštita intelektualnog vlasništva i prikrivanje programske logike koda koji se izvršava na klijentskoj strani kao što su ReCAPTCHA ili PerimeterX. Osim toga, može služiti zaštiti od piratstva, varanja ili izvlačenja podataka [6].

Iako se često koristi u istom kontekstu kao obfuskacija, potrebno je razlikovati pojmove obfuskacije i minifikacije. Obfuskaciji je cilj prikrivanje određene programske logike dok je minifikaciji primarni cilj ubrzavanje izvršavanja koda i optimizacija korištenja memorije, što postiže izbacivanjem znakova poput bjelina. Ipak, proces minifikacije može se koristiti kao jedan od koraka prilikom obfuskacije. Primjer izvornog koda je prikazan u ispisu 2.3 pa potom istog tog koda u minificiranoj verziji u ispisu 2.4.

```
function greet_user(username) {
alert('Hi ' + username + "!");
}
greet_user("Simon");
```

Kod 2.3: Izvorni kod

```
function gu(e){alert("Hi "+e+"!")}gu("Simon");
```

Kod 2.4: Minificirani kod

Također, potrebno je razlikovati pojmove šifriranja i obfuskacije. Šifriranje je proces kojim se vrši izmjena podataka tako da se poruka, odnosno informacije, učine nečitljivim za osobe koje ne posjeduju određeni ključ. Temeljna razlika između navedena dva pojma je što bi se šifrirana skripta prvo morala otključati pa tek onda izvršiti dok se obfuscirana skripta može i u obfusciranom obliku izvršavati.

2.1. Metode obfuskacije

Iako se kao metoda obfuskacije može definirati bilo koja transformacija koja uspješno prikriva programsku logiku, iz čega proizlazi da postoji mnoštvo takvih transformacija, u nastavku su navedeni pojedini koji se u obfuskatorima mogu vidjeti navedeni kao zasebne opcije.

2.1.1. Manipulacija niza znakova

Manipulacija niza znakova predstavlja bilo koji oblik transformacije niza znakova. Primjer manipulacije niza znakova je pretvaranje svakog pojedinog znaka u ASCII vrijed-

nost heksadekadskog oblika te potom spremanje u polje na slučajno odabrane indekse. Nad nizom znakova u izvornom kodu 2.5, mogu se vidjeti dvije prethodno opisane metode manipulacije nizova znakova u kodovima 2.6 i 2.7. Također, navedeno polje može sadržavati elemente na drugim indeksima, koji se ne koriste u daljnjem tijeku izvođenja programa.

```
console.log("Hello, world!" + 123);
```

Kod 2.5: Izvorni kod

```
console["\x6C\x6F\x67"]("\x48\x65\x6C\x6C\x6F\x2C\x20\x77\x6F
\x72\x6C\x64\x21\x20" + 123);
```

Kod 2.6: Pretvaranje niza znakova u ASCII vrijednost

```
var _0x8b75=["Hello,
   world!","log"];console[_0x8b75[1]](_0x8b75[0]+ 123)
```

Kod 2.7: Spremanje niza znakova u polje

Iz prethodna dva primjera može se zaključiti da je lako uočiti i identificirati nizove znakova u obfusciranom kodu, no postoje metode manipulacije niza znakova koje, kada se primijene, rezultiraju jako teškom rekonstrukcijom originalnog niza znakova. Navedene metode uobičajeno koriste manipulaciju bitova te razne JavaScript funkcije kao što su *String.fromCharCode*, *slice* ili *concat*.

Uz sve navedeno, također se koriste i razne vrste kodiranja nizova znakova koristeći kodere kao što su *base64*, *rc4* i sl.

2.1.2. Preimenovanje identifikatora

Preimenovanje identifikatora označava promjenu imena identifikatora u vrijednosti koje ne označavaju ono što identifikator predstavlja. Svi obfuskatori istraženi za potrebe ovoga rada, prema početnim postavkama, preimenuju identifikatore u određene heksadecimalne vrijednosti, ali postoje i drugi načini. Jedan od drugih načina, prikazan na kodovima 2.8 i 2.9, je zamjena imena identifikatora redom slovima abecede. Uz prikazani način, vrijedno je spomenuti i mogućnost zamjene slovima izmiješane abecede.

```
let oneTwoThree = 123;
let hello = "Hello, world! ";
console.log(hello + oneTwoThree);
```

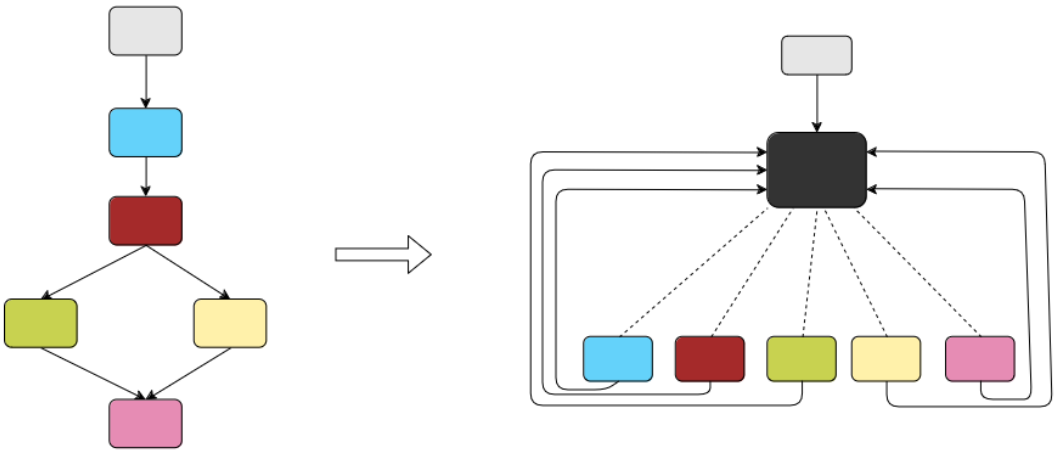
Kod 2.8: Izvorni kod

```
(function () {
var a = 123;
var b = 'Hello, world! ';
console.log(b + a);
}())
```

Kod 2.9: Proveden postupak preimenovanja identifikatora

2.1.3. Kontroliranje toka izravnavanjem

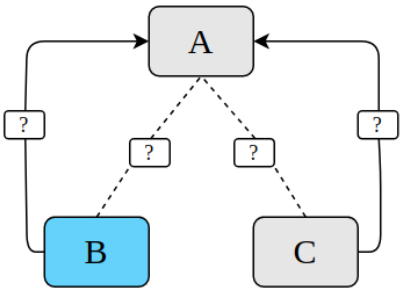
Kontroliranje toka izravnavanjem (engl. *Control Flow Flattening*) ima za cilj obfuscirati tok izvođenja programa. Da bi se to postiglo, transformacija razdvaja sav izvorni kod na blokove, kao što su tijelo funkcije, petlje i uvjetne grane, i stavlja ih sve unutar jedne beskonačne petlje s naredbom *switch* koja upravlja tokom izvođenja. Zbog toga je tok izvođenja programa znatno teže slijediti jer su konstrukti koji olakšavaju čitanje koda sada nestali. Na slici 2.1 je prikazana ilustracija kontroliranja toka izravnavanjem.



Slika 2.1: Kontroliranje toka izravnavanjem

Također, postoji mogućnost obfusciranja *switch* varijable, prikazan na slici 2.2, čime

se puno teže može odrediti koji će idući *switch case* biti izvršen.

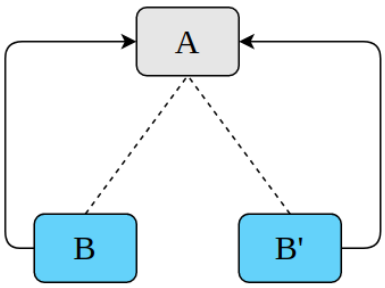


Slika 2.2: Obfusciranje *switch* varijable

2.1.4. Umetanje mrtvog koda

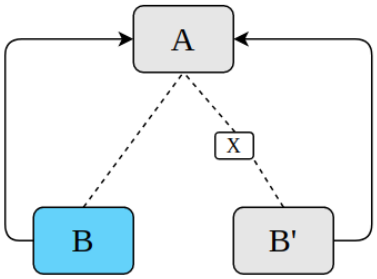
Umetanje mrtvog koda je metoda kojom se konačni kod čini dužim i težim za razumijevanje na način da se umeće kod koji se nikada neće izvršiti ili čiji rezultat nikako ne utječe na krajnji rezultat i tijek izvođenja programa.

Jedan od načina umetanja mrtvog koda prikazan na slici 2.3 je umetanje klonova, semantički ekvivalentnih kopija dijelova koda koji se mogu izvršavati naizmjenično s originalnim dijelovima koda. Korištenjem prethodno spomenute metode kontroliranja toka izravnavanjem može se učiniti da program na slučajno odabran način odabire kada će se izvršiti original, a kada klon.



Slika 2.3: Umetanje klonova

Još jedna od opcija, prikazana na slici 2.4, je umetanje mrtvih klonova, lažnih kopija koje su slične originalnom kodu, ali se nikada ne izvršavaju.



Slika 2.4: Umetanje mrtvih klonova

2.1.5. Ostale metode

Uz prethodno navedene metode, vrijedno bi bilo spomenuti i metodu obfuskacije dosegom (engl. *scope obfuscation*) koja može izgledati kao jednostavnija verzija umetanja mrtvog koda, ali ključna je razlika što uvodi više dosega s dupliciranim identifikatorima. Primjer toga bi bilo da varijabla ima jednu vrijednost kao globalna varijabla, drugu unutar određene funkcije, a treću unutar nekog bloka. Također, već spomenuta minifikacija može biti dio procesa obfuskacije na način da ukloni bjeline iz cijelog koda.

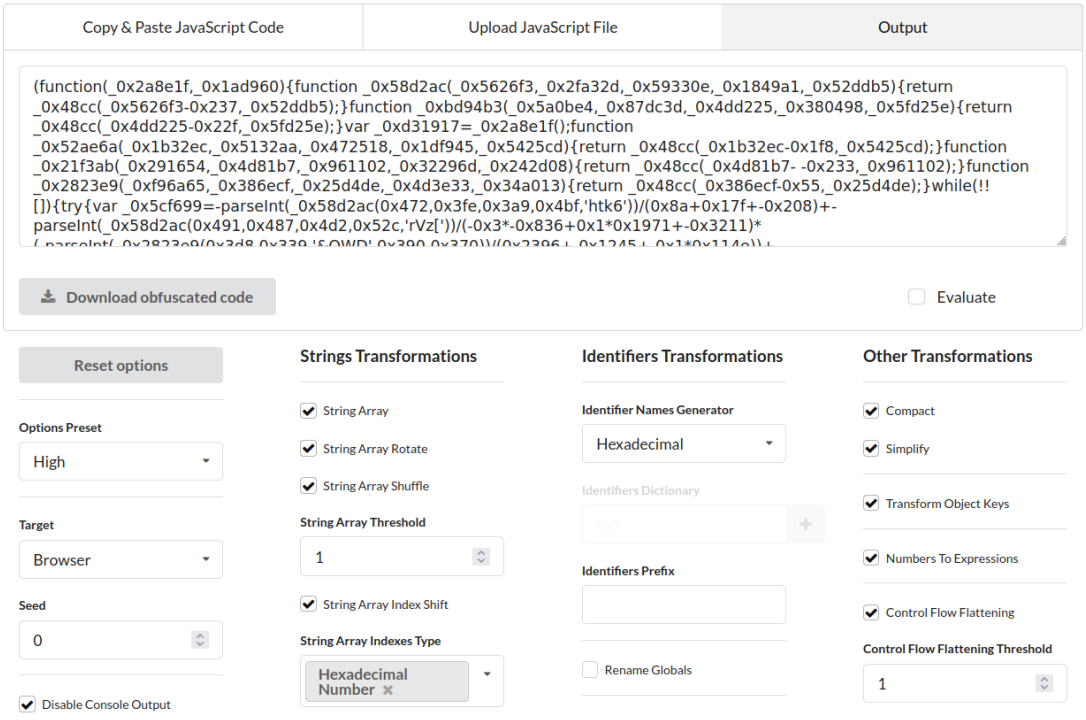
Kao što je već rečeno, broj metoda obfuskacije nije lako mjerljiv s obzirom na to da bilo koji neuobičajen način korištenja određenih JavaScript funkcija ili neuobičajeni način strukturiranja koda može dovesti do svojstava koje ima i obfuscirani kod.

3. Obfuskatori

Obfuskatori su alati koji provode obfuskaciju implementiranjem raznih metoda obfuskacije, od kojih su neke opisane u prethodnom poglavlju. Javno dostupnih besplatnih obfuskatora nema puno te će se sukladno tome u ovom poglavlju dati detaljniji pregled obfuskatora *obfuscator.io* te kratki pregled preostalih obfuskatora.

3.1. Obfuscator.io

Obfuscator.io [2] je obfuskator otvorenog koda koji nudi mnoštvo opcija i načina obfuskacije od kojih su gotovo sve objašnjene u prethodnom poglavlju.



Slika 3.1: Obfuscator.io - izgled sučelja

3.1.1. Opcije i mogućnosti

Načini obfuskacije koje nudi obfuscator.io će biti navedeni redoslijedom kojim su i opisani u trećem poglavlju.

Obfuscator.io nudi najviše opcija za transformaciju i manipulaciju nizova znakova. Jedna od njih je *String Array* opcija čijom aktivacijom obfuskator sve nizove znakova stavlja u posebno polje. Bitno je napomenuti da opcija *String Array*, kao i mnoge druge kasnije spomenute, ima i mogućnost definiranja vjerojatnosti. Konkretno se u ovoj opciji definira vjerojatnost od 0 do 1 da će određeni niz znakova biti uvršten u polje. Opcijama *String Array Rotate* i *String Array Shuffle* u navedeno polje se mogu staviti dodatni elementi te sve elemente polja postaviti na slučajno odabrane indekse. A opcija *String Array Index Shift* omogućava dodatni pomak indeksa elemenata.

Dohvaćanje navedenog polja može se dodatno otežati korištenjem *String Array Calls Transform* opcije te definiranjem broja omatajućih funkcija ili varijabli u opciji *String Array Wrappers Count*. Za kraj, postoje opcije kodiranja nizova znakova s *base64* ili *rc4* koderima te pretvaranja nizova znakova u *unicode escape* sekvence. Isječak obfusciranog koda koristeći prvo *unicode escape* sekvence te potom omatajuće funkcije je prikazan u kodu 3.1.

```
var _0x3c024c = [
   '\x6c\x6f\x67',
   '\x48\x65\x6c\x6c\x6f\x20\x77\x6f\x72\x6c\x64\x21'
];
function _0x5aec06(_0x18db9d, _0x49e13d) {
   return _0x205c(_0x18db9d - -'\x30\x78\x32\x31\x35',
      _0x49e13d);
}
function _0x43de73(_0x23afe2, _0x4ece17) {
   return _0x205c(_0x23afe2 - '\x30\x78\x31\x63\x64',
      _0x4ece17);
}
...
```

Kod 3.1: Isječak obfusciranog koda koristeći omatajuće funkcije i *unicode* sekvence

Dalje, postoje opcije preimenovanja identifikatora. Gotovo svi obfuskatori u početnim postavkama pretvaraju identifikatore u heksadecimalne vrijednosti, a osim te opcije obfuscator.io nudi i opciju preimenovanja identifikatora redom po abecedi ili koristeći

izmiješanu abecedu. Također, postoje opcije obfuskacije globalnih varijabli i funkcija te naziva elemenata objekata, ali treba biti oprezan s korištenjem neke od prethodne dvije navedene opcije jer mogu dovesti do toga da program nije moguće izvršiti.

Također, obfuscator.io implementira i kontroliranje toka izravnavanjem te umetanje mrtvog koda koji dolaze do izražaja jedino kod dužih programskih skripti.

Od ostalih metoda obfuskacije, implementirana je i minifikacija kroz opcije *Simplify* i *Compact* koje uklanjaju sve bjeline iz obfusciranog koda. Za kraj, postoji zanimljiva metoda obfuskacije implementirana u opciji *Numbers To Expressions*, koja nije spomenuta u drugom poglavlju. Navedena opcija, koja je prikazana u kodu 3.2, pretvara numeričke konstante u izraz, koji izračunava na određeni način koristeći heksadekadske brojeve.

```
//ulaz
const foo = 1234;
//izlaz
const foo=-0xd93+-0x10b4+0x41*0x67+0x84e*0x3+-0xff8;
```

Kod 3.2: Primjer korištenja opcije *Numbers To Expressions*

Uz već navedene opcije, u obfuskatoru postoje mogućnosti izbjegavanja obfuskacije koje napadači, ukoliko žele prikriti način funkcioniranja svog malicioznog koda, sigurno ne bi koristili kao što su opcije *ReservedNames* i *ReservedString* koje onemogućavaju obfuskaciju identifikatora odnosno nizova znakova koji im se navedu dok opcija *Exclude* onemogućava obfuskaciju cijelih datoteka.

Osim navedenih metoda obfuskacije, obfuscator.io pruža i određene funkcionalnosti koje se, također u obfusciranom obliku, dodaju ispred ili iza obfusciranog izvornog koda. Funkcionalnost *Disable Console Output* onemogućava pozivanje metoda *console* objekta zamjenjujući ih praznim funkcijama čime se otežava debugiranje. Nadalje, korištenjem funkcionalnosti *Self Defending* onemogućava se mijenjanje obfusciranog koda. Točnije, ukoliko se obfuscirani kod pokuša formatirati ili se probaju promijeniti imena varijabli, aktivira se navedena opcija te se kod više ne može izvršiti. Preostala je funkcionalnost *Debug Protection* koja korištenje funkcionalnosti debugera u pregledniku čini gotovo nemogućom. Prilikom pokušaja otvaranja alata *Developer Tools*, zamrzne se preglednik.

Prethodno navedene funkcionalnosti bi napadači koji prikrivaju svoj zloćudni kod gotovo sigurno koristili. No, kao što je već spomenuto obfuskacija se može koristiti i u

drugačije, korisne, svrhe te samim time alat obfuscator.io pruža podršku za to. Prva opcija su mape izvornog koda (eng. *source maps*) koje omogućuju transformaciju iz obfusciranog koda u originalni kod te njegovo lakše debugiranje u pregledniku. Druga opcija je zaključavanje domene (eng. *domain lock*) čime se obfuscirani kod mora pokretati isključivo na propisanim domenama i poddomenama. Ovime se otežava kopiranje te krađa intelektualnog vlasništva. Ipak, i kod ove opcije postoji moguća zlonamjerna upotreba u podopciji *Domain Lock Redirect URL* kojom se preglednik preusmjerava na propisani URL ako se kod ne pokreće na specificiranoj domeni.

3.1.2. Pokretanje i upotreba

Obfuscator.io se na vrlo jednostavan način može koristiti putem preglednika na istoimenom URL-u gdje su sve prethodno opisane opcije izlistane ispod forme za unos koda. Također, postoji opcija instalacije putem alata Yarn i NPM. Ako je alat instaliran, može se uključiti u JavaScript programski kod koristeći *require* ključnu riječ te definirane metode *obfuscate*, *obfuscateMultiple* i sl. No, način na koji je alat korišten za generiranje obfusciranih datoteka za potrebe ovoga rada je koristeći komandnu liniju. Kao što je prikazano u kodu 3.3, poslije imena datoteke s izvornim kodom, koriste se zastavice poput zastavice *output* koja određuje u koju datoteku će se obfuscirani kod pohraniti.

```
//Obfusciranje jedne datoteke
javascript-obfuscator primjer.js --output
   obfusciraniPrimjer.js --compact true --dead-code-injection
   false

//Obfusciranje cijelog direktorija
javascript-obfuscator ./dir --output ./dir/obfuscirani
   --compact true --dead-code-injection false
```

Kod 3.3: Primjer korištenja alata obfuscator.io u komandnoj liniji

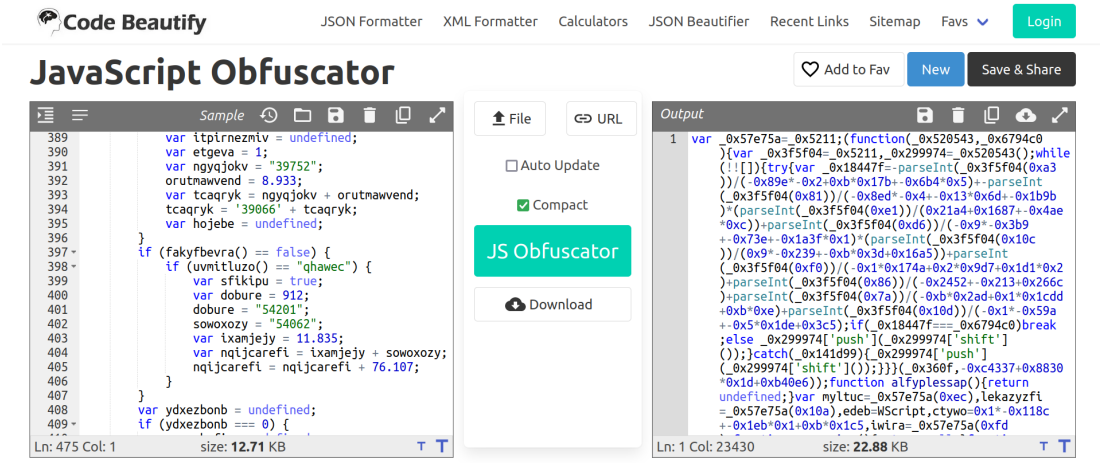
Kako ne bi bilo potrebno u komandnu liniju upisivati svaku opciju kojoj se žele promijeniti početne postavke alat nudi i opciju definiranja jačine obfuskacije. Točnije, alat nudi mogućnosti *Low*, *Medium* i *High* koje aktiviraju manju, srednju ili veću količinu opcija. Koje konkretno metode obfuskacije su aktivirane pod kojom opcijom se može provjeriti u službenoj dokumentaciji [7]. Za potrebe ovog rada uvijek je korištena mogućnost *High*.

3.2. Ostali obfuskatori

U ovom potpoglavlju će se dati kratki pregled javno dostupnih besplatnih obfuskatora. Treba napomenuti da postoji još obfuskatora dostupnih na drugim web stranicama, ali svi koji su pronađeni daju praktički jednake izlaze i nude iste opcije kao jedan od obfuskatora navedenih u ovom radu.

3.2.1. Code Beautify

Code Beautify [8] je obfuskator koji, kao što je prikazano na slici 3.2, nema puno opcija kojima korisnik može utjecati na način obfuskacije. Ističe se jedino opcija *Compact* koja uklanja bjeline iz obfusciranog koda. Uspoređujući izlaz s izlazom alata obfuscator.io može se prepoznati niz već viđenih metoda obfuskacije kao što su heksadecimalna imena identifikatora te omatajuće funkcije i varijable. Mnoštvo poziva funkcije *parseInt* također je već viđeno aktivacijom obfuscator.io opcije *String Array Rotate* kojom se otežava razumijevanje načina dohvaćanja indeksa niza znakova u polju.

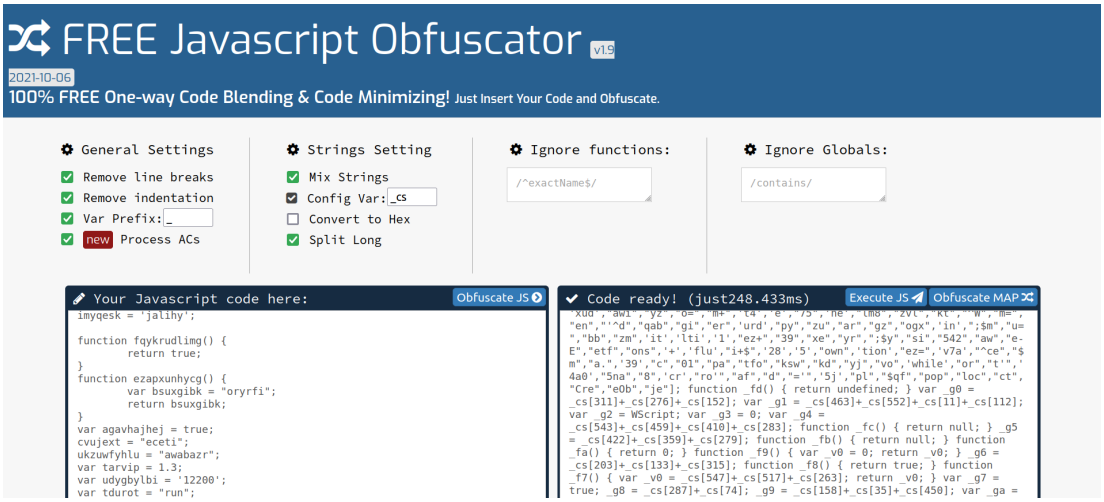


Slika 3.2: Code Beautify - izgled sučelja

3.2.2. FREE JavaScript Obfuscator

FREE JavaScript Obfuscator [9] nije pretjerano moćan obfuskator ako gledamo količinu metoda obfuskacije koje nudi i koje su prikazane na slici 3.3. Sve dostupne metode su prikazane korisniku i može ih modificirati. Opcije koje ovaj obfuskator nudi su pretvaranje nizova znakova u heksadekadske brojeve i spremanje istih u polje te uklanjanje bjelina. Kao što se može zaključiti navedene mogućnosti su lako ostva-

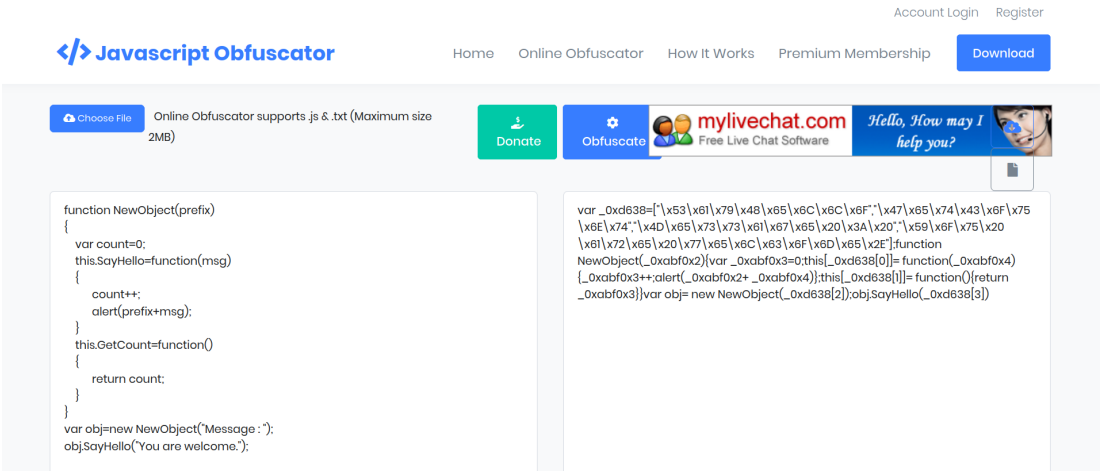
rive i koristeći obfuscator.io, konkretno, opcijama *Unicode Escape Sequence*, *Compact* i *String Array*.



Slika 3.3: FREE JavaScript Obfuscator - izgled sučelja

3.2.3. Javascript Obfuscator

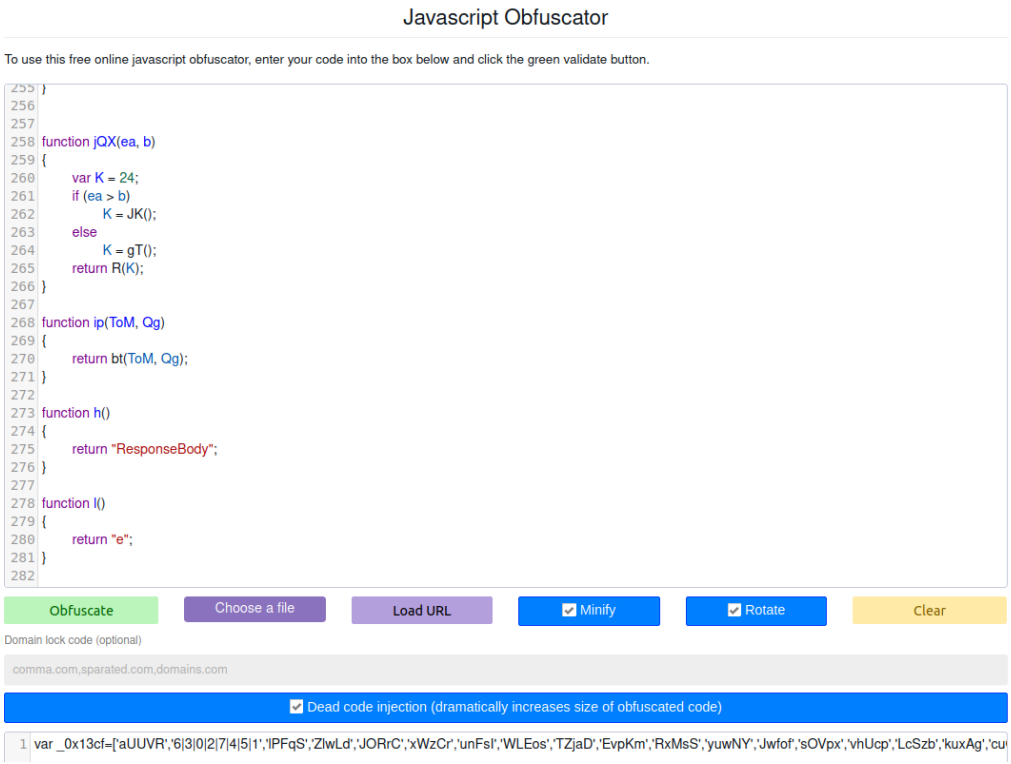
Javascript Obfuscator [10] je primjer obfuskatora koji nudi besplatnu i plaćenu verziju svoje usluge. Besplatna verzija je vrlo slična obfuskatoru FREE JavaScript Obfuscator jer također od opcija nudi samo uklanjanje bjelina, kodiranje nizova znakova te pohranjivanje istih u polje što je opet lako moguće ostvariti i korištenjem obfuscator.io. Plaćena verzija nudi određene naprednije opcije poput umetanja mrtvog koda, ali način na koje funkcioniraju naprednije opcije nije testiran. Izgled sučelja ovog obfuskatora je prikazan na slici 3.4.



Slika 3.4: Javascript Obfuscator - izgled sučelja

3.2.4. HTML Strip

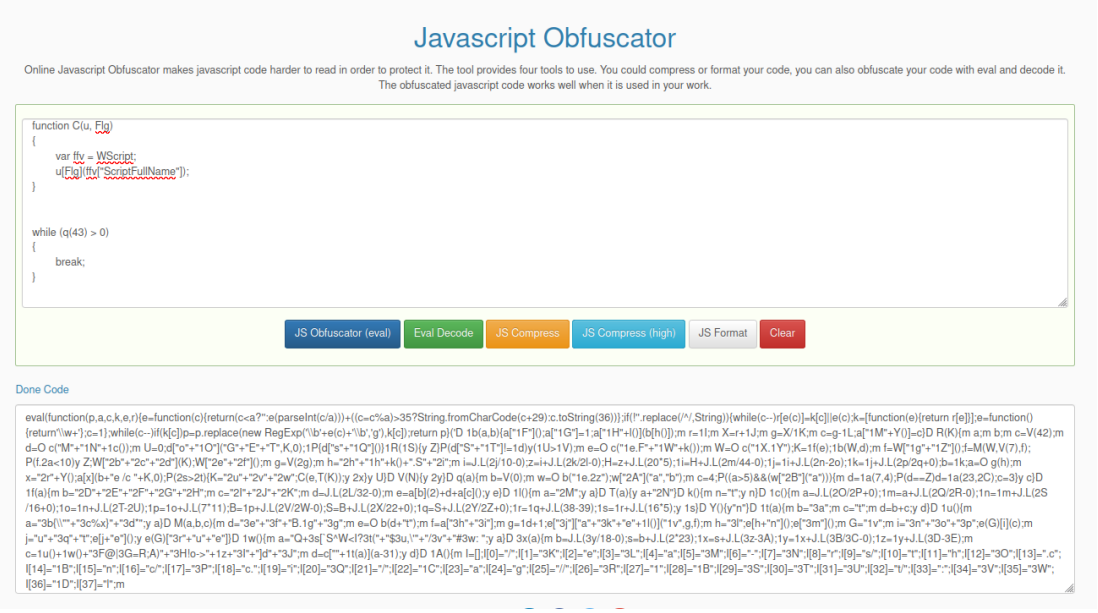
HTML Strip [11] je, kao i prethodno spomenuti Javascript Obfuscator, vrlo siromašan obfuskator po pitanju broja opcija kao što je prikazano na slici 3.5. Nudi opcije uklanjanja bjelina i umetanja mrtvog koda. Uz njih, također nudi opciju *Rotate* koja postavlja elemente na slučajno odabrane indekse u polje, što je ekvivalent metodi *String Array Rotate* u alatu obfuscator.io.



Slika 3.5: HTML Strip - izgled sučelja

3.2.5. Beautify Converter

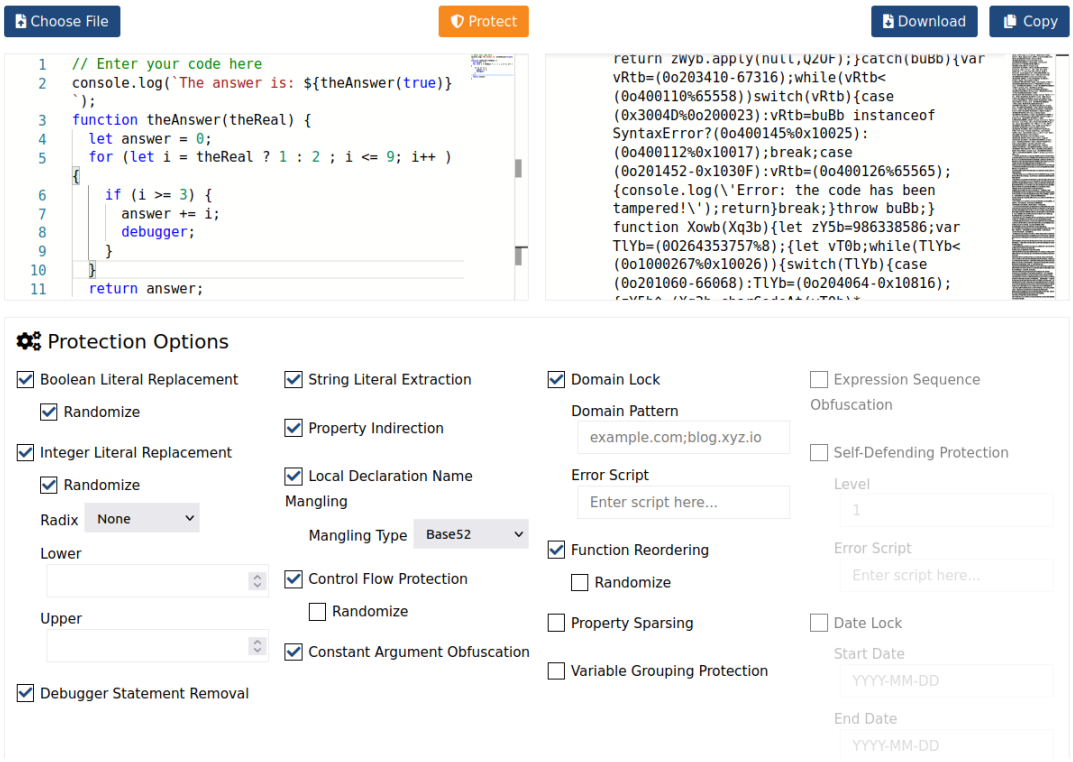
Kao što je već spomenuto, praktički isti obfuskatori postoje dostupni na više web stranica. Tako da obfuskatori Beautify Converter [12], Online-Toolz [13], SEOSniffer.com [14], Beautify Tools [15] i wTools.io [16] funkcioniraju na isti način koristeći razdvajanje i konkatenaciju nizova znakova te regularne izraze što se konačno predaje kao argument JavaScript funkciji *eval* koja izvršava navedeni niz znakova kao naredbu. Shodno tome, obfuscator.io, ne može generirati obfuscirani kod koji funkcionira na navedenom principu koristeći funkciju *eval*, te navedeno pretvaranje programskog koda u niz znakova, pa se može zaključiti da navedeni obfuskatori daju različit izlazni kod od obfuscator.io.



Slika 3.6: Beautify Converter - izgled sučelja

3.2.6. PreEmptive JS Defender

PreEmptive JS Defender [17] kao i već opisani Javascript Obfuscator nudi besplatnu i plaćenu verziju svog alata. Ipak, treba reći da i besplatna verzija nudi mnoštvo mogućnosti obfuskacije koje su prikazane na slici 3.7. Prilikom obfuskacije, nizove znakova pohranjuje u varijable te navedene varijable inicijalizira iz kodiranih nizova znakova. Također, ima opciju pretvaranja vrijednosti cjelobrojnih i *boolean* varijabli u određene izraze koji sadržavaju heksadecimalne ili binarne brojeve te se prilikom izvršavanja programa izračunavaju. Uz sve to, pruža i nekoliko vrsta preimenovanja identifikatora. Između ostalih, pretvaranje u heksadecimalne vrijednosti, kodiranje koristeći *Base62* te pretvaranje u glagoljičko pismo. Testirajući navedeni obfuskator, lako je vidljivo da način manipulacije niza znakova kao ni tolike mogućnosti preimenovanja identifikatora nisu moguće koristeći obfuscator.io te uspoređujući duljinu obfusciranog koda s obzirom na izvorni kod može se zaključiti da navedeni obfuskator i obfuscator.io daju različite izlaze.



Slika 3.7: JS Defender - izgled sučelja

4. Implementacija alata za usporedbu izlaza obfuskatora

Kao što je rečeno u zadatku, potrebno je utvrditi jesu li opcije i izlazi koje nude ostali obfuskatori dovoljno različiti od obfuskatora obfuscator.io. Dovoljnom dobrim razumijevanjem načina funkcioniranja opcija obfuskatora obfuscator.io te opcija svakog idućeg obfuskatora s kojim se uspoređuje moguće je utvrditi sličnosti u izlazima obfuskatora. Spomenuti način ručne usporedbe izlaza je donekle primijenjen u poglavlju 3.2. prilikom opisa obfuskatora. No, kako to zahtijeva dosta vremena, potrebno je navedeni postupak automatizirati i probati kreirati određeni alat koji bi svakom pojavom novog obfuskatora mogao dati odgovor na pitanje koliko su izlazi navedenog obfuskatora slični izlazima alata obfuscator.io ili nekog drugog obfuskatora.

Utvrđivanje sličnosti između izlaza obfuskatora tj. obfusciranih kodova nije lako s obzirom na to da se radi o poprilično nerazumljivom kodu. Također, sami obfuskatori funkcioniraju na pseudoslučajan način. Točnije, za jednak ulaz i jednako definirane opcije, ovisno o pokretanju, daju različite izlaze poput različitog redoslijeda zapisa ili različitih imena identifikatora. Primjer pokretanja alata obfuscator.io za isti jednostavan ulaz i samo nekoliko aktiviranih opcija te ispis različitih izlaza je prikazan u kodovima 4.1, 4.2 i 4.3.

```
console.log("Hello world!");
```

Kod 4.1: Izvorni kod

```
var _0x5ae277=_0x3130;function
   _0x3130(_0x24b834,_0x313028){var _0x5b38d1=_0x24b8();
_0x3130=function(_0x4cf2b8,_0xf85d9d){_0x4cf2b8=_0x4cf2b8-0x0;
var _0x29b9d8=_0x5b38d1[_0x4cf2b8];return _0x29b9d8;}
;return _0x3130(_0x24b834,_0x313028);}
...
```

Kod 4.2: Izlaz prilikom prvog pokretanja

```
function _0x4b18(_0x10d313,_0x4b1807)
{var _0x2c577=_0x10d3();_0x4b18=
function(_0x278117,_0x2793ce){_0x278117=_0x278117-0x0;
var _0x5d79f9=_0x2c577[_0x278117];return _0x5d79f9;};return
   _0x4b18(_0x10d313,_0x4b1807);}var _0xa96cc4=_0x4b18;
...
```

Kod 4.3: Izlaz prilikom drugog pokretanja

Samim time, bilo kakve metode usporedbe "znak po znak" ili usporedbe redoslijeda nisu moguće. Također, ne dolazi u obzir brojanje ukupne količine znakova ili količine identifikatora jer dva obfuskatora mogu funkcionirati praktički na isti način, ali koristiti više ili manje omatajućih funkcija ili varijabli te time zavarati navedeni način usporedbe.

Analizirajući dovoljno velik broj izlaza obfuskatora, može se zamijetiti da su ključne riječi, ugrađeni objekti i ostali dijelovi JavaScript programskog jezika jedini dijelovi obfusciranog koda koje svaki obfuskator ostavlja u izvornom obliku. No, zbog već spomenutog primjera problema omatajućih funkcija ili varijabli, brojanje navedenih dijelova JavaScript programskog jezika neće dovesti do ispravnog rješenja. Konačno, metoda koja je korištena za implementaciju alata je brojanje različitih ključnih riječi i ostalih dijelova JavaScript programskog jezika u izlazu obfuskatora čime se može utvrditi koristi li određeni obfuskator drugačije mogućnosti JavaScript programskog jezika od obfuskatora s kojim se uspoređuje.

4.1. Korištene tehnologije i implementacija

Jednostavni programski alat za usporedbu obfuskatora implementiran je u programskom jeziku Go, verzije 1.20.1, tako da je za ispravno pokretanje ovog alata potrebno imati instaliran programski jezik Go. Go je odabran iz razloga što pruža mogućnost izvođenja u raznim okruženjima, jednostavan je za korištenje te pruža veliku brzinu izvođenja [18].

Programska skripta se pokreće naredbom prikazanom u kodu 4.4 te joj se moraju predati dva argumenta. Argumenti mogu biti dvije datoteke ili dva direktorija. Jasno,

efikasniji i točniji način korištenja alata je usporedbom direktorija, gdje svaki direktorij sadrži izlaze jednog obfuskatora u kojem se nalazi veći broj datoteka s obfusciranim kodovima. Također, bitno je napomenuti da prvi direktorij ili prva datoteka uvijek moraju biti od obfuskatora za kojeg se provjerava može li rekreirati i nadomjestiti sve izlaze drugog obfuskatora. U primjeru ovog rada prvi obfuskator će uvijek biti obfuscator.io jer se želi provjeriti može li on svojim opcijama nadomjestiti sve druge obfuskatore koji se navedu kao drugi argument.

```
go run main.go file1 file2
// ili
go run main.go dir1 dir2
```

Kod 4.4: Pokretanje alata u komandnoj liniji

Kao što je objašnjeno na početku poglavlja, potrebno je uspoređivati obfuskatore na temelju dijelova JavaScript programskog jezika, a kako bi to bilo moguće potrebno je sve ključne riječi [19], ugrađene objekte i ostale dijelove programskog jezika [20] pohraniti u datoteku. Za potrebe ovog rada, svi elementi su pohranjeni u csv datoteku imena jsWords.

Predajom direktorija kao argumenata slijedi iteriranje po svakom od direktorija te usporedba datoteka po principu prva s prvom, druga s drugom i sl. Time korisnik koji generira navedene direktorije treba paziti o redoslijedu datoteka. Točnije, usporedba datoteka mora biti između dvaju obfusciranih kodova dobivenih od dvaju različitih obfuskatora iz istog izvornog koda. Najefikasniji način za to bi bio da datoteke koje su obfuscirale isti izvorni kod budu jednako nazvane, samo u različitim direktorijima. Prvo se iz datoteka izvlače sve riječi jednostavnim regularnim izrazom \w+ koji pronalazi sva podudaranja jednog ili više znakova malog ili velikog slova, broja ili donje crte [21].

Svaka se pojedina riječ potom provjerava je li dio programskog jezika JavaScript, točnije nalazi li se u datoteci jsWords.csv te ako se nalazi, dodaje se u strukturu podataka u kojoj su sadržane svi dijelovi rječnika JavaScripta za tu datoteku. Nakon što su kreirane strukture podataka za oba obfuskatora, točnije, mape gdje su ključevi navedene riječi, slijedi usporedba. Iterira se po strukturi podataka drugog obfuskatora te se svaka riječ provjerava je li sadržana u strukturi podataka prvog obfuskatora. Ako jest, onda se brojač povećava za jedan. Po završetku iteracije ukupan broj riječi strukture podataka drugog obfuskatora se dijeli s brojačem i zatim se množi sa 100 tj. dobiva se

postotak riječi iz druge datoteke koje se mogu pronaći u prvoj datoteci. Zatim se navedeni proces ponavlja za sve datoteke u direktoriju i izračunava se srednja vrijednost. Ako navedena srednja vrijednost postotka prelazi 80%, navedena dva obfuskatora su dovoljno slična i moguće je prvim obfuskatorom nadomjestiti drugi, a ako je ta brojka manja, dva obfuskatora funkcioniraju na različite načine. Dolazak do navedenog praga od 80% i detaljnije o alatu je opisano u potpoglavlju 4.2.

4.2. Realizacija alata na početnom skupu podataka

Kako bi alat funkcionirao za bilo koje obfuskatore, potrebno je utvrditi kakve rezultate daje za obfuskatore za koje se već zna da su dovoljno slični ili različiti. Pošto je u četvrtom poglavlju utvrđeno da je obfuskator Code Beautify dovoljno sličan obfuskatoru obfuscator.io, a da mu je PreEmptive JS Defender dovoljno različit, navedena tri obfuskatora će se koristiti prilikom početnog testiranja i generiranja početnog skupa podataka. Za početni skup podataka, ali i bilo koje iduće korištenje alata, potrebno je aktivirati sve opcije na obfuskatoru iz kojeg se dobiva obfuscirani kod. Razlog tomu je što nijedna opcija obfuskatora ne negira drugu. Na primjer, korištenjem opcije umetanja mrtvog koda stvaramo uniju, a ne presjek, s dijelom koda koji se odnosi na kontroliranje toka izravnavanjem. A ovime se smanjuje broj potrebnih testnih primjera jer će se odmah moći vidjeti koje ključne riječi i ostale dijelove programskog jezika JavaScript koristi pojedini obfuskator u svojim metodama obfuskacije.

Kako se za generiranje obfusciranog koda ne bi koristile relativno kratke programske skripte, korišten je skup javno dostupnih malicioznih JavaScript kodova [22] koji su dovoljno dugački da se vide sve mogućnosti obfuskatora. Za početno generiranje korišteno je 15 obfusciranih skripti iz tri prethodno spomenuta obfuskatora. Za navedeno testiranje nije potreban velik broj kodova za usporedbu s obzirom na to da su izvorni kodovi dovoljno dugački i dovoljno različiti da obfuskatori prikažu sve dijelove JavaScript koda koje koriste, kojih u svakom slučaju ni nema sveukupno prevelik broj. Pokretanjem alata i predajom obfuscator.io direktorija kao prvog argumenta i Code Beautify direktorija kao drugog argumenta dobivena je sličnost od 89.65%. A usporedbom obfuscator.io direktorija i PreEmptive JS Defender direktorija dobivena je sličnost od 59.52%. Naposljetku, definiran je prag od 80%. Ako sličnost prelazi navedeni prag, obfuskatori su dovoljno slični, a ako ne, različiti su. Razlog tolerancije od 20% je što se dijelovi JavaScript koda mogu pojaviti u izvornom kodu kao dijelovi imena identifikatora ili unutar nizova znakova što alat može krivo pribrojiti. Također,

otvorena je i mogućnost da obfuskator uistinu koristi određenu funkcionalnost programskog jezika prilikom obfuskacije koja se ne koristi u prvom obfuskatoru, ali se to, radi velike količine podataka i velikog postotka sličnosti, zanemaruje jer alat nema mogućnost utvrditi koliko puta se navedeni dio u obfusciranom kodu koristi i kolika mu je uloga.

4.3. Testiranje alata

Nakon što je alat završen, preostalo je testiranje alata na svim obfuskatorima te provjera daje li točne rezultate. Kreirani su direktoriji za svaki obfuskator te u svakom direktoriju 30 datoteka s obfusciranim kodovima malicioznih skripti. Kao što je već rečeno, mogućnosti JavaScript programskog jezika nema puno, a s obzirom na duljinu i različitost izvornih kodova skup ne treba biti veći, ali bitno je i da nije puno manji s obzirom na to da neki obfuskatori skripte velike duljine ne uspijevaju obfuscirati ili javljaju greške prilikom obfuskacije tako da može doći do svakako manjeg skupa. A treba uzeti i u obzir da i izvorni kodovi sami po sebi mogu biti sintaksno neispravni za što neki obfuskatori također javljaju greške. U bilo kojem od prethodno opisanih slučajeva datoteka je ostavljena prazna te ju alat preskače i ne uračunava u postupak usporedbe. Kao što je vidljivo na tablici 4.1, svi obfuskatori, označeni po potpoglavljima u kojima su opisani, su uspoređeni s obfuscator.io.

	Code Beautify	FREE JavaScript Obfuscator	Javascript Obfuscator	HTML Strip	Beautify Converter	PreEmptive JS Defender
Sličnost s obfuscator.io	89.52%	87.14%	90.27%	89.14%	75.30%	60.75%

Tablica 4.1: Postotak sličnosti obfuskatora s alatom obfuscator.io

Vidljivo je da alat za obfuskatore koji su klasificirani u četvrtom poglavlju kao dovoljno slični obfuscator.io daje sličnost preko 80%, a vrijednost manju od te za obfuskatore koji su tada klasificirani kao nedovoljno slični. Iz ovoga proizlazi da alat daje u potpunosti ispravne rezultate za usporedbu obfuskatora i moguće mu je do jedne mjere

vjerovati i koristiti ga za obfuskatore koji se naknadno pojave na Internetu ili za one koji nisu obuhvaćeni ovim radom.

5. Zaključak

Obfusciranje JavaScript koda široko je područje koje će se zasigurno s vremenom sve više razvijati s obzirom na to da se obfuskacija ne koristi isključivo u maliciozne svrhe. Stoga razvojem ovog područja i napadači će tražiti nove načine zaobilaženja mehanizama detekcije i deobfuskacije pa je rješavanju problema te detekciji malicioznog sadržaja potrebno pristupiti odgovorno, pri čemu je jedna od opcija i detekcija obfusciranog koda strojnim učenjem.

Iako obfuskatora JavaScript koda nema puno, za proces generiranja podataka za strojno učenje ipak je potrebno uz obfuscator.io uključiti još dodatnih obfuskatora koje obfuscator.io ne može nadomjestiti. Što se tiče trenutno javno dostupnih obfuskatora analiza sličnosti s alatom obfuscator.io je u ovom radu provedena, ali ako se pojave novi obfuskatori ili obfuskatori koji nisu zahvaćeni ovim radom, mogu se bez dublje analize, uz pomoć alata za usporedbu izlaza obfuskatora opisanog u četvrtom poglavlju, usporediti s obfuscator.io ili nekim drugim obfuskatorom te utvrditi je li navedeni obfuskator dovoljno različit od onog s kojim se uspoređuje ili točnije, može li jedan obfuskator svojim opcijama i izlazom nadomjestiti drugi obfuskator.

6. LITERATURA

[1] Usage statistics of JavaScript as client-side programming language on websites, 2023. URL https://w3techs.com/technologies/details/cp-javascript. Pristupio: 21.5.2023.

[2] Obfuscator.io, 2023. URL https://github.com/javascript-obfuscator/javascript-obfuscator. Pristupio: 21.5.2023.

[3] JavaScript Obfuscation: The Definitive Guide, 2022. URL https://blog.jscrambler.com/javascript-obfuscation-the-definitive-guide. Pristupio: 21.5.2023.

[4] Christian Collberg, Clark David Thomborson, Douglas Low. A Taxonomy of Obfuscating Transformations, 1997. URL https://www.researchgate.net/publication/37987523_A_Taxonomy_of_Obfuscating_Transformations. Pristupio: 21.5.2023.

[5] Maurice H. Halstead. *Elements of Software Science*. Elsevier North-Holland, Inc., 1977. ISBN 0444002057.

[6] Beyond Obfuscation: JavaScript Protection and In-Depth Security, 2020. URL https://blog.jscrambler.com/beyond-obfuscation-javascript-protection-and-in-depth-security/. Pristupio: 21.5.2023.

[7] Obfuscator.io - JavaScript obfuscator options, 2023. URL https://github.com/javascript-obfuscator/javascript-obfuscator#javascript-obfuscator-options. Pristupio: 24.5.2023.

[8] Code Beautify, 2023. URL https://codebeautify.org/javascript-obfuscator. Pristupio: 24.5.2023.

[9] FREE Javascript obfuscator, 2023. URL http://www.freejsobfuscator.com/. Pristupio: 24.5.2023.

[10] Javascript obfuscator, 2023. URL https://www.javascriptobfuscator.com/Javascript-Obfuscator.aspx. Pristupio: 24.5.2023.

[11] HTML Strip - Javascript obfuscator, 2023. URL https://www.htmlstrip.com/javascript-obfuscator. Pristupio: 24.5.2023.

[12] Beautify Converter - Javascript obfuscator, 2023. URL https://www.beautifyconverter.com/javascript-obfuscator.php. Pristupio: 24.5.2023.

[13] Online Toolz - Javascript obfuscator, 2023. URL https://www.online-toolz.com/tools/javascript-obfuscator.php. Pristupio: 24.5.2023.

[14] SEO Sniffer - JavaScript obfuscator, 2023. URL https://www.seosniffer.com/javascript-obfuscator. Pristupio: 24.5.2023.

[15] Beautify Tools - Javascript obfuscator, 2023. URL https://beautifytools.com/javascript-obfuscator.php. Pristupio: 24.5.2023.

[16] wTools.io - JavaScript obfuscator, 2023. URL https://wtools.io/javascript-obfuscator. Pristupio: 24.5.2023.

[17] PreEmptive JS Defender, 2023. URL https://www.preemptive.com/products/jsdefender/online-javascript-obfuscator-demo/. Pristupio: 24.5.2023.

[18] John Biggs, Ben Popper. What's so great about Go, 2020. URL https://stackoverflow.blog/2020/11/02/go-golang-learn-fast-programming-languages/. Pristupio: 26.5.2023.

[19] W3Schools - JavaScript keywords, 2023. URL https://www.w3schools.in/javascript/keywords/. Pristupio: 24.5.2023.

[20] MDN Web Docs - JavaScript reference, 2023. URL https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference#additional_reference_pages. Pristupio: 24.5.2023.

[21] Jan Bodnar. ZetCode - Go regular expressions, 2023. URL https://zetcode.com/golang/regex/. Pristupio: 24.5.2023.

[22] Hynek Petrak. Javascript Malware Collection, 2023. URL https://github.com/HynekPetrak/javascript-malware-collection. Pristupio: 24.5.2023.

Analiza i mjerenje izlaza obfuskatora JavaScript koda

Sažetak

Velika upotreba programskog jezika JavaScript u web stranicama postaje meta koju napadači žele iskoristiti. Jedna od metoda koju koriste je obfuskacija, tehnika otežavanja čitanja izvornog koda. Detekcijom obfuskacije moguće je otkriti potencijalno zlоćudni kod, a jedan od pristupa detekciji je strojno učenje.

U ovom radu je opisana obfuskacija i metode kojom se postiže te je potom opisan alat obfuscator.io i ukratko uspoređen sa svim ostalim javno dostupnim besplatnim obfuskatorima na temelju opcija i izlaza. Automatizacija usporedbe se može provesti alatom izrađenim za potrebe ovog rada kojim se došlo do zaključka da za potrebe generiranja skupa podataka za strojno učenje nije dovoljno koristiti jedan obfuskator, nego više njih s obzirom na to da funkcioniraju na različite načine i koriste različite mogućnosti JavaScript programskog jezika.

Ključne riječi: obfuskacija, JavaScript, kibernetička sigurnost, obfuskatori

Analysis and Measurement of JavaScript Code Obfuscators' Output

Abstract

The large use of the JavaScript programming language in web pages is becoming a target that attackers want to exploit. One of the methods they use is obfuscation, a technique to make the source code harder to read. Detecting obfuscation can lead to revealing potentially malicious code, and one approach to detection is machine learning.

This paper describes obfuscation and the methods by which it is achieved. Later on it describes the obfuscator.io tool and briefly compares it, based on options and output, to all other publicly available free obfuscators. The automation of the comparison can be carried out with a tool created for the purposes of this work by help of which it was concluded that for the purposes of generating a dataset for machine learning, it is not enough to use one obfuscator, but several, because they function in different ways and use different capabilities of the JavaScript programming language.

Keywords: obfuscation, JavaScript, cybersecurity, obfuscators