Sadržaj

1 Uvod

Web je danas sveprisutna tehnologija na kojoj se temelje mnoge aktivnosti i zahvaljujući Webu informacije se prenose brže nego ikada te također imaju daleko širi doseg nego što su to imale druge metode u povijesti.

Uz sve načine na koje Web olakšava prijenos informacija i povećava produktivnost svakodnevnih aktivnosti u modernom svijetu, zbog zlouporabe različitih zlonamjernih dionika donosi i brojne opasnosti te rizike za sve svoje korisnike. Napadači koriste Web kako bi u njega ubacili zlonamjerni HTML i JavaScript kod uz pomoć kojeg pokušavaju preuzeti nadzor nad uređajem korisnika, neovlašteno pribaviti privatne i osjetljive korisničke podatke te iskoristiti kompromitirane uređaje za daljnje napade ili širenje zlonamjernog programskog koda, a sve u svrhu direktnog ili indirektnog ostvarenja zarade. Pravovremena detekcija takvih stranica izuzetno je bitna kako bi se zaštitili svi oni koji pristupaju Webu uz očuvanje svih prednosti koje donosi Web tehnologija.

Detekcija ubačenih JavaScript i HTML fragmenata nije trivijalna te je mnoštvo predloženih metoda koje pokušavaju riješiti taj problem, uključujući metode temeljene na statičkim atributima stranice, metode temeljene na dinamičkoj analizi, metode koje koriste strojno učenje, metode koje koriste heuristiku i metode koje koriste druge proizvoljne tehnologije primjenjive na detekciji zlonamjernog koda. Cilj svakog od predloženih algoritama je pronaći maksimalni broj zlonamjernih stranica uz minimalni broj lažno pozitivnih detekcija u minimalnom vremenu izvođenja i uz maksimalnu učinkovitost. U sklopu ovog diplomskog rada objasnit će se i implementirati odabrani načini detekcije ubačenih zlonamjernih HTML i JavaScript fragmenata u Web stranice. Metode kojima će se diplomski rad baviti koristit će isključivo HTML Web stranice, odnosno neće imati mogućnost pristupa izvornom kodu Web aplikacije. Razlog je što takve metode u praktičnoj primjeni moraju analizirati javno dostupne Web stranice do kojih se može doći pristupom Internetu, ne imajući pritom pristup samom poslužitelju niti kodu koji se na njemu nalazi i koji generira sadržaj HTML stranice.

Rad je strukturiran na sljedeći način. U drugom poglavlju uvodi se pojam i problematika zlonamjernih Web stranica, objašnjavaju se napadi klijentske strane zasnovani na Webu te se daje kratki pregled metoda detekcije. U trećem poglavlju predstavlja se algoritam detekcije zasnovane na asocijativnoj klasifikaciji te se daje pregled i opisuje implementacija algoritma statističke heuristike, algoritma zasnovanog na mehanizmu bodovanja i algoritma zasnovanog na Yara pravilima, uz prijedloge potencijalnih optimizacija izvornih implementacija. U četvrtom poglavlju opisuju se metodologija testiranja i testni skupovi podataka za algoritam statičke heuristike, algoritam zasnovan na bodovanju i algoritam zasnovan na Yara pravilima. Peto poglavlje daje pregled implementacije i rezultata optimizacija algoritma statičke heuristike, algoritma zasnovanog na bodovanju te algoritma zasnovanog na Yara pravilima uz pregled općenitih dodatnih optimizacija mogućih za opisane algoritme. Zaključak sažima rezultate dobivene implementacijom i testiranjem odabranih algoritama te daje pregled nad usporedbom njihove učinkovitosti i primjenjivosti.

2 Zlonamjerne Web stranice

2.1 Problematika zlonamjernih Web stranica

2.1.1 Pojam zlonamjernih Web stranica i njihove detekcije

Pojam zlonamjerne Web stranice [1] odnosi se na internetsku stranicu koja sadrži zlonamjerni sadržaj koji može iskoristiti ranjivosti klijentskog računalnog sustava, a napad koji dolazi do klijentskog preglednika kad je zlonamjerna Web stranica zatražena naziva se napad s klijentske strane zasnovan na webu (engl. *web-based client-side attack*).

Napadi koji su dostavljeni kao dio Web stranice mogu iskoristiti klijentske ranjivosti kao što su mane u implementaciji funkcionalnosti preglednika, zastarjelih dodataka kao što su ActiveX ili Flash ili operacijskog sustava. Rezultat napada je često instalacija zlonamjerne aplikacije na klijentsko računalo bez pristanka korisnika te neovlašteno pribavljanje korisničkih podataka, a napadač često preuzima i kontrolu nad računalom korisnika te ga može iskoristiti za pokretanje slanja masovne neželjene elektroničke pošte ili raspodijeljenih napada uskraćivanja usluge (engl. *Distributed Denial of Service*, skraćeno DDOS).

Detekcija i označavanje zlonamjernih Web stranica postala je vrlo bitna problematika, a jedan od pristupa je stvaranje virtualnih okolina poput *honeypot* klijenata gdje su stranice učitane i izvršene te se prati potencijalne zlonamjerne aktivnosti ili ponašanja, pri čemu virtualno okruženje sprječava utjecaj zlonamjernih aplikacija na računalo domaćina [2]. Metoda interaktivnih *honeypot* klijenata vrlo je učinkovita u otkrivanju nepoznatih napada, ali također zahtjeva značajne resurse i provjera pojedine stranice traje relativno dugo. Iz tog razloga, uvode se hibridni sustavi koji kombiniraju manje zahtjevne metode zasnovane na statičkim atributima HTML stranice s *honeypot* klijentom kojem se prosljeđuju stranice detektirane kao zlonamjerne na provjeru radi eliminacije lažno pozitivnih detekcija.

Takvi sustavi i metode nisu idealni te postoje brojni izazovi vezani uz njih. Mehanizam implementiran u tu svrhu mora trošiti malu količinu resursa. Zato se radi o mehanizmu koji koristi značajke izdvojene iz statičke internetske stranice umjesto značajki otkrivenih tijekom skupog dinamičkog izvođenja. Potrebno je izdvojiti upravo one najprimjerenije značajke koje mogu ukazivati na potencijalno zlonamjerne Web stranice, a određivanje tih značajki nije trivijalan zadatak. Mehanizam također mora biti prilagodljiv tako da se regulira broj stranica koje će se proslijediti zahtjevnijim mehanizmima poput *honeypot* klijenta. Jedan od predloženih načina za to jest izračun numeričke vrijednosti potencijalne zlonamjernosti umjesto binarnog klasifikatora zlonamjernosti odnosno benignosti, pri čemu se određuje prag kojeg stranica mora prijeći kako bi se proslijedila učinkovitijem, ali sporijem i zahtjevnijem algoritmu te se prilagođavanjem tog praga može utjecati na performanse. Mehanizam treba biti tako dizajniran da se minimiziraju lažno negativne detekcije koje su gore od lažno pozitivnih detekcija koje će se proslijediti na provjeru pouzdanijem i zahtjevnijem algoritmu, budući da će u slučaju lažno negativne detekcije zlonamjerne stranice ostati nezamijećene. Pritom postoji veza između praga kojeg stranica mora prijeći da bi se proslijedila i postotka lažno negativnih detekcija.

2.1.2 Napadi klijentske strane zasnovani na webu

Rastom broja korisnika Interneta, napadi zasnovani na webu koji preko zlonamjernih Web stranica mogu iskoristiti ranjivosti sustava korisnika postale su glavni problem u internetskoj sigurnosti, budući da se prilikom posjeta Web stranici mogu iskoristiti ranjivosti preglednika, dodataka pregledniku ili samog operacijskog sustava kako bi se kompromitiralo računalo korisnika.

Web aplikacija može se definirati kao mrežna aplikacija koja tipično međudjeluje s internetskim preglednikom preko Interneta, a pružatelji informacijskih usluga koriste Web aplikacije kako bi dostavili svoje usluge korisnicima. Pružatelji usluga pritom implementiraju svoju poslovnu logiku unutar Web aplikacija na internetskim poslužiteljima do kojih se dolazi preko javno dostupnih URL-ova te pritom mogu koristiti više od jednog Web poslužitelja, odnosno pozadinskog poslužitelja i aplikacija koje surađuju kako bi dostavile sve usluge korisnicima. Na klijentskoj strani postoji glavna aplikacija internetskog preglednika kojom se korisnici služe kako bi pristupili pruženim uslugama, no preglednici podržavaju i dodatke treće strane poput Adobe Acrobata i Apple QuickTimea, a postojali su i AdobeFlash i ActiveX koji nisu više podržani.

Kako bi napadač dostavio zlonamjerni sadržaj do klijentske strane najprije ga mora objaviti na Internetu, a uobičajeni način je kompromitacijom internetskog poslužitelja. Neki od načina kompromitacije internetske stranice su iskorištavanje ranjivosti internetskog poslužitelja, iskorištavanje ranjive Web aplikacije ili ranjivih SQL aplikacija napadima kao što su SQL ubacivanje (engl. *SQL injection*) ili JavaScript ubacivanje (engl. *JavaScript injection*), a rezultat je umetanje zlonamjernih fragmenata u Web stranice koji se mogu dostaviti do sustava korisnika. Web 2.0 tehnologija postala je dobra okolina za širenje zlonamjernih sadržaja budući da dopušta posjetiteljima postavljanje proizvoljnog sadržaja što može uključivati zlonamjerni kod, poveznice na zlonamjerne stranice ili postavljanje zlonamjernih datoteka.

Nakon postavljanja zlonamjernog sadržaja na Internet, napadači trebaju navesti korisnike da posjete zlonamjerne Web stranice kako bi se iskoristile ranjivosti. Masovno slanje elektroničke pošte koja sadrži poveznice na zlonamjerne Web stranice i ostale metode društvenog inženjeringa, primjerice putem blogova ili društvenih mreža, mogu se koristiti za privlačenje korisnika, neke legitimne stranice mogu imati nekontrolirani sadržaj treće strane poput oglasa koje upućuju na takve stranice, a internetski preglednici također se mogu zlouporabiti u tu svrhu korištenjem ključnih riječi zbog kojih će se zlonamjerne Web stranice prikazati kao rezultati popularnih pretraga.

Uz navedene metode postoje alati za iskorištavanje ranjivosti Weba koji lako kompromitiraju poslužitelje te dostavljaju zlonamjerni sadržaj korisnicima, što često rezultira preusmjeravanjem njihovih zahtijeva na mreže za distribuciju zlonamjernih programa, a određena istraživanja pokazuju da se takvim mrežama često i dostavljaju same zlonamjerne Web stranice.

Kako bi uočili ranjivosti klijentskog sustava, uključujući preglednike i njihove dodatke, napadi koriste podršku za skripte, primjerice preko JavaScripta ili Visual Basica, kako bi

sakupili informacije o napadnutom računalu, a kod kojim se iskorištavaju ranjivosti često se obfuscira, odnosno čini manje čitljivim kako bi se teže detektirao i analizirao.

2.1.3 Kratki pregled metoda detekcije

U analiziranom članku [1] razmatraju se tri glavna pristupa detekcije: pristup baziran na potpisu (engl. *signature-based*), pristup baziran na promjeni stanja (engl. *state-change*) i pristup baziran na strojnom učenju.

U pristupima, odnosno metodama zasnovanima na potpisu, sustavi za detekciju koriste poznate potpise, odnosno uzorke zlonamjernog koda ili sadržaja kako bi detektirali zlonamjerne Web stranice, a potpisi se često nalaze unutar baza podataka na nekim poznatim sustavima za detekciju uljeza (engl. *Intrusion Detection Systems*, skraćeno IDS) ili antivirusnim programima. Za razliku od *honeypot* klijenata visoke interaktivnosti koji su više usredotočeni na dinamičku analizu te šalju puno veći broj zahtjeva, metode zasnovane na potpisu često se koriste u sustavima detekcije s *honeypot* klijentom niske interaktivnosti. HTTP odgovori poslužitelja također mogu biti pretvoreni u odgovarajući XML format koji se onda analizira s obzirom na poznate potpise ili uzorke, a internetske stranice se također mogu dohvatiti i spremiti u datoteke te skenirati antivirusnom aplikacijom kako što je ClamAV [3]. Nedostatak ovakvih metoda je što obfuskacija koda ili male izmjene koda koje rezultiraju istom ili gotovo istom funkcionalnošću onemogućuju detekciju metodama zasnovanima na potpisu.

Metode zasnovane na promjeni stanja (engl. *state-change*) nazivaju se još i tehnike zasnovane na pravilima (engl. *rule-based*), a koriste ih *honeypot* klijenti visoke interakcije. Glavna je ideja nadzor promjene stanja u klijentskom sustavu u trenutku pristupa određenom URL-u. Ako dođe do bilo koje neovlaštene promjene stanja koja nije eksplicitno dopuštena te popraćena odgovarajućim sistemskim logovima, stranica se klasificira kao zlonamjerna. U pojedinim sustavima kao što je Strider HoneyMonkeys, aplikacija učita preglednik te mu daje uputu da posjeti svaki URL i neko vrijeme pričeka pojavu procesa preuzimanja koji bi se mogli pojaviti te se onda u sustavu provjerava postoje li neautorizirana kreiranja izvršnih datoteka ili unosa registara.

Drive-by-download napad odnosi se na napad u kojem kad internetski preglednik zatraži sadržaj od poslužitelja, poslužitelj vraća kod koji iskorištava ranjivost, a taj je kod ugrađen u internetsku stranicu te omogućuje kontrolu nad klijentskim sustavom. Kako bi se detektirao *drive-by-download* napad neki sustavi u radu s metodama zasnovanim na promjeni stanja koriste okidače događaja (engl. *event triggers*). U mehanizmu okidača događaja kreiraju se uvjeti okidača kako bi se pratile neovlaštene aktivnosti u kreiranju procesa, datotečnom sustavu i sistemskom registru, a uvjeti također uključuju događaje koji uzrokuju rušenje preglednika i sustava. Ako URL uzrokuje pokretanje okidača tijekom njegovog posjećivanja, on se označava kao nesiguran.

Neki *honeypot* klijentski sustavi koriste modul za nadzor ponašanja kako bi pratili zlonamjerno ponašanje tako da putem nativnih API-ja i DLL funkcija nadziru sve aktivnosti koje uzrokuju prelijevanje spremnika (engl. *buffer overflow*) te pristupanje resursima sustava kao što su procesi, mrežni resursi, datoteke i registar.

Kao primjer metoda strojnog učenja koje se koriste za detekciju zlonamjernih stranica, naveden je primjer detekcije zasnovane na asocijativnoj klasifikaciji koji je opisan u poglavlju 3.1, no sam se rad fokusira na analizu i implementaciju metoda koje nisu zasnovane na strojnom učenju i koje uzimaju u obzir statičke atribute stranice, tako da se ova skupina metoda neće detaljno razmatrati.

Osim metode zasnovane na asocijativnoj klasifikaciji čija će osnovna ideja i mehanizam biti opisani, u radu će biti predstavljena tri algoritma, uključujući njihov teoretski pregled i opis implementacije koja je provedena u okviru diplomskog zadatka: statička heuristika (engl. *static heuristics*), algoritam temeljen na mehanizmu bodovanja (engl. *scoring mechanism*) i algoritam temeljen na Yara pravilima (engl. *yara rules*), a svaki od algoritama testirat će se i optimizirati na skupu od 75 492 stranice preuzete iz dane MongoDB baze podataka.

Programski kod implementacije diplomskog rada dostupan je na javnom Github repozitoriju [4].

2.2 Metodologija implementacije i testiranja odabranih algoritama

2.2.1 Osnovni alati i metode izrade diplomskog rada

Za izradu diplomskog rada, odnosno razvoj implementacije algoritama detekcije zaraženih stranica bit će korišten Python na operacijskom sustavu Windows 10 te je na računalo potrebno instalirati Python razvojno okruženje sa službene Python stranice [5]. Preuzima se najnovija verzija Pythona 3.9.2, a prilikom instalacije odabire se instalacija na putanju prikazanu u ispisu 2.1, uz odabir opcije instalacije Python pokretača za sve korisnike te dodavanja Pythona 3.9 u sistemsku putanju.

Testni setovi podataka za provjeru implementacije diplomskog rada bit će dostupni u MongoDB bazi podataka na koju će se imati pristup čitanja te je potrebno na računalo instalirati MongoDB razvojno okruženje. Instalacija MongoDB poslužitelja preuzima se sa službene MongoDB Community stranice [6]. Uz instalaciju je potrebno odabrati opciju potpune instalacije, pri čemu će na 64-bitnom Windows 10 operacijskom sustavu bazni direktorij poslužitelja biti kao što je prikazano u ispisu 2.2. Uz instalaciju dolazi programski alat MongoDB Compass u koji je potrebno unijeti podatke o vezi koja može biti u standardnom ili SRV formatu, a primjer je dan u ispisu 2.3. Danoj MongoDB bazi podataka za koju će biti dostupni podaci za povezivanje pristupat će se iz MongoDB Compass alata unosom odgovarajućeg niza znakova za spajanje (engl. *connection string*).

Prije same izrade diplomskog rada i implementacije, potrebno je ponoviti osnove programskog jezika Python, proučiti postojeće algoritme i metode detekcije zlonamjernih HTML stranica te odlučiti koji će se konkretni algoritmi implementirati u Python kodu. U kasnijim fazama rada kad će se implementirane metode testirati na danim skupovima podataka, potrebno je proučiti način stvaranja veze na aktivnu MongoDB instancu iz Pythona te učitavanja podataka kako bi se testiranje moglo izvršiti.

Programi pisanu u Pythonu trebaju imati ekstenziju .py te se izvršavaju naredbom prikazanom u ispisu 2.4.

```
C:\Users\<korisničko-ime>\AppData\Local\Programs\Python\Python39
```

Ispis 2.1: Putanja instalacije Python razvojnog okruženja

```
C:\Program Files\MongoDB
```

Ispis 2.2: Instalacijska putanja MongoDB alata

```
mongodb+SRV://username:password@cluster0-jtpxd-mondodb.net/admin
```

Ispis 2.3: Podaci o vezi na MongoDB bazu podataka u MongoDB Compass alatu

```
python <putanja do programa>
```

Ispis 2.4: Pokretanje Python programa

Također je moguće koristiti interaktivnu ljusku za izvršavanje python naredbi unosom naredbe `python` u Windows PowerSheel ljusku ili pokretanjem Python 3.9 aplikacije iz Start izbornika koja nudi svoju konzolu, a iz interaktivne ljuske izlazi se naredbom `exit()`. Takvo interaktivno pokretanje bit će korisno za testiranje jednostavnih naredbi, modula i isječaka koda prije nego se oni uklope u glavni Python program.

2.2.2 Učitavanje HTML stranice iz testne baze podataka

MongoDB bazi podataka u kojoj se nalaze testne Web stranice pristupa se preko podataka koji će biti specificirani nizom znakova za spajanje, a pristup je moguć putem MongoDB Compass alata ili korištenjem `pymongo` modula u Pythonu.

Budući da se bazi podataka može pristupiti samo preko lokalne adrese (127.0.0.1), odnosno nije javno dostupna preko Interneta, potrebno je postaviti SSH tuneliranje prije nego se moguće povezati bilo preko MongoDB Compass alata, bilo korištenjem `pymongo` modula. Postoji način uspostavljanja SSH tuneliranja u MongoDB Compass alatu, no to ne rješava problem povezivanja korištenjem `pymongo` modula pa je zato odabran način uspostavljanja SSH tuneliranja preko PowerShell ljuske. Povezivanje na SSH i tuneliranje preko PowerShell terminala postiže se naredbom prikazanom u ispisu 2.5.

```
ssh -L 27017:127.0.0.1:27017 <korisničko-ime>@<ip-adresa>
```

Ispis 2.5: Pokretanje Python programa

Nakon uspješnog uspostavljanja SSH tunela moguće je pristupiti bazi podataka pomoću specificiranog niza znakova za spajanje u MongoDB Compass alatu bez dodatnih postavki, s tim da SSH veza u PowerShellu pritom mora biti otvorena. Format niza znakova za spajanje na MongoDB bazu podataka prikazan je u ispisu 2.6.

```
mongodb://<korisničko-ime>:<lozinka>@127.0.0.1:27017/<ime-
baze>?authSource=<ime-baze>
```

Ispis 2.6: Format niza znakova za spajanje na MongoDB bazu podataka

Pritom MongoDB korisnik koji će biti specificiran u ovom slučaju ima *read-only* pristup, odnosno omogućeno mu je samo čitanje podataka iz baze, ali ne i njihov unos, brisanje ili izmjena. Ako je na lokalnom računalo već pokrenut MongoDB poslužitelj na istom portu, potrebno ga je zaustaviti, što se na Windows 10 operacijskom sustavu postiže u sistemskom izborniku Services.

U slučaju problema sa stvaranjem SSH tunela korištenjem PowerShell ljuske, u MongoDB Compass alatu moguće je to postići tako da se odabere opcija 'Fill in connections fields individually' -> 'More options' -> 'SSH Tunnel' i upiše odgovarajuće podatke SSH veze.

Povezivanje na MongoDB bazu podataka korištenjem `pymongo` modula izvodi se u skladu s danim uputama na službenoj MongoDB stranici [7].

Kako bi se omogućilo povezivanje na MongoDB bazu podataka, najprije je potrebno instalirati `pymongo` modul naredbom u PowerShell ljusci prikazanom u ispisu 2.7.

```
python -m pip install pymongo
```

Ispis 2.7: Instalacija pymongo modula

Mongo klijent iz spomenutog modula uključuje se u kod naredbom prikazanom u ispisu 2.8.

```
from pymongo import MongoClient
```

Ispis 2.8: Uključivanje pymongo modula u kod

Iako će se za povlačenje cijelog testnog skupa podataka iz baze koristiti potpuno drugačija logika koja određuje sve pohranjene Web stranice u HTML formatu te ih pohranjuje lokalno na računalo kako bi se kasnije nad njima izvodilo testiranje, ovdje se razmatra osnovni način povlačenja stranice sa zadanim identifikatorom iz određene baze podataka i određene kolekcije na danom poslužitelju.

U `pagefetch` modul koji se kreira u svrhu omogućavanja funkcionalnosti dohvaćanja sadržaja stranica koje će se analizirati, dodaje se funkcija `fetchMongoHTML` kojoj je zadaća na temelju specificiranog identifikatora stranice dohvatiti HTML pohranjen u testnoj MongoDB bazi podataka. Ta će funkcija koristiti MongoDB vezu definiranu unutar modula i kreiranu unutar funkcije `setMongoConnection` čija je zadaća povezati se na bazu podataka putem specificiranog niza znakova za povezivanje. Funkcijom za povezivanje također će se definirati ime baze podataka te ime kolekcije u bazi podataka u kojoj se nalaze podaci o HTML stranicama.

Inicijalizacija veze na MongoDB poslužitelj te postavljanje podataka potrebnih za dohvaćanje stranice implementira se kako je prikazano u ispisu 2.9.

```
def setMongoConnection(connectionString, databaseName, collectionName):
    global mongoConnection, webpagesDatabaseName,
    webpagesCollectionName
    mongoConnection = MongoClient(connectionString)
    webpagesDatabaseName = databaseName
    webpagesCollectionName = collectionName
```

Ispis 2.9: Inicijalizacija veze na MongoDB poslužitelj

Pri dohvaćanju stranice sa specificiranim identifikatorom treba se koristiti `ObjectId` dodatak [8] koji se u Python program mora uvesti naredbom prikazanom u ispisu 2.10.

```
from bson.objectid import ObjectId
```

Ispis 2.10: Uvođenje ObjectId dodatka u Python program

U ovom slučaju podaci o stranicama nalaze se u kolekciji `crawled_data_pages_v0`, te je svaka stranica identificirana jedinstvenim identifikatorom `_id` i sadrži komponentu `page` koja ima sadržaj HTML-a. Funkcija za dohvaćanje stranice prema identifikatoru iz specificirane baze podataka, odnosno kolekcije izgledat će kako je prikazano u ispisu 2.11.

```
def fetchMongoHTML(pageId):
    if mongoConnection is None:
        raise ConnectionNotSetException("MongoDB connection not set")
    webpagesDatabase = mongoConnection[webpagesDatabaseName]
    webpagesCollection = webpagesDatabase[webpagesCollectionName]
    html = webpagesCollection.find_one({"_id":
    ObjectId(pageId)})['page'];
    return html
```

Ispis 2.11: Dohvaćanje stranice iz MongoDB baze podataka

Pritom veza na bazu podataka mora biti uspostavljena, inače se generirana iznimka `ConnectionNotSetException` definirana u istom modulu kako je prikazano u ispisu 2.12.

```
class ConnectionNotSetException(Exception):
    pass
```

Ispis 2.12: Generiranje iznimke ako veza na bazu podataka nije uspostavljena

3 Algoritmi detekcije zlonamjernih Web stranica

3.1 Detekcija zasnovana na asocijativnoj klasifikaciji

3.1.1 Motivacija za metodu asocijativne klasifikacije

Metoda detekcije zlonamjernih URL-ova zasnovana na asocijativnoj klasifikaciji jedan je od pristupa koji uključuje strojno učenje, a opisana je u članku 'Malicious URL Detection Based on Associative Classification' [9].

U svrhu detekcije zlonamjernog sadržaja na Web stranicama, odnosno zlonamjernih URL-ova, istraživači specijalizirani za to područje predlažu različite pristupe obrane, uključujući statičku analizu, dinamičku analizu, metode zasnovane na crnim listama te pristupe zasnovane na statičkoj heuristici.

Tehnike statičke analize koriste statičke značajke HTML stranica kako bi ispitale stranice bez potpunog izvršavanja i prikazivanja stranica u internetskom pregledniku. Metode dinamičke analize kao Cuckoo [10] i SpyProxy [11] koriste okolinu za analizu ponašanja kako bi detektirale zlonamjerne skripte, no njihova je mana što napadači lako mogu otkriti takvu okolinu za analizu, što povećava njihovu šansu izbjegavanja procesa nadzora ponašanja.

Kod metodi baziranih na crnim listama, URL-ovi za koje se želi doznati jesu li zlonamjerni provjeravaju se uzimajući u obzir prethodno definirani popis zlonamjernih URL-ova, ali takve metode nisu uspješne u detekciji novih zlonamjernih Web stranica koje su nedavno postale dostupne na Internetu.

Tehnike bazirane na heuristici stvaraju potpise (engl. *signatures*) korisnih tereta poznatih napada za skeniranje i provjeru stranica, no sustave zasnovane na prethodno definiranim potpisima napadači mogu lako izbjeći, čime se propušta detekcija napada.

Istraživanja su također predlagala puno različitih tehnika dubinske analize podataka (engl. *data mining*) za detekciju zlonamjernih URL-ova, ali postoji ograničen broj radova koji govore o pristupu dubinske analize podataka zasnovanom na asocijativnoj klasifikaciji (AC). AC spaja dva područja dubinske analize podataka: dubinska analiza zasnovana na klasifikacijskim pravilima i dubinska analiza zasnovana na asocijativnim pravilima, kako bi izgradila točne klasifikatore koji se mogu lako interpretirati koristeći pravila asocijacije sa skupom podataka. Dubinska analiza zasnovana na klasifikacijskim pravilima otkriva mali skup pravila za formiranje točnih klasifikatora, dok dubinska analiza zasnovana na asocijativnim pravilima ima za cilj opisati skup podataka koristeći pouzdane asocijacije među uzorcima.

Za razliku od neuronskih mreža i vjerojatnosnih pristupa čiji su rezultat klasifikacijski modeli koje je teško razumjeti, pravila koja nastaju u procesu asocijativne klasifikacije krajnji korisnici mogu lako interpretirati, a asocijativna klasifikacija ima i tu prednosti da otkriva korisne skrivene informacije u skupovima podataka koje drugi klasifikacijski modeli najčešće propuštaju.

Nekoliko postojećih sustava koji funkcioniraju na principu asocijativne klasifikacije za detekciju zlonamjernih URL-ova fokusiraju se na određeni tip napada, na primjer *phishing*. *Phishing* [12] je vrsta napada društvenim inženjeringom koja se često koristi za krađu korisničkih podataka, uključujući podatke za prijavu i brojeve kreditnih kartica. Odvija se tako da napadač koji se pretvara da je strana od povjerenja namami žrtvu da otvori poruku elektroničke pošte nakon čega ju se navede da klikne na zlonamjernu poveznicu. Nakon klika na poveznicu dolazi do instalacije zlonamjernog programa poput ucjenjivačkog zlonamjernog programa na računalo žrtve ili neovlaštenog otkrivanja osjetljivih osobnih podataka.

Neka istraživanja predlažu i metode detekcije *phishing* stranica na temelju značajki samih URL-ova, no napadači mogu lako izbjeći detekciju takvih sustava tako da oponašaju leksičke značajke benignih URL-ova. Kad je zlonamjerni kod s klijentske strane ubačen u Web stranicu, URL te stranice nije izmijenjen pa sustavi zasnovani na leksičkim značajkama samih URL-ova imaju visoku stopu lažno negativnih rezultata. Unaprijeđene verzije takvog pristupa također uključuju analizu značajki baziranih na sadržaju poput statičkog sadržaja HTML stranica, no ne uključuju analizu obfusciranog JavaScript koda i funkcija koji su danas sve češći.

Cilj je stvoriti efikasnu i fleksibilnu metodu za detekciju zlonamjernih Web stranica kako bi se pratio razvoj načina na koji se zlonamjerni sadržaj uključuje i skriva na stranicama. Ta metoda neće imati ograničenja prisutna kod prethodno opisanih metoda te će učinkovito otkrivati zlonamjerne internetske stranice, uključujući *phishing*, stranice koje sadrže zloćudni program (engl. *malware*) i *drive-by-download* stranice.

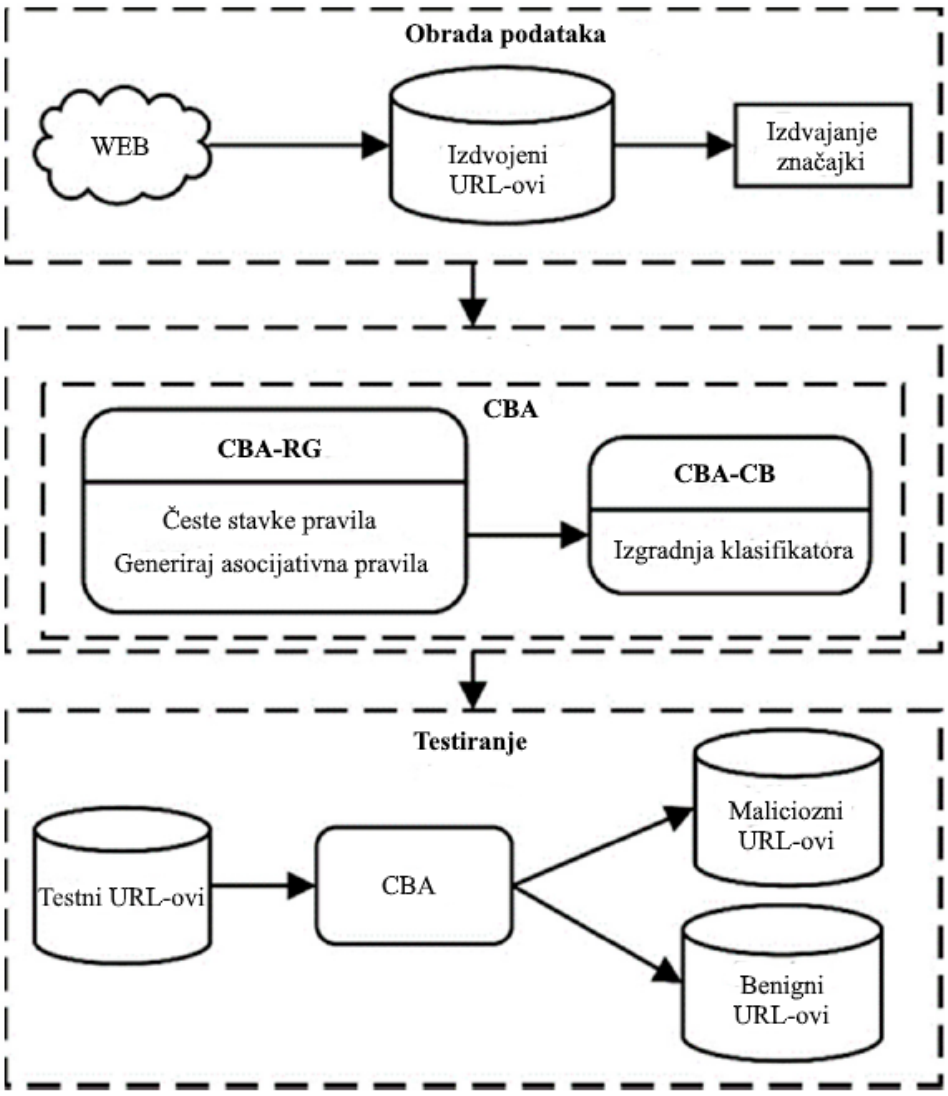
U analiziranom članku koristi se algoritam klasifikacije bazirane na asocijaciji (engl. *clasification based on association*, skraćeno CBA), kako bi se detektirali zlonamjerni URL-ovi na temelju značajki samog URL-a i na temelju značajki sadržaja Web stranice. CBA se sastoji od dva dijela: generator pravila (engl. *rule generator*, skraćeno CBA-RG) i graditelj klasifikatora (engl. *classifier builder*, skraćeno CBA-CB)

CBA stvaranje pravila dubinskom analizom podataka (engl. *rule mining*) koristi apriori algoritam kako bi našao korelaciju između značajki izdvojenih iz Web stranica te izgrađuje klasifikator zasnovan na pravilima koji spaja klasifikaciju s dubinskom analizom podataka zasnovanom na asocijativnim pravilima.

Dobivena otkrića metode trebala bi pomoći stručnjacima u razumijevanju načina na koji zlonamjerni URL-ovi evoluiraju te veza između njihovih značajki, odnosno atributa, a eksperimentalni rezultati pokazuju da stvaranje pravila dubinskom analizom podataka CBA algoritma postiže odlične rezultate na stvarnim skupovima podataka što se tiče točnosti, brzine i stope lažno pozitivnih rezultata (engl. *false positive rate*, skraćeno FPR).

3.1.2 Metodologija klasifikacije zasnovane na asocijaciji

Predložena metodologija ima za cilj analizirati i klasificirati URL kao zlonamjeran ili benigan, a arhitektura sustava čije će relevantne komponente biti detaljno opisane u ovom poglavlju prikazana je na slici 3.1.



Slika 3.1: Arhitektura sustava klasifikacije zasnovane na asocijaciji

U fazi obrade podataka najprije se s Interneta dobiva određeni skup URL-ova različitim metodama kako što su ubacivanje najpopularnijih fraza pretrage u internetske preglednike i sakupljanje određenog skupa najviše rangiranih rezultata dobivenih takvom pretragom. Nakon toga se iz tih URL-ova u drugom dijelu faze obrade podataka izdvajaju značajke na temelju samih URL-ova i sadržaja (HTML i JavaScript), nakon čega će se koristiti CBA model za treniranje i pravljenje predviđanja.

Izdvajanje značajki Web stranica pomaže u detekciji bilo kakve zlonamjerne aktivnosti, a opisani pristup se tiče zlonamjernih URL-ova koji su povezani s *phishingom*, stranicama sa zloćudnim programima i stranicama koje uključuju *drive-by-download* napade. Značajke se izdvajaju iz pribavljenih Web stranica kako bi ih se klasificiralo kao benigne ili zlonamjerne, a pritom se izdvajaju dva tipa značajki - značajke URL-ova te značajke sadržaja stranice, odnosno HTML i JavaScript koda.

Značajke URL-ova prikupljene su leksičkim skeniranjem niza znakova URL-a, a značajke sadržaja Web stranica prikupljene su posjećujući stranice preko Selenium WebDrive alata [13] i *headless* Chrome preglednika. Općenito se značajke URL-ova mogu podijeliti u dvije kategorije: leksičke značajke i značajke zasnovane na domaćinu. No kako bi se odredile razlike između zlonamjernih i benignih URL-ova koriste se samo leksičke odnosno tekstualne značajke URL-ova, a ne uključuju se značajke zasnovane na domaćinu kao što su podaci o IP adresi, WHOIS podaci ili geolokacijski podaci, budući da izdvajanje takvih značajki troši puno vremena i resursa.

Leksičke značajke URL-ova koje se uzimaju u obzir u analizi uključuju specijalne znakove, entropiju domenskog imena i osjetljive riječi.

Napadači koriste specijalne znakove za napade enkodirane u URL-ovima kako bi izbjegli validacijsku logiku pa se zato određuje broj specijalnih znakova u URL-u, uključujući ';', '+=', '_', '?', '=', '&', '[', ']', '#', '~', '%', '@', '$', '*', '+', '!' i '|'.

Entropija domenskog imena mjeri faktor nasumičnosti ili nesigurnosti u URL-ovima, odnosno što je viša entropija, to je viši faktor nasumičnosti u URL-u, pri čemu se entropija koristi kako bi se detektirala nasumično generirana (engl. *randomized*) domenska imena. Neki zlonamjerni URL-ovi koriste algoritme za generiranje domena (engl. *domain generation algorithms*, skraćeno DGA) kako bi često mijenjali domene zato da stavljanje takvih URL-ova na crnu listu ne budu učinkovito. DGA je program koji pruža zlonamjernoj aplikaciji nove domene na zahtjev ili automatizirano i periodički pa su zato URL-ovi s visokom entropijom značajni pokazatelji zlonamjernog ponašanja. Entropija može pomoći u detekciji zlonamjernih URL-ova postavljanjem pragova na temelju entropije benignih i pouzdanih URL-ova. Entropija domenskih imena izračunava se Shannonovom formulom prikazanom u formuli 3.1.

$$H(x) = -\sum_{i=0}^{n} p(x_i) log_b p(x_i)$$

Formula 3.1: Shannonova formula za entropiju

Pritom je `H(x)` Shannonova entropija niza znakova `x`, `b` jest baza logaritma koja se koristi, a `p(xi)` je funkcija vjerojatnosne mase.

Postoji lista osjetljivih riječi koje se najčešće nalaze u *phishing* URL-ovima te se URL podijeli u tokene kako bi se izbrojale osjetljive riječi u njemu, na primjer 'confirm', 'account',

'secure', 'banking', 'login', 'signin', pri čemu metoda u analiziranom članku uzima u obzir samo osjetljive riječi engleskog jezika.

Uz značajke samih URL-ova, značajke sadržaja Web stranica predstavljaju vrlo bitan pokazatelj zlonamjernosti. HTML oznake (engl. *tags*) omogućuju djelovanje zlonamjernih programa preusmjeravanjem korisnika na kompromitirane Web stranice, a zlonamjerni JavaScript kod sastoji se od sumnjivih uzoraka i funkcija za pokretanje *drive-by-download* napada i distribuciju zlonamjernih programa. Napadači također često koriste JavaScript kod kako bi izbjegli detekciju te su kodovi za preusmjeravanje ubačeni su u obfuscirani JavaScript kod kako bi sakrili odredišta preusmjeravanja, a značajke `location` objekta koriste se kako bi preusmjerili korisnike na zlonamjerne stranice. Pritom JavaScript funkcije koje napadači često koriste između ostalog uključuju `eval()`, `escape()`, `unescape()`, `replace()`, `exec()` i `unbound()`. Također, skrivene poveznice ubacuju se u internetske stranice kako bi pratile aktivnost korisnika.

Relevantne značajke Web stranice koje se izdvajaju za korištenje u navedenom algoritmu su značajke zasnovane na veličini Web stranica, `iframe` oznake, `object` oznake, elementi male veličine, `script` oznake, DOM funkcije, obfuskacija i sumnjive funkcije, učestalost `var` i `functions` ključnih riječi u JavaScript kodu i udio *whitespace* znakova.

Kao relevantne značajke na temelju veličine Web stranica izdvajaju se broj linija i maksimalna duljina linije, a za zlonamjerne stranice je vjerojatnije da će imati jednu dugu liniju koda. U promatranom skupu podataka otkriveno je da 25% zlonamjernih URL-ova s međusobno jedinstvenim domenskim imenima imaju isti ukupni broj linija koda, a benigne stranice s jednom linijom koda imale su maksimalnu duljinu linije između 6 i 496 znakova, dakle ako je ukupan broj linija koda 1, a maksimalna duljina linije veća od 500 znakova, za Web stranicu se smatra da je visoko zlonamjerna.

`Iframe` oznake uzimaju se u obzir jer ih napadači mogu koristiti za ubacivanje i učitavanje zlonamjernog koda u Web stranice, a elementi male veličine bitni su jer napadači koriste male vrijednosti visine i širine oznaka kako bi napadi bili nevidljivi korisnicima.

Sadržaj Web stranice statistički se analizira s obzirom na prisutnost unutarnjih i vanjskih script oznaka na Web stranici. Napadači ugrađuju maliciozan JavaScript kod u Web stranice kako bi pristupili osjetljivim korisničkim podacima, a ubacivanje zlonamjernog JavaScript koda može dovesti do *cross-site request forgery* (CSRF) ranjivosti koja omogućuje napadaču da iskoristi kolačiće i dopuštenja korisničkog preglednika žrtve za izvođenje zlonamjerne akcije na nekoj drugoj Web stranici. Benigne stranice najčešće uključuju vanjske skripte u svrhu oglašavanja i analitike, a zlonamjerne Web stranice ih koriste s namjerom preusmjeravanja korisnika na kompromitirane Web stranice.

Što se tiče DOM funkcija, napadači koriste JavaScript kako bi manipulirali elementima u DOM (engl. *document object model*) stablu stranice, a funkcije koje to rade napadači koriste kako bi neovlašteno mijenjali prikaz podataka koje su kreirali korisnici na Web stranici. Ovaj algoritam analizira sadržaj JavaScript koda kako bi se provjerilo prisustvo DOM funkcija kao što su `appendChild`, `createElement`, `getElementByTagName` i `getElementById` na Web stranici.

Napadači koriste funkcije za obfuskaciju kako bi izbjegli detekciju i otežali analizu koda ili uključuju zlonamjerne JavaScript privitke koji se pokreću preko Windows programa poput `WScript.Shell`, a sumnjive funkcije kao što su `ActiveXObject`, `CreateObject`, `CreateTextFile`, `FileSystemObject` i `FileExists` koriste se kako bi se kreirale mape ili datoteke ili im se pristupilo, ili kako bi se stvorio *backdoor* za nadzor aktivnosti na računalu bez znanja korisnika.

Također se provjerava učestalost pojavljivanja `var` i `function` ključnih riječi na Web stranicama, a broj njihovog pojavljivanja manji je kod zlonamjernih URL-ova budući da je njihov izvorni JavaScript kod koji bi sadržavao te riječi obfusciran ili skripta ima *`src`* atribut, a ne uključuje *inline* kod.

Whitespace obfuskacija najjednostavnija je tehnika obfuskacije kojom napadači uključuju nasumične *whitespace* znakove (uključujući razmak, tabulator i znak za novi redak) unutar izvornog koda kako bi zbunili alate za statičku ili automatiziranu analizu.

CBA u svojoj glavnoj fazi koja dolazi nakon faze obrade podataka integrirani je algoritam koji se fokusira na sakupljanje (engl. *mining*) pravila asocijacije iz dane baze podataka zvane *Class Association Rules* (CARs) i gradi klasifikator na osnovi otkrivenog CAR seta.

CBA, koji predstavlja ujedno i fazu treninga u metodologiji predstavljenoj analiziranim člankom sastoji se od dvije faze: generator pravila (engl. *Rule Generator*, skraćeno CBA-RG) i graditelj klasifikatora (engl. *Classifier builder*, skraćeno CBA-CB).

CBA-RG, odnosno generator pravila baziran je na apriori algoritmu koji pronalazi sve CAR-ove koji zadovoljavaju minimalni prag potpore (engl. *minsup*) te minimalni prag pouzdanosti (engl. *minconf*) koje definira korisnik. Potpora predstavlja mjeru učestalosti određenog skupa elemenata u svim transakcijama, a pouzdanost je definirana kao mjera sigurnosti povezane sa svakim otkrivenim skupom podataka, odnosno vjerojatnosti da će se unija skupova A i B pojaviti u transakciji ako se pojavi skup elemenata A. Predloženi apriori algoritam koristi se za sakupljanje (engl. *mining*) čestih skupova podataka za pravila asocijacije koja su tipa `Boolean`, a CBA-RG prelazi preko povijesno izdvojenih atributa URL-ova koji predstavljaju skup podataka za trening kako bi se otkrila sva česta pravila, odnosno stavke pravila (engl. *rule items*) od kojih je CAR generiran.

Formula 3.2. definira potporu skupa pravila `A` kao omjer između broja transakcija koji sadrži A i ukupnog broja transakcija.

$$Support(A) = \frac{support\_count(A)}{total\ number\ of\ transactions}$$

Formula 3.2: Potpora skupa pravila

Formula 3.3. definira pouzdanost pravila, pri čemu je `A` prethodnik (preduvjet), `B` je posljedica, `support_count(A U B)` je broj transakcija koji sadrže uniju skupova pravila `A U B`, a `support_count(A)` broj transakcija koje sadrže skup pravila `A`.

$$Confidence(A \rightarrow B) = \frac{support\_count(A \cup B)}{support\_count(A)}$$

Formula 3.3: Pouzdanost pravila

Pritom je pravilo (engl. *rule item*) u formi `<condset,y>` te predstavlja pravilo `condset => y`, gdje je `condset` skup određenih stavki, primjerice vrijednosti atributa, koje predstavljaju uvjete, a `y ∈ Y` je oznaka klase koja može biti `'malicious'` ili `'benign'`. Formula 3.4 primjer prikazuje pravilo za klasifikaciju URL-a, gdje su A i B atributi, pri čemu su pravila interpretirana kao ako-onda tvrdnje:

$$\{(A{=}1),(B{=}1)\} => \{\text{class} = \text{benign}\}$$

Formula 3.4: Primjer pravila za klasifikaciju URL-a

U navedenom primjeru, ako je `A` jednako 1 i `B` jednako 1, onda će URL biti klasificiran kao benigni.

Graditelj klasifikatora (CBA-CB) izgrađuje klasifikator koristeći generirane CAR-ove. Kako bi se izgradio točni klasifikator, CBA-CB koristi proceduru sortiranja pravila i postupnog uklanjanja pokrivenosti podataka kako bi se odbacila redundantna pravila. Pravila za izgradnju klasifikatora odabrana su na temelju redoslijeda rangiranja, na primjer pravila, odnosno skupovi pravila s visokom pouzdanošću i potporom odabrana su prva. Kao procedure uklanjanja pokrivenosti podataka pritom se koriste M1, odnosno direktna verzija CBA algoritma i poboljšana verzija M2.

M1 algoritam prolazi CAR bazu više puta kako bi pronašao optimalni broj pravila koja će se koristiti za izgradnju točnog klasifikatora, a autori CBA algoritma predstavljaju i poboljšanu verziju algoritma zvanu M2 kojoj je svrha smanjiti pristup podacima kad podataka ima previše da bi ih se sve pohranilo u radnoj memoriji, no budući da se dostupna radna memorija povećala i performanse računala su se poboljšale, M2 algoritam je postao manje relevantan. Analize performansi samih algoritama također pokazuju da je M1 verzija CBA algoritma brža od M2 verzije algoritma u većini slučajeva, tako da će se kao metoda uklanjanja za odbacivanje redundantnih pravila koristiti algoritam M1.

Algoritam M1 ima 3 koraka:

1) Sortiranje generiranih pravila prema operatoru prioriteta (>) s obzirom na potporu i pouzdanost.

2) Odabir pravila za klasifikator prema sortiranoj sekvenci, pri čemu se pravilima prolazi po bazi podataka D kako bi se našle instance koje zadovoljavaju pravila, a ako pravilo ispravno klasificira instancu u bazi podataka, označeno je i umetnuto na kraj klasifikatora C. Potom se instance koje su odabrana pravila ispravno klasificirala uklanjaju iz baze podataka. Pri ovom koraku odabire se pretpostavljena klasa, na primjer klasa koju ima većina preostalih instanci,

kako bi se osiguralo da je instanca klasificirana čak i ako se niti jedno drugo pravilo s njome ne podudara u klasifikatoru.

3) Odbacuju se pravila u klasifikatoru C koja ne poboljšavaju točnost klasifikatora, a razlog odbacivanja je što generiraju više pogrešaka.

Algoritam M1 također se može prikazati pseudokodom kako je prikazano u ispisu 3.1.

```
Neka je R skup generiranih pravila (CAR-ova), a D podaci za trening
Ulaz: Skup generiranih pravila, R, trening podaci, D
Izlaz: Klasifikator, C
R = sortiraj(R)
za svako pravilo r ∈ R u sekvenciji činiti
    temp = ∅;
    za svaku instancu d ∈ D činiti
        ako d zadovoljava uvjete od r onda
            pohrani d.id u temp i označi r ako ispravno klasificira d;
   ako je r označeno onda
       umetni r na kraj od C;
       obriši sve instance s id-ovima u temp iz D;
       odaberi pretpostavljenu klasu za trenutni C;
       izračunaj ukupni broj pogrešaka u C;
   kraj
kraj
pronađi prvo pravilo p u C s najnižim ukupnim brojem pogrešaka i ukloni
sva pravila nakon p iz C;
dodaj pretpostavljenu klasu povezanu s p na kraj od C;
vrati C;
```

Ispis 3.1: Pseudokod algoritma M1 za proceduru uklanjanja pokrivenosti podataka

Nakon faza generatora pravila i izgradnje klasifikatora koje čine fazu treninga s trening setom URL-ova, konačni klasifikator C koristi se za klasifikaciju testnih URL-ova na benigne i zlonamjerne URL-ove.

Cjelokupni algoritam detekcije zlonamjernih URL-ova koristeći CBA model može se sažeto prikazati pseudokodom prikazanim u ispisu 3.2.

Ulaz: Trening podaci D, minsup i minconf pragovi

Priprema za obradu: Diskretizacija kontinuiranih značajki ako takve postoje

Izlaz: Rezultati klasifikacije (Benigni i Zlonamjerni URL-ovi)

Korak 1: Generator pravila

a. Skeniranje trening podataka radi detekcije čestih pravila

b. Generiranje CAR-ova R iz čestih pravila, gdje je $r_i \in R$

Korak 2: Izgradnja klasifikatora

a. Sortiraj skup pravila prema pouzdanosti i potpori

b. Generiraj Cr: Skeniraj D radi pronalaska instanci pokrivenih s r_i i ispravno klasificiranih kao instanci u D

c. Izgradi klasifikator C koristeći pravila iz Cr

Korak 3: Klasifikacija URL-ova

a. Iskoristi C za klasifikaciju testnih podataka

Ispis 3.2: Pseudokod detekcije malicioznih stranica koristeći CBA model

3.1.3 Rezultati i usporedba performansi s drugim algoritmima

Kao relevantne parametre za procjenu performansi predloženog pristupa uzimaju se preciznost (engl. *precision*), *recall* omjer, matrica zbrke (engl. *confusion matrix*) i točnost (engl. *accuracy*).

Matrica zbrke C iskazana formulom 3.5 koristi se za evaluaciju preciznosti klasifikatora, a sadrži predviđene i stvarne klasifikacije koje klasifikator obavlja.

$$C = \begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

Formula 3.5: Matrica zbrke algoritma

Pritom `TP` (engl. *true positives*) ili stvarni pozitivni rezultati predstavljaju ispravna predviđanja zlonamjernih URL-ova, `TN` (engl. *true negatives*) ili stvarni negativni rezultati predstavljaju ispravna predviđanja benignih URL-ova, `FP` (engl. *false positives*) ili lažno pozitivni rezultati predstavljaju neispravna predviđanja zlonamjernih URL-ova, a `FN` (engl. *false negatives*) ili lažno negativni rezultati predstavljaju neispravna predviđanja benignih URL-ova.

Točnost je omjer točnih predviđanja i ukupnog broja uzoraka te je definirana formulom 3.6.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Formula 3.6: Izračun točnosti algoritma

Preciznost je omjer broja stvarnih pozitivnih rezultata i ukupnog broja stvarnih pozitivnih i lažno pozitivnih rezultata te je definirana formulom 3.7.

$$Precision = \frac{TP}{TP + FP}$$

Formula 3.7: Izračun preciznosti algoritma

Recall je definiran kao omjer broja stvarnih pozitivnih rezultata i ukupnog broja stvarnih pozitivnih i lažnih negativnih rezultata te se može izraziti formulom 3.8.

$$Recall = \frac{TP}{TP + FN}$$

Formula 3.8: Izračun recall omjera algoritma

Za sve eksperimente u analiziranom članku koristilo se Desktop računalo s Intel Core i5-7500 @ 3.40 GHz procesorom i 64-bitnim Windows 10 operacijskim sustavom i 16 GB radne memorije.

Skup podataka benignih i zlonamjernih URL-ova skupljen je iz reprezentativnih izvora, benigni URL-ovi sakupljeni su obilaskom Alexa top 500 stranica, a zlonamjerni URL-ovi skupljeni su iz različitih baza podataka kao što su OpenPhish [14], VxVault [15] i URLhaus [16] te su sakupljeni benigni i zlonamjerni URL-ovi označeni s B, odnosno M.

Za parsiranje URL-ova i izvlačenje njihovih značajki koristile su se ugrađene Python biblioteke, a rezultati izvlačenja značajki transformirani su u .csv datoteku.

CBA model klasifikacije URL-ova implementiran je u programskom jeziku R koristeći `arulesCBA` R programski paket, a CBA algoritam evaluiran je koristeći deseterostruku unakrsnu provjeru valjanosti za testiranje 1 200 označenih URL-ova, od kojih je 700 zlonamjerno, a 500 benigno. CBA algoritam pritom koristi dani set podataka kao povijesnu bazu URL-ova kako bi se otkrila pravila povezana s benignim i zlonamjernim URL-ovima.

Prag minimalne potpore ima jak utjecaj na točnost klasifikatora jer ako je postavljen previsoko, CAR-ovi možda neće uspjeti pokriti sve instance u setu podataka za treniranje. Sljedeće kombinacije (engl. *grid*) pragova potpore i pouzdanosti su istražene kako bi se izabrale najbolje vrijednosti pragova za generiranje CAR-ova: *GridSupport* = [1,2,3,4,5] i *GridConfidence* = [50, 60, 70, 80, 90 ,100]. Pritom je ustanovljeno da ako je *minsup* spušten na 1-2% s pragom pouzdanosti od 90%, klasifikator ima bolje performanse, a u konkretnom primjeru gdje se otkrivaju CAR-ovi prag minimalne potpore postavljen je na 1% te prag minimalne pouzdanosti na 90% uz maksimalnu duljina pravila 3.

Primjenom ovih postavki i parametara na skup podataka, skupljena su 52 pravila za analizu URL-ova s pretpostavljenom klasom `'malicious'`. Kontinuirane brojčane vrijednosti izdvojenih atributa diskretizirane su prije generiranja CAR-ova, a zato što CA radi s asocijacijskim pravilima, svi podaci moraju biti pretvoreni u binarne ili kategorički iskazane stavke. Diskretizacija je transformacija podataka kojom se atributi s kontinuiranim vrijednostima pretvaraju u kategoričke podatke. CBA algoritam takve podatke diskretizira koristeći entropijsku metodu koju su predložili Fayyad i Irani [17] i koja se u analiziranom članku ne opisuje u detalje, ali bit je da se kontinuirane vrijednosti diskretiziraju particioniranjem raspona vrijednosti u podraspone do određene najsitnije granulacije.

Uz odabir praga potpore od 1% i odabir praga pouzdanosti od 90%, točnost metode je visokih 95.83% s vrlo niskom stopom lažno pozitivnih i lažno negativnih rezultata, a atributi koji su se pokazali najznačajnijima su entropija domenskog imena, JavaScript oznake, DOM funkcije, postotak *whitespace* znakova te učestalost `var` i `function` ključnih riječi. Svi izdvojeni atributi korišteni kod algoritma asocijativne klasifikacije poredani od najznačajnijeg do najmanje značajnog prikazani su u tablici 3.1.

Značajka	Važnost
Entropija domenskog imena	0.419320
JavaScript oznake	0.190276
DOM funkcije	0.075120
Postotak whitespace znakova	0.073060
Učestalost var i function ključnih riječi	0.065900
Udio specijalnih znakova	0.042401
Broj linija	0.040867
Broj iframe oznaka	0.040005
Maksimalna duljina linije	0.024783
Obfuskacija i sumnjive funkcije	0.019935
Elementi s malom površinom	0.007263
Osjetljive riječi	0.001088

Tablica 3.1: Važnost značajki kod algoritma asocijativne klasifikacije

Metoda predstavljena u članku uspoređuje se s metodom Jeeva et al. [18] koja također koristi asocijativnu klasifikaciju, ali za razliku od pristupa opisanog u članku uzima u obzir samo leksičke značajke iz URL-ova i fokusira se samo na specifični tip URL-ova, odnosno napada, na primjer *phishing*. Eksperimenti za drugu metodu izvršeni su uporabom WEKA softvera koristeći parametre koje su specificirali autori druge metode na skupu podataka korištenom u članku. WEKA je softver otvorenog koda koji uključujuće kolekciju algoritama strojnog učenja za zadatke dubinske analize podataka [19].

Metoda s kojom s uspoređuje pokazuje dobre performanse, ali ne uključuje analizu i detekciju obfusciranog JavaScript koda koji je u današnje vrijeme primarno korišten kod *drive-by-download* napada te ima visoku stopu lažno negativnih rezultata od 7.57% u usporedbi sa stopom od 1.35% metode prezentirane u analiziranom članku, zbog čega je ta metoda sama po sebi manje učinkovita u detekciji zlonamjernih URL-ova. Razlog je što svaki napadač može imitirati tekstualne, odnosno leksičke značajke benignih URL-ova, tako da bi se URL-ovi trebali kombinirati sa značajkama baziranima na sadržaju (HTML i JavaScript) kako bi se poboljšale performanse sustava.

Kako bi se još preciznije procijenila učinkovitost CBA algoritma, njegove su performanse također uspoređene s dobro poznatim klasifikacijskim algoritmima, uključujući *support vector machine* (SVM) [20], naivni Bayes (engl. *naive Bayes*) [21] i logističku regresiju (engl. *logistic regression*) [22], a ti su klasifikatori bili korišteni u radu autora J. Ma et al. [23] za detekciju zlonamjernih stranica.

SVM konstruira linearnu hiperravninu koja razdvaja set podataka u različite klase te nalazi optimalno rješenje maksimizirajući marginu hiperravnine, odnosno udaljenost između hiperravnine i najbliže točke obaju klasa, a u radu je korištena `libsvm` biblioteka.

Naivni Bayes je vjerojatnosni klasifikator zasnovan na Bayesovom teoremu s pretpostavkom nezavisnosti između prediktora.

Formula Bayesovog teorema prikazana je formulom 3.9.

$$\mathrm{P(A|B)} = \frac{P(B|A)P(A)}{P(B)}$$

Formula 3.9: Bayesov teorem

Pritom `P(A|B)` predstavlja vjerojatnost `A` uz uvjet `B`, a `P(B|A)` vjerojatnost `B` uz uvjet `A`.

Logistička regresija je proširenje linearne regresije, a radi se o statističkoj metodi predviđanja binarne klasifikacije, pri čemu je varijabla o kojoj ovisi predikcija kategorička. U ovom se slučaju koristi binarna logistička regresija gdje ciljana varijabla ima dva ishoda koji uključuju benigni i zlonamjerni URL.

Eksperimenti za SVM, naivnog Bayesa i logističku regresiju također su izvršeni u WEKA softveru, pri čemu je zadržana pretpostavljena postavka svi klasifikatora dostupnih u WEKA API-ju za set podataka analiziran u članku koji se sastojao od 1200 URL-ova.

Uspoređuje se preciznost, *recall,* površina ispod ROC krivulje i stopa lažno pozitivnih rezultata svih navedenih klasifikatora s CBA metodom na istom setu podataka koristeći deseterostruku unakrsnu provjeru valjanosti. ROC predstavlja skraćenicu za krivulju radne karakteristike prijemnika (engl. *receiver operating characteristic*), a to je grafički prikaz koji ilustrira sposobnost binarnog sustava klasifikatora s promjenom praga diskriminacije.

Analiza pokazuje da je CBA metoda bolja od algoritma naivnog Bayesa s preciznošću od 91.30% i *recall* parametrom od 96.67% (u usporedbi s 90.9%, odnosno 90.3% algoritma naivnog Bayesa), dok logistička regresija i SVM imaju visoku stopu negativnih rezultata od 19.2%, odnosno 16.2%. Cjelokupni rezultati usporedbe prikazani su u tablici 3.2.

Klasifikator	**Preciznost(%)**	**Recall(%)**	**ROC površina (%)**	**Lažno Negativna Stopa**
SVM	83.4	80	81.9	16.2
Naivni Bayes	90.9	90.3	95.2	8.4
Logistička regresija	85.9	84.8	94.1	19.7
CBA	**91.30**	**97.67**	**96.2**	**8.6**

Tablica 3.2: Usporedba CBA s ostalim klasifikacijskim modelima

Sveukupno gledano, CBA daje veliku učinkovitost u usporedbi s klasifikatorima dobro poznatih algoritama s kojima se uspoređivala, a također omogućuje lakšu interpretaciju rezultata koja pomaže sigurnosnim analitičarima i krajnjim korisnicima da lakše razumiju evoluciju zlonamjernih URL-ova.

3.2 Algoritam statičke heuristike

3.2.1 Pregled algoritma statičke heuristike

Za implementaciju detekcije zlonamjernih Web stranica u ovom diplomskom radu fokusira se na metode i algoritme koji nisu temeljeni na strojnom učenju, a za lakšu realizaciju algoritama predlaže se korištenje statičkih metoda analize koje uzimaju u obzir sadržaj stranica kakvog ga vrati poslužitelj, bez uzimanja u obzir dinamičkih promjena na stranici. Kao preporučena alternativa strojnom učenju mogu se koristiti metode koje se temelje na heuristici ili nekim drugim mehanizmima koje je moguće implementirati kroz niz eksplicitno određenih i diskretiziranih koraka provjere. Predložena metoda detekcije temelji se na statičkoj heuristici opisanoj u članku 'Identification of Malicious Web Pages with Static Heuristics' [24].

Klijentski *honeypot* uređaji s visokom interakcijom su sigurnosni uređaji koji mogu detektirati zlonamjerne Web stranice na mreži interaktivnim slanjem određenih zahtjeva poslužitelju i praćenjem odgovora te promjena stanja na klijentu. No njihov je nedostatak što zahtijevaju velike resurse i često mogu propustiti napade jer ih poslužitelji lako mogu detektirati. Stoga je potrebno uvesti metode uočavanja zlonamjernih Web stranica koje uključuju ispitivanje inicijalnih statičkih atributa njihovog HTTP odgovora i HTML koda. Većina zlonamjernih Web stranica može se uočiti tim metodama budući da zlonamjerne Web stranice uvode zlonamjerne fragmente koda s udaljenih Web lokacija te ih skrivaju, a moguća je detekcija statičkih atributa koji upućuju na te akcije. Kombiniranje ovakvih metoda s klijentskim *honeypot* uređajima u hibridne sustave može znatno povećati učinkovitost takvih sustava.

Postoje implementirane obrane protiv tradicionalnih metoda napada pa napadači kontinuirano rade na novim metodama napada za koje ne postoji zaštita, a bitna skupina tih napada su napadi koji se fokusiraju na klijentske aplikacije (engl. *client-side*). Mehanizam napada fokusiranih na klijentske aplikacije temelji se na tome da tijekom pristupa zlonamjernom poslužitelju, on klijentu dostavlja napad kao dio svog odgovora te potom Web poslužitelj pokreće *drive-by-download* napad u internetskom pregledniku. Kad internetski preglednik zatraži sadržaj od poslužitelja, poslužitelj vraća kod koji iskorištava ranjivost, a taj je kod ugrađen u internetsku stranicu te omogućuje kontrolu nad klijentskim sustavom. Tradicionalni mehanizmi obrane nisu učinkoviti protiv takvih prijetnji jer ne pružaju adekvatnu obranu protiv napada koji se temelje na tek otkrivenim ranjivostima (engl. *zero-day attacks*).

Potrebna je detekcija i ispitivanje zlonamjernih internetskih poslužitelja kako bi se razvili novi obrambeni mehanizmi, a to se postiže *honeypot* klijentima visoke interaktivnosti koji koriste ranjive računalne sustave za interakciju s potencijalno zlonamjernim poslužiteljima. Klijentski *honeypot* prima odgovore poslužitelja te ih obrađuje i nadzire bilo kakve neovlaštene promjene u sustavu, odnosno promjene koje nisu logički povezane s pozadinskim sistemskim procesima poput kreiranja logova.

Ipak, postoje brojni izazovi *honeypot* klijentskih aplikacija te nedostaci vezani uz njih. Zahtijevaju značajne računalne resurse kako bi funkcionirali, što negativno utječe na njihovu

učinkovitost. Često ne uspijevaju detektirati zlonamjerne internetske stranice koje zahtijevaju unos s klijentske strane, primjerice putem korisničke interakcije ili sistemskih poziva. Rad klijentskih *honeypot* sustava pretpostavlja da će sam pristup takvoj stranici uzrokovati napad te ako je za realizaciju napada potrebna reakcija korisnika, neće biti neovlaštenih promjena koje se mogu detektirati.

Postoje tri glavna elementa svojstvena za zlonamjerne internetske stranice koje će algoritam statističke heuristike uzimati u obzir: sam napad odnosno iskorištavanje ranjivosti, mehanizam dostave zlonamjernog sadržaja do internetskog preglednika te mehanizmi skrivanja napada ili dostava od mehanizma detekcije, a detaljni pregled danih elemenata prikazan je u tablici 3.3. uz pojašnjenja u daljnjem tekstu.

Kategorija	**Značajke**	**Opis**
Iskorištavanje ranjivosti	Dodaci	Broj applet i object oznaka
	Script oznake	Broj script oznaka
	XML processing instrukcije	Broj XML processing instrukcija, uključujući posebne XML processing instrukcije
Mehanizam dostave ranjivosti	Frame elementi	Broj frame i iframe oznaka, uključujući informaciju o izvoru
	Preusmjeravanja	Pokazatelji preusmjeravanja, uključujući kod odgovora, meta refresh oznake i JavaScript kod
	Script oznake	Broj script oznake, uključujući informacije o izvoru
Skrivanje	Obfuskacija skripti	Funkcije i elementi koji ukazuju na obfuskaciju skripti, uključujući enkodirane nizove znakova i funkcije za dekodiranje
	Frame elementi	Informacije o vidljivosti i veličini iframe oznaka

Tablica 3.3: Atributi relevantni za metodu statičke heuristike

Iskorištavanja ranjivosti (engl. *exploits*) svojstveni su dio zlonamjernih Web stranica koji mora biti prisutan kako bi se stranica uopće smatrala zlonamjernom. To je napadački kod koji cilja na određenu ranjivosti preglednika, njegovih dodataka ili operacijskog sustava na kojem se nalazi. Napad je specifičan za pojedinu ranjivost koju iskorištava te može koristiti različite tehnike. Na primjer, manje očiti napadi često se pronalaze u slikama, a najčešći napadi pogađaju ranjivosti u dodacima poput ActiveX komponenti koje su stoga izbačene iz uporabe u internetskim preglednicima.

Iako je iskorištavanje ranjivosti svojstveni dio internetske stranice, internetska stranica ne mora direktno sadržavati napadački kod, nego se napadački kod može uvesti sa središnjeg poslužitelja kako bi se omogućio mehanizam za ponovno iskoristivi i modularni dizajn napada, a za to je potreban odgovarajući mehanizam dostave ranjivosti (engl. *exploit delivery mechanism*). Postoje dva tipa dostave zlonamjernog koda: direktno uključivanje resursa i

preusmjeravanja. Direktno uključivanje resursa prirodno je podržano HTML standardnom, budući da `src` atribut koji postoji na nekoliko HTML oznaka (engl. *tag*) može uvesti resurse s lokalnih ili udaljenih internetskih poslužitelja. Čak i ako pojedina oznaka ne podupire `src` atribut izravno, skripte mogu učinkovito pribaviti udaljeni resurs i dinamički dodati `src` atribut, budući da mogu po želji modificirati HTML stranicu preko DOM-a (engl. *document object model*) te tako uvesti čitave HTML elemente s udaljenih izvora. Preusmjeravanja se mogu koristiti ako napadač ne uvodi napadački kod direktno, nego daje uputu pregledniku da dohvati novu internetsku stranicu s potpuno druge lokacije. Preusmjeravanja s poslužiteljske strane daju pregledniku upute da dohvati stranicu s druge lokacije putem HTTP koda odgovora `3xx` i polja u zaglavlju naziva `location`, dok preusmjeravanja s klijentske strane to čine putem HTML ili JavaScript koda te se pokreću nakon što je stranica učitana.

U svrhu skrivanja zlonamjernog ponašanja, samo iskorištavanje ranjivosti te mehanizam dostave koda koji iskorištava ranjivost uobičajeno se skrivaju obfuskacijom koja je često svojstvo zlonamjernih Web stranica. Kod skripte dohvaća se u obfusciranom obliku uz proizvoljnu funkciju za deobfuskaciju koja onda može pretvoriti obfuscirani isječak koda u čistu formu koja se potom može izvršiti. Algoritmi detekcije temeljeni na potpisu (engl. *signature-based algorithms*) koji traže karakteristične uzorke koda poznate kao sredstvo iskorištavanja pojedinih ranjivosti ne mogu analizirati obfuscirani JavaScript kod.

Opisani elementi koje algoritam statičke heuristike uzima u obzir mogu imaju i svoje legitimne namjene i biti prisutni i na benignim Web stranicama, ali ih napadači također mogu iskoristiti za zlonamjerne radnje. Iz tog razloga, stranica se ne može uvijek jednostavno klasificirati kao zlonamjerna ili benigna samo na temelju prisutnosti tih elemenata u HTTP odgovoru te je potreban složeniji algoritam za analizu relevantnih značajki.

U daljnjem tekstu opisano je uspostavljanje algoritma statičke heuristike koji će se u konačnici implementirati u okviru diplomskog rada kako je opisano u članku gdje je dani algoritam predstavljen [24].

U listopadu i studenom 2007., nekoliko tisuća potencijalno zlonamjernih Web stranica ispitano je korištenje *honeypot* klijenta visoke interaktivnosti Capture-HPC v2.1 [25] konfiguriranog s Windows XP SP2 operacijskim sustavom koristeći preglednik Internet Explorer 6 SP2 te je pritom bilježen mrežni promet i pohranjen je odgovarajući HTTP odgovor za stranice koje su detektirane i označene kao zlonamjerne. Na sličan je način zabilježen HTTP odgovor prilikom interakcije s zlonamjernim internetskim stranicama koristeći isti sustav tako da su engleski 5-grami nasumično izabrani iz kolekcije internetskih stranica povezanih s DMOZ *Open Directory Project* [26] te su dani Yahoo internetskom pregledniku nakon čega je provjereno prvih 50 URL-ova danih rezultata pretrage, a stranice koje nisu klasificirane kao zlonamjerne označene su kao benigne i odgovarajući HTTP odgovor je pohranjen.

Nakon prikupljanja tih HTTP odgovora, izdvojeni su atributi HTTP odgovora i ugrađenog HTML koda putem kojih bi se mogli uočiti karakteristični elementi zlonamjernih Web stranica, odnosno ti atributi uključuju karakteristike koji bi mogle ukazivati na potencijalna iskorištavanja ranjivosti, mehanizme dostave napadačkog koda te pokušaje obsfuskacije.

Iz svake skupine elemenata svojstvenih za zlonamjerne Web stranice mogu se izdvojiti relevantni atributi koji mogu ukazivati na te elemente. Atributi relevantni za iskorištavanje

ranjivosti su dodaci (engl. *plugin*) - broj applet, object i embed oznaka, `script` oznake - broj script oznaka te XML processing instrukcije - broj XML processing instrukcija kao što je `xml-stylesheet`. Atributi relevantni za mehanizam dostave napadačkog koda su okviri - broj `frame` i `iframe` oznaka, uključujući informacije o izvoru, preusmjeravanja - pokazatelji preusmjeravanja, što uključuje kod odgovora, `meta` oznake za preusmjeravanje i JavaScript kod te `script` oznake - broj script oznaka, uključujući informacije o izvoru. Atributi koji se odnose na kategoriju skrivanja iskorištavanja ranjivosti i dostave napadačkog koda su atributi vezani za obfuskaciju skripti, odnosno funkcije i elementi koji ukazuju na samu obfuskaciju, poput enkodiranih vrijednosti nizova znakova i funkcija za dekodiranje.

Atributi koji su izdvojeni iz prikupljenog skupa stranica ubačeni su u implementaciju algoritma učenja J4.8 stabla odlučivanja realiziranu putem Weka biblioteke za strojno učenje Waikato sveučilišta [19], a prediktivna vrijednost generiranog klasifikatora bila je generirana na novim stranicama koje nisu bile korištene u fazi učenja.

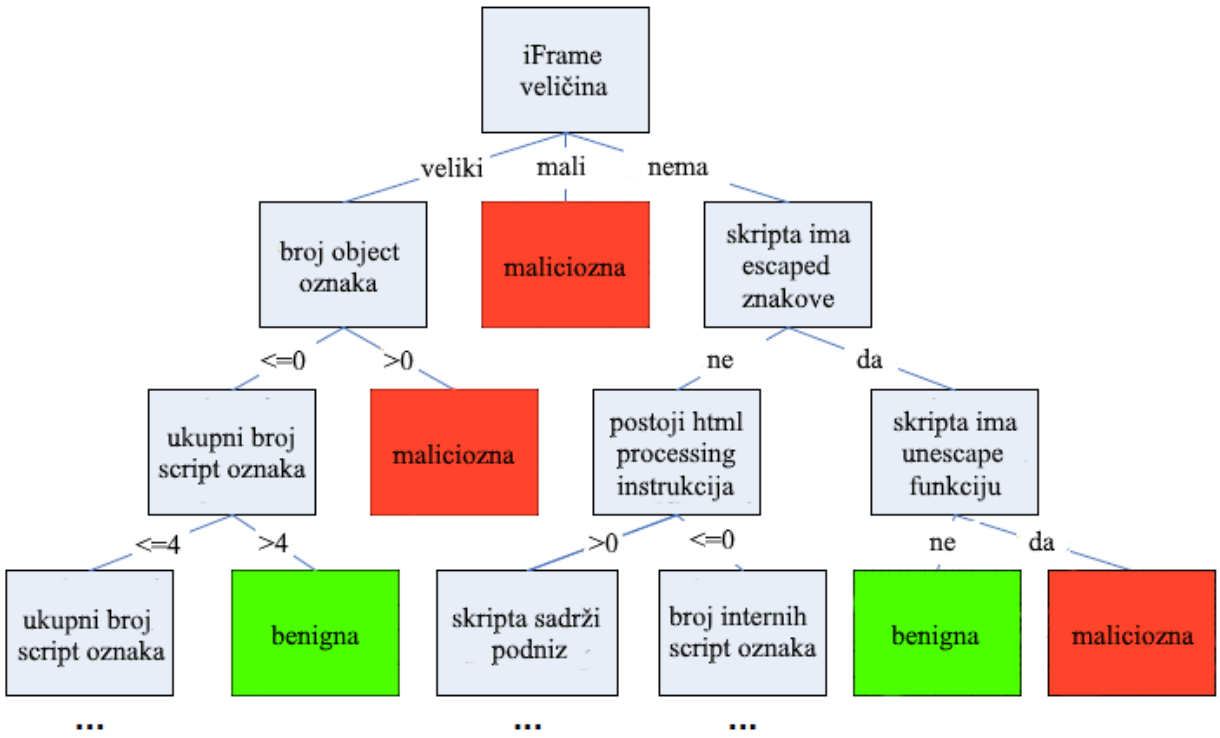
Najprije je skup od 61 000 tisuće URL-ova nasumično odabranih korištenjem 5-grama korišten za procjenu učinkovitosti metode, pri čemu su svi URL-ovi klasificirani metodom temeljenom na statičkoj heuristici, a potom korištenjem *honeypot* klijenta visoke interaktivnosti Capture-HPC v2.1 te su određeni udjeli lažno pozitivni i lažno negativnih rezultata u usporedbi s rezultatima koje je dao *honeypot* klijent.

Predložena klasifikacijska metoda također je ispitana koristeći skup od 500 000 URL-ova koje je pružio pružatelj usluga za zaštitu od internetskih prijetnji HauteSecure koji su već analizirani te su u njima otkriveni *drive-by-download* napadi, pri čemu je korištenje ovih URL-ova smanjilo pristranost uzrokovanu tehnologijom *honeypot* klijenta i nasumičnom metodom odabira URL-a.

Uzorak od 61 000 URL-a također je korišten kako bi se procijenilo poboljšanje performansi dobiveno ovim pristupom koristeći Amazon EC2 instancu s 1.7 GB RAM-a te je trajanje klasifikacije uspoređeno s trajanjem uz korištenje *honeypot* klijenata na drugom testnom računalu sa sličnim performansama, uz korištenje podijeli i pobjedi (engl. *divide-and-conquer*) algoritma, vrijeme usluge od približno 10 sekundi i mogućnost za istodobno pokretanje triju *honeypot* klijenata.

Za opisano ispitivanje najprije je 5 678 zlonamjernih i 16 006 benignih Web stranica ubačeno u algoritam strojnog učenja te je generirani klasifikator korišten kako bi se generirao novi uzorak. Opisana metoda klasifikacije iz skupa od 61000 URL-ova detektirala je 3590 URL-ova kao zlonamjerne te je nakon usporedbe s rezultatima *honeypot* klijenta visoke interaktivnosti ustanovljen postotak lažno pozitivne detekcije od 5.88% i lažne negativne detekcije od 46.15% za novu klasifikacijsku metodu, a skup od 500 000 URL-ova dao je slične rezultate.

Nakon što su stranice bile ubačene u algoritam strojnog učenja, analizira se stablo odlučivanja generirano bibliotekom za strojno učenje koje će se koristiti za analizu, odnosno klasifikaciju nepoznatih Web stranica. Stablo odlučivanja koje je generirano tim algoritmom prikazano je na slici 3.2, a bit će pojašnjeno u nastavku teksta.



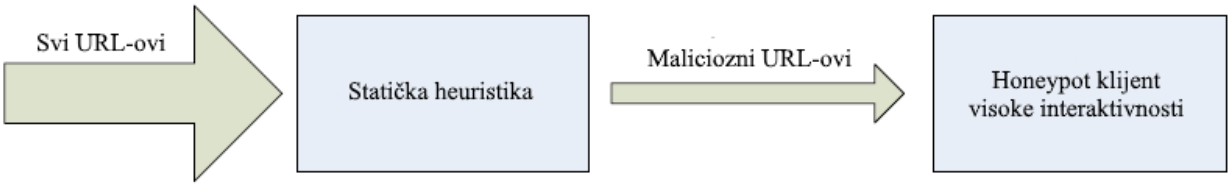
Slika 3.2: Stablo odlučivanja algoritma statičke heuristike

Za razumijevanje značenja stabla odlučivanja najprije je potrebno pojasniti njegovu notaciju, odnosno način kako se ono ispravno interpretira. Počevši od korijenskog čvora, vrijednost atributa prikazanog na čvoru stabla se evaluira, što dovodi do određenog čvora djeteta. Nakon toga se atributi rekurzivno evaluiraju dok se ne dođe do čvora odluke, u ovom slučaju čvora koji pokazuje da je stranica benigna, odnosno zlonamjerna. U analiziranom znanstvenom članku dano je samo djelomično stablo odlučivanja kojeg je generirao algoritam strojnog učenja, a ono će biti prošireno u konkretnoj implementaciji dodavanjem novih grana odlučivanja.

Postojanje `iframe` elemenata dobar je klasifikator zlonamjernosti internetskih stranica. Ako postoji `iframe` koji je malen, to je pokazatelj zlonamjernosti Web stranice. Ako `iframe` element ne postoji, provjeravaju se drugi atributi, primjerice prisutnost *escaped* znakova u JavaScript kodu koji su predstavljeni heksadekadskim ASCII kodom znaka s prefiksom %, što može ukazivati na enkodirane znakove i postojanje `unescape()` funkcije koja se najčešće koristi za deobfuskaciju. Istodobna prisutnost *escaped* znakova i `unescape()` funkcije dovodi do odluke da se radi o zlonamjernoj Web stranici, budući da to direktno ukazuje na postojanje mehanizma dostave napadačkog koda i obfuskacije. Ako uz nepostojanje `iframe` elemenata skripta ima znakove koji su *escaped*, ali nema `unescape()` funkcije, stranica je vjerojatno benigna. Ako postoji veliki `iframe` element, a pritom postoje `object` oznake na stranici, to je dobar pokazatelj da je stranica zlonamjerna, no ako `object` oznake ne postoje, a postoji više od 4 `script` oznake, stranica se može smatrati benignom.

Učinkovitost klasifikacije HTTP odgovora ovom metodom klasifikacije značajno je veća od učinkovitosti pri korištenju tradicionalnog *honeypot* klijenta visoke interaktivnosti. Tijekom 49 minuta ovom metodom statičke heuristike uspjelo se dohvatiti i analizirati 61 000 URL-ova, dok se klijentskom *honeypot* tehnologijom u istom razdoblju uspjelo analizirati samo 996 URL-ova, što ukazuje na 61 puta veću učinkovitost algoritma statičke heuristike.

Iako prezentirana metoda klasifikacije pokazuje bolju učinkovitost u odnosu na klijentsku *honeypot* tehnologiju, producira određen broj lažno pozitivnih rezultata i propuštenih detekcija, odnosno lažno negativnih rezultata pa je uputno korištenje hibridnih sustava koji će kombinirati ovu metodu s *honeypot* klijentskom tehnologijom. Kako bi predstavljena metoda bila i učinkovita i pouzdana, mora biti korištena u sprezi s *honeypot* klijentom visoke interakcije. Najprije se stranice zadane URL-ovima povlače i klasificiraju opisanim metodama statičke heuristike, a budući da je postotak lažno pozitivnih rezultata relativno visok, svi URL-ovi koji su klasificirani kao zlonamjerni prosljeđuju se drugom dijelu hibridnog sustava - *honeypot* klijentu koji donosi konačnu odluku, kako je prikazano na slici 3.3. Tehnologija *honeypot* klijenta visoke interaktivnosti ima nultu stopu lažno pozitivne detekcije pa su time lažno pozitivni rezultati koji su imali stopu od 5.88% eliminirani.



Slika 3.3 Hibridni sustav statičke heuristike i honeypot klijenta

Brzina detekcije hibridnog sustava pod velikim je utjecajem brzina njegovih individualnih komponenti. Budući da je klijentski *honeypot* visoke interakcije znatno sporiji od metode statičke heuristike, drugi će pregled stranice imati veliki utjecaj na učinkovitost hibridnog sustava, dakle glavni čimbenik koji utječe na brzinu jest broj internetskih stranica koje su metodom statičke heuristike klasificirane kao zlonamjerne i proslijeđene *honeypot* klijentu u svrhu konačne odluke.

Broj zlonamjernih klasifikacija prezentirane klasifikacijske metode može se izračunati na sljedeći način:

Ako je p_m postotak zlonamjernih stranica u skupu N koji se ispituje, broj zlonamjernih stranica N_M može se izračunati formulom 3.10.

$$N_M = p_m * N$$

Formula 3.10: Izračun broja zlonamjernih stranica

Broj benignih stranica može se izračunati formulom 3.11.

$N_B = N - N_M$

Formula 3.11: Izračun broja benignih stranica

Uzimajući u obzir lažno pozitivnu i lažno negativnu stopu rezultata navedene metode FP_C, odnosno FN_C, broj zlonamjernih klasifikacija $alerts_c$ može se izračunati formulom 3.12.

$alerts_c = FP_C * N_B + (1 - FN_C) * N_M$

Formula 3.12: Izračun broja zlonamjernih klasifikacija

Postotak zlonamjernih klasifikacija može izraziti formulom 3.13.

$P_{alertc} = alerts_c/N$

Formula 3.13: Izračun postotak zlonamjernih klasifikacija

Zbrajanjem vremena za ispitivanje skupa od N stranica prezentiranom metodom statičke heuristike i vremena da se ponovno ispita $alerts_c$ stranica *honeypot* klijentom visoke interaktivnosti dobiva se ukupno vrijeme hibridnog sustava za ispitivanje N stranica TH_y, a to se može izraziti formulom 3.14.

$T_{Hy} = T_C * N + alerts_c * T_H$

Formula 3.14: Izračun ukupnog vremena analize hibridnog sustava

Pritom T_C predstavlja vrijeme obrade po stranici za predstavljenu metodu klasifikacije, a T_H vrijeme obrade po stranici koristeći *honeypot* klijent.

Implementacija hibridnog sustava s vremenima obrade T_C od 0.05 sekundi i T_H od 2.95 sekundi klasificiralo je uzorak od 61 000 URL-ova s 3590 zlonamjernih klasifikacija za 227 minuta što je oko 13 puta kraće od metode koju bi koristi *honeypot* klijent kojem bi za to bilo potrebno oko 2999 minuta.

Što se tiče procjene točnosti detekcije ovih hibridnim sustavom, budući da *honeypot* klijent ima postotak lažno pozitivnih detekcija nula, ovako konstruiran sustav također ima postotak lažno pozitivnih detekcija nula, no neke zlonamjerne stranice mogu ostati neotkrivene te su neki napadi propušteni, uzimajući u obzir da je stopa lažno negativnih rezultata statičke heuristike čak 46.15%.

Budući da je samo $alerts_c$ stranica proslijeđeno *honeypot* klijentu, stopa lažno negativnih rezultata hibridnog sustava FNH_y identična je uniji stope lažno negativnih rezultata

predstavljene klasifikacijske metode FN_C i stope lažno negativnih rezultata *honeypot* klijenta FN_H, što je prikazano formulom 3.15.

$$FN_{Hy} = FN_C \cup FN_H = FN_C + FN_H - (FN_C, FN_H)$$

Formula 3.15: Izračun stope lažno negativnih rezultata hibridnog sustava

Na primjer, ako skup URL-ova koji je analiziran hibridnom metodom ima postotak negativnih rezultata 10.0% za *honeypot* klijent, a kombinirano metode propuste otkrivanje 5.00% svih napada, rezultantni postotak lažnih rezultata hibridnog sustava je 46.15% + 10.0% - 5% = 51.15%. Ako se pretpostavi da *honeypot* klijent ima postotak lažno negativnih rezultata nula, tada stopa lažno negativnih rezultata hibridnog sustava pada na lažno negativnu stopu predstavljene metode, koja iznosi 46.15%.

Budući da je postotak lažno negativnih rezultata od 46.15% neprihvatljiv za mnoge slučajeve korištenja, on se može poboljšati u hibridnom sustavu ako se dio URL-ova koje metoda nije otkrila kao zlonamjerne nasumično proslijedi *honeypot* klijentu. Određen postotak URL-ova $p_{noAlertL}$ koji nisu detektirani metodom koja koristi statičku heuristiku prosljeđuje se *honeypot* klijentu te imaju priliku biti ispravno klasificirani *honeypot* klijentom i pritom postotak lažno negativnih rezultata opada i može se izračunati formulom 3.16.

$$FN_{Hy} = FN_C \cup FN_H = FN_C + FN_H - (FN_C, FN_H) - (p_{noAlertL} * FN_C)$$

Formula 3.16: Izračun smanjene stope lažno negativnih rezultata hibridnog sustava

Primjerice, ako se 20% URL-ova koje metoda statičke heuristike nije detektirala kao zlonamjerne proslijedi *honeypot* klijentu, uz pretpostavku da je njegov postotak lažno negativnih detekcija nula, postotak lažno negativne detekcije hibridnog sustava bit će smanjen s 46.15% na približno 36.92%.

Prosljeđivanje određenog broja URL-ova koji nisu detektirani kao zlonamjerni *honeypot* klijentu visoke interakcije imat će učinak na brzinu sustava, dakle postoji kompromis između točnosti detekcije i poboljšanja učinkovitosti jer kako se $p_{noAlertL}$ povećava, postotak lažno negativnih rezultata opada, ali smanjuje se i brzina sustava, odnosno pogoršavaju performanse.

U ovom diplomskom radu neće se implementirati predloženi hibridni sustav, nego samo metoda statičke heuristike koja čini njegov prvi dio, tako da se očekuje stopa lažno pozitivnih rezultata približno jednaka stopi od 5.88% navedenoj u analiziranom članku, no predložit će se i provesti optimizacije koje će imati za cilj smanjiti navedenu stopu.

3.2.2 Implementacija algoritma statičke heuristike

Potrebno je implementirati izdvajanje atributa koji su relevantni za detekciju zlonamjernih stranica u skladu s generiranim stablom odlučivanja za metodu koja se temelji na statičkoj heuristici, a potom implementirati samo stablo odlučivanja koje dovodi do odluke o tome je li stranica benigna ili zlonamjerna te daje dijagnozu zlonamjernosti ako je stranica zlonamjerna.

Kako bi se mogla analizirati struktura HTML dokumenta u Pythonu, potrebno je koristiti određene gotove biblioteke koje će statički HTML niz znakova pretvoriti u DOM objekt radi lakše analize postojanja određenih HTML oznaka te njihovih atributa i sadržaja koje su bitne za donošenje odluke o tome je li stranica zlonamjerna ili benigna.

Za početak će se implementirati skraćena verzija stabla odlučivanja prikazana u članku koji se bavi detekcijom zlonamjernih stranica statičkom heuristikom [24] koja uključuje najbolje pokazatelje zlonamjernosti Web stranica. Time bi se trebala obuhvatiti većina implementacije koja je na teorijskom nivou opisana u analiziranom članku, odnosno za nepoznate ishode stabla odlučivanja gdje se ono prekida pretpostavit će se da stranica nije zlonamjerna. Nakon toga će se stablo odlučivanja proširiti te implementirati provjera dodatnih atributa iz relevantnih kategorija zlonamjernosti kako bi detekcija bila točnija i obuhvatila veći broj potencijalno zlonamjernih stranica.

Dodatna pomoćna biblioteka koja se može koristiti za dobivanje podataka o HTML kodu je Beautiful Soup biblioteka [27] kojom se može konstruirati objekt iz HTML niza znakova na kojem se onda rade određeni upiti. Kako bi se funkcionalnosti tog modula mogle koristiti, potrebno ga je u PowerShell ljusci instalirati naredbom prikazanom u ispisu 3.3, a unutar Python programa uvesti naredbom prikazanom u ispisu 3.4.

```
pip install beautifulsoup4
```

Ispis 3.3: Instalacija Beautiful Soup Python biblioteke

```
from bs4 import BeautifulSoup
```

Ispis 3.4: Umetanje Beautiful Soup dodatka u Python program

U svrhu analize HTML dokumenta za realizaciju ovog algoritma kreirat će se poseban modul `htmlhandler` koji će se koristiti za izdvajanje određenih atributa ili brojenje određenih elemenata direktno iz HTML niza znakova.

Pretvaranje HTML niza znakova u kompaktni objekt s kojim se može jednostavnije raditi putem Beautiful Soup biblioteke može se realizirati Python funkcijom prikazanom u ispisu 3.5.

```
def makeBeautifulSoup(html):
    soup = BeautifulSoup(html, 'html.parser')
    return soup
```

Ispis 3.5: Pretvaranje HTML niza znakova u Beautiful Soup objekt

Najprije je cilj napraviti funkcije koje će pobrojiti postojanje `iframe` oznaka [28], `script` oznaka [29], `object` oznaka [30] i `embed` oznaka [31], budući da se njihovo postojanje koristi kao bitan pokazatelj zlonamjernosti Web stranice u prikazanom stablu odlučivanja.

Brojanje pojedinih oznaka koristeći Beautiful Soup biblioteku može se implementirati odgovarajućim funkcijama korištenjem `find_all` funkcije nad Beautiful Soup objektom koja kreira liste određenih elemenata unutar HTML koda. Budući da `find_all` vraća objekt tipa `ResultSet`, on se treba predati funkciji `len` kako bi se dobio broj elemenata u njemu.

Funkcije za brojenje `script`, `object`, `iframe` i `embed` oznaka implementirat će se kako je prikazano u ispisu 3.6.

```
def countScriptTags(soup):
    return len(soup.find_all('script'))

def countObjectTags(soup):
    return len(soup.find_all('object'))

def countIFrameTags(soup):
    return len(soup.find_all('iframe'))

def countEmbedTags(soup):
    return len(soup.find_all('embed'))
```

Ispis 3.6: Brojenje script, object, iframe i embed oznaka

Broj XML processing instrukcija [32] također može biti relevantan za određivanje zlonamjernosti i potrebno je izbrojiti koliko takvih instrukcija ima, no korištenje Beautiful Soup objekta nije praktično za rad s takvim instrukcijama, budući da se ne radi o elementima koji su klasični čvorovi u DOM-u nego o instrukcijama koje diktiraju kako će se XML kod umetnut u dokument Web stranice transformirati u vizualnu reprezentaciju podataka, a te su oznake formata prikazanog u ispisu 3.7.

```
<?xml-stylesheet href="style.css" type="text/css"?>
```

Ispis 3.7: Format XML processing instrukcije

Prisustvo XML struktura u Web aplikaciji općenito može povećati rizik od napada ubacivanjem XML-a (engl. *XML injection attack*) [33] kojim je moguće manipulirati ili zlouporabiti logiku XML aplikacije ili servisa, odnosno bilo kojeg XML sadržaja kojeg aplikacija generira, budući da ubacivanje neprimjerenog XML sadržaja ili strukture u XML poruku može promijeniti logiku čitave aplikacije ili omogućiti uključivanje dodatnog zlonamjernog sadržaja.

Za određivanje broja pojavljivanja xml instrukcija moguće je tražiti broj pojavljivanje podniza znakova `'<?xml-stylesheet'` u nizu znakova koji predstavlja HTML i to se implementira funkcijom kojoj se direktno predaje HTML niz znakova prikazanom u ispisu 3.8.

```
def countXMLProcessingInstructions(html):
    xmlInstructionSubstring = '<?xml-stylesheet';
    numberOfInstructions = html.count(xmlInstructionSubstring)
    return numberOfInstructions
```

Ispis 3.8: Brojenje XML processing instrukcija

Dodatno je uz funkciju brojenja XML processing instrukciju korisno dodati provjeru da li postoje *inline* skripte koje dinamički ubacuju XML čvorove u stranicu jer bi oni mogli biti zlonamjerni u kombinaciji s postojećim XML instrukcijama. To se postiže tako da se unutar skripte traže uzorci koji ukazuju na generiranje XML čvora, uzimajući u obzir bitne primjere generiranja XML-a unutar JavaScript koda [34].

Kreiranje novog XML DOM objekta vrši se funkcijom prikazanom u ispisu 3.9.

```
document.implementation.createDocument
```

Ispis 3.9: Kreiranje XML DOM objekta

XML datoteka može također se može dohvaćati putem objekta klase `XMLHttpRequest`, kako je prikazano u ispisu 3.10.

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (xhttp.readyState == 4 && xhttp.status == 200) {

    var xmlDoc = xhttp.responseXML;
}
xhttp.open("GET", "example.xml", true);
xhttp.send();
```

Ispis 3.10: Dohvaćanje XML datoteke putem objekta klase XMLHttpRequest

Moguće je i parsirati niz znakova koji sadrži serijalizirani XML, kako je prikazano u ispisu 3.11.

```
var xmlString = "<root></root>";
var parser = new DOMParser();
var xmlDoc = parser.parseFromString(xmlString, "text/xml");
```

Ispis 3.11: Parsiranje niza znakova koji sadrži serijalizirani XML

Iz navedenih karakterističnih primjera dohvaćanja XML-a putem JavaScripta, mogu se odrediti karakteristični nizovi znakova koje ako JavaScript sadrži vrlo vjerojatno ubacuje, dohvaća ili generira XML kojeg lako može dodati u dokument stranice. Smatrat će se da JavaScript kod generira XML na stranici ako se pojavljuju nizovi `'document.implementation.createDocument'`, `'XMLHttpRequest'` i `'responseXML'` ili ako se pojavljuju nizovi `'DOMParser'` i `'text/xml'`.

Navedena će se provjera implementirati kako je prikazano u ispisu 3.12.

```
def scriptsInjectXML(soup):
    webpageScripts = soup.find_all('script');
    for script in webpageScripts:
        scriptText = ''.join(script.contents)
        if scriptText != '':
```

```
            if ('document.implementation.createDocument' in scriptText)
or ('XMLHttpRequest' in scriptText and 'responseXML' in scriptText) or
('DOMParser' in scriptText and 'text/xml' in scriptText):

                return True

    return False
```

Ispis 3.12: Detekcija ubacivanja XML čvorova

Osim brojenja HTML oznaka, za implementaciju algoritma statičke heuristike ključna je i njihova klasifikacija. Za skripte, koje su predstavljene `script` oznakama, bit će korisno ustanoviti radi li se o *inline* skriptama unutar kojih su direktno specificirane JavaScript naredbe, unutarnjim skriptama koje putem `src` atributa učitavaju JavaScript datoteke s iste domene ili vanjskim skriptama koje putem `src` atributa učitavaju JavaScript datoteke s druge vanjske domene. Za `iframe`, `embed` i `object` oznake također će biti korisno ustanoviti radi li se o unutarnjim dokumentima koji se učitavaju s iste domene odnosno lokalne putanje ili o vanjskim dokumentima koji se učitavaju s druge vanjske domene.

Navedena klasifikacija postiže se provjerom `src` atributa objekta koji se promatra, s tim da kod script oznake `src` atribut ne mora postojati pa se smatra da se radi o *inline* skripti ako ne postoji. Budući da korištenjem `find_all` upita Beautiful Soup vraća `ResultSet` objekt po kojem se može iterirati, moguće je za svaki pojedini element provjeriti *`src`* atribut i na temelju njega donijeti odluku o kojem se tipu oznaka radi. Implementacija brojenja *inline*, unutarnjih i vanjskih skripti može se realizirati tako da se kreira objekt rječnika koji sadrži broj svake pojedine vrste skripti, a onda se specijaliziranim funkcijama iz tog objekta izvlači vrijednost za vrstu čiji se broj želi dobiti. Time se izbjegava ponavljanje koda i višestruka izvršavanja pregleda skupa skripti detektiranih na stranici. Opcionalno se funkcijama koje određuju broj unutarnjih, odnosno vanjskih skripti kao parametar može dati domena stranice ako je poznata domena na kojoj se učitani HTML nalazi kako bi se primjerice na domeni 'example.com' i uključivanje s apsolutnom putanjom prikazano u ispisu 3.13 i uključivanje s relativnom putanjom prikazano u ispisu 3.14 ispravno detektirali kao uključivanje unutarnje skripte, a ako se domena ne specificira, onda će se kao interna skripta u obzir uzeti samo ona `script` oznaka čiji je `src` atribut oblika prikazanog u ispisu 3.14.

```
<script src= 'https://example.com/script.js'>
```

Ispis 3.13: Uključivanje lokalne skripte s apsolutnom putanjom

```
<script src= 'script.js'></script>
```

Ispis 3.14: Uključivanje lokalne skripte s relativnom putanjom

To je moguće postići uvođenjem pretpostavljenog argumenta domene koji će ako se ne specificira imati vrijednost `None` [35].

Kako bi se mogla odrediti domena `src` atributa, dobro je koristiti `urllib.parse` biblioteku kojom će se odrediti *host*, odnosno domena URL-a koja se uvozi naredbom prikazanom u ispisu 3.15.

```
import urllib.parse
```

Ispis 3.15: Uvoz urlib.parse biblioteke u Python kod

Generiranje objekta rječnika u kojeg se pohranjuje broj pojedinih vrsti skripti na nekoj HTML stranici realizira se funkcijom prikazanom u ispisu 3.16.

```
def classifyScripts(soup, domain = None):
    scriptsByType = {'inline' : 0, 'internal' : 0, 'external' : 0}
    webpageScripts = soup.find_all('script');
    for script in webpageScripts:
        scriptSource = script.get('src')
        if scriptSource is None:
            scriptsByType['inline'] += 1
        else:
            scriptOnLocalPath = isPathLocal(scriptSource, domain)
            if scriptOnLocalPath:
                scriptsByType['internal'] += 1
            else:
                scriptsByType['external'] += 1
    return scriptsByType
```

Ispis 3.16: Klasifikacija skripti na HTML stranici

Određivanje broja pojedinih vrsta `script` oznaka realizira se funkcijama prikazanima u ispisu 3.17, redom za *inline*, unutarnje i vanjske skripte.

```
def countInlineScripts(soup):
    scriptsByType = classifyScripts(soup)
    inlineScripts = scriptsByType['inline']
    return inlineScripts
```

```
def countInternalScripts(soup, domain = None):
    …
    return internalScripts


def countExternalScripts(soup, domain = None):
    …
    return externalScripts
```

Ispis 3.17: Određivanje broja inline, unutarnjih i vanjskih skripti

Na sličan način kao što je to učinjeno za skripte, klasificira se broj unutarnjih, odnosno vanjskih `iframe`, `object` i `embed` oznaka, s tim da bi one morale imati `src` atribut i *inline* kategorija ovdje ne postoji.

Kako bi se izbjeglo ponavljanje koda, implementirat će se slična funkcija onoj `classifyScripts` koja će se nazivati `classifyCustomTags`, samo što će ona primati dodatni argument koji će predstavljati oznaku tagova nad kojima se gradi objekt rječnika s ključevima `'internal'` i `'external'`. Pritom funkcija mora uzeti u obzir za koju se oznaku radi provjera i u skladu s time provjeravati određeni atribut, budući da `iframe` i `embed` oznake imaju `src` atribut, a `object` oznaka ima `data` atribut. Implementacije je prikazana na ispisu 3.18.

```
def classifyCustomTags(soup, tag, domain = None):
    if tag == 'embed':
        attribute = 'data'
    else:
        attribute = 'src'
    …
    for element in webpageElements:
        elementSource = element.get(attribute)
        …
    return elementsByType
```

Ispis 3.18: Klasifikacija različitih vrsta HTML oznaka

Određivanje broja vanjskih i unutarnjih `object`, `embed` i `iframe` elemenata implementira se na sljedeći način koristeći ranije implementiranu funkciju `classifyCustomTags`, kako je prikazano u ispisu 3.19.

```
def countInternalObjects(soup, domain = None):
    objectsByType = classifyCustomTags(soup, 'object', domain)
    internalObjects    = objectsByType['internal']
    return internalObjects

def countExternalObjects(soup, domain = None):
    …
    return externalObjects

def countInternalEmbeds(soup, domain = None):
    …
    return internalEmbeds
…
def countInternalIFrames(soup, domain = None):
    …
    return internalIFrames
…
```

Ispis 3.19: Brojenje vanjskih i unutarnjih object, embed i iframe elemenata

Određivanje je li putanja dokumenta lokalna, odnosno je li s iste domene koja je učitava, čime se razlikuju unutarnji od vanjskih dokumenata, realizira se sljedećom funkcijom kojoj se može ili ne mora predati domena, pri čemu se inačice domena s i bez `'www.'` prefiksa smatraju identičnima. Provjera je li putanja dokumenta lokalna prikazana je u ispisu 3.20.

```
def isPathLocal(scriptSource, domain = None):
    parsedUrl =
    urllib.parse.urlparse(scriptSource.replace('[','').replace(']',''))
    scriptDomain = parsedUrl.netloc
```

```
    if not scriptDomain or domain is not None and
    scriptDomain.replace('www.','') == domain.replace('www.',''):
        return True
    else:
        return False
```

Ispis 3.20: Provjera je li putanja dokumenta lokalna

Za `iframe` objekte potrebno je odrediti postoje li mali `iframe` objekti, primjerice oni kojima je visina ili širina postavljena na 0, koji se time sakrivaju od korisnika i po mogućnosti sa sobom nose zlonamjerni kod. Kako bi implementacija heuristike bila što fleksibilnija, odnosno kako bi se uz minimalni broj provjera otkrilo najprije postoje li `iframe` objekti uopće, a potom postoje li mali `iframe` objekti, implementirat će se brojenje objekata unutar iste funkcije naziva `searchIFrames` kako je prikazano u ispisu 3.21, i to uz ključeve `'small'`, `'large'` i `'total'`, pri čemu će se provjeravati vrijednost atributa `width` i `height`. Pritom treba uzeti u obzir da će veličina možda biti specificirana uz jedinice, primjerice kao `'0px'`, `'0%'` ili `'0em'` pa se jedinice trebaju zanemariti u provjeri kako bi se detektirali svi mali `iframe` objekti:

```
def searchIFrames(soup):
    …
    iFramesByType['total'] = len(webpageIFrames)
    for iFrame in webpageIFrames:
        width = iFrame.get('width')
        height = iFrame.get('height')
        if (width is not None):
            iFrameWidth = re.sub("[^0-9]", "", width)
        else:
            iFrameWidth = None
        if (height is not None):
        …
        if iFrameWidth == '0' or iFrameHeight == '0':
            iFramesByType['small'] += 1
        else:
            iFramesByType['large'] += 1
```

```
    return iFramesByType
```

Ispis 3.21: Klasifikacija iframe objekata po veličini

Navedena se funkcija koristi unutar funkcija koje broje koliko ima malih `iframe` elementa, koliko ima velikih `iframe` elemenata te koliko ukupno ima `iframe` elemenata, kako je prikazano u ispisu 3.22.

```
def countTotalIFrames(soup):
    detectedIFrames = searchIFrames(soup)
    totalIFrames = detectedIFrames['total']
    return totalIFrames


def countSmallIFrames(soup):
    …
    return smallIFrames


def countLargeIFrames(soup):
    …
    return largeIFrames
```

Ispis 3.22: Određivanje broja svih, malih i velikih iframe elemenata

Potrebno je implementirati i funkcije koje se bave detekcijom obfuskacije JavaScript koda. Otkrivanje postojanja *escaped* znakova unutar JavaScript koda uz istovremeno postojanje `unescape()` funkcije ukazuje na postojanje obfusciranog JavaScript koda koji skriva zlonamjernost od metoda detekcije koje se temelje na potpisu i zato je bitno odrediti koliko u kodu postoji skripti koje sadrže takve znakove, a da istovremeno sadrže i `unescape()` funkciju.

Osim pojedinih atributa otkrivenih elemenata, Beautiful Soup biblioteka može detektirati i sadržaj pojedinih elemenata odnosno podatke u njima, što je potrebno kako bi se analizirao sadržaj JavaScript koda i došlo do podataka o tome postoje li *escaped* znakovi i `unescape()` funkcija. Podacima pohranjenima unutar pojedine skripte pristupa se putem *contents* atributa, a *find_all* funkcija omogućuje pronalaženje svih skripti. Podatke o tome koliko skripti unutar stranice sadrži *escaped* znakove, koliko ih sadrži `unescape()` funkciju, a koliko ih sadrži i jedno i drugo može se dobiti analizom sadržaja skripti, što će se implementirati unutar iste

funkcije. Pristup sadržaju skripte obavlja se tako da se iz elementa skripte izvuče atribut `contents` koji će dati listu sadržaja skripte i onda se ti svi sadržaji spoje funkcijom `join`. Očekuje se da će biti jedan element u listi ili nijedan ako se radi o skripti koja nije *inline*. Ako skripta nema teksta, sadržaj koji se izvuče bit će prazan niz znakova, a samo one skripte čiji niz znakova nije prazan uzet će se u obzir pri analizi pojavljivanja obfuskacije. Unutar funkcija prisutnost `unescape()` funkcije u JavaScript kodu traži se pronalaskom postojanja 'unescape' niza znakova, a za *escaped* znakove koristi se regularni izraz koji traži dvoznamenkaste heksadekadske brojeve ispred kojih je znak '%'.

Navedena se logika implementira funkcijom prikazanom u ispisu 3.23.

```
def detectObfuscation(soup):
    …
    for script in webpageScripts:
        …
        scriptText = ''.join(script.contents)
        if scriptText != '':
            if scriptText.find('unescape(') != -1:
                scriptsByObfuscation['unescape_function'] += 1
                escapedFound = True
            if re.search('%[0-9a-f]{2}',scriptText):
                scriptsByObfuscation['escaped_characters'] += 1
                unescapeFound = True
            if escapedFound and unescapeFound:
                scriptsByObfuscation['obfuscated'] += 1
    return scriptsByObfuscation
```

Ispis 3.23: Detekcija obfuskacije JavaScript koda

Funkcije koje koriste ovu glavnu funkciju za određivanje broja skripti s *escaped* znakovima, odnosno `unescape()` funkcijom implementiraju se kako je prikazano u ispisu 3.24, pri čemu je najbitnija funkcija `countObfuscatedScripts`:

```
def countObfuscatedScripts(soup):
    detectedScripts = detectObfuscation(soup)
    obfuscatedScripts = detectedScripts['obfuscated']
```

```
    return obfuscatedScripts


def countEscapedCharactersScripts(soup):
    …
    return escapedCharactersScripts


def countUnescapeFunctionScripts(soup):
    …
    return unescapeFunctionScripts
```

Ispis 3.24: Brojenje obfusciranih skripti

Budući da je nekontrolirano preusmjeravanje jedan od bitnih pokazatelja zlonamjernosti, bitna je detekcija koda koji uzrokuje preusmjeravanje. Uz izravno uključivanje zlonamjernih resursa moguće je uključivanje putem preusmjeravanja koje može biti realizirano na različite načine. Implementirat će se algoritmi koji uočavaju preusmjeravanje unutar HTML koda ili JavaScript koda, a preusmjeravanje koje bi eventualno bilo uključeno u kod odgovara stranice neće biti implementirano budući da se ispitivanje zlonamjernih stranica vrši nad skupom statičkih HTML stranica i bilo kakav dinamički tok poput daljnjeg preusmjeravanja na druge stranice ne bi bio praćen te se uzima u obzir samo HTML sadržaj.

Jedan on načina na koji je moguće postići preusmjeravanje jest postojanje `meta` oznaka za preusmjeravanje (engl. *meta-refresh tags*) [36] u HTML-u koje je moguće relativno lako detektirati, a podaci o njihovom postojanju mogu kasnije poslužiti za donošenje odluke je li stranica zlonamjerna u kombinaciji s nekim drugim značajkama. U praksi je općenito bolje koristiti `301 Redirect` kod odgovora za preusmjeravanje stranica umjesto `meta` oznaka za preusmjeravanje. Ipak, u nekim slučajevima poput nemogućnosti korištenja `.htaccess` datoteke ili želje da se preusmjeri samo jedna datoteka unutar direktorija koji sadrži puno drugih datoteka, korištenje meta-refresh oznake je nužno, no ono također može ukazivati na zlonamjerna preusmjeravanja.

Primjer formata `meta` oznake za preusmjeravanje prikazan je u ispisu 3.25.

```
<meta http-equiv="refresh" content="2;url=http://example.com/" />
```

Ispis 3.25: Meta oznaka za preusmjeravanje

Ova `meta` oznaka sadrži atribut `http-equiv` čija je vrijednost `'refresh'`, a polje `content` sadrži odvojene točka-zarezom vrijeme u sekundama nakon kojeg će se dogoditi preusmjeravanje te URL na koji će se preusmjeriti.

Putem Beautiful Soup objekta može se detektirati da postoji `meta` oznaka s atributom `http-equiv` koji ima vrijednost `'refresh'` te proizvoljnim `content` atributom tako da se prođe kroz sve detektirane `meta` oznake i provjeri da li ti atributi uopće postoje te ako postoje je li vrijednost atributa `http-equiv` niz znakova `'refresh'`.

Navedena funkcionalnosti implementira se kako je prikazano u ispisu 3.26.

```
def metaRefreshTagsExist(soup):
    metaTags = soup.find_all('meta')
    for meta in metaTags:
        httpEquiv = meta.get('http-equiv')
        content = meta.get('content')
        if (httpEquiv is not None and httpEquiv == 'refresh' and
        content is not None):
            return True
    return False
```

Ispis 3.26: Detekcija meta oznaka za preusmjeravanje

Još jedan način preusmjeravanja koji bi se trebao moći detektirati je JavaScript preusmjeravanje koje se događa spontano bez interakcije s korisnikom ili odabira korisnika. Preusmjeravanje u JavaScriptu [37] vrši se na dva različita načina prikazana u ispisu 3.27.

```
window.location.href = "http://www.example.com";
window.location.replace("http://www.example.com");
```

Ispis 3.27: Načini preusmjeravanja u JavaScriptu

Razlika između ta dva načina je što funkcija `replace` uklanja trenutni URL iz povijesti dokumenata pa se nije moguće vratiti na stranicu pritiskom na tipku za povratak, odnosno prvi je način primjereniji za simuliranje klika miša koji vodi na drugu stranicu uz mogućnost povratka, a drugi se način češće koristi za preusmjeravanja.

Prisutnost ovih naredbi u JavaScript kodu može ukazivati na automatizirano zlonamjerno preusmjeravanje, no treba isključiti pojavljivanje naredbi koje su smještene unutar neke funkcije koja je pokrenuta klikom korisnika jer bi klasificiranje takvih naredbi kao spontanog preusmjeravanja do kakvog primjerice dovodi `meta` oznaka za preusmjeravanje bilo neosnovano.

Iako nije u potpunosti pouzdan, jedan od mogućih pristupa koji može ukazivati na automatska i zlonamjerna preusmjeravanja, a ne zahtijeva dinamičku analizu te nije izrazito kompleksan je detekcija `window.location.replace()` ili *`window.location.href`* naredbi, ali u JavaScript kodu iz kojeg su uklonjene funkcije, pri čemu se anonimne funkcije koje se mogu pozivati unutar `setTimeout` ili `setInterval` blokova unutar glavnog JavaScript koda ovdje ne uzimaju u obzir. Kako bi se pronašlo pojavljivanje funkcije, mora se uzeti u obzir da su JavaScript funkcije označene ključnom riječi `function` nakon koje dolazi identifikator, a nakon njega otvorena obla zagrada, opcionalni sadržaj unutar nje koji predstavlja listu argumenata i zatim zatvorena zagrada. Dozvoljeni nazivi funkcija proizvoljno su dugački te sadrže alfanumeričke znakove i znak '_', uz iznimku da prvi znak u nazivu funkcije ne smije biti znamenka. Regularni izraz koji uočava pojavljivanje funkcija koristit će se u naredbi *`re.sub`* kako bi se sva takva podudaranja iz JavaScript koda uklonila [38].

Pojavljivanju funkcija unutar JavaScript koda, uključujući njihovo tijelo, odgovara regularni izraz prikazan u ispisu 3.28.

```
'function[ \t]+[A-Za-z_][0-9A-Za-z_]+[ \t]*\([a-zA-Z0-9 _,]+\){[^\}]*}'
```

Ispis 3.28: Regularni izraz za JavaScript funkcije

Kako bi se uočilo pojavljivanje znakova preusmjeravanja unutar koda iz kojeg su uklonjene funkcije, potrebno je koristiti regularni izraz koji će pokriti obje funkcije za preusmjeravanje prikazan u ispisu 3.29.

```
'window.location.href[ \t]*=|window.location.replace[ \t]*\('
```

Ispis 3.29: Regularni izraz za preusmjeravanje

Funkcionalnost detekcije koda preusmjeravanja implementira se funkcijom prikazanom u ispisu 3.30. koja koristeći Beautiful Soup objekt prolazi kroz sve skripte na stranici i pronalazi one koje sadrže preusmjeravanja.

```
def containsRedirectingScripts(soup):
    webpageScripts = soup.find_all('script');
    for script in webpageScripts:
        scriptText = ''.join(script.contents)
        if scriptText != '':
            textWithoutFunctions = re.sub('function[ \t]+[A-Za-z_][0-
9A-Za-z_]+[ \t]*\(.*\)[ \t]*{[^\}]*}',  '',
scriptText.replace('\n',''))
```

```
            if re.search('window.location.href[
\t]*=|window.location.replace[ \t]*\(',textWithoutFunctions):
                return True
    return False
```

Ispis 3.30: Detekcija JavaScript preusmjeravanja

Određivanje vrijednosti pojedinih atributa koji se uzimaju obzir u djelomično prikazanom stablu u analiziranom članku [24] već je implementirano, ali još je potrebno proširiti stablo odlučivanja dano u članku kako bi se uzeli u obzir svi relevantni atributi za određivanje potencijalno zlonamjernih Web stranica te napraviti njegovu adekvatnu implementaciju. Određuje se nekoliko dodatnih uvjeta koji se dodaju u stablo odlučivanja:

1) Ako stranica uz to što ima iframe element ima i embed elemente, tada je zlonamjerna - dosad je taj uvjet postojao samo za kombinaciju iframe i object elemenata.

2) Ako stranica uz to što ima `iframe` element ima manje ili jednako od 2 `script` elementa te sadrži preusmjeravanje bilo preko `meta` oznaka za preusmjeravanje ili skripti, onda je zlonamjerna. Uz postojanje `iframe` objekta i mali broj korisnih `script` elemenata dodatna prisutnost automatskih preusmjeravanja može ukazivati na zlonamjernost.

3) Ako stranica sadrži skripte s *escaped* znakovima, sadrži XML processing instrukcije, a istovremeno sadrži interne skripte, tada je zlonamjerna. To ukazuje na mogućnost da stranica učitava određene skripte koje će deobfuscirati *escaped* znakove, odnosno koje sadrže *`unescape`*`()` funkciju ili skripte koje će umetnuti zlonamjerne XML čvorove na koje će XML processing instrukcije biti primijenjene.

4) Ako stranica sadrži skripte s *escaped* znakovima, sadrži XML processing instrukcije, ne sadrži interne skripte, ali *inline* skripte sadrže ubacivanje XML elemenata, tada je zlonamjerna. To ukazuje na mogućnost da se ubacuju zlonamjerni XML čvorovi na koje će XML processing instrukcije djelovati, moguće i tako da iskoriste *escaped* znakove.

Nakon dodavanja navedenih uvjeta djelomičnom stablu odlučivanja prikazanom na slici 3.2, potpuni pseudokod novog stabla odlučivanja koje će biti implementirano u radu prikazan je u ispisu 3.31.

```
ako stranica sadrži više od 0 iframe elemenata:
    ako stranica sadrži više 0 malih iframe elemenata:
        stranica zlonamjerna jer sadrži male iframe elemente
    inače:
        ako stranica sadrži više od 0 object elemenata:
            stranica zlonamjerna jer sadrži iframe element i dodatne
            objekt elemente
```

```
        inače:
            ako stranica sadrži više od 0 embed elemenata:
            stranica zlonamjerna jer sadrži iframe element i dodatne
            embed elemente
            inače:
               ako stranica sadrži više od 4 script elementa:
                   stranica benigna
               inače:
                   ako stranica sadrži manje ili jednako od 2 script
                   elementa:
                       ako stranica sadrži preusmjeravanja:
                           stranica zlonamjerna jer sadrži mali broj script
                           elemenata i preusmjeravanja
                        inače:
                           stranica benigna
                   inače:
                       stranica benigna
inače:
    ako stranica sadrži više od 0 skripti s escaped znakovima:
        ako stranica sadrži više od 0 obfusciranih skripti:
            stranica zlonamjerna jer sadrži obfuscirane skripte
        inače:
            stranica benigna
    inače:
        ako stranica sadrži više od 0 XML processing instrukcija:
            ako stranica sadrži više od 0 internih skripti:
                stranica zlonamjerna jer sadrži escaped znakove, xml
                processing instrukcije i interne skripte
            inače:
                ako interne skripte ubacuju XML:
                    stranica zlonamjerna jer sadrži escaped znakove, xml
                    processing instrukcije i skripte ubacuju XML
                inače:
                   stranica benigna
```

```
inače:
    stranica benigna
```

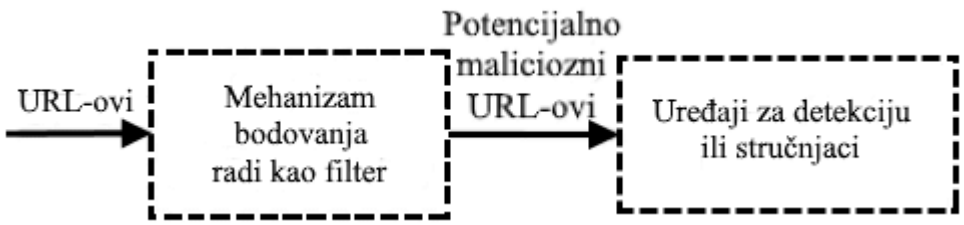
Ispis 3.31: Pseudokod algoritma statičke heuristike

U konkretnoj implementaciji, konstruirano stablo odlučivanja implementira se unutar glavnog modula za provjeru stranica `pagechecker` korištenjem `if-elif-else` grananja uz pozivanje gotovih funkcija iz modula `htmlhandler` koji na temelju HTML niza znakova određuje relevantne atribute ili skupove atributa korištene u stablu odlučivanja.

3.3 Algoritam zasnovan na mehanizmu bodovanja

3.3.1 Pregled algoritma zasnovanog na mehanizmu bodovanja

Cilj je smanjiti broj sumnjivih Web stranica te istovremeno minimizirati lažne negativne detekcije, a mehanizam bodovanja je predloženi filter koji funkcionira tako da stranice klasificira u benigne i potencijalno zlonamjerne, a samo potencijalno zlonamjerne stranice prosljeđuju se uređajima za detekciju ili stručnjacima za detekciju, kako je prikazano na slici 3.4.



Slika 3.4: Mehanizam bodovanja

Moguće je implementirati i algoritam koji stranice s vrlo visokim rezultatom bodovanja označava kao zlonamjerne uz informaciju o rezultatu koje su ostvarile i takva inačica algoritma koristit će se u ovom diplomskom radu, budući da u konkretnoj implementaciji neće postojati uređaji za detekciju kojima će se stranice prosljeđivati, a neće svi URL-ovi koji su proglašeni zlonamjernima biti ručno pregledani.

Razlozi za prijedlog mehanizma bodovanja su što radi kao filter, a ne kao krajnji klasifikator tako da radi procjenu zlonamjernosti Web stranica, koristi statičke značajke koje se mogu odrediti bez izvršavanja koda Web stranica, a također omogućuje implementaciju kompromisa između broja detektiranih potencijalno zlonamjernih Web stranica i lažno negativnih rezultata, odnosno neotkrivenih napada.

Statički atributi daju manje korisnih informacija od dinamičkih atributa koji se dobivaju izvršavanjem stranice pa su primjereniji za procjenu nego konačnu klasifikaciju, a ideja je smanjiti broj sumnjivih Web stranica koje trebaju biti ispitane zahtjevnijim algoritmima ili ručno, bez da se propusti detektirati napade.

U procesu odabira atributa za bodovanje stranica potrebno je identificirati potencijalno zlonamjerne značajke koje razlikuju zlonamjerne Web stranice od benignih, a analiza odabranih zlonamjernih stranica pokazuje da postoje tri glavne grupe zlonamjernih sadržaja čiji je pregled prikazan u tablici 3.4.

Značajka	Grupa 1: Strani sadržaj
1	Broj preusmjeravanja
2	Broj iframe oznaka
3	Broj vanjskih poveznica u iframe oznakama
4	Duljina poveznica iframe oznaka: medijan
5	Udio samoglasnika u iframe oznakama: minimum
6	Udio specijalnih znakova u iframe oznakama: minimum
7	Broj vanjskih poveznica (osim iframe poveznica)
8	Duljina ostalih poveznica: minimum
	Grupa 2: Sadržaj skripti
9	Broj skripti
10	Broj linija skripti
11	Broj riječi skripti
12	Udio posebnih znakova u skriptama
13	Duljina skripti: minimum
14	Duljina linija skripti: minimum
15	Duljina nizova znakova skripti: maksimum
16	Duljina riječi skripti: minimum
17	Duljina argumenata funkcija skripti: minimum
	Grupa 3: Sadržaj za iskorištavanje ranjivosti
18	Broj object oznaka
19	Duljina poveznica object oznaka: maksimum
20	Udio posebnih znakova u poveznicama object oznaka
21	Udio samoglasnika u poveznicama object oznaka
22	Broj atributa object oznaka: medijan

Tablica 3.4: Relevantne značajke za detekciju potencijalno zlonamjernih Web stranica

Strani sadržaj je zlonamjerni sadržaj koji se učitava izvana sa sumnjivim Web stranicama, a može se učitati kao izvor određenih zlonamjernih HTML oznaka kao što su `img` ili `iframe`, pri čemu je `iframe` posebno poznat kao metoda učitavanja zlonamjernih vanjskih Web stranica pa se u algoritmu promatraju atributi poput duljine poveznica, udjela samoglasnika i udjela specijaliziranih znakova u `iframe` oznakama, ali i u drugim vanjskim poveznicama. U gotovo svim slučajevima, uzrok stranog zlonamjernog sadržaja su kompromitacija poslužitelja ili nekontrolirani sadržaji treće strane poput oglasa ili brojača posjeta stranice.

Sadržaji u obliku skripti kao što je JavaScript poznati su kao najčešći zlonamjerni sadržaj zlonamjernih Web stranica, a u većini slučajeva koriste se za dvije glavne svrhe: dostavu zlonamjernog koda i skrivanje zlonamjernog koda obfuskacijom. Neke od potencijalno zlonamjernih značajki identificiraju se iz JavaScript koda, kao što su veličina skripte, veličina nizova znakova, veličina riječi, veličina argumenata i distribucija znakova.

Sadržaji za iskorištavanje ranjivosti su glavni dijelovi zlonamjernih Web stranica koji ciljaju određene ranjivosti u internetskim preglednicima, dodacima i operacijskim sustavima, a neki od HTML elemenata kojima se često učitavaju potencijalno zlonamjerni kod su `applet`, `object` i `embed`, s tim da je `applet` kao oznaka zastario i više se ne koristi. Takvi sadržaji se rijetko mogu pronaći u direktom obliku te su u gotovo svim slučajevima enkodirani u obfuscirane skripte kako bi bili skriveni od uređaja za detekciju.

Prema analizi skupa poznatih zlonamjernih stranica, odabiru se 52 atributa iz triju glavnih skupina zlonamjernih sadržaja, ako se atribut pojavljuje više puta uzimaju se 4 vrijednosti: maksimum, minimum, medijan i prosjek, no samo se jedna vrijednost uzima u obzir u izračunu rezultata.

Za odabir samo onih atributa koji su značajni, mjeri se količina dobivenih informacija (engl. *information gain,* skraćeno IG) svakog pojedinog atributa koja se računa po formulu 3.17.

$$\mathrm{IG(S,a)} = \mathrm{Entropy(S)} - \sum_{v \epsilon a} \frac{|S_v|}{|S|} * Entropy(S_v)$$

Formula 3.17: Izračun količina dobivenih informacija

Pritom je `S` kolekcija instanci, na primjer Web stranica, a `Sv` je podskup od `S` koji ima relevantnu vrijednost `v` atributa `a`.

Što je veća količina dobivenih informacija, to atribut više doprinosi procesu otkrivanja zlonamjernih Web stranica, a skup za trening koji se koristi za izračun količine dobivenih informacija mora sadržavati i zlonamjerne i benigne stranice.

Na temelju količine dobivenih informacija, od 52 atributa odabrano je njih 26, s tim da se u razmatranju članka neće uzimati u obzir niti spominjati atributi koji se odnose na `frame` ili `applet` oznake budući da su one zastarjele i više se ne koriste iako se u članku spominju pa će zato broj promatranih atributa koji su prikazani u tablici 3.3.1.1 biti manji od 26.

Iz skupine atributa koji čine strani sadržaj uzimaju se u obzir broj preusmjeravanja, broj `iframe` oznaka, broj vanjskih poveznica u `iframe` oznakama, medijan duljine poveznica u `iframe` oznakama, minimalni udio samoglasnika u pojedinim `iframe` oznakama, minimalni udio posebnih znakova u pojedinim `iframe` oznakama, broj vanjskih poveznica osim `iframe` oznaka te minimalna duljina pojedinih drugih poveznica.

Iz skupine atributa koji čine skripte uzimaju se u obzir broj skripti, broj linija unutar skripti, broj riječi unutar skripti, postotak posebnih znakova unutar skripti, minimalna duljina pojedinih skripti, minimalna duljina pojedine linije skripti, maksimalna duljina pojedinog niza znakova unutar skripte, minimalna duljina pojedine riječi unutar skripte te minimalna duljina pojedinog argumenta funkcije skripti.

Iz skupina atributa koji čine sadržaj za potencijalno iskorištavanje ranjivosti uzimaju se u obzir broj `object` oznaka, maksimalna duljina poveznice `object` oznaka, udio posebnih znakova u `object` oznakama, udio samoglasnika u `object` oznakama te medijan broja atributa `object` oznaka.

Predloženi algoritam bodovanja radi na temelju koncepta standardnog rezultata koji mjeri koliko je standardnih devijacija vrijednost promatranog rezultata udaljena od prosjeka skupa. Svaka instanca, odnosno stranica ima tri tipa rezultata na temelju tri grupe sadržaja Web stranica: rezultat stranog sadržaja (engl. *foreign content score*), rezultat sadržaja skripti (engl. *script content score*) i rezultat sadržaja za iskorištavanje ranjivosti (engl. *exploit content score*).

Grupni rezultat instance x, odnosno stranice x izračunava se formulom 3.18.

$$GS_{g \epsilon G}(x) = \sum_{a \epsilon g} \frac{|x_a - \mu_a|}{\delta_a}$$

Formula 3.18: Izračun grupnog rezultata stranice x

Pritom je g grupa atributa - može biti grupa stranog sadržaja, grupa sadržaja skripti ili grupa sadržaja za iskorištavanje ranjivosti, oznaka a predstavlja atribut unutar grupe g, x_a je vrijednost atributa instance x, δ_a je standardna devijacija atributa a koja je procijenjena tijekom treniranja na skupu benignih stranica, a μ_a je prosjek atributa a koji je procijenjen tijekom treninga na setu benignih stranica.

Što je veći rezultat instance, odnosno stranice x u svakoj grupi, to je vjerojatnije da će ona biti klasificirana kao potencijalno zlonamjerna. Ako je T_g izabran kao prag za grupu sadržaja kako bi se identificirale potencijalno zlonamjerne stranice, pravilo klasifikacije je takvo da će svaka stranica koja ima rezultat bilo koje grupe iznad praga te grupe biti klasificirana kao zlonamjerna, a to se može iskazati formulom 3.19.

$$x = \begin{cases} potencijalno\ maliozna\ ako\ \exists g\ \epsilon\ G: GS_g(x) > T_g \\ benigna\ inače \end{cases}$$

Formula 3.19: Klasifikacija algoritmom zasnovanim na mehanizmu bodovanja

U analiziranom članku provedeni su eksperimenti učinkovitosti kojima je bio cilj odrediti minimalni udio stranica koje su označene kao potencijalno zlonamjerne uz kojeg će stopa lažno negativnih rezultata, odnosno broj propuštenih zlonamjernih stranica biti zanemariva ili nula.

Kako bi se dobio skup podataka za eksperimente, potrebno je skupiti Web stranice kandidate koje sadrže i zlonamjerne i benigne Web stranice. Benigne stranice skupljene su tako da su najpopularnije fraze pretraživanja skupljene iz Google Search Engine-a i predane Yahoo API websearch alatu [39] kako bi se dohvatilo prvih 10 URL-ova iz rezultata pretrage, zlonamjerne Web stranice skupljene su iz skupa javno poznatih zlonamjernih stranica s listi kao što je malwaredomainlist.com [40], a obje skupine stranica verificirane su korištenjem *honeypot* klijenta visoke interakcije.

Nakon skupljanja stranica kreira se *honeypot* klijent niske interakcije koji od Web poslužitelja zahtijeva navedene stranice te su iz HTTP odgovora poslužitelja izdvojeni atributi i njihove potencijalne vrijednosti iz skupa od 26 relevantnih atributa.

Ukupno je sakupljeno 33 646 Web stranica, od kojih je 33 422 benigno, a 224 zlonamjerno.

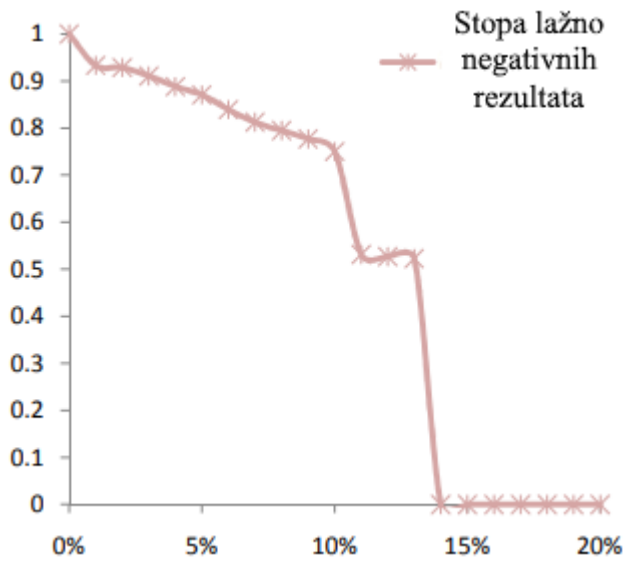
Skup podataka za trening sastoji se od 20 000 stranica i koristi se za izračunavanje prosjeka i standardne devijacije za svaki atribut koji će se koristiti u mehanizmu bodovanja, a skup

podataka za testiranje sastoji se od 13 646 stranice, uključujući 13 422 benignu stranicu i 224 zlonamjernih stranica te će se koristi za testiranje mehanizma bodovanja.

Eksperiment se provodi u tri koraka.

U prvom je koraku skup podataka za treniranje predan mehanizmu za bodovanje kako bi se izračunale potrebne statističke vrijednosti atributa, odnosno prosjek i standardna devijacija. U drugom se koraku izračunavaju grupni rezultati za svaku stranicu u testnom skupu podataka, pri čemu svaka stranica ima tri tipa rezultata: rezultat stranog sadržaja, rezultat sadržaja skripti i rezultat sadržaja za iskorištavanja ranjivosti. U trećem koraku prilagođavaju se vrijednosti praga rezultata u svakoj grupi kako bi se našla veza između udjela lažno negativnih detekcija, odnosno propuštenih detekcija napada i broja identificiranih potencijalno zlonamjernih Web stranica.

Vrijednost praga započinje od maksimalne vrijednosti za svaki rezultat grupe i onda se smanjuje na vrijednost koja odgovara određenom postotku otkrivenih potencijalno zlonamjernih Web stranica. Kako se broj potencijalno zlonamjernih Web stranica povećava, postotak lažno negativnih rezultata se smanjuje, a cilj je minimizirati postotak lažno negativnih stranica, što se postigne kad je broj potencijalno zlonamjernih Web stranica jednak 14% ukupnog broja stranica u skupu za testiranje. To znači da se može eliminirati 86% potencijalno zlonamjernih Web stranica bez da se propusti ijedan napad, kako je prikazano na slici 3.5.



Slika 3.5: Veza između lažno negativne stope i broja potencijalno zlonamjernih Web stranica

U konkretnoj implementaciji unutar diplomskog rada postojat će skup od 75 492 stranica koji će se koristiti kao trening set za izračun srednjih vrijednosti i standardnih devijacija, ali isti će se skup koristiti i kao testni skup, što znači da će određeni skup stranica koji odudara od prosjeka ostalih stranica biti označen kao potencijalno maliciozan. Razlog tome je što je pretpostavljeni udio zlonamjernih stranica u uzorku vrlo mali budući da skup ne sadrži poznate zlonamjerne stranice i mogućnost da neki zlonamjerni atributi budu zajednički kod

većine stranica te time onemoguće njihovu detekciju značajnim utjecajem na srednje vrijednosti i standardne devijacije skupa gotovo je nikakva.

3.3.2 Implementacija algoritma zasnovanog na mehanizmu bodovanja

Uz neke funkcije koje već postoje jer su bile potrebne za implementaciju algoritam statičke heuristike, u modul `htmlhandler` potrebno je dodati funkcije za izdvajanje ostalih potrebnih atributa čiji se rezultat koristi kao kriterij algoritma temeljenog na mehanizmu bodovanja.

Za veliki broj relevantnih atributa koje je potrebno analizirati traži se maksimalna, minimalna, prosječna vrijednost ili medijan pojedine liste, primjerice minimalna duljina linije određene skripte. Zato će se koristiti ugrađene funkcije `min` i `max` te funkcije `median` i `mean` iz modula `statistics` koje je potrebno uvesti u modul `htmlhandler` naredbom prikazanom u ispisu 3.32.

```
from statistics import median, mean
```

Ispis 3.32: Uvođenje modula statistics u Python program

Te se funkcije pozivaju nad listom cijelih ili decimalnih brojeva.

Također, mnogi atributi sadrže određena leksička svojstva nizova znakova, kao što su broj linija, broj riječi, udio samoglasnika ili udio posebnih znakova i za to je korisno kreirati pomoćni modul `lexicutils` koji će se uvesti u modul `htmlhandler` naredbom prikazanom u ispisu 3.33.

```
import lexicutils
```

Ispis 3.33: Uvođenje modula lexicutils u Python program

Iz skupine atributa koji čine strani sadržaj relevantni su broj preusmjeravanja, broj `iframe` oznaka, broj vanjskih poveznica u `iframe` oznakama, medijan duljine poveznica u `iframe` oznakama, minimalni udio samoglasnika u pojedinim `iframe` poveznicama, minimalni udio posebnih znakova u pojedinim `iframe` poveznicama, broj vanjskih poveznica osim `iframe` oznaka te minimalna duljina pojedinih drugih poveznica.

Što se tiče implementacije određivanja broja preusmjeravanja, već je implementirana funkcija koja određuje postoje li `meta` oznake koje sadrže preusmjeravanje te `script` oznake koje sadrže preusmjeravanje, sada je potrebno napraviti funkciju koja će odrediti ukupni broj takvih preusmjeravanja korištenjem postojećih dijelova implementiranih funkcija, a navedeno se implementira kako je prikazao u ispisu 3.34.

```
def countRedirects(soup):
    …
    for meta in metaTags:
        …
        if (httpEquiv is not None and httpEquiv == 'refresh' and
        content is not None):
            redirects +=1
    webpageScripts = soup.find_all('script');
    for script in webpageScripts:
        scriptText = ''.join(script.contents)
        if scriptText != '':
            …
            redirects += redirectMatches
    return redirects
```

Ispis 3.34: Određivanje broja preusmjeravanja HTML stranice

Funkcija za broj iframe oznaka već je implementirana kao funkcija `countIFrameTags`, a funkcija za broj vanjskih poveznica u iframe oznakama već je implementirana kao `countExternalIframes` koja određuje broj iframe oznaka s vanjskom poveznicom uzimajući u obzir i opcionalni parametar domene stranice.

Za određivanje medijana duljina poveznica, minimalnog udjela samoglasnika te minimalnog udjela posebnih znakova u `iframe` poveznicama potrebno je implementirati posebnu funkciju koja će davati sve te informacije u jednom objektu rječnika.

Za određivanje udjela samoglasnika i udjela posebnih znakova u poveznicama potrebno je implementirati pomoćne funkcije `vowelRatio` i `specialCharRatio` unutar `lexicutils` modula kako je prikazano u ispisu 3.35.

```
def vowelRatio(str):
    …
    for letter in str:
        if letter in vowel:
            vowelCount += 1
    letterCount = len(str)
    if(letterCount > 0):
```

```
        ratio = vowelCount/letterCount
    else:
        ratio = 0.00
    return ratio

def specialCharRatio(str):
    specialChar = [';', '+=', '_', '?', '=', '&', '[', ']', '#', '~',
    '%','@', '$', '*', '+', '!', '|']
    …
    if(letterCount > 0):
        ratio = specialCharCount/letterCount
    else:
        ratio = 0.00
    return ratio
```

Ispis 3.35: Određivanje udjela samoglasnika i udjela posebnih znakova

Funkcija za određivanje statistike `iframe` poveznica implementira se kako je prikazano u ispisu 3.36.

```
def getIFrameLinksStatistics(soup):
    …
    for iFrame in webpageIFrames:
        iFrameSource = iFrame.get('src')
        if iFrameSource is not None:
            iFrameLinkLength = len(iFrameSource)
            iFrameLinkVowelRatio = lexicutils.vowelRatio(iFrameSource)
            …
    iFrameLinksStatistics['medianLength'] = median(iFrameLinkLengths)
    if (len(iFrameLinkLengths) > 0) else 0
    iFrameLinksStatistics['minVowelRatio'] = min(iFrameLinkVowelRatios)
    if (len(iFrameLinkVowelRatios) > 0) else 0.00
    …
```

```
    return iFrameLinksStatistics
```

Ispis 3.36: Određivanje statistika iframe poveznica

Za određivanje broja vanjskih poveznica i minimalne duljine ostalih poveznica potrebno je implementirati posebnu funkciju koja će uzimati u obzir sve ostale elemente osim `iframe` elemenata koji imaju poveznice te određivati koji su od njih vanjski i na kraju odrediti minimalnu duljinu poveznice među njima. Oznake koje će se uzimati u obzir za statistiku poveznica su `a` oznake, odnosno njihov `href` atribut, `script` oznake, odnosno njihov `src` atribut, `embed` oznake, odnosno njihov `src` atribut, `object` oznake, odnosno njihov `data` atribut te `link` oznake, odnosno njihov `rel` atribut.

Funkcija za određivanje statistike vanjskih poveznica implementira se kako je prikazano u ispisu 3.37.

```
def getLinksStatistics(soup, domain = None):
    …
    for link in links:
        href = link.get('href')
        if(href is not None):
            allLinks.append(href)
    scripts = soup.find_all('script')
    …
    objects = soup.find_all('object')
    for object in objects:
        data = object.get('data')
        if(data is not None):
            allLinks.append(data)
    linkElements = soup.find_all('link')
    …
    for link in allLinks:
        if (not isPathLocal(link,domain)):
            externalLinksNumber += 1
            linkLengths.append(len(link))
    linksStatistics['minLength'] = min(linkLengths)
    if (len(linkLengths) > 0) else 0
```

```
    linksStatistics['externalLinks'] = externalLinksNumber
    return linksStatistics
```

Ispis 3.37: Određivanje statistike vanjskih poveznica

Iz skupine atributa koji čine skripte uzimaju se u obzir broj skripti, ukupni broj linija unutar skripti, ukupan broj riječi unutar skripti, postotak posebnih znakova unutar skripti, minimalna duljina pojedinih skripti, minimalna duljina pojedine linije skripti, maksimalna duljina pojedinog niza znakova unutar skripte, minimalna duljina pojedine riječi unutar skripte te minimalna duljina pojedinog argumenta funkcije skripti.

Funkcija za određivanje broja skripti već je implementirana kao `countScriptTags`.

Ostali atributi određuju se putem posebne funkcije koja vraća statistiku skripti.

Funkcije za određivanje tih atributa svake pojedine skripte koja se predaje kao niz znakova nalazit će se unutar pomoćnog modula `lexicutils`.

Implementacija određivanja broja linija i broja riječi unutar skripte prikazana je u ispisu 3.38.

```
def linesNumber(str):
    return len(str.splitlines())


def wordsNumber(str):
    return len(str.split())
```

Ispis 3.38: Određivanje broja linija i broja riječi unutar skripte

Određivanje udjela specijalnih znakova u nizu znakova već je implementirano funkcijom `specialCharRatio`.

Implementacija određivanja minimalne duljine linije skripte, maksimalne duljine nizova znakova unutar skripte, minimalne duljine riječi unutar skripte i minimalne duljine argumenta funkcije postignuta je kako je prikazano u ispisu 3.39.

```
def minimumLineLength(str):
    lines = str.splitlines()
    if (len(lines) > 0):
        return len(min(lines, key=len))
    else:
```

```
        return 0

def maximumStringLength(str):
    stringsList = re.findall("['\"]([0-9a-zA-
    Z\.\?\+\*\^\$\\\(\)\[\]\{\}\|]*)['\"]",str)
    if (len(stringsList) > 0):
        return len(max(stringsList, key = len))
    else:
        return 0

def minimumWordLength(str):
    words = str.split()
    if (len(words) > 0):
        return len(min(words, key=len))
    else:
        return 0

def minimumFunctionArgLength(str):
    …
    for functionOccurance in functionOccurances:
        argumentsString =
        functionOccurance.split('(')[1].replace(')','').replace(' ','')
        argumentsList = argumentsList + argumentsString.split(',')
    if (len(argumentsList) > 0):
        return len(min(argumentsList, key = len))
    else:
        return 0
```

Ispis 3.39: Određivanje dodatnih leksičkih obilježja skripte

Nakon implementacije funkcija unutar pomoćnog modula `lexicutils`, funkcija za dohvaćanje ukupne statistike skripti implementira se kako je prikazano u ispisu 3.40.

```
def getScriptStatistics(soup):
    …
    for script in scripts:
        scriptText = ''.join(script.contents)
        if scriptText != '':
            linesNumbers.append(lexicutils.linesNumber(scriptText))

            wordsNumbers.append(lexicutils.wordsNumber(scriptText))
            …
            scriptLengths.append(len(scriptText))
            …
    scriptStatistics['linesNumber'] = sum(linesNumbers) if
    (len(linesNumbers) > 0) else 0
    scriptStatistics['wordsNumber'] = sum(wordsNumbers) if
    (len(wordsNumbers) > 0) else 0
    …
    return scriptStatistics
```

Ispis 3.40: Određivanje ukupnih statistika skripti

Iz skupina atributa koji čine sadržaj za potencijalno iskorištavanje ranjivosti uzimaju se u obzir broj `object` oznaka, maksimalna duljina poveznice `object` oznaka, udio posebnih znakova u `object` oznakama, udio samoglasnika u `object` oznakama te medijan broja atributa `object` oznaka.

Funkcija koja računa broj `object` oznaka već je implementirana kao `countObjectTags`.

Određivanje ostalih relevantnih atributa `object` oznaka implementirat će se u posebnoj funkciji kako je prikazano u ispisu 3.41.

```
def getObjectStatistics(soup):
    …
    for object in webpageObjects:
        objectSource = object.get('data')
        if objectSource is not None:
            objectLinkLength = len(objectSource)
```

```
            objectLinkVowelRatio = lexicutils.vowelRatio(objectSource)
            …
            objectLinkLengths.append(objectLinkLength)
            objectLinkVowelRatios.append(objectLinkVowelRatio)
            …
    objectStatistics['maxLinkLength'] = max(objectLinkLengths) if
    (len(objectLinkLengths) > 0) else 0
    objectStatistics['vowelRatio'] = avg(objectLinkVowelRatios) if
    (len(objectLinkVowelRatios) > 0) else 0.00
    …
    return objectStatistics
```

Ispis 3.41: Određivanje statistika object oznaka

Pojedini relevantni atributi mehanizma bodovanja za koji je implementirano izračunavanje unutar niza funkcija sada se trebaju koristiti u funkciji `checkByScoringMechanism` modula `pagechecker`.

Funkciji se kao argument predaje `scoringParametersPath` putanja iz koje se učitavaju parametri bodovanja koji sadrže parametre poput srednjih vrijednosti i devijacija relevantnih atributa s obzirom na koje se računa rezultat komponente te pragova za pojedine kategorije bodovanja.

Nazivi skupina atributa koje će se računati su `'foreign'`, odnosno strani sadržaj, `'scripts'`, odnosno sadržaj skripti te `'exploit'`, odnosno sadržaj za potencijalno iskorištavanje ranjivosti. Nakon određivanja svih atributa, na temelju parametara bodovanja koji su predani funkciji određuju se rezultati svake pojedine skupine atributa zlonamjernosti te ako neka od vrijednosti prelazi prag određen za tu skupinu atributa, onda se stranica označava kao zlonamjerna i to se dodaje u objašnjenje zlonamjernosti.

Za generiranje `scoringParameters` objekta iz `scoringParametersPath` putanje te za generiranje objašnjenja zlonamjernosti iz `scoringParameters` objekta koji predstavlja generirane rezultate za pojedinu stranicu treba uvesti poseban modul `scoringchecker` koji će se unutar modula `pagechecker` uvesti naredbom

```
import scoringchecker
```

Unutar modula `scoringchecker` postoji funkcija `setScoringParameters` koja unutar modula inicijalizira objekt `scoringParameters` iz JSON niza znakova pohranjenog na putanji `scoringParametersPath`, a implementirana je kako je prikazano u ispisu 3.42.

```
def setScoringParameters(path):
    global scoringParameters

    scoringParametersFile = open(path, encoding='utf-8')
    scoringParametersString = scoringParametersFile.read()

    scoringParametersFile.close();
    scoringParameters = json.loads(scoringParametersString)
```

Ispis 3.42: Postavljanje parametara bodovanja

Određivanje da li stranica prelazi neki od pragova vrši se tako da se prolazi kroz sve atribute koji su izračunati za stranicu te računa koliko standardnih devijacija odstupa od prosječne stranice trening skupa za svaki od atributa te vrijednost odstupanja pribraja ukupnom rezultatu za odgovarajuću skupinu atributa.

Unutar modula postoji funkcija `generateMaliciousExplanations` koja na temelju postavljenih parametara bodovanja i rezultata stranice provjerava postoji li neka skupina atributa koja je kod stranice zlonamjerna te generira objašnjenje za detekciju zlonamjernosti, a implementira se kako je prikazano u ispisu 3.43.

```
def generateMaliciousExplanations(scores):
    …
    for foreignKey in list(foreignScores.keys()):
        foreignResult += abs(foreignScores[foreignKey]-
foreignParameters[foreignKey]['average'])/foreignParameters[foreignKey]
['deviation']
    for scriptsKey in list(scriptsScores.keys()):
        …
    for exploitKey in list(exploitScores.keys()):
        …
    if foreignResult > foreignThreshold:
        maliciousExplanations.append('Foreign content score passed
        threshold of ' + str(foreignThreshold) + ' with result ' +
        str(foreignResult))
    if scriptsResult > scriptsThreshold:
        …
```

```
    if exploitResult > exploitThreshold:
        …
    return maliciousExplanations
```

Ispis 3.43: Provjera prelaska pragova bodovanja za rezultate različitih skupina atributa

Prije izvođenja algoritma potrebno je generirati ispravnu `scoring_parameters.json` datoteku uzimajući u obzir srednje vrijednosti i atribute unutar promatranog skupa podataka te postaviti pragove kako bi optimalni broj stranica bio otkriven kao potencijalno zlonamjeran, odnosno kako bi vjerojatnost propuštanja detekcije bila minimalna, uz prihvatljivi broj lažno pozitivnih rezultata.

Najprije je unutar pomoćnog `scoringchecker` modula potrebno implementirati generiranje svih vrijednosti atributa za pojedinu stranicu kako bi se ona mogla koristiti i kod izračuna srednjih vrijednosti i devijacija cijelog skupa za pojedine atribute i za izračun rezultata za svaku pojedinu stranicu pri procjeni njene zlonamjernosti, kako je prikazano u ispisu 3.44.

```
def calculateScores(html, domain = None):
    …
    scores['foreign']['redirectsCount'] =
    htmlhandler.countRedirects(soup)
    …
    iframeLinksStatistics = htmlhandler.getIFrameLinksStatistics(soup)
    scores['foreign']['iframeLinksMedianLength'] =
    iframeLinksStatistics['medianLength']
    …
    linksStatistics = htmlhandler.getLinksStatistics(soup, domain)
    scores['foreign']['minLinkLength'] = linksStatistics['minLength']
    …

    scores['scripts']['scriptsCount'] =
    htmlhandler.countScriptTags(soup)
    scriptStatistics = htmlhandler.getScriptStatistics(soup)
    scores['scripts']['linesNumber'] = scriptStatistics['linesNumber']
    …
```

```
    scores['exploit']['objectsCount'] =
    htmlhandler.countObjectTags(soup)
    objectStatistics = htmlhandler.getObjectStatistics(soup)
    scores['exploit']['objectsMaxLinkLength'] =
    objectStatistics['maxLinkLength']
    …
    return scores
```

Ispis 3.44: Izračun vrijednosti svih relevantnih atributa za stranicu

Priprema za izvođenje algoritma mora biti provedena implementacijom posebne funkcije koja u obzir uzima sve relevantne atribute te za njih računa srednje vrijednosti i standardne devijacije kako bi se generirala ispravna `scoring_parameters.json` datoteka.

Toj funkciji će se predati lista HTML stranica i putanja do datoteke u koji mora pohraniti izračunate parametre bodovanja, a bit će implementirana tako da izračunava sve atribute za svaku od stranica, zatim ih spaja u liste i iz tih lista računa standardnu devijaciju i srednje vrijednosti.

Pragovi koje stranica treba prijeći da bi postala zlonamjerna bit će postavljeni najprije na 100, a nakon toga u skladu s rezultatima, odnosno postotku zlonamjernosti otkrivenom u skeniranju s tim postavljenim pragovima, oni će se podešavati.

Navedena funkcija implementirana je kako je prikazano u ispisu 3.45.

```
def prepareScoringParameters(webpagesList, path):
    …
    for webpage in webpagesList:
        htmlScores = calculateScores(webpage['html'])
        foreignScores = htmlScores['foreign']
        …
        for foreignKey in list(foreignScores.keys()):
            …
            scoresList['foreign'][foreignKey].append(
            foreignScores[foreignKey])
       …
    foreignScoresList = scoresList['foreign']
    …
```

```
    for foreignKey in list(foreignScoresList.keys()):
        …
        scoringParameters['foreign'][foreignKey]['average'] =
        mean(foreignScoresList[foreignKey])
        scoringParameters['foreign'][foreignKey]['deviation'] =
        stdev(foreignScoresList[foreignKey])
    …
    scoringParametersFile = open(path, "w", encoding='utf-8')
    scoringParametersFile.write(json.dumps(scoringParameters))
    scoringParametersFile.close()
```

Ispis 3.45: Priprema parametara bodovanja na testnom skupu stranica

Konačna implementacija algoritma zasnovanog na mehanizmu bodovanja unutar modula *pagechecker*, koristeći sve spomenute pomoćne funkcije dana je u ispisu 3.46.

```
def checkByScoringMechanism(html, scoringParametersPath, domain =
None):

    scoringchecker.setScoringParameters(scoringParametersPath)
    analysisResults = {'status':'not classified', 'details':''}
    scores = scoringchecker.calculateScores(html, domain)
    explanations = scoringchecker.generateMaliciousExplanations(scores)

    if len(explanations) > 0:
        analysisResults['status'] = 'malicious'
        analysisResults['details'] = "; ".join(explanations)
    else:
        analysisResults['status'] = 'benign'
        analysisResults['details'] = 'no malicious properties detected'
    return analysisResults
```

Ispis 3.46: Implementacija algoritma zasnovanog na mehanizmu bodovanja

Prilikom pokretanja funkcije rezultat `prepareScoringParameters` nad trening skupom podataka rezultat je JSON formata prikazanog u ispisu 3.47 koji će se koristiti za određivanje rezultata kod algoritma temeljenog na mehanizmu bodovanja.


```
{"foreignThreshold": 100, "scriptsThreshold": 100, "exploitThreshold":
100, "foreign": {"redirectsCount": {"average": 0.03579139787794895,
"deviation": 0.2890028489079589}, …}, "scripts": {"scriptsCount":
{"average": 15.597975971282105, "deviation": 18.72353448289455},…},
"exploit": {"objectsCount": {"average": 0.035301286211966675,
"deviation": 0.33327217035917134},…}}
```


Ispis 3.47: Primjer .json konfiguracijske datoteke za parametre bodovanja

U članku koji opisuje algoritam temeljen na bodovanju [1] utvrđeno je da je stopa lažno negativne detekcije 0 postignuta onda kad 14% stranica bude proglašeno potencijalno zlonamjernima. No budući da je ovaj algoritam napravljen zasebno, a ne kao dio hibridnog sustava u kojem se potencijalno zlonamjerne stranice prosljeđuju stručnjacima i uređajima za detekciju, kao ciljani postotak uzet će se nešto niži postotak, između 5% i 10% kako bi stopa lažno negativnih rezultata ostala minimalna uz stopu lažno pozitivnih rezultata koja je razmjerna stopi lažno pozitivnih rezultata statičke heuristike od 5.88%.

Za brzo i efikasno podešavanje pragova na vrijednosti koje će dati željeni postotak zlonamjernih stranica, algoritam će se u nekoliko iteracija pokrenuti na 367 nasumičnih stranica iz cijelog skupa kako bi se prikazali rezultati i vidjelo koje su vrijednosti rezultata stranice dostigle, odredio postotak stranica koji prelazi pragove te se u skladu s time pragove smanjilo ili povećalo.

Pritom će su u algoritmu dodati pomoćni ispis koji pokazuje rezultate po pojedinim skupinama atributa za određene stranice, kako bi se moglo pretpostaviti na koje nove vrijednosti pragove treba ažurirati. Kod za generiranje pomoćnog ispisa prikazan je u ispisu 3.48.

```
for foreignKey in list(foreignScores.keys()):

        foreignResult += abs(foreignScores[foreignKey]-
foreignParameters[foreignKey]['average'])/foreignParameters[foreignKey]
['deviation']

…

print("foreign result", foreignResult)

print("script result", scriptsResult)

print("exploit result", exploitResult)
```

Ispis 3.48: Generiranje pomoćnog ispisa rezultata pojedinih skupina atributa

Pritom se dobije lista rezultata u formatu prikazanom na ispisu 3.49.

```
foreign result 2.0986314366780126
script result 2.382323718184738
exploit result 0.4425835470971211
foreign result 2.120830959838992
script result 5.408355437538288
exploit result 0.4425835470971211
foreign result 2.108307923973732
script result 1.2202187346364626
exploit result 0.4425835470971211
…
```

Ispis 3.49: Rezultati pojedinih stranica po skupinama atributa

Nakon nekoliko iteracija podešavanja pragova na podskupu od 367 stranica, dobiju se optimalni pragovi od 9.0 za skupinu atributa stranog sadržaja, 6.0 za skupinu atributa skripti i 5.0 za skupinu atributa sadržaja za iskorištavanje ranjivosti.

Pri takvim pragovima, postotak detekcija na pomoćnom podskupu od 367 stranica iznosi 9.26%, što odgovara prihvatljivom postotku koji se želi postići, s tim da se pritom prihvaća određeni broj lažno negativnih rezultata. Zbog toga bi pragovi u slučaju implementacije koja je dio hibridnog sustava bili nešto niži, a očekivani postotak koji se prosljeđuje uređajima za detekciju ili stručnjacima nešto viši.

Očekuje se da će kod analize cijelog skupa podataka od 75 492 stranice postotak detekcije biti sličan ili barem neće biti viši od 14% predloženih u literaturi.

3.4 Algoritam zasnovan na Yara pravilima

3.4.1 Yara pravila i standard

Yara pravila [41] koriste se za klasifikaciju i identifikaciju kreirajući standardizirane porodice zlćudnih programa na temelju tekstualnih ili binarnih uzoraka. Yara pravila funkcioniraju tako da definiraju određene varijable koje sadrže uzorke koji se mogu pronaći u nekom zlćudnom programu. Ako su neki od uvjeta ili svi uvjeti ispunjeni, onda ta podudarnost ovisno o pravilu može biti iskorištena kako bi se uspješno identificirao zlonamjerni sadržaj.

Tijekom analize zlonamjernog sadržaja, istraživači će identificirati jedinstvene uzorke i nizove znakova sadržane u zlćudnom programu koji će im omogućiti da identificiraju kojoj grupi prijetnji i porodica zlćudnih programa se određeni uzorak može pridružiti. Stvaranjem Yara pravila iz nekoliko uzoraka iz iste porodice zlćudnih programa, također je moguće identificirati više uzoraka koji su svi moguće povezani s istom kampanjom ili kreatorom prijetnji.

Kada istražuje određeni zlćudni program, analitičar može kreirati Yara pravilo za novi uzorak kojeg istražuje i onda to pravilo može iskoristiti za traženje sličnih uzoraka u vlastitoj baze zlćudnih programa ili u online repozitoriju kao što je VirusTotal [42].

Ako analitičar zlćudnih programa radi za organizaciju koja primjenjuje sustav za prevenciju uljeza (engl. *intrustion prevention system*, skraćeno IPS) ili drugu platformu koja se koristi za zaštitu od zlćudnih programa i podržava Yara standard, tada se Yara pravila mogu koristiti kao alat za reakciju na incidente (engl. *incident response*) koji detektira zlonamjerne binarne sadržaje unutar organizacije.

Yara je iznimno popularna unutar zajednice informacijske sigurnost upravo zbog velikog broja slučajevima u kojima je njena implementacija korisna, uključujući identifikaciju i klasifikaciju zlćudnih programa i pronalazak novih zlonamjernih uzoraka na temelju uzoraka specifičnih za porodicu zlćudnih programa. Prilikom reakcije na incidente, Yara pravila se mogu primijeniti za identifikaciju zlonamjernih uzoraka i kompromitiranih uređaja, a proaktivna primjena proizvoljnih Yara pravila može poboljšati opću obranu organizacije.

Kako bi se moglo izgraditi korisno Yara pravilo te razumjeti i po potrebi mijenjati već definirana Yara pravila, potrebno je poznavati različite elemente koji se koriste u Yara sintaksi prilikom izgradnje proizvoljnog Yara pravila.

Meta podaci (engl. `metadata`) ne utječu na to što će Yara pravilo tražiti, već pružaju korisne informacije o samom pravilu. Polje `author` može sadržavati ime, email adresu ili korisničko ime na društvenim mrežama, `date` označava datum kad je pravilo kreirano, `version` označava broj verzije pravila potreban da bi se lakše pratili nove verzije i nadogradnje, `reference` sadrži poveznicu na popratni članak ili poveznicu za preuzimanje uzorka, a služi za pružanje relevantnih informacija o zlonamjernom uzorku kojeg pravilo treba detektirati, `description` predstavlja kratki opis namjene pravila i zlćudnog programa koji detektira, a `hash` predstavlja listu sažetaka uzoraka koji su bili korišteni pri kreiranju Yara pravila.

Uobičajeno je da se unutar zlonamjernog uzorka pronađu zanimljivi nizovi znakova te su oni idealni za izgradnju Yara pravila. Kako bi se definirao niz znakova unutar Yara pravila, on treba biti deklariran kao varijabla unutar sekcije pravila `strings` kako je prikazano u ispisu 3.50.

```
$a="string from malware sample"
```

Ispis 3.50: Definicija niza znakova unutar Yara pravila

U svrhu finog podešavanja pretrage, također je moguće dodati određene ključne riječi zvane modifikatori nakon deklariranog niza znakova. Modifikator `fullword` nakon deklariranog niza znakova uzrokovat će podudaranje samo s točnom riječi, primjerice pravilo prikazano u ispisu 3.51 uzrokovat će podudaranje kod niza znakova `'www.malwarestring.com'`, ali ne i kod niza znakova `'www.abcmalwarestring.com'`.

```
$a="malwarestring" fullword
```

Ispis 3.51: Fullword modifikator Yara niza znakova

Modifikator `wide` uzorkovat će podudaranje kod unicode nizova znakova koji su razdvojeni null bajtovima, primjerice pravilo prikazano u ispisu 3.52 uzrokovat će podudaranje kod niza znakova w.w.w..m.a.l.w.a.r.e.s.t.r.i.n.g..c.o.m.

```
$a="malwarestring" wide
```

Ispis 3.52: Wide modifikator Yara niza znakova

Modifikator `wide ascii` u pravilu će omogućiti podudaranje i kod unicode i kod ascii znakova, a modifikator `nocase` uzrokovat će podudaranje ne uzimajući u obzir radi li se o malom ili velikom slovu.

Sekcija pravila `strings` definira kriterije pretrage koji će biti korišteni za Yara pravilo, a `conditions` sekcija definira način na koji će kriteriji za pravilo biti okidač za uspješno podudaranje. Provjeru zaglavlja datoteke uvijek je korisno uključiti u uvjete Yara pravila pa primjerice uvjet prikazan u ispisu 3.53 provjerava radi li se o Windows izvršnoj datoteci, budući da se heksadekadske vrijednosti 4D 5A uvijek nalaze na početku takve datoteke, pri čemu se ovdje koristi *little-endian* poredak, a uvjet prikazan u ispisu 3.54 provjerava radi li se o Linux ili MacOS binarnoj datoteci.

```
uint16(0) == 0x5A4D
```

Ispis 3.53: Yara uvjet za detekciju Windows izvršne datoteke

```
uint32(0)==0x464c457f) or (uint32(0) == 0xfeedfacf) or (uint32(0) ==
0xcffaedfe) or (uint32(0) == 0xfeedface) or (uint32(0) == 0xcefaedfe)
```

Ispis 3.54: Yara uvjet za detekciju Linux i MacOS binarne datoteke

Također, uvjeti Yara pravila također mogu ispitivati duljine datoteke kako je prikazano u ispisu 3.55.

```
(filesize>512)

(filesize<5000000)

(filesize<5MB)
```

Ispis 3.55: Yara uvjet za ispitivanje duljine datoteke

Nakon što su nizovi znakova deklarirani unutar `strings` sekcije, moguće je unutar `conditions` sekcije specificirati koliko i kojih podudaranja nizova znakova mora biti ostvareno kako bi se smatralo da je pravilo uspješno zadovoljeno, pri čemu je poželjno koristiti nekoliko različitih skupina uvjeta kako bi se kreiralo pouzdano pravilo uz izbjegavanje lažno pozitivnih rezultata. Primjeri vidljivi u ispisu 3.56. pokazuju uvjete u kojima se koriste prethodno definirani nizovi.

```
2 of ($a,$b,$c)

3 of them

4 of ($a*)

all of them

any of them

$a and not $b
```

Ispis 3.56: Yara uvjeti koji koriste definirane nizove znakova

Primjer u ispisu 3.57 prikazuje Yara pravilo koje detektira replica watches adware koji prikazuje neželjene oglase na jednoj od analiziranih stranica iz baze podataka www.e-skole.hr.

```
rule replica_watches_malware

{

    strings:

        $a = "replica watches" wide ascii

        $b = "www.replica-watches.is" wide ascii

        $c = "www.fake-watches.is" wide ascii

        $d = "cc-click-11"

        $e = "cc-contents-11"

        $f = "eval"

        $g = "String.fromCharCode"

        $h =
"cc|11|document|getElementById|contents|style|click|display|oInner|even
t|var|obox|block|target|getAttribute|id|if|function|none|addEventListen
er|else|relative|parentNode|position"

      condition:

        all of them

}
```

Ispis 3.57: Primjer Yara pravila

3.4.2 Implementacija algoritma zasnovanog na Yara pravilima

Kao prvi korak pripreme za implementaciju detekcije zlonamjernih stranica Yara pravilima, potrebno je izvršiti instalaciju i usvojiti načine korištenja Python biblioteke `yara-python` koja ima za ulogu olakšati primjenu Yara pravila [44]. Kako bi se `yara` modul mogao koristiti, potrebno ga je instalirati naredbom prikazanom u ispisu 3.58. u ljusci operacijskog sustava.

```
pip install yara-python
```

Ispis 3.58: Instalacija Yara Python biblioteke

Yara modul se uvozi u pozivajući modul Python koda naredbom prikazanom u ispisu 3.59.

```
import yara
```

Ispis 3.59: Umetanje yara modula u Python program

Objekt Yara pravila generira se pozivanjem funkcije `compile` modula `yara` kojoj se predaje odgovarajuća definicija Yara pravila, kao što je prikazano u ispisu 3.60.

```
rule = yara.compile(source='rule foo: bar {strings: $a = "lmn" condition: $a}')
```

Ispis 3.60: Generiranje objekta Yara pravila

U ovom slučaju pravilo (engl. *rule*) se naziva `'foo'` i ima oznaku podudaranja (engl. *tag*) `'bar'` te sadrži uvjet koji predstavlja pojavljivanje niza znakova `'lmn'` unutar podataka koji se provjeravaju, a koji je definiran kao varijabla `$a` u sekciji `strings`.

Lista podudaranja unutar analiziranih podataka dobiva se pozivanjem funkcije `match` nad objektom Yara pravila kojoj se predaju određeni podaci `data`, odnosno niz znakova nad kojim se Yara pravila provjeravaju, kako je prikazano u ispisu 3.61.

```
matches = rule.match(data='abcdefgjiklmnoprstuvwxyz')
```

Ispis 3.61: Dobivanje liste podudaranja podataka s Yara pravilima

Kad se lista ispisuje kao takva, dobivaju se redom nazivi pravila koja su se našla kao podudaranja s predanim nizom znakova, a primjer izlaza dobivenog kod ispisa liste podudaranja prikazan je u ispisu 3.62.

```
print(matches)
[foo]
```

Ispis 3.62: Lista podudaranja s Yara pravilima

Dodatno, svaki element liste podudaranja ima atribute `rule`, `tags` i `strings`. Atribut `rule` u ovom slučaju ispisuje da je pronađeno poklapanju u skladu s pravilom 'foo', atribut `tags` ispisuje listu oznaka pravila prema kojima je nađeno podudaranje, u ovom slučaju je to jedna oznaka 'bar', a `strings` daje podatke o konkretnih nizovima znakova, odnosno uvjetima koji su uzrokovali podudaranje navodeći informaciju o imenu uvjeta za kojeg je pronađeno podudaranje, nizu znakova koji je ostvario podudaranje i poziciju unutar ulaznih podataka gdje je podudaranje pronađeno.

Prikaz svih relevantnih atributa Yara podudaranja dan je u ispisu 3.63.

```
print(matches[0].rule)
print(matches[0].tags)
print(matches[0].strings)

foo
['bar']
[(10L, '$a', 'lmn')]
```

Ispis 3.63: Relevantni atributi Yara podudaranja

Yara specificira pravila kako detektirati zlonamjerni binarni kod, ali se koristiti i za detekciju zlonamjernog koda na Web stranicama i kompromitiranih Web stranica. Skup pravila zvan 'Burp-yara-rules' predstavlja pravila koja se mogu koristiti za detekciju na Web stranicama i koja će biti korištena za detekciju zlonamjernih Web stranica u okviru implementacije ovog algoritma u diplomskom radu [41]. Burp-yara-rules sadrži kolekciju Yara pravila koja je izgrađena na temelju zlonamjernih uzoraka koda pronađenih na Web stranicama te Yara pravila koja su kreirale treće strane, a koja pronalaze maliciozan software koji se često može pronaći na Web stranicama. Pravila koja se nalaze u navedenom repozitoriju traže znakove inficiranog HTML koda, CSS koda, JavaScript koda te inficiranih PDF, Silverlight XAP, JAR te Flash SWF datoteka.

Za konkretnu primjenu u ovoj implementaciji bit će dovoljna pravila koja nalaze znakove inficiranog HTML, JavaScript i CSS koda. Kako bi se ostvarila implementacija algoritma, potrebno je skinuti skup svih Yara pravila s repozitorija [45], prekopirati sadržaj u datoteku `yara_rules.yar` unutar mape za implementaciju te ukloniti sva Yara pravila koja se ne odnose na HTML, JavaScript ili CSS, dakle odnose se na PDF, Jara, Flash, Silverlight ili Office. Suvišna pravila se mogu detektirati prema njihovom sadržaju za kojeg je jasno da ne upućuje na HTML, CSS, odnosno JavaScript nego neke druge specifične dokumente, prema njihovom nazivu ili `tag` polju, po polju `sample_filetype` u `meta` sekciji pravila ili po nazivu datoteke na listi drugih .yar datoteka u koju je pojedino pravilo na repozitoriju koji se promatra svrstano. Niz znakova `'java'` u nazivu ukazuje na povezanost s JAR datotekama, a `'quicktime'` i `'swf'` ukazuju na povezanost s Flash objektima.

Primjeri pravila koja se mogu ukloniti su `blackhole2_pdf`, `malicious_author : PDF`, `suspicious_version : PDF` i `suspicious_creation : PDF` koja se odnose na PDF dokumente, `blackhole2_jar`, `blackhole2_jar`, `bleedinglife2_jar2` i `fragus_js_java` koja se odnose na JAR dokumente, `angler_flash`, `bleedinglife2_adobe_2010_1297_exploit` i `fragus_js_quicktime` koja se odnose na Adobe Flash dokumente, `cve_2013_0074` koje se odnosi na Silverlight dokumente jer sadrži niz znakova SilverApp1 te `maldoc_indirect_function_call_1 : maldoc`, `maldoc_indirect_function_call_2: maldoc` i `maldoc_find_kernel32_base_method_2: maldoc` koja se odnose na Office dokumente.

Nakon uklanjanja bespotrebnih pravila, preostala se pravila mogu grupirati u sljedeće ranjivosti i kompromitacije na koje se ona odnose: 0x88 Exploit Kit, Angler EK Exploit Kit, Eleonore Exploit Kit, Fragus Exploit Kit, Phoenix Exploit Kit, Zeus Exploit Kit, ZeroAccess Exploit Kit, BlackHole2 Exploit Kit, osnovne detekcije uzoraka kompleta za eksploataciju i općenite zlonamjernosti JavaScript koda.

Pod osnovne detekcije uzoraka kompleta za eksploataciju spadaju `redkit_bin_basic` koji predstavlja detekciju Red Kit Exploit Kita te `blackhole_basic` koji predstavlja detekciju Blackhold Exploit Kita.

Većina Yara pravila otkriva uključivanja određenih kompleta za eksploataciju. Kompleti za eksploataciju (engl. *exploit kit*) [46] su automatizirane prijetnje koje koriste kompromitirane stranice kako bi preusmjeravale internetski promet, tražile ranjive aplikacije internetskog preglednika i pokretale zlonamjerni software. Koriste se kako bi automatski i neprimjetno iskoristili ranjivosti na računalima žrtava tijekom pretraživanja Weba, a postali su jedne od najpopularnijih metoda među kriminalnim skupinama za distribuciju masovnih zloćudnih programa i alata za udaljenu kontrolu. Kompleti za eksploataciju također mogu biti učinkoviti izvor profita za zlonamjerne sudionike, budući da korištenje tih kompleta nude drugim kriminalnim skupinama kao uslugu koja se naplaćuje na mjesečnoj bazi i čija cijena može doseći tisuće američkih dolara mjesečno. Napadači koriste komplete za eksploataciju kako bi uspostavili kontrolu nad uređajem na automatski i pojednostavljen način, a kako bi infekcija bila uspješna, niz događaja se mora uspješno odviti, uključujući početnu stranicu, izvršavanje iskorištavanja ranjivosti i dostavu korisnog tereta (engl. *payload*) te svaka faza mora biti uspješno izvršena kako bi se preuzela kontrola nad računalom žrtve.

Kompleti za iskorištavanja ranjivosti počinju svoje djelovanje na stranici koja mora biti kompromitirana te će ona diskretno preusmjeriti internetski promet na drugu početnu stranicu gdje se nalazi programski kod koji će analizirati postojanje mogućih ranjivosti preglednika na uređaju žrtve. Promet i djelovanje kompleta za eksploataciju ranjivosti tada će prestati ako uređaj sadrži najnoviji software iz kojeg su uklonjene ranjivosti, a ako su ranjivosti pronađene, kompromitirana će stranica diskretno preusmjeriti promet prema izvoru ranjivosti.

Kod za iskorištavanje ranjivosti koristi ranjivu aplikaciju kako bi potajice pokrenuo zlćudni program na računalu, a aplikacije koje su na meti uključuju Adobe Flash Player, Java Runtime Environment i Microsoft Silverlight. Njihovi izvori ranjivosti su datoteke, primjerice JAR ili SWF datoteke, i internetski preglednik čiji je kod za iskorištavanje ranjivosti poslan kao kod putem internetskog prometa.

Ako je iskorištavanje ranjivosti uspješno, komplet za iskorištavanje ranjivosti šalje korisni teret kako bi zarazio računalo domaćina, a korisni teret može biti software koji preuzima drugi zlćudni program ili sam ciljani zlćudni program koji se putem iskorištavanja ranjivosti želi poslati. Napredniji alati omogućuju slanje korisnog tereta kao kriptiranog binarnog sadržaja preko mreže koji se dekriptira i izvršava na računalu žrtve. Najčešći korisni teret je ucjenjivački zlćudni program (engl. *ransomware*), ali prisutni su i zlćudni program mreže kompromitiranih računala (engl. *botnet malware*), zlćudni program za krađu informacija i bankarski Trojanac, odnosno financijski zlćudni program.

Budući da kompleti za iskorištavanje ranjivosti postaju sve učestalije korišteni među napadačima različitih vještina, važno je zaštiti sustave od napada, a to se može postići tako da se smanji površina napada, blokiraju poznati zlćudni programi i kodovi za iskorištavanje ranjivosti te brzo identificiraju i zaustave nove prijetnje putem različitih sustava koji su za to specijalizirani.

Browser Exploatation Framework (BeeF) [47] je alat za penetracijsko testiranje koji se može koristiti i u zlonamjerne svrhe ako se pokrene bez znanja žrtve na njenom računalu, a fokusira se na sam internetski preglednik. Kao penetracijski alat koristan je otkrivanju ranjivosti na napade bazirane na webu na klijentske uređaje, uključujući mobilne uređaje, ali isto tako može biti korišten da bi otkrio i iskoristio potencijalne ranjivosti žrtvinog računala. Njegov je cilj ispitati sigurnosno stanje ciljane okoline, no umjesto testiranja ranjivosti na niskoj mrežnoj razini klijentskog sustava, tražit će ranjivosti direktno u kontekstu Web preglednika.

Priključivanje na preglednik (engl. *browser hooking*) je proces u kojem se BeeF poveže na jedan ili više preglednika kako bi njih koristio kao sredstva za pokretanje direktnih komandnih modula i daljnjih napada protiv sustava iz samog konteksta internetskog preglednika.

Posebno će se objasniti općenite zlonamjernosti JavaScript koda koje su sadržane u nekoliko vrlo bitnih pravila.

Pravilom `src_ptheft_command` može se detektirati pokušaj krađe korisničkih podataka pojavljivanjem lažne autentifikacijske forme koja traži upis podataka za prijavu na Apple račun. JavaScript obfuskacija otkrivena pravilom `generic_javascript_obfuscation` pretvara standardni i čitljivi JavaScript kod u kod koji je teško čitljiv ljudima kao i automatiziranim alatima za analizu JavaScript funkcija, ali izvršava iste funkcionalnosti. Često je svrha obfuskacije prikrivanje prisustva zlonamjernog koda ili dostave zlonamjernog

sadržaja. JavaScript uključivanje ili paketizirane funkcije predstavljene su pravilom `possible_includes_base64_packed_functions`. Slično procesu minifikacije, funkcije se mogu uključiti kodirane u base64 formatu ili paketizirane na određen sažeti način te ih funkcije poput `eval` onda pretvaraju u kod na klijentskoj strani i izvršavaju te je ovakvo uključivanje zapravo jedan poseban vid obfuskacije koja je dobar pokazatelj zlonamjernosti. U praski se i na benignim stranicama paketiziranje funkcija JavaScript *packer* alatima može koristiti kao alternativa minifikaciji koja stvara manje skripte s istom funkcionalnošću, ali uz sporije izvršavanje koje uključuje i proces otpakiranja (engl. *unpack*) [48]. Pravilo `BeEF_browser_hooked` u ovom skupu Yara pravila predstavlja priključivanje na preglednik korištenjem Browser Exploitation Frameworka.

Kako bi se omogućila implementacija algoritma koji detektira stranice koristeći Yara pravila, potrebno je određene funkcionalnosti, uključujući učitavanja Yara pravila iz datoteke i obradu rezultata poklapanju izdvojiti u pomoćni modul `yarachecker`.

Unutar modula najprije je potrebno uvesti modul `yara` naredbom prikazanom u ispisu 3.64 kako bi se sve potrebne funkcionalnosti omogućile.

```
import yara
```

Ispis 3.64: Umetanje yara modula u Python programski kod

Zatim je potrebno implementirati funkciju koja će iz putanje Yara pravila generirati objekt Yara pravila s kojim će se provjeravati zadane HTML stranice.

Funkcija će se implementirati kako je prikazano u ispisu 3.65 i njeno pozivanje je preduvjet za bilo koje druge operacije nad modulom `yarachecker`.

```
def setYaraRulesObject(path):
    global yaraRules
    yaraRulesFile = open(path, encoding='utf-8')
    yaraRulesString = yaraRulesFile.read()
    yaraRulesFile.close();
    yaraRules = yara.compile(source = yaraRulesString)
```

Ispis 3.65: Postavljanje objekta Yara pravila

Iduća funkcija koju treba implementirati jest funkcija `getYaraMatches` koja iz postavljenog objekta Yara pravila i HTML niza znakova generira listu eventualnih poklapanja koja su pronađena, a koja bi mogla ukazivati na zlonamjernost Web stranice.

Ako se funkcija zove prije nego što je postavljen objekt Yara pravila, generirat će se iznimka `YaraRulesNotSetException` koja govori da Yara pravila unutar modula još nisu postavljena, kao što je prikazano u ispisu 3.66.

```
class YaraRulesNotSetException(Exception):
    pass
```

Ispis 3.66: Generiranje iznimke kada objekt Yara pravila nije postavljen

Funkcija `getYaraMatches` koja generira listu svih poklapanja bit će implementirana kako je prikazano u ispisu 3.67.

```
def getYaraMatches(html):
    if yaraRules is None:
        raise YaraRulesNotSetException("Yara rules object not set")
    matches = yaraRules.match(data=html)
    return matches
```

Ispis 3.67: Generiranje liste poklapanja sadržaja s Yara pravilima

Također, unutar modula `yarautils` treba izdvojiti pravila pridruživanja koja će određenim Yara pravilima pridružiti objašnjenja zlonamjernosti stranica koristeći objekt rječnika naziva `rulesInterpretation`. Pritom je bitno napomenuti da će više Yara pravila ukazivati na isto objašnjenje zlonamjernosti, primjerice i `angler_html` i `angler_html2` pravilo ukazuju na detekciju Angler Exploit Kita u HTML kodu.

`RulesInterpretation` objekt bit će realiziran kako je prikazano u ispisu 3.68.

```
rulesInterpretation = {
'zerox88_js2': '0x88 Exploit Kit detected in JavaScript code',
'zerox88_js3': '0x88 Exploit Kit detected in JavaScript code',
'angler_ek_checkpoint': 'Angler Exploit Kit checkpoint detected',
'AnglerEKredirector': 'Angler Exploit Kit redirector detected',
'angler_html': 'Angler Exploit Kit detected in HTML code',
'angler_html2': 'Angler Exploit Kit detected in HTML code',
```

```
'angler_js': 'Angler Exploit Kit detected in JavaScript code',
'blackhole_basic': 'Blackhole Exploit kit usage detected',
'eleonore_js': 'Elenore Exploit Kit detected in JavaScript code',
…
'fragus_htm': 'Fragus Exploit Kit detected in HTML code',
'fragus_js': 'Fragus Exploit Kit detected in JavaScript code',
…
'generic_javascript_obfuscation': 'JavaScript obfuscation detected',
'possible_includes_base64_packed_functions ': 'Possible includes and
packed functions detected',
'BeEF_browser_hooked': 'Possible browser hooking detection using
Browser Exploatation Framework detected',
'src_ptheft_command': 'Possible confidential data theft using Browser
Exploitation Framework detected',
'phoenix_html': 'Phoenix Exploit Kit detected in HTML code',
…
'redkit_bin_basic': 'Red Kit Exploit Kit usage detected',
'zeus_js': 'Zeus Exploit Kit detected in JavaScript code',
'zeroaccess_htm': 'Zeroaccess Exploit Kit detected in HTML code',
…
'blackhole2_css': 'Blackhole Exploit Kit detected in CSS code',
'blackhole2_htm': 'Blackhole Exploit Kit detected in HTML code',
…
}
```

Ispis 3.68: Pridruživanje objašnjenja podudaranjima s pojedinim Yara pravilom

Modul `yarachecker` te interpretacija pronađenih podudaranja moraju se uvesti u glavni modul za provjeru zlonamjernosti stranica `pagechecker` naredbama prikazanima u ispisu 3.69.

```
import yarachecker
from yarachecker import rulesInterpretation
```

Ispis 3.69: Umetanje pomoćnih Yara funkcionalnosti u modul provjere stranica

Funkcija koja provjerava zlonamjernost stranice korištenjem algoritma zasnovanog na Yara pravilima prolazit će kroz listu pronađenih podudaranja s pravilima te generirati listu dijagnoza zlonamjernosti. Ako je lista prazna, stranica će se označiti kao benigna, a ako lista nije prazna, stranica će se označiti kao zlonamjerna te će se elementi ulančati u opis odvojeni točka-zarezom nakon eliminacije duplih elemenata iz liste.

Funkcija algoritma bit će implementirana kako je prikazano u ispisu 3.70.

```
def checkByYaraRules(html, yaraRulesPath):
    …
    matches = yarachecker.getYaraMatches(html)
    maliciousProperties = []
    for match in matches:
        matchedRule = match.rule
        maliciousProperty = rulesInterpretation[matchedRule]
        maliciousProperties.append(maliciousProperty)
    if len(maliciousProperties) > 0:
        analysisResults['status'] = 'malicious'
        maliciousProperties = list(dict.fromkeys(maliciousProperties))
        analysisResults['details'] = "; ".join(maliciousProperties)
    else:
        analysisResults['status'] = 'benign'
        analysisResults['details'] = 'no malicious properties detected'

    return analysisResults
```

Ispis 3.70: Implementacija algoritma zasnovanog na Yara pravilima

Postoji stranica za koju se zna da je zlonamjerna, a radi se o stranici www.e-skole.hr koja sadrži skriveni HTML s reklamnim materijalima o replikama satova s pripadnim CSS-om koji se aktivira određenim isječkom kompromitiranog JavaScript koda umetnutog na stranicu.

Cilj je prilagoditi algoritam Yara pravila tako da detektira navedenu zlonamjernost na toj stranici i potencijalno na nekim drugim stranicama ako takva zlonamjernost postoji. To je moguće relativno jednostavno postići dodavanjem određenog Yara pravila, budući da taj algoritam može naći konkretne nizove znakova koji ukazuju na zlonamjernost, a ne ovisi o općenitim obilježjima statičkih stranica kao što je slučaj kod statičke heuristike i algoritma zasnovanog na mehanizmu bodovanja.

Analizom sadržaja HTML koda stranice mogu se izdvojiti određeni nizovi znakova koji su karakteristični za navedenu zlonamjernost:

`'replica watches'` - fraza koja upućuje na navedenu zlonamjernost

`'www.replica-watches.is'` - Web stranica do koje vode poveznice prikazane u HTML-u kao rezultat navedene zlonamjernosti

`'www.fake-watches.is'` - Web stranica do koje vode poveznice prikazane u HTML-u kao rezultat navedene zlonamjernosti

`'cc-click-11'` - klasa div oznake unutar koje se nalazi navedeni zlonamjerni sadržaj u statičkom HTML-u

`'cc-contents-11'` - klasa div oznake unutar koje se nalazi navedeni zlonamjerni sadržaj u statičkom HTML-u

`'eval'`, `'String.fromCharCode'` - funkcije koje upućuju na pojavljivanje obfusciranog JavaScript koda

`'cc|11|document|getElementById|contents|style|click|display|oInner|even t|var|obox|block|target|getAttribute|id|if|function|none|addEventListen er|else|relative|parentNode|position'` - niz znakova koji sadrži karakteristični uzorak obfusciranog JavaScript koda gdje se spominje i 'cc' te '11' koji su fragmenti imena karakterističnih klasa u kodu

Navedeno Yara pravilo koje će se dodati u `yara_rules.yar` može se napisati kako je prikazano u ispisu 3.71.

```
rule replica_watches_malware {
    strings:
        $a = "replica watches" wide ascii
        $b = "www.replica-watches.is" wide ascii
        …
        $h =
"cc|11|document|getElementById|contents|style|click|display|oInner|even
t|var|obox|block|target|getAttribute|id|if|function|none|addEventListen
er|else|relative|parentNode|position"
      condition:
        all of them
}
```

Ispis 3.71: Yara pravilo za detekciju replica watches adwarea

Također je potrebno u modulu `yarachecker` dodati u objekt rječnika `rulesInterpretation` ključ `'replica_watches_malware'` s vrijednošću `'Replica watches adware detected'`.

4 Testiranje implementacije algoritama

4.1 Metodologija i testni skupovi podataka

Prije obavljanja samog testiranja i generiranja ukupnih rezultata te njihove statističke analize i analize pojedinačnih detekcija, potrebno je pripremiti testni skup podataka.

Za testiranje implementiranih algoritama s optimalnim performansama, potrebno je izdvojiti sve validne HTML stranice iz baze te ih pohraniti lokalno na računalu, što će uštedjeti vrijeme testiranja i znatno olakšati ponavljajuća testiranja. Samo povlačenje pojedine stranice iz baze podataka znatno je dugotrajnije od testiranja zlonamjernosti te također stabilnost procesa testiranja ne bi smjela ovisiti o stabilnosti i brzini internetske veze koje u slučaju povlačenja iz baze podataka imaju značajnu ulogu.

Taj se dohvat neće implementirati tako da se iz baze podataka povlači specificirani broj sirovih HTML stranica za koje regularni izraz utvrdi da predstavljaju validni HTML kao što je napravljeno u prethodnim metodama povlačenja od kojih se odustalo, nego će na temelju kolekcije pohranjenih URL-ova koji su provjeravani dohvatiti sve takve stranice iz baze podataka koje se referenciraju na zadnju provjeru URL-ova.

Kasnije će se unutar modula `pagefetch` koji služi za dohvat HTML stranica također implementirati odgovarajuća funkcija koja na temelju putanje direktorija gdje su stranice lokalno pohranjene kreira listu HTML stranica za analizu.

Funkcionalnost dohvaćanja iz baze podataka bit će implementirana unutar funkcije `fetchMongoHTMLDatabase` te se neće uzimati u obzir dano ime kolekcije iz koje se pojedinačne stranice dohvaćaju nego će se implementirati proizvoljna logika koja uzima u obzir više kolekcija, generira listu relevantnih ID-ova stranica i zatim vrši pojedinačno dohvaćanje stranica.

Prvi korak jest dohvaćanje liste URL-ova koji nemaju postavljene atribute `redirects_to` i `disabled` iz kolekcije `url`. To znači da ne preusmjeravaju nego pokazuju na konačne i validne HTML stranice te nije označeno da ih sustav ne provjerava i da se ne bi trebali uzimati u obzir.

Pritom je iz ove kolekcije relevantno dohvatiti samo listu URL-ova, a ne i njihove ID-eve, stoga se u specifikaciji liste polja koja će se uzimati iz odgovarajućih dokumenata iz kolekcije URL-ova označava da se uzima podatak o URL-u, ali ne i podatak o ID-u, i to upitom prikazanim u ispisu 4.1.

```
{'_id': 0, 'url' : 1}
```

Ispis 4.1: MongoDB upit za odabir liste stranica s URL-ovima bez ID-a

Nakon toga koraka dobiveni broj URL-ova iz baze podataka koji valja proći je oko 143 600 od 267 800 URL-ova jer su isključeni URL-ovi koji su onemogućeni ili koji sadrže atribut preusmjeravanja.

U drugom koraku, iz kolekcije `crawled_data_urls_v0` potrebno je dohvatiti hash zadnje provjere svakog od tih URL-ova kako bi se u konačnici znalo koju stranicu iz kolekcije Web stranica dohvatiti, pod uvjetom da provjere da za taj URL postoje. Budući da one ne postoje za sve URL-ove, broj konačno dohvaćenih stranica bit će manji od 143 600.

To se postiže tako da se iz polja `checks` svakog od URL-ova dohvaća zadnji objekt te njegov `hash` atribut kako bi se u konačnici znalo koju stranicu dohvatiti iz baze podataka, a u tom slučaju `_id` atribut također ne treba dohvaćati, što se u pretrazi kolekcije treba označiti upitom prikazanim u ispisu 4.2.

```
{'_id': 0, 'checks' : 1}
```

Ispis 4.2: MongoDB upit za odabir liste provjera stranica bez ID-a

Posljednji korak je dohvat `page` atributa iz kolekcije `crawled_data_webpages_v0` koji predstavlja HTML sadržaj stranice, pri čemu se u listi dohvaćenih stranica sprema URL dohvaćenog sadržaja te ID stranice iz kolekcije `crawled_data_pages_v0`.

Pritom će se sadržaj stranica zapisivati u direktorij `database_webpages` gdje će nazivi datoteka biti ID-ova iz kolekcije `crawled_data_pages_v0`, a URL-ovi stranica u direktorij `database_webpage_urls` gdje će u datoteku čije je ime ID stranice biti zapisan URL koji odgovara ID-u stranice. Zapisivanje URL-a stranica korisno je kako bi se moglo znati o kojem se URL-u radi ako se pojedina stranica detektira kao zlonamjerna.

Pri zapisivanju sadržaja u datoteke potrebno je taj sadržaj kodirati u UTF-8 kako ne bi došlo do pogreške pri zapisivanju, odnosno datoteka se treba otvoriti s UTF-8 kodiranjem pa se funkciju zapisivanja treba implementirati kako je prikazano u ispisu 4.3.

```
def writeHTMLToFile(path, html):
    htmlFile = open(path, "w", encoding='utf-8')
    htmlFile.write(html)
    htmlFile.close()
```

Ispis 4.3: Zapis HTML stranice u datoteku

Budući da je cijeli proces dohvaćanja dugotrajan i može biti prekinut zbog problema s vezom s bazom podataka, korisna je mogućnost da se dohvaćanje nastavi nakon određenog broja dohvaćenih stranica i zato je poželjno dodati opcionalan parametar `offset` kako bi se određeni broj već dohvaćenih URL-ova mogao preskočiti tijekom dohvaćanja iz kolekcije

URL-ova `urls`. Ako se želi ostaviti mogućnost dohvaćanja manjeg broja stranica, odnosno dohvaćanje dio po dio, poželjno je i omogućiti postavljanje parametra `numberOfPages`.

Ovakvo dohvaćanje iz baze podataka implementirano je kako je prikazano u ispisu 4.4.

```
def fetchMongoHTMLDatabase(numberOfPages = 268000, offset = 0):
    …
    webpagesDatabase = mongoConnection[webpagesDatabaseName]
    …
    urlsCollection = webpagesDatabase['urls']
    urlCrawlsCollection = webpagesDatabase['crawled_data_urls_v0']
    webpagesCollection = webpagesDatabase['crawled_data_pages_v0']
    urlsSetCursor = urlsCollection.find({'redirects_to' : { '$exists' :
    False }, 'disabled' : { '$exists' : False }},{'_id': 0, 'url' :
    1}).skip(offset).limit(numberOfPages)
    for url in urlsSetCursor:
        urlsList.append(url['url'])
        webpageUrl = url['url']
        urlCrawlsCursor = urlCrawlsCollection.find({'url' : { '$eq' :
        webpageUrl}},{'_id': 0, 'checks' : 1})
        for urlCrawl in urlCrawlsCursor:
            hash = urlCrawl['checks'][-1]['hash']
            if hash is not None:
                webpagesCursor = webpagesCollection.find({'hash' : {
                '$eq' : hash }},{'page' : 1})
                for webpage in webpagesCursor:
                    htmlPath = 'database_webpages/' +
                    str(webpage['_id'])
                    urlPath = 'database_webpage_urls/' +
                    str(webpage['_id'])
                    writeHTMLToFile(htmlPath, webpage['page'])
                    writeHTMLToFile(urlPath, webpageUrl)
```

Ispis 4.4: Dohvaćanje skupa stranica iz MongoDB baze podataka

Nakon provedenog postupka dohvaćanja, rezultat je 75 492 HTML dokumenta pohranjeni svaki u svojoj datoteci i za svaki od njih postoji datoteka u kojoj se sprema njegov odgovarajući URL.

Na ovaj način dohvaćeni podaci kasnije će se predavati funkciji za usporedbu dvaju algoritama i funkciji za skupnu analizu svih algoritama.

Za dohvaćanje stranica u listu koja će se predavati algoritmima za testiranje koristit će se funkcija koje će iz putanje direktorija gdje su Web stranice pohranjene kao datoteke generirati listu koja će za svaku stranicu imati pohranjen HTML niz znakova i ID izdvojen iz putanje pojedine Web stranice, a ta će funkcija biti implementirana kako je prikazano u ispisu 4.5.

```
def readHTMLListFromDatabaseDir(directory):
    …
    childList = listdir(directory)
    for child in childList:
        fullPath = join(directory, child)
        if isfile(fullPath):
            html = readHTMLFromFile(fullPath)
            htmlList.append({'id': child, 'html': html})
    return htmlList
```

Ispis 4.5: Dohvaćanje liste lokalno pohranjenih stranica iz direktorija

Kako bi učitavanje bilo moguće, potrebno je implementirati funkciju za čitanje sadržaja iz HTML datoteke tako da se kao kodiranje datoteka iz kojih se čita specificira UTF-8, kako je prikazano u ispisu 4.6.

```
def readHTMLFromFile(path):
    htmlFile = open(path, encoding='utf-8')
    html = htmlFile.read()
    htmlFile.close();
    return html
```

Ispis 4.6: Čitanje HTML koda stranice iz datoteke

Nakon pripreme testnih podataka i funkcija za njihovo dohvaćanje u listu koja će se predavati testovima algoritama, testiranje će se provoditi za sva tri algoritma na sve 75 492 stranice. Rezultati testiranja za svaki pojedini algoritam bih će pohranjeni u liste čiji će elementi sadržavati ID stranice, status zlonamjernosti koji može biti `'benign'` ili `'malicious'` te detalje koji mogu biti objašnjenje da je stranica benigna ili dijagnoza zlonamjernosti.

Dijagnoza zlonamjernosti za algoritam statičke heuristike sadržavat će opis odluke u stablu odlučivanja koja je dovela do klasifikacije stranice kao zlonamjerne, dijagnoza zlonamjernosti za algoritam temeljen na mehanizmu bodovanja sadržavat će popis skupina atributa za koje je stranica prešla prag s navedenom vrijednošću rezultata i vrijednošću praga koji je prijeđen, a dijagnoza zlonamjernosti za algoritam temeljen na Yara pravilima sadržavat će popis objašnjenja Yara pravila za zlonamjernost s kojima je pronađeno podudaranje u kodu stranice.

U listi stranica za svaki će se algoritam također izračunati broj i postotak benignih stranica te broj i postotak zlonamjernih stranica, a program će pojedinačne podatke o zlonamjernosti za sve 75 492 stranice te statističke izračune broja i postotka benignih i zlonamjernih stranica izvesti u .csv datoteku.

Koristeći Office 365 Excel alat [49], navedena će se statistika iskoristiti za generiranje dodatnih pivot tablica kako bi se generirale dodatne statistike, poput distribucije dijagnoza zlonamjernosti za svaki od algoritama među stranicama iz cijelog skupa podataka koje je taj algoritam otkrio kao zlonamjerne. Također, tortni dijagrami distribucije dijagnoza zlonamjernosti za algoritam statičke heuristike, algoritam zasnovan na Yara pravilima, optimizirani algoritam statičke heuristike i optimizirani algoritam zasnovan na Yara pravilima generirati su online alatom Meta-Chart [50] te su iskorišteni za dodatnu vizualizaciju.

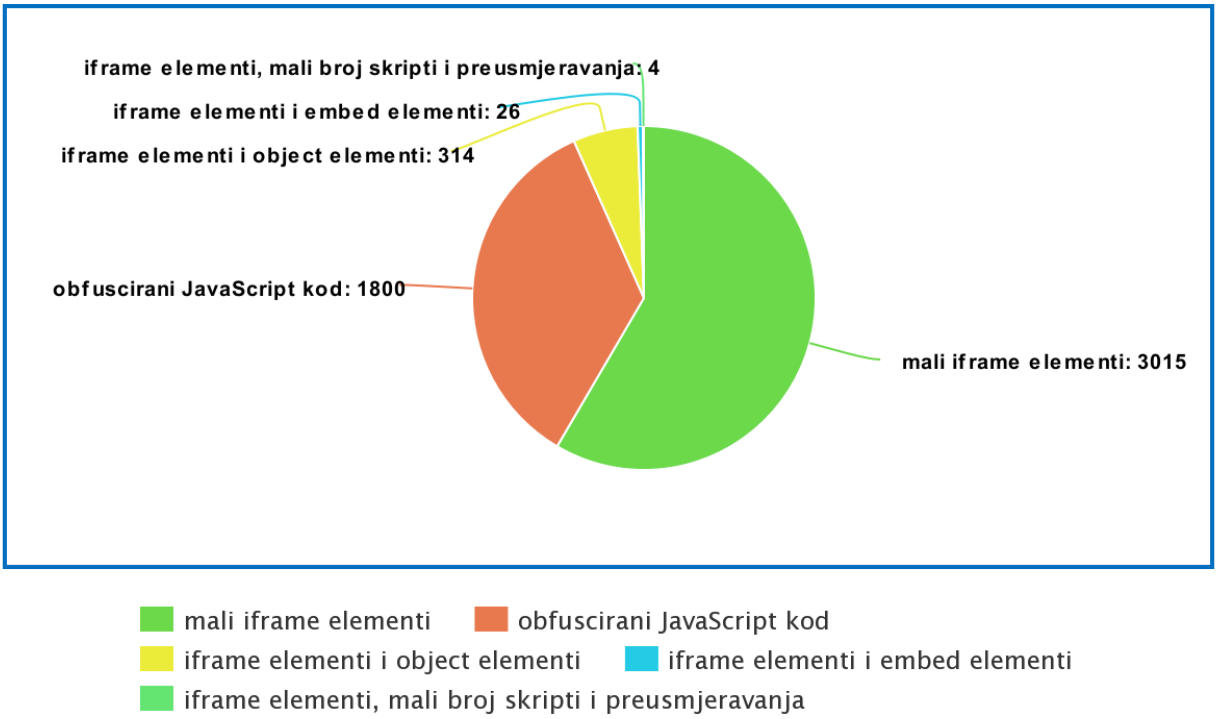
Brojevi i udjeli benignih i zlonamjernih stranica pojedinih algoritama implementiranih u diplomskom radu prikazani su u tablici 4.1, a detaljniji pregled rezultata i konkretnih dijagnoza zlonamjernosti bit će dan u poglavljima 4.2, 4.3 i 4.4.

Algoritam	**Ukupni broj stranica**	**Broj benignih stranica**	**Broj zlonamjernih stranica**	**Udio benignih stranica**	**Udio zlonamjernih stranica**
Statička heuristika	75 492	70 333	5 159	93.17%	**6.83%**
Mehanizam bodovanja	75 492	66 928	8 564	88.66%	**11.34%**
Yara pravila	75 492	70 005	5 487	92.73%	**7.27%**

Tablica 4.1: Statistika benignih i zlonamjernih Web stranica s odabranim algoritmima

4.2 Rezultati algoritma statičke heuristike

Udio zlonamjernih stranica dobivenih analizom algoritmom statičke heuristike iznosi 6.83%, odnosno 5159/75492 stranica. Dijagnoze zlonamjernosti koje se pojavljuju poredane silazno po učestalosti pojavljivanja su mali `iframe` elementi (3015), obfuscirani JavaScript kod (1800), `iframe` element i `object` element (314), `iframe` element i `embed` element (26) te postojanje `iframe` elemenata uz mali broj skripti i prisutnost preusmjeravanja (4). Opisana distribucija zlonamjernosti stranica algoritma statičke heuristike prikazana je na slici 4.1.



Slika 4.1: Distribucija dijagnoza zlonamjernosti algoritma statičke heuristike

Budući da je detektiran vrlo velik broj stranica koje sadrži male iframe elemente, za pretpostaviti je da postoji veliki broj lažno pozitivnih detekcija u toj grupi zlonamjernih Web stranica pa će se dio stranica trebati ručno provjeriti kako bi se otkrile moguće pogreške algoritma i predložile moguće optimizacije.

Pregledat će se nekoliko nasumičnih stranica s tom detekcijom kako bi se ustanovio razlog lažno pozitivne detekcije zlonamjernosti ili stvarna zlonamjernost:

orvasyachting.com, villsy.com i www.taxicroatiatransfer.com.

U svakoj od navedenih stranica pronađen je isječak koda oblika prikazanog u ispisu 4.7.

```
<!-- Google Tag Manager (noscript) -->
<noscript>
```

```
<iframe src="https://www.googletagmanager.com/ns.html?id=ID" height="0"
width="0" style="display:none;visibility:hidden"></iframe>

</noscript>

<!-- End Google Tag Manager (noscript) -->
```

Ispis 4.7: Isječak koda pronađen na stranici s detektiranim malim iframe oznakama

Može se zaključiti da je za optimizaciju algoritma poželjno uvesti listu dozvoljenih domena koje se smiju nalaziti kao izvor u malom `iframe` elementu uz koje stranica ne bi bila detektirana kao zlonamjerna.

Zatim će se analizirati nekoliko nasumičnih stranica gdje je dijagnoza zlonamjernosti obfuscirani JavaScript kod:

www.dom-jalzabet.hr, feid.hr, www.salon-vodotehnika.hr i delmata-travel.hr.

U JavaScript konzoli Chrome preglednika izvršit će se `unescape()` funkcija nad obfusciranim nizovima znakova navedenima u skripti kako bi se otkrio sadržaj, a to se postiže jednostavnim kopiranjem `unescape` izraza iz JavaScript koda.

Na svim ovim stranicama pronađen je unutar `unescape()` funkcije validni CSS kod koji će se dinamički dodati određenoj HTML oznaci.

Zajednički JavaScript kod pronađen na tim stranicama prikazan je u ispisu 4.8.

```
<script>

var htmlDivCss = unescape("obfusciraniCSSKod");

var htmlDiv = document.getElementById('rs-plugin-settings-inline-css');

if(htmlDiv) {

      htmlDiv.innerHTML = htmlDiv.innerHTML + htmlDivCss;

}

else{

      var htmlDiv = document.createElement('div');

            htmlDiv.innerHTML = '<style>' + htmlDivCss + '</style>';

      document.getElementsByTagName('head')[0].appendChild(htmlDiv.chil
dNodes[0]);

}

</script>
```

Ispis 4.8: JavaScript kod stranica s detektiranom JavaScript obfuskacijom

Može se zaključiti da se radi o benignom CSS kodu za RS plugin postavke, a karakteristični niz znakova koji ga identificira je `'rs-plugin-settings-inline-css'`. Taj je dodatak često korišten unutar WordPressa za omogućavanje raznih vizualnih elemenata poput slidera i efekata na stranicama koji mogu značajno poboljšati izgled stranice i korisničko iskustvo [51].

Za optimizaciju algoritma zato je poželjno uvesti listu dozvoljenih uzoraka kako bi se omogućilo da se one skripte koje sadrži obfuscirane znakove i `unescape()` funkciju, ali pritom sadrže i neki od uzoraka iz liste ne smatraju zlonamjernima.

Stranice koje imaju dijagnozu postojanja `iframe` elemenata uz mali broj skripti i prisutnost preusmjeravanja su sljedeće: tumir.hr, policijazaustavlja.com, www.boem-mps.hr i scholar.google.hr.

Stranica tumir.hr sadrži `meta` oznaku za preusmjeravanje na samu tu stranicu koja je prikazana u ispisu 4.9, dakle stranica nije zlonamjerna.

```
<meta http-equiv="refresh" content="60;URL=http://www.tumir.hr" />
```

Ispis 4.9: Meta oznaka za preusmjeravanje na istu stranicu

Stranica policijazaustavlja.com sadrži `meta` oznaku za preusmjeravanje koja uzrokuje osvježavanje svakih 10 minuta, dakle nije zlonamjerna. Navedena oznaka prikazana je u ispisu 4.10.

```
<meta http-equiv="refresh" content="600">
```

Ispis 4.10: Meta oznaka za preusmjeravanje koja uzrokuje osvježavanje svakih 10 minuta

Stranica www.boem-mps.hr sadrži `meta` oznaku za preusmjeravanje koja uzrokuje osvježavanje svakih sat vremena, dakle nije zlonamjerna.

U slučaju kad URL preusmjeravanja u `meta` oznaci za preusmjeravanje nije naveden ili je naveden URL stranice na kojoj se sama oznaka nalazi, ta se oznaka zapravo ne interpretira kao preusmjeravanje, nego kao osvježavanje stranice.

Stranica scholar.google.hr sadrži isječke minificiranog JavaScript koda Closure Library JavaScript biblioteke koji sadrži određene nizove znakova koji ukazuju na preusmjeravanja poput `'window.location.href'` i `'window.location.replace'`. Iako se takva preusmjeravanja izvršavaju unutar kontrolirano pozivanih funkcija, budući da je JavaScript kod minimiziran i da je veliki broj tih funkcija zapisan u sažetog notaciji ili drugačije ubačen u kod, algoritam nije uspio otkriti da se radi o preusmjeravanju definiranom unutar funkcije kao što na jednostavni način može otkriti za većinu drugih preusmjeravanja te je neispravno odredio na temelju tog isječka koda da se radi o spontanom preusmjeravanju. Po tome se može zaključiti da u ovom aspektu algoritam nije potpuno pouzdan i može dati lažno pozitivne rezultate.

Lažne detekcije preusmjeravanja mogu se spriječiti uvođenjem pravila da se meta-refresh oznake koje nemaju označenu domenu ili imaju označenu domenu stranice te koje imaju interval osvježavanja veći od nekog proizvoljnog vremena ne uzimaju u obzir kao potencijalno zlonamjerna preusmjeravanja. Također, poželjno je poboljšati razlikovanje između spontanog preusmjeravanja i preusmjeravanja definiranog unutar funkcija koje se pozivaju korisničkim akcijama uvođenjem nekih preciznijih mehanizama provjere poput dinamičke evaluacije.

Kad se rezultati ovog eksperimenta usporede sa stopom lažno pozitivne detekcije u analiziranom članku koji predstavlja metodu statičke heuristike [24] koja iznosi 5.88%, dobije se približno jednaki broj kao broj zlonamjernih detektiranih stranica od 6.83%, tako da se može zaključiti da je većina detekcija u ovom eksperimentu lažna i može se djelomično smanjiti uvođenjem navedenih optimizacija koje se uglavnom svode na uvođenje određenih listi dozvoljenih uzoraka.

U originalnom članku, algoritam kao takav i nije zamišljen kao algoritam bez lažno pozitivnih detekcija, već kao dio hibridnog sustava u kojem se detekcije algoritmom statičke heuristike prosljeđuju *honeypot* klijentu visoke interaktivnosti koji ima lažno pozitivnu stopu 0% kako bi se eliminirale takve detekcije. Zato se ovakvi rezultati eksperimenta prije provođenja navedenih optimizacija čine očekivanima.

4.3 Rezultati algoritma zasnovanog na mehanizmu bodovanja

Udio zlonamjernih stranica dobivenih analizom algoritmom zasnovanim na mehanizmu bodovanja iznosi 11.34%, odnosno 8564/75492 stranice, što je nešto manje od postotka od 14% navedenog u analiziranom članku, a nešto više od postotka 9.26% koji je postignut pri podešavanju pragova skupina atributa koje je prethodno napravljeno na manjem skupu od 367 stranica.

Prevladavaju stranice koje imaju dijagnozu da su prešle prag iz skupine atributa stranog sadržaja te stranice koje imaju dijagnozu da su prešle prag iz skupine atributa koji se odnose na skripte, određeni broj stranica prešle su oba ta praga, a dio stranica prešao je prag koji se tiče sadržaja iskorištavanja ranjivosti. U nastavku je dan primjer stranica koje prelaze pragove za svaku od grupa atributa.

Stranica mrak.org prešla je prag stranog sadržaja koji iznosi 9.0 s rezultatom od 11.03.

Ta skupina atributa odnosi se na broj preusmjeravanja, broj `iframe` oznaka, statistiku poveznica `iframe` oznaka koju čine udio samoglasnika, udio specijalnih znakova i duljina te statistiku vanjskih poveznica koju čine broj i minimalna duljina vanjskih poveznica.

Broj `iframe` oznaka na stranici je 6, što je relativno visoko te one sadrže poveznice na YouTube video zapise koje nemaju naročito veliki broj znakova niti udio samoglasnika ili specijalnih znakova.

Što se tiče vanjskih poveznica, postoji desetak vanjskih poveznica unutar teksta na stranici koje su generirali sami autori i pokazuju na legitimne domene. Uglavnom se radi o .hr domenama, ali 2 poveznice uključuju Facebook i Twitter tipku za dijeljenje sadržaja, a dio poveznica su skripte i CSS datoteke koje su referencirane apsolutnom putanjom s mrak.org domene pa se radi o njihovoj lažnoj detekciji kao vanjskih poveznica.

Stranica je ručnom analizom procijenjena kao benigna, a niti jedan od druga dva algoritma nije ju klasificirao kao zlonamjernu.

Mogućnost poboljšanja ovog algoritma kojom se dio ovakvih stranica može ukloniti s popisa lažnih detekcija je mogućnost detekcije poveznica s apsolutnom putanjom kao unutarnjih čak i bez poznate domene analizirane stranice. Ako velika većina poveznica na stranici općenito ili unutar neke skupine oznaka pokazuje na istu domenu, onda se može pretpostaviti da se učitavaju s iste domene na kojoj se nalaze uz navođenje apsolutne putanje te da su poveznice zapravo unutarnje.

No takva optimizacija ne garantira smanjenje broja lažno pozitivnih rezultata za ovu skupinu atributa. Vrlo mali broj vanjskih poveznica, primjerice na stranici koja sve sadržaje referencira s vlastite domene pa će za nju algoritam detektirati da ima 0 vanjskih poveznica, također može dovesti do odstupanja od prosječne vrijednosti broja vanjskih poveznica za veliki broj standardnih devijacija, što utječe na povećanje rezultata skupine atributa za tu stranicu.

Stranica www.eturist.info prešla je prag atributa skripti koji iznosi 6.0 s rezultatom od visokih 21.63.

Ta skupina atributa odnosi se na broj skripti i statistike skripti kao što su broj linija unutar skripti, broj riječi unutar skripti, postotak posebnih znakova unutar skripti, minimalna duljina pojedinih skripti, minimalna duljina pojedine linije skripti, maksimalna duljina pojedinog niza znakova unutar skripte, minimalna duljina pojedine riječi unutar skripte te minimalna duljina pojedinog argumenta funkcije skripti.

Stranica sama po sebi ne sadrži nikakav koristan sadržaj, nego jedno spontano preusmjeravanje unutar jedne jednolinijske skripte koje preusmjerava na samu tu stranicu s parametrom `'js'` unutar URL-a koji je vrlo dugačak i sadrži određeni broj specijalnih znakova. Taj je kod prikazan u ispisu 4.11.

```
window.location.replace('http://www.eturist.info/?js=eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ9.eyJhdWQiOiJKb2tlbiIsImV4cCI6MTYyMTk3NTA1NiwiaWF0Ijox
NjIxOTY3ODU2LCJpc3MiOiJKb2tlbiIsImpzIjoxLCJqdGkiOiIycTE2NmIwMWFsbzZhNzh
qaDAwZHFqMGEiLCJuYmYiOjE2MjE5Njc4NTYsInRzIjoxNjIxOTY3ODU2NDU4ODMxfQ.qk0
KajT_cUwkrv9lAwz0yklccCwDIfgh6IcJtogSuMQ&sid=46b6b84e-bd88-11eb-85ef-
546db98ca33e');
```

Ispis 4.11: JavaScript kod sa spontanim preusmjeravanjem

Naslov stranice je 'Loading…' i preusmjeravanje se obavlja na istu domenu. Iako je stranica vrlo loše postavljena u skladu sa SEO praksama, vjerojatno nije zlonamjerna budući da se preusmjeravanje radi trenutačno i na istu domenu, a niti jedan od druga dva algoritma nije ju procijenio kao zlonamjernu.

Prijedlog optimizacije ovog algoritma je da se ne uzimaju u obzir preusmjeravanja koja su na istu domenu na kojoj se nalazi stranica ili da takva preusmjeravanja manje utječu na ukupan rezultat.

Stranica vilajasmin.blogspot.com prešla je prag skupine atributa sadržaja iskorištavanja ranjivosti koji iznosi 5.0 s rezultatom od 6.29.

Ta skupina atributa odnosi se na statistike `object` oznaka kao što su broj `object` oznaka, maksimalna duljina poveznice `object` oznaka, udio posebnih znakova u `object` oznakama, udio samoglasnika u `object` oznakama te medijan broja atributa `object` oznaka.

Stranica sadrži jednu `object` oznaku s relativno malim brojem atributa (samo `width` i `height`) koja ima nekoliko djece s oznakom `param`. Te `param` oznake uključuju ime videa, poveznicu na Youtube video te specificiraju da se dopusti pristup skriptama i *fullscreen* način rada. Dijete tog `object` elementa je `embed` element kojem je izvor taj isti Youtube video. Tip tog `embed` elementa je `'application/x-shockwave-flash'` te on traži dopuštenje pristupu skriptama, a sadržaj videa vezan je za sadržaj stranice i pripada istom autoru. Opisani `object` element prikazan je u ispisu 4.12.

```
<object height="385" width="440"><param name="movie"
value="http://www.youtube.com/v/8YBTurl-QHo?fs=1&amp;hl=hr_HR"><param
```

```
name="allowFullScreen" value="true"><param name="allowscriptaccess"
value="always"><embed src="http://www.youtube.com/v/8YBTurl-
QHo?fs=1&amp;hl=hr_HR" type="application/x-shockwave-flash"
allowscriptaccess="always" allowfullscreen="true" width="440"
height="385"></embed></object>
```

Ispis 4.12: Object element stranice koja prelazi prag sadržaja iskorištavanja ranjivosti

Ova stranica nije zlonamjerna, što je ustanovljeno ručnom provjerom, pa se radi o lažno pozitivnoj detekciji. Drugi algoritam koji ju je detektirao kao zlonamjernu je algoritam zasnovan na Yara pravilima, no ta je detekcija lažno pozitivna zbog `'base64;'` uzorka koji se trebao tražiti u skriptama kako bi se našle potencijalne paketizirane funkcije, no umjesto toga je nađen u `img` oznakama, kako je prikazano u ispisu 4.13.

```
background-image: url(…);
```

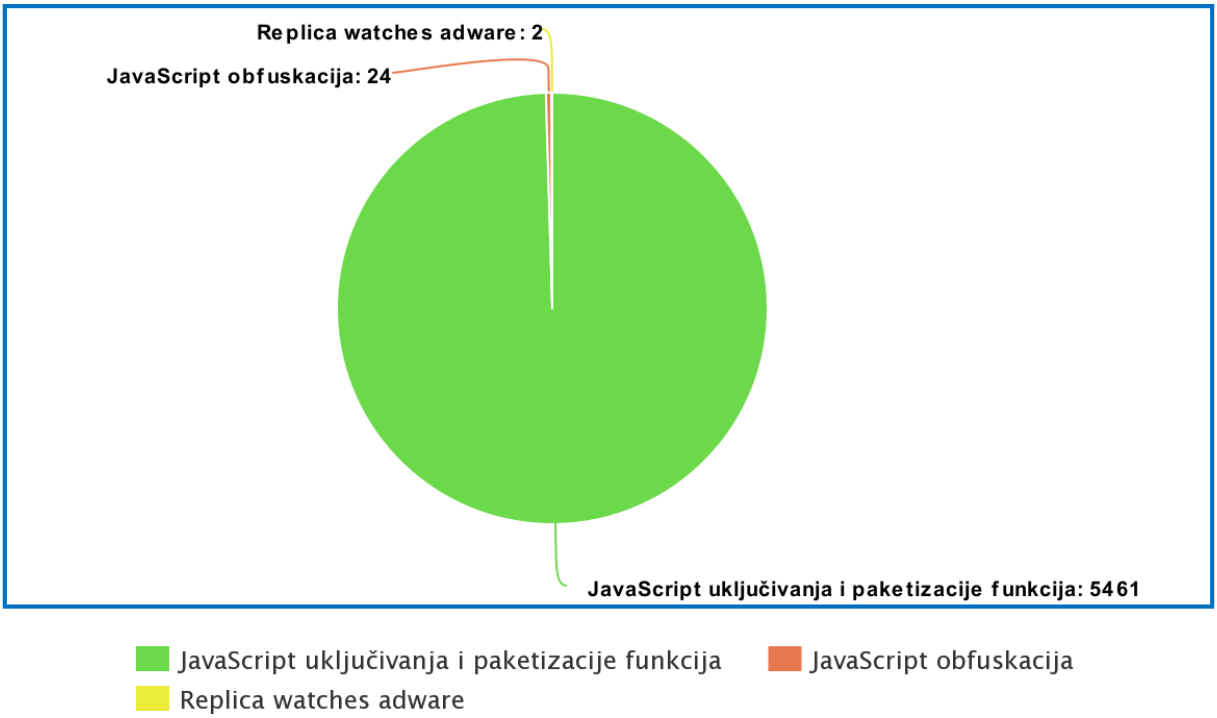
Ispis 4.13: Pronalazak base64; uzorka u img oznakama

Iako mogućnost optimizacije za ovu skupinu atributa nije naročito velika, optimizaciju je moguće postići uzimanjem u obzir parametara danih unutar param djece object oznake uz atribute koji su specificirani direktno kao atributi `object` oznake, na primjer `data`, `width` i `height`, u izračunu ukupnog broja atributa object oznake. To bi dovelo do pravednije procjene stvarnog broja atribute te nešto preciznije klasifikacije stranica kao zlonamjerne za ovu skupinu atributa.

Zaključuje se da je stopa lažno pozitivnih rezultata ovog algoritma relativno visoka uz manje mogućnosti optimizacije nego kod statističke heuristike. Algoritam ne uzima u obzir konkretne atribute nego njihove kombinacije, a i kao takav je zamišljen ne kao algoritam koji će stranicu binarno klasificirati kao zlonamjernu ili benignu, nego kao potencijalno zlonamjernu te ju proslijediti odgovarajućim uređajima za detekciju ili stručnjacima za informacijsku sigurnost na ručnu analizu koji s tim algoritmom čine drugu komponentu hibridnog sustava za ispitivanje zlonamjernosti stranica.

4.4 Rezultati algoritma zasnovanog na Yara pravilima

Udio zlonamjernih stranica dobivenih analizom algoritmom Yara pravila iznosi 7.27%, odnosno 5487/75439 stranica. Dijagnoze zlonamjernosti koje se pojavljuju poredane silazno po učestalosti pojavljivanja su JavaScript uključivanja i paketizacije funkcija (5461), JavaScript obfuskacija (24) i replica watches adware (2). Opisana distribucija zlonamjernosti stranica algoritma temeljenog na Yara pravilima prikazana je na slici 4.2.



Slika 4.2: Distribucija dijagnoza zlonamjernosti algoritma Yara pravila

Primjeri stranica koje sadrže dijagnozu mogućih JavaScript uključivanja i paketizacije funkcija su ustanova-kristofor.hr, www.inyourpocket.com i giga-green.com.

Sve 3 stranice sadrže `'base64;'` uzorak samo u `src` atributu slika koje su označene `img` oznakama, što znači da je detekcija lažno pozitivna i da se Yara pravilom koje detektira moguću paketizaciju funkcija treba provjeravati samo sadržaj skripti unutar stranice, a ne cijela stranica.

Primjeri stranica koji sadrže dijagnozu JavaScript obfuskacije su www.apartmentcroatia-zadar.com, planetzoe.hr i finder.hr.

Na svim stranicama pronađeni su obfuscirani uzorci JavaScript koda koji su svi vrlo sličnog formata, a deobfuskacija skripti bit će provedena online alatom za obfuskaciji i depaketizaciju JavaScript funkcija kako bi se otkrilo koju funkcionalnost imaju obfuscirane skripte [52].

Na stranici www.apartmentcroatia-zadar.com pronađen je obfuscirani sadržaj prikazan u ispisu 4.14.

```
var _0xd747=["\x63\x6F\x6E\x74\x65\x78\x74\x6D\x65\x6E\x75","\x53\...
\x62\x69\x64\x64\x65\x6E\x21","\x70\x72\x65\x76\x65\x6E\x74\x44\x65\x66
\x61\x75\x6C\x74","\x62\x69\x6E\x64","\x69\x6D\x67","\x6B\x65\x79\x43\x
6F\x64\x65","\x6B\x65\x79\x75\x70","\x72\x65\x61\x64\x79"];$(document)[
_0xd747[7]](function (){$(_0xd747[4])[_0xd747[3]](_0xd747[0],function
(_0x2ad3x1){alert(_0xd747[1]);_0x2ad3x1[_0xd747[2]]();}
);$(window)[_0xd747[6]](function
(_0x2ad3x1){if(_0x2ad3x1[_0xd747[5]]==44){alert(_0xd747[1]);} ;} );} );
```

Ispis 4.14: Izvorni obfuscirani JavaScript kod sa stranice www.apartmentcroatia-zadar.com

Sadržaj dobivenim deobfuskacijom ovog sadržaja prikaz je u ispisu 4.15.

```
var _0xd747 = ["contextmenu", "Svako neovlašteno kopiranje i
preuzimanje fotografija je strogo zabranjeno!...", "preventDefault",
"bind", "img", "keyCode", "keyup", "ready"];

$(document)[_0xd747[7]](function () {

    $(_0xd747[4])[_0xd747[3]](_0xd747[0], function (_0x2ad3x1) {

        alert(_0xd747[1]);

        _0x2ad3x1[_0xd747[2]]();

    });

    $(window)[_0xd747[6]](function (_0x2ad3x1) {

        if (_0x2ad3x1[_0xd747[5]] == 44) {

            alert(_0xd747[1]);

        };

    });

});
```

Ispis 4.15: Deobfuscirani JavaScript kod sa stranice www.apartmentcroatia-zadar.com

Ovaj sadržaj na stranici uveo je sam autor stranice kako bi prikazao upozorenje da kopiranje fotografija nije dozvoljeno kad se klikne na fotografiju.

Deobfuskacijom izvornog JavaScript koda sa stranica planetzoe.hr i finder.hr dobiva se sadržaj prikazan u ispisu 4.16.

```
var _0xb787 = ["<a
href=\'mailto:narudzbe@planetzoe.hr\'>narudzbe@planetzoe.hr</a>",
"write"];

document[_0xb787[1]](_0xb787[0])

…


var _0x25f5 = ["<a
href=\"mailto:prodaja@finder.hr\">prodaja@finder.hr</a>", "write"];

document[_0x25f5[1]](_0x25f5[0])
```

Ispis 4.16: Deobfuscirani JavaScript kod sa stranica planetzoe.hr i finder.hr

Obfuskacije na drugoj i trećoj stranici uvedene su kako web pauci (engl. *crawleri*) ne bi mogli automatski doći do email adrese tijekom procesa masovnog prikupljana email adresa (engl. *mail harvesting*) koji se koristi kod pribavljanja adresa za slanje masovnih neželjenih poruka, budući da se u sadržaju stranice ne preporučuje uključivati nizove znakova s poveznicama email adresa.

Niti na jednoj od ove tri stranice ne radi se o zlonamjernoj obfuskaciji, nego o obfuskaciji koju autori stranice koriste za zaštitu određenog koda od automatiziranih i zlonamjernih web pauka, tako da algoritam zasnovan na Yara pravilima za ovo pravilo ima određeni postotak lažno pozitivnih detekcija.

Dvije stranice za koje postoji dijagnoza detekcije replica watches adwarea obje se odnose na URL www.e-skole.hr.

Radi se o istinitim pozitivnim detekcijama specifičnog zlonamjernog koda adwarea koji reklamira stranice www.replica-watches.is i www.fake-watches.is. Taj kod drugi općeniti algoritmi poput statičke heuristike i mehanizma bodovanja ne mogu otkriti.

Općenito algoritam zasnovan na Yara pravilima ima manju stopu lažno pozitivne detekcije te može preciznije otkriti određene zlonamjerne uzorke i biti prilagođen za nove vrste napada u usporedbi s druga dva algoritma te se može smatrati boljim algoritmom.

5 Optimizacija algoritama

5.1 Implementacija optimizacije algoritama

Kako bi se izbjeglo pokretanje ispitivanja nad cijelim skupom podataka nakon implementacije svakog pojedinog poboljšanja algoritama, poboljšanja će se testirati na nasumičnom skupu od 367 stranica kako bi se vidio učinak smanjenja broja stranica otkrivenih kao zlonamjernih nakon svake uvedene promjene.

Kad se radi o ispitivanju na podskupu od 367 stranica koristeći izvorne implementacije algoritma, stope zlonamjernosti su 4.9% za algoritam statičke heuristike, 9.26% za algoritam zasnovan na mehanizmu bodovanja i 5.72% za algoritam zasnovan na Yara pravilima.

5.1.1 Optimizacija algoritma statičke heuristike

Kao optimizacija za algoritam statičke heuristike uvodi se lista dozvoljenih domena koje se smiju naći kao izvori `iframe` oznaka u funkciju `searchIFrames` modula `pagechecker` te uz koje se čak i `iframe` oznake s `width` ili `height` atributom 0 neće označavati kao mali `iframe` elementi koji predstavljaju direktno obilježje zlonamjernosti. Za to se uvodi pomoćna funkcija prikazana u ispisu 5.1 koja na temelju URL-a izvora `iframe` elementa određuje da li se njegova domena nalazi na specificiranoj listi dozvoljenih domena definiranoj unutar modula, pri čemu se u listi na početku uvodi domena www.googletagmanager.com.

```
iFramesWhitelist = ['www.googletagmanager.com']
…
iFrameSource = iFrame.get('src')
…
if (iFrameWidth == '0' or iFrameHeight == '0') and (not
isSourceWhitelisted(iFrameSource)):
            iFramesByType['small'] += 1
        else:
            iFramesByType['large'] += 1
…
def isSourceWhitelisted(iFrameSource):

    if iFrameSource is None:
        return False
```

```
    parsedUrl =
    urllib.parse.urlparse(iFrameSource.replace('[','').replace(']',''))
    iFrameDomain = parsedUrl.netloc
    if iFrameDomain in iFramesWhitelist:
        return True
    else:
        return False
```

Ispis 5.1: Implementacija liste dozvoljenih domena za male iframe elemente

Nakon uvedene optimizacije ponovno se pokreće algoritam na uzorku od 367 stranica i stopa zlonamjernosti za algoritam statičke heuristike pala je s 4.9% na 0.82%, što znači da je velika većina lažno negativnih detekcija eliminirana.

Budući da je stopa zlonamjernosti stranica na ovom skupu podataka za algoritam statičke heuristike sada vrlo niska uz samo 3 zlonamjerne stranice, kako bi se detektiralo poboljšanje uvođenjem daljnjih optimizacija, potrebno je podskupu dodati još stranica te nakon dodavanja skup sadrži ukupno 2 628 stranica.

Testiranjem tog seta stranica ustanovljena je stopa zlonamjernosti algoritmom statičke heuristike od 3.92%, odnosno 103/2628 stranica.

Druga optimizacija je uvođenje liste dozvoljenih uzoraka obfuskacije JavaScripta, odnosno ako se ustanovi da je obfuscirani JavaScript kod koji sadrži `unescape()` funkciju dio koda komponente standardne biblioteke kao što je RS plugin, neće se smatrati obfusciranim JavaScript kodom u kontekstu ovog algoritma za detekciju zlonamjernosti.

Iz tog razloga potrebno je promijeniti funkciju `detectObfuscation` te uvesti funkciju koja će otkriti slučajeve u kojima skripta predstavlja dobro poznate dodatke i biblioteke koji unutar sebe koriste `unescape()` funkciju te tako izuzeti skriptu s popisa zlonamjerno obfusciranih. Navedena promjena prikazana je u ispisu 5.2.

```
scriptsWhitelist = ['document.getElementById(\'rs-plugin-settings-
inline-css\')', 'document.getElementById("rs-plugin-settings-inline-
css")']
…
if escapedFound and unescapeFound and (not
isScriptWhitelisted(scriptText)):
    scriptsByObfuscation['obfuscated'] += 1
…
def isScriptWhitelisted(scriptText):
```

```
    for listed in scriptsWhitelist:
       if scriptText.find(listed) != -1:
           return True
    return False
```

Ispis 5.2: Implementacija liste dozvoljenih uzoraka za obfuscirani JavaScript kod

Treća optimizacija je zanemarivanje onih `meta` oznaka za preusmjeravanje koje nemaju oznaku domene preusmjeravanja ili preusmjeravaju na istu domenu koja je domena stranice te pritom imaju interval preusmjeravanja 30 sekundi ili više, budući da bi `meta` oznake koje preusmjeravaju jako brzo na istu stranicu mogle ukazivati na umetanje zlonamjerne `meta` oznake koja namjerno uzrokuje česta osvježavanja s namjerom da omete korisničko iskustvo.

U tu je svrhu potrebno promijeniti funkciju `metaRefreshTagsExist` u funkciju `metaRedirectExists` koja će realizirati navedena svojstva korištenjem pomoćne funkcije koja određuje treba li se određena `meta` oznaka za preusmjeravanje izuzeti iz ukupnog broja preusmjeravanja. Optimizacija je prikazana u ispisu 5.3.

```
def metaRefreshTagsExist(soup, domain = None):
        if (httpEquiv is not None and httpEquiv == 'refresh' and
content is not None and (not isMetaRefreshContentWhitelisted(content,
domain))):
            return True
…
def isMetaRefreshContentWhitelisted(content, domain):
    contentParts = content.split(';')
    urlPartFound = False
    acceptableTime = False
    minimumRedirectValue = 30
    for contentPart in contentParts:
        if contentPart.lower().startswith("url="):
            urlPartFound = True
            …
            if domain is not None and redirectDomain.replace('www.','')
            == domain.replace('www.',''):
                sameDomainRedirect = True
```

```
            else:
                sameDomainRedirect = False
        else:
            if (contentPart.isnumeric() and int(contentPart) >=
            minimumRedirectValue):
                acceptableTime = True
    …
    return sameDomainRedirect and acceptableTime
```

Ispis 5.3: Uvođenje iznimki za meta oznake za preusmjeravanje

5.1.2 Optimizacija algoritma zasnovanog na mehanizmu bodovanja

Algoritam temeljen na mehanizmu bodovanja sam je po sebi zamišljen tako da određeni postotak stranica označi kao potencijalno zlonamjerne te stranice analizira relativno u odnosu na veliki skup stranica koje imaju određene atribute, a ne u odnosu na apsolutne i individualne kriterije. Iz tog razloga postotak lažno pozitivnih detekcija uz kojeg postotak lažno negativnih detekcija ostaje minimalan ne može biti jako nizak i nema puno prostora za optimizaciju uz ovakvu paradigmu. Ipak, moguće je provesti optimizacije koje će na pravedniji način procijeniti određene atribute i možda smanjiti devijacije pojedinih stranica zbog kojih je njihov rezultat neopravdano visok, ili će postotak potencijalno zlonamjernih stranica ostati isti, ali će pretpostavke biti točnije, odnosno veći postotak tih stranica proslijeđenih uređajima za detekciju ili stručnjacima bit će detektirane kao zlonamjerne.

Prije optimizacije, na skupu od 2 628 stranica koje su podskup 75 492 stranice na temelju kojih su određeni parametri bodovanja, postotak zlonamjernih detekcija je 12.75%, odnosno 335/2628. To je malo manje od očekivanog postotka za taj algoritam koji je u literaturi naveden kao 14%, a malo više od ciljanog postotka koji iznosi oko 10%.

Jedna od predloženih optimizacija uključuje pravednije prebrojavanje atributa `object` oznaka tako da se kao atributi ne uzimaju samo direktno označeni atributi `object` oznake, nego i podaci dani kao `param` oznake koji su djeca `object` oznaka, primjerice oznaka prikazana u ispisu 5.4 prema trenutnom algoritmu ima 2 atributa, no ispravnije bi bilo označiti da ima 5 atributa, uključujući 2 atributa same oznake i 3 `param` djece koja također pridjeljuju određene atribute.

```
<object height="385" width="440"><param name="movie"
value="http://www.youtube.com/v/8YBTurl-QHo?fs=1&amp;hl=hr_HR"><param
name="allowFullScreen" value="true"><param name="allowscriptaccess"
value="always"><embed src="http://www.youtube.com/v/8YBTurl-
QHo?fs=1&amp;hl=hr_HR" type="application/x-shockwave-flash"
allowscriptaccess="always" allowfullscreen="true" width="440"
height="385"></embed></object>
```

Ispis 5.4: Object oznaka s 2 atributa i 3 param djece

U tu svrhu, potrebno je modificirati funkciju `getObjectStatistics` koristeći pomoćnu funkciju za brojenje atributa objekata, kako je prikazano u ispis 5.5.

```
objectAttributesNumber = countObjectAttributes(object)
…
def countObjectAttributes(object):
    inlineAttributesCount = len(object.attrs.keys())
```

```
    paramsCount = len(object.find_all('param'))
    totalAttributes = inlineAttributesCount + paramsCount
    return totalAttributes
```

Ispis 5.5: Promjena načina brojenja atributa object oznaka

Druga moguća optimizacija jest razlikovanje poveznica koje su vanjske od poveznica koje zapravo povlače sadržaj s iste domene u slučaju kad domena nije eksplicitno predana algoritmu. To se može postići tako da se pretpostavi da ako više od nekog arbitrarnog visokog postotka poveznica, primjerice 80%, ima određenu domenu, onda se u najvećem broju slučajeva zaključuje da je to zapravo domena analizirane stranice ili neka druga administrativno povezana domena ili poddomena poput CDN-a na kojoj se resursi učitavaju apsolutnom putanjom. Iako domena kao parametar algoritmu nije predana, on će analizom elemenata stranice zaključiti koje su poveznice uistinu vanjske, a koje nisu. Ova se analiza ne odnosi na `iframe` objekte jer se u njima vrlo često uključuju vanjske poveznice i sadržaji, puno češće nego primjerice u `script` ili `rel` oznakama koje češće referenciraju neku lokalnu skriptu ili CSS datoteku.

Spomenute će se promjene uvesti unutar modula `htmlhandler` u funkciji `getLinksStatistics` tako da se kroz poveznice prolazi dvaput, prvi puta kako bi se odredilo koja domena prevladava te ju proglasilo domenom stranice, a drugi puta kako bi se obavio prolazak u kojem se broje vanjske poveznice. Navedena je optimizacija prikazana u ispisu 5.6.

```
def getLinksStatistics(soup, domain = None):
…
linkDomains = []
    localDomainThreshold = 0.80
    for link in allLinks:
        linkDomain =
        urllib.parse.urlparse(link.replace('[','').replace(']',''))
        .netloc
        if linkDomain:
            linkDomains.append(linkDomain)
    if domain is None:
        if len(linkDomains) > 0:
            mostCommonDomain = max(set(linkDomains),
            key=linkDomains.count)
```

```
            mostCommonDomainPercentage = (len([domain for domain in
            linkDomains if domain == mostCommonDomain])/len(linkDomains
            ))
            if mostCommonDomainPercentage >= localDomainThreshold:
                websiteDomain = mostCommonDomain
    …
    for link in allLinks:
        if (not isPathLocal(link,websiteDomain)):
            externalLinksNumber += 1
            linkLengths.append(len(link))
    linksStatistics['minLength'] = min(linkLengths)
    if (len(linkLengths) > 0) else 0
    linksStatistics['externalLinks'] = externalLinksNumber
…
```

Ispis 5.6: Promjena načina razlikovanja unutarnjih poveznica od vanjskih

Kako bi optimizacije imale smisla, potrebno je ponovno konstruirati objekt `scoring_parameters.json` iz cijelog skupa podataka od 75 492 stranice te tek onda ponovno pokrenuti algoritam evaluacije nad manjim skupom podataka koji je njegov podskup, budući da se način izračuna jednog od atributa mijenja za cijeli skup podataka. Očekivano je da će se nakon optimizacija prosječni broj atributa object oznaka povećati, a broj vanjskih poveznica smanjiti.

Nakon ponovnog određivanja parametara oni su se promijenili kako je prikazano u ispisu 5.7.

```
"objectsMedianAttributesNumber": {"average": 0.08279575589789782,
"deviation": 0.599835667897434}} -> "objectsMedianAttributesNumber":
{"average": 0.15157235203730196, "deviation": 1.1449438600991682}

"minLinkLength": {"average": 22.622084166743935, "deviation":
16.939422890495646}, "externalLinks": {"average": 43.1950511967997,
"deviation": 98.48164504051593} ->

"minLinkLength": {"average": 17.47880570126636, "deviation":
18.51112178445883}, "externalLinks": {"average": 12.004503788480898,
"deviation": 42.21399702942435}
```

Ispis 5.7: Novi parametri duljine poveznica i broja atributa objekata nakon optimizacije

Nakon ponovnog pokretanja optimiziranog algoritma nad skupom od 2 628 stranica i prilagodbe praga za strani sadržaj na 10.2, postotak detekcije unutar skupa od 2 628 stranica je 11.57%, što je nešto niže od prethodnog postotka. Postotak detektiranih stranica nije se značajno smanjio, no pretpostavlja se da se korektnijom procjenom pojedinih atributa algoritam učinio preciznijim.

5.1.3 Optimizacija algoritma zasnovanog na Yara pravilima

Glavni problem algoritma zasnovanog na Yara pravilima je što se pravila koja se odnose samo na JavaScript kod primjenjuju na cijelu HTML stranici i iz tog razloga ga je potrebno preraditi. Prije optimizacije, set od 2 628 stranica sadrži čak 11.04%, odnosno 290/2628 stranica koje su označene kao zlonamjerne na temelju Yara pravila i to većinom s dijagnozom `'Possible includes and packed functions detected'`.

Najprije je potrebno iz yara_rules.yar izdvojiti ona pravila koja se odnose na JavaScript kod te ih postaviti unutar datoteke `yara_rules_scripts.yar`, budući da će se preko njih provjeravati samo JavaScript kod posebno. No za efikasnu optimizaciju ne treba premještati sva pravila, pogotovo ona koja u sebi uključuju `'<script>'` niz znakova, nego samo ona koja prave probleme s lažno pozitivnim detekcijama, a to su `generic_javascript_obfuscation` i `possible_includes_base64_packed_functions`.

Od putanje do glavne datoteke s Yara pravilima funkcija će konstruirati putanju do datoteke koja sadrži pravila za skripte dodavanjem sufiksa 'scripts' prije .yar ekstenzije, tako da se ta putanja ne bi morala predavati kao dodatni argument. Također, sadržaj skripti izdvojit će se iz HTML-a koristeći Beautiful soup biblioteku.

Funkcija za inicijalizaciju Yara objekata `setYaraRulesObject` izmijenit će se na način prikazan u ispisu 5.8.

```
def setYaraRulesObject(path):
    …

    yaraRulesFile = open(path, encoding='utf-8')
    yaraRulesString = yaraRulesFile.read()
    yaraRulesFile.close()
    yaraRules = yara.compile(source = yaraRulesString)

    pathJavaScript = path.replace('.yar','_scripts.yar')
    yaraRulesFileJavaScript = open(pathJavaScript, encoding='utf-8')
    yaraRulesStringJavaScript = yaraRulesFileJavaScript.read()
    yaraRulesFileJavaScript.close();
    yaraRulesJavaScript = yara.compile(source =
    yaraRulesStringJavaScript)
```

Ispis 5.8: Postavljanje objekta Yara pravila u optimiziranoj verziji algoritma

Unutar modula `htmlhandler` bit će dodana funkcija koja će iz HTML dokumenta izdvojiti sav JavaScript kod te ga spojiti u jedan niz znakova, a prikaza je u ispisu 5.9.

```
def extractJavaScriptCode(html):
    soup = makeBeautifulSoup(html)
    scripts = soup.find_all('script')
    scriptTexts = []
    for script in scripts:
        scriptText = ''.join(script.contents)
        if scriptText != '':
            scriptTexts.append(scriptText)
    return "\n".join(scriptTexts)
```

Ispis 5.9: Izdvajanje JavaScript koda iz HTML stranice

Funkcija `getYaraMatches` unutar modula `yarachecker` bit će izmijenjena na način prikaza u ispisu 5.10.

```
def getYaraMatches(html):
    if yaraRules is None:
        raise YaraRulesNotSetException("Yara rules object not set")
    scriptCode = htmlhandler.extractJavaScriptCode(html)
    matchesHTML = yaraRules.match(data=html)
    matchesJavaScript = yaraRulesJavaScript.match(data=scriptCode)
    matches = matchesHTML + matchesJavaScript
    return matches
```

Ispis 5.10: Optimizirana verzija algoritma zasnovanog na Yara pravilima

5.2 Rezultati optimiziranih algoritama

Nakon optimizacije algoritama navedenim predloženim tehnikama, stopa detekcije zlonamjernih Web stranica značajno se smanjila za algoritme statičke heuristike i algoritam zasnovan na Yara pravilima, dok je za algoritam zasnovan na mehanizmu bodovanju stupanj detekcije ostao podjednak, no daljnje smanjivanje moglo bi povećati rizik od većeg broja lažno negativnih rezultata i stranice označene tim algoritmom mogu se smatrati potencijalno zlonamjernima umjesto zlonamjernima. Budući da se pretpostavlja da velika većina stranica nije zlonamjerna, sve stope detekcija približno su jednake stopi lažno pozitivnih rezultata algoritma, pogotovo zato što se za konkretne primjere stranica otkrivene kao zlonamjerne prije optimizacije algoritama ručnom analizom ustanovilo da su benigne.

Brojevi i udjeli benignih i zlonamjernih stranica pojedinih algoritama nakon optimizacije prikazani su u tablici 5.1.

Algoritam	**Ukupni broj stranica**	**Broj benignih stranica**	**Broj zlonamjernih stranica**	**Udio benignih stranica**	**Udio zlonamjernih stranica**
Statička heuristika	75 492	74 991	501	99.34%	**0.66%**
Mehanizam bodovanja	75 492	68 084	7 408	90.19%	**9.81%**
Yara pravila	75 492	74 672	820	98.91%	**1.09%**

Tablica 5.1: Statistika benignih i zlonamjernih Web stranica s optimiziranim algoritmima

5.2.1 Rezultati algoritma statičke heuristike

Stopa detekcije za algoritam statičke heuristike jest 0.66%, odnosno 501/75492 stranice, u usporedbi sa stopom detekcije od 6.83%, odnosno 5156/75492 prije provedene optimizacije.

Stranice orvasyachting.com, villsy.com i www.taxicroatiatransfer.com koje su prethodno otkrivene kao zlonamjerne zbog malih `iframe` elemenata koji sadrže Google tag manager HTML stranicu i za koje je ustanovljeno da se radi o lažno pozitivnoj detekciji više nisu označene kao zlonamjerne tim algoritmom, niti jednim od ostala 2 algoritma.

Stranice www.dom-jalzabet.hr, feid.hr, www.salon-vodotehnika.hr i delmata-travel.hr koje su prethodno otkrivene kao zlonamjerne zbog obfusciranog JavaScript koda legitimne RS plugin biblioteke i za koje je ustanovljeno da se radi o lažno pozitivnoj detekciji nisu više označene kao zlonamjerne tim algoritmom, niti jednih od ostala dvaju algoritama.

Stranice dubrovnikcruising.tripod.com i www.os-rivarela-novigrad.skole.hr za koje je detektirano da imaju kombinaciju `iframe` i `object` oznake te stranice www.os-pantovcak-zg.skole.hr i spinevital.blogspot.com za koje je detektirano da imaju kombinaciju `iframe` i

`embed` oznake i dalje su označene kao zlonamjerne budući da nije implementirana optimizacija na razini heuristike koji bi eliminirala takve lažno pozitivne rezultate.

Stranica dubrovnikcruising.tripod.com pritom je također označena algoritmom temeljenim na mehanizmu bodovanja kao zlonamjerna s objašnjenjem da prelazi prag sadržaja za iskorištavanje ranjivosti koji se odnosi na statistike `object` oznaka, ali nije označena algoritmom Yara pravila, stranica www.os-rivarela-novigrad.skole.hr također je označena algoritmom temeljenim na mehanizmu bodovanja kao zlonamjerna s objašnjenjem da prelazi prag sadržaja za iskorištavanja ranjivosti, ali nije označena algoritmom Yara pravila, a stranica spinevital.blogspot.com označena je algoritmom temeljenih na mehanizmu bodovanja uz objašnjenje da rezultat atributa stranog sadržaja koji se odnosi na statistike poveznica i `iframe` objekata prelazi prag, ali također nije označena algoritmom Yara pravila.

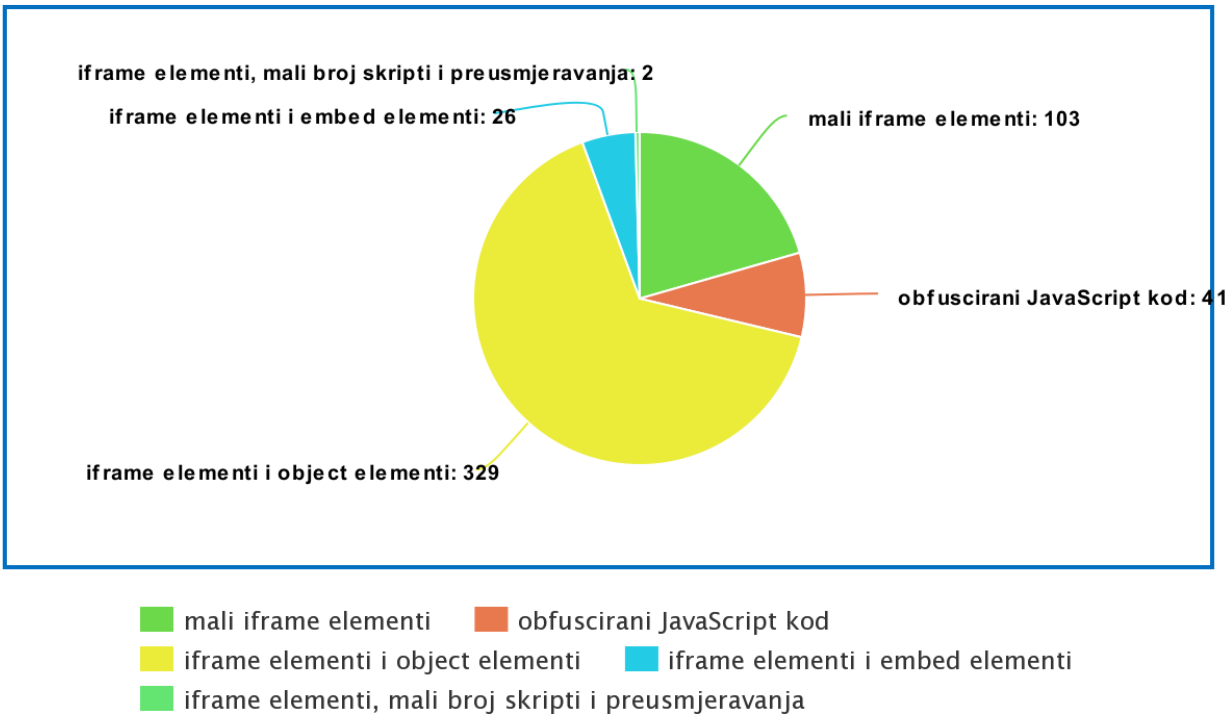
Stranice policijazaustavlja.com i www.boem-mps.hr za koje je dijagnoza bila `'webpage contains iframe elements, small number of script tags and redirects'` nisu više označene kao zlonamjerne algoritmom statičke heuristike, budući da sadrže `meta` oznaku za preusmjeravanje bez URL-a, što znači da preusmjeravaju na istu tu stranicu, s prihvatljivim vremenom osvježavanja, a pritom je stranica www.boem-mps.hr označena algoritmom temeljenom na mehanizmu bodovanja kao zlonamjerna uz objašnjenje da prelazi prag skupine atributa koje se odnose na skripte.

Stranica tumir.hr koja sadrži `meta` oznaku za preusmjeravanje s preusmjeravanjem na istu domenu i dalje je označena kao zlonamjerna jer algoritam u ovom slučaju nije primio podatak o tome o kojoj se domeni radi i ne može ustanoviti da se zapravo radi o osvježavanju, odnosno preusmjeravanju na istu stranicu, a također je označena algoritmom bodovanja kao stranica koja prelazi prag atributa koji se odnose na strani sadržaj. U ovom slučaju, radi se o lažno pozitivnoj detekciji.

Stranica scholar.google.hr koja sadrži isječke minificiranog JavaScript koda Closure Library JavaScript biblioteke koji ima određene izraze koji ukazuju na preusmjeravanja poput `'window.location.href'` i `'window.location.replace'` i dalje je označena kao zlonamjerna zbog nesavršenosti implementacije algoritma koji teško može sa sigurnošću ustanoviti koja su preusmjeravanja unutar JavaScript koda spontana, a koja nisu te je također označena algoritmom bodovanja kao stranica koja prelazi prag atributa koji se odnose na strani sadržaj. I u ovom slučaju, radi se o lažno pozitivnoj detekciji.

Budući da je za sve te stranice ručnom provjerom ustanovljeno da su lažno pozitivne detekcije, može se zaključiti da algoritam statičke heuristike te algoritam zasnovan na mehanizmu bodovanja mogu u sličnim slučajevima dati lažno negativne rezultate, dok je algoritam zasnovan na Yara pravilima u ovom slučaju pouzdaniji. Također, za stranice koje su klasificirane kao zlonamjerne algoritmom statičke heuristike vjerojatnije je da će biti isto tako klasificirane algoritmom temeljenim na mehanizmu bodovanja te postoji određena korelacija skupine atributa za koje će biti označene. Na primjer, ako statička heuristike otkrije da stranica sadrži kombinaciju `iframe` oznaka i `object` oznaka, najvjerojatnija dijagnoza koju će postaviti algoritam zasnovan na mehanizmu bodovanja jest prelazak praga za atribute koji se odnose na sadržaj iskorištavanja ranjivosti jer se oni odnose na statistike `object` oznaka i njihovih poveznica. Ako dijagnoza algoritma statičke heuristike sadrži prisutnost preusmjeravanja, najvjerojatnija dijagnoza algoritma zasnovanog na mehanizmu bodovanja je prelazak praga za atribute skripti, pogotovo ako se radi o JavaScript preusmjeravanju.

Što se tiče distribucije dijagnoza zlonamjernosti među stranicama koje su otkrivene kao zlonamjerne optimiziranom verzijom algoritma, detekcije malih `iframe` elemenata više nisu najzastupljenija dijagnoza, nego je najviše stranica detektirano kao zlonamjerno zato što sadrže `iframe` element i `object` element (329). Zatim po silaznom redoslijedu zastupljenosti kao objašnjenja zlonamjernosti slijede mali `iframe` elementi (103), obfuscirani JavaScript kod (41), postojanje `iframe` elementa i `embed` elementa (26) te postojanje `iframe` elemenata uz mali broj skripti i prisutnost preusmjeravanja (2). Opisana distribucija zlonamjernosti stranica optimizirane verzije algoritma statičke heuristike prikazana je na slici 5.1.



Slika 5.1: Distribucija dijagnoza zlonamjernosti optimizirane statičke heuristike

5.2.2 Rezultati algoritma zasnovanog na mehanizmu bodovanja

Stopa detekcije algoritma mehanizma bodovanja iznosi 9.81%, odnosno 7408/75492 stranice, što nije puno niže u odnosu na prethodne rezultate koji su otkrili 11.34%, odnosno 8561/75492 kao zlonamjerne.

Stranica mrak.org koja je označena kao zlonamjerna jer skupina atributa koja se odnosi na strani sadržaj prelazi prag i dalje je označena kao zlonamjerna, a nije označena kao zlonamjerna nije jednim od drugih dvaju algoritama.

Stranica www.eturist.info koja je označena kao zlonamjerna jer skupina atributa koja se odnosi na skripte prelazi prag i dalje je označena kao zlonamjerna, a nije označena kao zlonamjerna nije jednim od drugih dvaju algoritama.

Stranica vilajasmin.blogspot.com koja je označena kao zlonamjerna jer skupina atributa koja se odnosi na sadržaj iskorištavanja ranjivosti prelazi prag i dalje je označena kao zlonamjerna, a nije označena kao zlonamjerna nije jednim od drugih dvaju algoritama.

Može se zaključiti da je algoritam temeljen na mehanizmu bodovanja najmanje pouzdan i da vrlo često otkriva stranice kao zlonamjerne iako nisu označene niti jednim od drugih algoritama, iako postoji određena ranije utvrđena korelacije između označenosti algoritmom statičke heuristike i algoritmom mehanizma bodovanja.

Optimizacije koje su provedene samo su malo smanjile njegovu stopu lažno negativne detekcije iako se očekuje da su povećali točnost, odnosno veća je šansa da od onih stranica koje su otkrivene algoritmom kao zlonamjerne neka uistinu bude zlonamjerna nego što je bila prije provedenih optimizacija.

5.2.3 Rezultati algoritma zasnovanog na Yara pravilima

Stopa detekcije algoritma Yara pravila nakon optimizacije iznosi 1.09%, odnosno 820/75492 stranice, znatno manje od 7.27%, odnosno 5487/75492 stranice koliko je otkriveno prije optimizacije algoritma.

Stranice ustanova-kristofor.hr, www.inyourpocket.com i giga-green.com za koje je ustanovljeno da su zlonamjerne s dijagnozom da sadrže moguća JavaScript uključivanja i paketizirane funkcije više nisu označene kao zlonamjerne, niti su označene nekim od drugih dvaju algoritama.

Stranice www.apartmentcroatia-zadar.com, planetzoe.hr i finder.hr u kojima je otkrivena JavaScript obfuskacija i dalje su označene kao zlonamjerne, budući da se uistinu radi o JavaScript obfuskaciji koja je ispravno otkrivena. No ovdje je riječ o JavaScript obfuskaciji koju je implementirao vlasnik stranice, a ne zlonamjernoj JavaScript obfuskaciji. Algoritam Yara pravila očekivano ne može razlikovati kontekst u kojem se obfuskacija odvija nego samo utvrditi postoji li ona, zbog čega nije potpuno pouzdan.

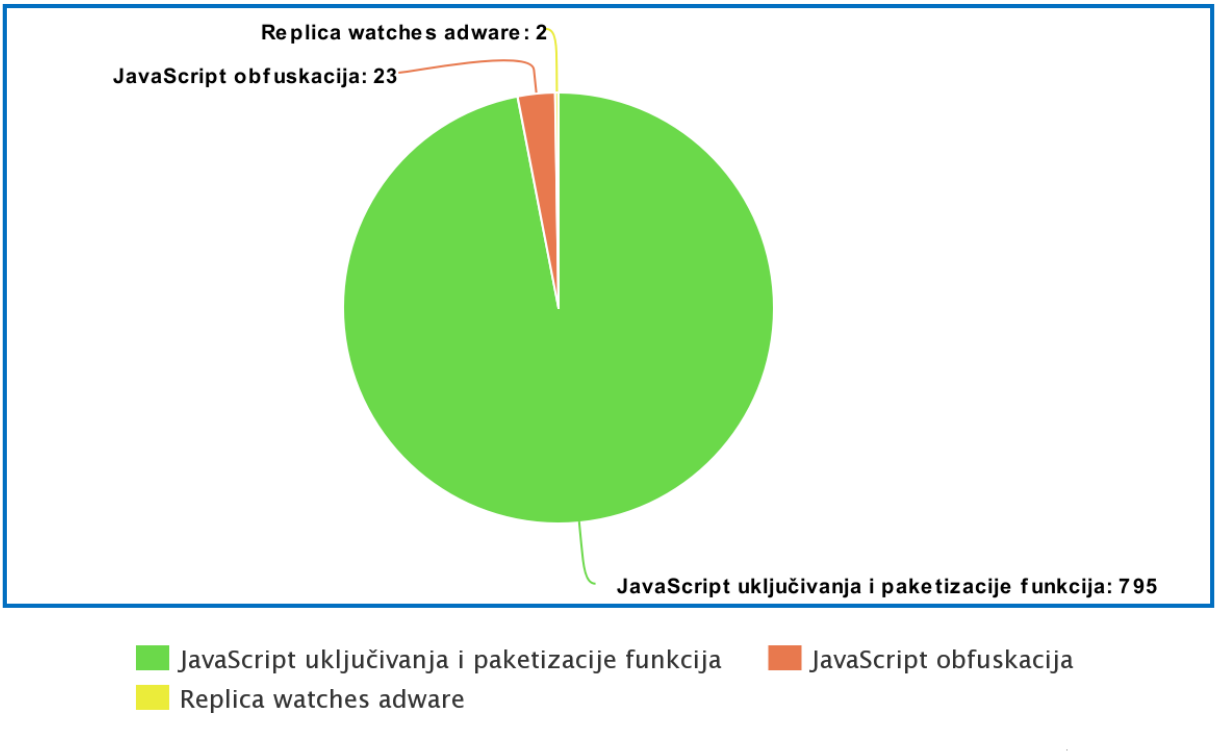
Na temelju provedene analizi može se zaključiti da ne postoji gotovo nikakva korelacija između označenosti algoritmom Yara pravila i označenosti drugim dvjema algoritmima zbog velike razlike u njihovom osnovnom pristupu, odnosno niti jedna ručno analizirana stranica koja je označena kao zlonamjerna algoritmom statičke heuristike, odnosno algoritmom mehanizma bodovanja nije označena zlonamjernom i algoritmom Yara pravila, i obrnuto.

Dvije stranice koje obje odgovaraju URL-u www.e-skole.hr ispravno su označene kao zlonamjerne s dijagnozom da je otkriven replica watches adware te je ranije taj URL ustanovljen kao maliciozan i ručnom analizom, dakle radi se stvarnom pozitivnom rezultatu.

Da se zaključiti da je algoritam zasnovan na Yara pravilima općenito pouzdaniji od ostalih dvaju algoritama jer ima malu stopu lažno pozitivnih rezultata, može otkriti konkretnije uzorke zlonamjernosti i dati točniju dijagnozu te je moguće dodavanjem Yara pravila ciljati određeni uski skup zlonamjernosti ili određenu zlonamjernost koju ostali algoritmi neće pronaći.

Ipak, algoritam zasnovan na Yara pravilima ima slabosti ostalih algoritama koji su zasnovani na potpisu (engl. *signature-based*), a to je određena stopa lažno negativnih rezultata zbog mogućnosti da Yara pravila ne detektiraju izmijenjene zlonamjerne isječke koda koji i dalje imaju istu funkcionalnost i nakon izmjene.

Što se tiče distribucije dijagnoza zlonamjernosti među stranicama koje su otkrivene kao zlonamjerne optimiziranom verzijom algoritma, poredak zastupljenosti dijagnoza nije se promijenio u odnosu na izvornu verziju algoritma, iako se ukupni broj zlonamjernih Web stranica značajno smanjio te je postojanje JavaScript uključivanja i paketizacija funkcija i dalje najzastupljenija dijagnoza. Po silaznom redoslijedu zastupljenosti kao objašnjenja zlonamjernosti navedena su JavaScript uključivanja i paketizacije funkcija (795), JavaScript obfuskacija (23) i replica watches adware (2). Opisana distribucija zlonamjernosti stranica optimizirane verzije algoritma zasnovanog na Yara pravilima prikazana je na slici 5.2.



Slika 5.2: Distribucija dijagnoza zlonamjernosti optimiziranog algoritma Yara pravila

5.3 Dodatne predložene optimizacije

Postoje općenite optimizacije za algoritme koje bi se mogle dodatno provesti uz već implementirane optimizacije, kako u samoj logici algoritma, tako i u konkretnoj implementaciji.

Implementacija dinamičke evaluacije pojedinih atributa mogla bi pomoći u njihovom preciznijem određivanju. Primjer je ispitivanje je li kod za preusmjeravanje unutar skripte definiran samo kao dio funkcije koja će se pozvati u određenom slučaju do kojeg dovodi kontrolirana korisnička akcija ili će se JavaScript preusmjeravanja odviti spontano samim otvaranjem stranice nakon nekog vremena.

Implementacija preciznijih mehanizama traženja određenih uzoraka pomogla bi u smanjenju broja lažno pozitivnih detekcija, primjerice sadašnja implementacija u nizu znakova `'%1e3'` koji znači 'mod 1000' pronalazi uzorak *escaped* heksadekadskih znakova.

Uvođenje dodatnih iznimki u algoritmima također bi omogućilo preciznije procjene i smanjenje broja lažno negativnih rezultat. Na primjeri, ako se niz `'base64;'` nalazi unutar skripte kao sadržaj koji se pridružuje određenom HTML slikovnom elementu, neće se uzimati u obzir kod Yara pravila za otkrivanje paketizacije funkcija. Također, ako je neka skripta za koju je Yara pravilima detektirano zlonamjerno svojstvo poznata skripta široko korištenog dodatka, onda se zlonamjerna svojstva ne bi trebala uzimati u obzir. Prisustvo koda poznatih dodataka unutar skripti može se detektirati određenim metodama zasnovanima na potpisu, no pritom treba biti oprezan kako napadači ne bi ubacili zlonamjerni kod koji oponaša takve dodatke i pritom sadrži malu količinu korisnog tereta koji se koristi za napad jer bi takvi slučajevi mogli dovesti do propuštanja napada i povećanja stope lažno negativnih rezultata.

6 Zaključak

Niti jedan učinkoviti algoritam za detekciju zlonamjernih stranica nije potpuno pouzdan i točan, odnosno svaki od njih ima određeni udio lažno pozitivnih rezultata te propušta detekciju određenog broja zlonamjernih stranica. Klasični pristup detekcija stranica korištenjem uređaja za detekciju visoke interaktivnosti i ručnim pregledom stručnjaka za informacijsku sigurnost, iako puno pouzdaniji, rezultira puno većim trajanjem provjere za svaku od Web stranica, što čini provjeru većeg skupa neprihvatljivo dugotrajnom za praktičnu primjenu. Opisani algoritmi uvode kompromis između brzine izvođenja, odnosno učinkovitosti i pouzdanosti, odnosno broja propuštenih napada i lažno pozitivnih detekcija.

Algoritam zasnovan na asocijativnoj klasifikaciji (engl. *classification based on association*, skraćeno CBA) koji je predstavljen u diplomskom radu ima točnost od visokih 95.83% s vrlo niskom stopom lažno pozitivnih i lažno negativnih rezultata. Performanse tog algoritma uspoređene su s dobro poznatim klasifikacijskim algoritmima, uključujući *support vector machine* (SVM), naivni Bayes i logističku regresiju. Analiza pokazuje da je CBA metoda bolja od naivnog Bayesa s preciznošću od 91.30% i *recall* parametrom od 96.67% (u usporedbi s 90.9%, odnosno 90.3% Bayesovog algoritma), dok logistička regresija i SVM imaju visoku stopu negativnih rezultata od 19.2%, odnosno 16.2%, u usporedbi sa stopom lažno negativnih rezultata od 8.6% koju ima CBA. CBA daje veliku učinkovitost u usporedbi s klasifikatorima dobro poznatih algoritama, a također omogućuje lakšu interpretaciju rezultata koja pomaže sigurnosnim analitičarima i krajnjim korisnicima da lakše razumiju evoluciju zlonamjernih URL-ova.

Rezultati eksperimenta nad danim skupom od 75 492 stranice korištenjem algoritma statičke heuristike daju stopu zlonamjernih detektiranih stranica od 6.83%, tako da se može zaključiti da je većina detekcija u ovom eksperimentu lažna i može se djelomično smanjiti uvođenjem određenih optimizacija koje se uglavnom svode na uvođenje listi dozvoljenih uzoraka. Dijagnoze zlonamjernosti koje se pojavljuju poredane silazno po učestalosti pojavljivanja su mali `iframe` elementi, obfuscirani JavaScript kod, `iframe` element i `object` element, `iframe` element i `embed` element te postojanje `iframe` elemenata uz mali broj skripti i prisutnost preusmjeravanja. Nakon optimizacije ovog algoritma stopa zlonamjernosti uzorka stranica iznosi 0.66%, što je blizu očekivanom postotku zlonamjernih stranica na stvarnom testnom skupu podataka te ukazuje na to da su optimizacije bitno smanjile stopu lažno negativne detekcije, pogotovo za detekciju malih i zlonamjernih iframe elemenata čiji se broj ovdje znatno smanjio.

Rezultati eksperimenta nad danim skupom od 75 492 stranice korištenjem algoritma zasnovanog na bodovanju daju stopu zlonamjernih detektiranih stranica od 11.34%. Zaključuje se da je stopa lažno pozitivnih rezultata ovog algoritma relativno visoka uz manje mogućnosti optimizacije nego kod statističke heuristike. Algoritam ne uzima u obzir konkretne atribute nego njihove kombinacije, a i zamišljen je ne kao algoritam koji će stranice binarno klasificirati kao zlonamjerne ili benigne, nego kao potencijalno zlonamjerne te ih proslijediti odgovarajućim uređajima za detekciju ili stručnjacima za sigurnost za ručnu analizu koji s tim algoritmom čine drugu komponentu hibridnog sustava za ispitivanje zlonamjernosti stranica. Nakon provedene optimizacije, stopa zlonamjernosti skupa stranica iznosi 9.81%. Optimizacije koje su provedene samo su malo smanjile njegovu stopu lažno

negativne detekcije iako se očekuje da su povećali njegovu točnost i pouzdanost. Algoritam temeljen na mehanizmu bodovanja najmanje je pouzdan od promatranih algoritama i vrlo često otkriva stranice kao zlonamjerne iako nisu označene niti jednim od drugih algoritama.

Algoritam statičke heuristike i algoritam zasnovan na bodovanju ne mogu biti korišteni kao pouzdani alati za detekciju zlonamjernosti bez dodatnih provjera budući da imaju relativno visoku stopu lažno pozitivne detekcije pa se njihovi rezultati trebaju prosljeđivati pouzdanijim algoritmima ili mehanizmima koji će donijeti konačnu odluku. Postoji određena korelacija između detekcija algoritmom statičke heuristike i algoritmom zasnovanim na mehanizmu bodovanja, odnosno veliki broj stranica koje su klasificirane kao zlonamjerne algoritmom statičke heuristike također će biti tako klasificirane algoritmom zasnovanim na mehanizmu bodovanja.

Udio zlonamjernih stranica dobivenih analizom algoritmom Yara pravila iznosi 7.27%. Dijagnoze zlonamjernosti koje se pojavljuju poredane silazno po učestalosti pojavljivanja su JavaScript uključivanja i paketizacije funkcija, JavaScript obfuskacija i replica watches adware. Algoritam zasnovan na Yara pravilima ima malu stopu lažno pozitivne detekcije te se može smatrati boljim algoritmom. Nakon optimizacije ovog algoritma stopa zlonamjernosti uzorka stranica iznosi 1.09%, što je blizu očekivanom postotku zlonamjernih stranica na stvarnom testnom skupu podataka te ukazuje na to da su optimizacije znatno smanjile stopu lažno pozitivne detekcije, pogotovo za detekciju JavaScript uključivanja i paketizaciju funkcija.

Algoritam zasnovan na Yara pravilima općenito je pouzdaniji od ostalih dvaju algoritama jer ima malu stopu lažno pozitivnih rezultata, može preciznije otkriti konkretnije uzorke zlonamjernosti, biti lakše prilagođen za nove vrste napada i dati točniju dijagnozu u usporedbi s drugim dvjema algoritmima. Također je moguće dodavanjem Yara pravila ciljati određeni uski skup zlonamjernosti ili određenu zlonamjernost koju ostali algoritmi neće pronaći. Ipak, algoritam zasnovan na Yara pravilima ima slabosti svih algoritama koji su zasnovani na potpisu, a to je određena stopa lažno negativnih rezultata zbog mogućnosti da Yara pravila ne detektiraju izmijenjene zlonamjerne isječke koda koji i dalje imaju istu funkcionalnost i nakon izmjene.

7 Literatura

[1] V. L. Le, I. Welch, X. Gao, and P. Komisarczuk, Victoria University of Wellington, "Identification of Potential Malicious Web Pages," *UWL Repository*, 17.1.2011. [Online]. Dostupno: http://repository.uwl.ac.uk/id/eprint/773/1/CRPITV116Le.pdf. [Pristup: 14.5.2021].

[2] Wikimedia Foundation, "Client honeypot," *Wikipedia*, 24.7.2020. [Online]. Dostupno: https://en.wikipedia.org/wiki/Client_honeypot. [Pristup: 9.4.2021].

[3] Cisco and/or its affiliates, "Clamav, " *ClamavNet*. [Online]. Dostupno: https://www.clamav.net/. [Pristup: 14.5.2021].

[4] T. Zukina, "Detecting injected malicious HTML and JavaScript fragments in Web pages," *GitHub*, 5.4.2021. [Online]. Dostupno: https://github.com/tibor-zukina/malicious-web-pages-detection. [Pristup: 5.4.2021].

[5] Python Software Foundation, "Download Python," *Python.org*. [Online]. Dostupno: https://www.python.org/downloads/. [Pristup: 16.3.2021].

[6] MongoDB, Inc., "MongoDB Community Download," *MongoDB*. [Online]. Dostupno: https://www.mongodb.com/try/download/community. [Pristup: 16.3.2021].

[7] R. Walters, "Getting Started with Python and MongoDB: MongoDB Blog," *MongoDB*, 16.3.2021. [Online]. Dostupno: https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb. [Pristup: 16.4.2021].

[8] MongoDB, Inc., "objectid – Tools for working with MongoDB ObjectIds," *objectid – Tools for working with MongoDB ObjectIds - PyMongo 3.11.4 documentation*. [Online]. Dostupno: https://pymongo.readthedocs.io/en/stable/api/bson/objectid.html. [Pristup: 16.4.2021].

[9] S. Kumi, C. H. Lim, and S.-G. Lee, "Malicious URL Detection Based on Associative Classification," *MDPI*, 31-Jan-2021. [Online]. Dostupno: https://www.mdpi.com/1099-4300/23/2/182/pdf. [Pristup: 28.4.2021].

[10] Stichting Cuckoo Foundation, "Automated Malware Analysis," *Cuckoo Sandbox - Automated Malware Analysis*, 19.6.2019. [Online]. Dostupno: https://cuckoosandbox.org/. [Pristup: 28.4.2021].

[11] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, i H. M. Levy, Department of Computer Science and Engineering University of Washington, "SpyProxy," *SpyProxy: Execution-based Detection of Malicious Web Content*. [Online]. Dostupno: https://www.usenix.org/legacy/event/sec07/tech/full_papers/moshchuk_new/moshchuk_new_html/index.html. [Pristup: 28.4.2021].

[12] Imperva, "What is phishing: Attack techniques & scam examples: Imperva," *Learning Center*, 17.6.2020. [Online]. Dostupno: https://www.imperva.com/learn/application-security/phishing-attack-scam/. [Pristup: 28.4.2021].

[13] Software Freedom Conservancy, "Selenium automates browsers. That's it!," SeleniumHQ Browser Automation. [Online]. Dostupno: https://www.selenium.dev/. [Pristup: 23.4.2021].

[14] OpenPhish, "Phishing Intelligence" *OpenPhish*. [Online]. Dostupno: https://openphish.com/. [Pristup: 2.5.2021].

[15] VXVault, "VXVault," *VX Vault*. [Online]. Dostupno: http://vxvault.net/ViriList.php. [Pristup: 2.5.2021].

[16] Abuse.ch, "Malware URL exchange," *URLhaus*. [Online]. Dostupno: https://urlhaus.abuse.ch/. [Pristup: 2.5.2021].
[17] Fayyad, U.M.; Irani, K.B. Multi-interval discretization of continuous-valued attributes for classification learning. In Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambéry, France, 28.8.-3.9.1993; Volume 2.
[18] Jeeva, S.C.; Rajsingh, E.B. Intelligent phishing url detection using association rule mining. Hum. Centric Comput. Inf. Sci. 2016, 6.
[19] A. Bifet, B. Durrant, E. Frank, L. Hunt, and G. Holmes, "Weka 3: Machine Learning Software in Java," *Weka 3 - Data Mining with Open Source Machine Learning Software in Java*. [Online]. Dostupno: https://www.cs.waikato.ac.nz/ml/weka/. [Pristup: 2.5.2021].
[20] Wikimedia Foundation, "Support-vector machine," *Wikipedia*. [Online]. Dostupno: https://en.wikipedia.org/wiki/Support-vector_machine. [Pristup: 2.5.2021].
[21] Wikimedia Foundation, "Naive Bayes classifier," *Wikipedia*. [Online]. Dostupno: https://en.wikipedia.org/wiki/Naive_Bayes_classifier. [Pristup: 2.5.2021].
[22] Wikimedia Foundation, "Logistic regression," *Wikipedia*. [Online]. Dostupno: https://en.wikipedia.org/wiki/Logistic_regression. [Pristup: 2.5.2021].
[23] Ma, J.; Saul, L.K.; Savage, S.; Voelker, G.M. Beyond blacklists: Learning to detect malicious web sites from suspicious URLs. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28.6.–1.7. 2009; pp. 1245–1253.
[24] Seifert, I. Welch i P. Komisarczuk, Victoria University of Wellington, "Identification of Malicious Web Pages with Static Heuristics," *ResearchGate*, 31.1.2009. [Online]. Dostupno: https://www.researchgate.net/profile/Christian-Seifert-3/publication/202141488_Identification_of_Malicious_Web_Pages_with_Static_Heuristics/links/020ed9612599fbb7983d2250/Identification-of-Malicious-Web-Pages-with-Static-Heuristics.pdf. [Pristup: 3.5.2021].
[25] The Honeynet Project, "Capture-HPC," *The Honeynet Project*. [Online]. Dostupno: https://www.honeynet.org/projects/old/capture-hpc/. [Pristup: 4.5.2021].
[26] AOL Inc., "DMOZ," *DMOZ - The Directory of the Web*, 10.1.2017. [Online]. Dostupno: https://dmoz-odp.org/. [Pristup: 4.5.2021].
[27] S. Agnew, "Web Scraping and Parsing HTML in Python with Beautiful Soup," *Twilio Blog*, 22.10.2019. [Online]. Dostupno: https://www.twilio.com/blog/web-scraping-and-parsing-html-in-python-with-beautiful-soup. [Pristup: 6.5.2021].
[28] W3Schools, "HTML <iframe> Tag," *W3Schools*. [Online]. Dostupno: https://www.w3schools.com/tags/tag_iframe.asp. [Pristup: 6.5.2021].
[29] W3Schools, "HTML <script> Tag," *W3Schools*. [Online]. Dostupno: https://www.w3schools.com/tags/tag_script.asp. [Pristup: 6.5.2021].
[30] W3Schools, "HTML <object> Tag," *W3Schools*. [Online]. Dostupno: https://www.w3schools.com/tags/tag_object.asp. [Pristup: 6.5.2021].
[31] W3Schools, "HTML <embed> Tag," *W3Schools*. [Online]. Dostupno: https://www.w3schools.com/tags/tag_embed.asp. [Pristup: 6.5.2021].
[32] W3Schools, "XSLT <xsl:processing-instruction>," *W3Schools*. [Online]. Dostupno: https://www.w3schools.com/xml/ref_xsl_el_processing-instruction.asp. [Pristup: 6.5.2021].
[33] R. Auger, "XML Injection," *The Web Application Security Consortium*, 30.12.2009. [Online]. Dostupno: http://projects.webappsec.org/w/page/13247004/XML%20Injection. [Pristup: 6.5.2021].

[34] BigBug, "Create XML in Javascript," *Stack Overflow*, 15.1.2013. [Online]. Dostupno: https://stackoverflow.com/questions/14340894/create-xml-in-javascript/51464792. [Pristup: 6.5.2021].
[35] P. Gujral, "Default arguments in Python," *GeeksforGeeks*. [Online]. Dostupno: https://www.geeksforgeeks.org/default-arguments-in-python/. [Pristup: 7.5.2021].
[36] Media Temple, "How do I redirect my site using a META Tag?," *Media Temple Community*. [Online]. Dostupno: https://mediatemple.net/community/products/dv/204645160/how-do-i-redirect-my-site-using-a-meta-tag. [Pristup: 10.5.2021].
[37] W3Schools, "How TO - Redirect to Another Webpage," *W3Schools*. [Online]. Dostupno: https://www.w3schools.com/howto/howto_js_redirect_webpage.asp. [Pristup: 11.5.2021].
[38] LZone, "Python re.sub," *LZone*. [Online]. Dostupno: https://lzone.de/examples/Python%20re.sub. [Pristup: 11.5.2021].
[39] P. W. Pwtempuser, "Yahoo Search," *ProgrammableWeb*, 15.8.2014. [Online]. Dostupno: https://www.programmableweb.com/api/yahoo-search. [Pristup: 16.5.2021].
[40] www.malwaredomainlist.com, "Query Malware Domain List or alternatively, Submit malware urls and share information in our Forums Follow us on Twitter," *MDL*. [Online]. Dostupno: http://www.malwaredomainlist.com/. [Pristup: 16.5.2021].
[41] N. Fox, "YARA Rules Guide: Learning this Malware Research Tool: Varonis," *Inside Out Security*, 20.5.2021. [Online]. Dostupno: https://www.varonis.com/blog/yara-rules/. [Pristup: 20.5.2021].
[42] VirusTotal, *VirusTotal*. [Online]. Dostupno: https://www.virustotal.com/gui/. [Pristup: 24.5.2021].
[43] V. M. Alvarez, "The Python interface for YARA," *GitHub*, 27.4.2021. [Online]. Dostupno: https://github.com/VirusTotal/yara-python. [Pristup: 20.5.2021].
[44] J. Bery, "Yara rules to be used with the Burp Yara-Scanner extension," *GitHub*, 27.6.2016. [Online]. Dostupno: https://github.com/codewatchorg/Burp-Yara-Rules. [Pristup: 20.5.2021].
[45] J. Bery, "codewatchorg/Burp-Yara-Rules," *GitHub*, 27.6.2016. [Online]. Dostupno: https://github.com/codewatchorg/Burp-Yara-Rules/blob/master/all.yar. [Pristup: 10.5.2021].
[46] Palo Alto Networks, "What is an Exploit Kit?," *Palo Alto Networks*. [Online]. Dostupno: https://www.paloaltonetworks.com/cyberpedia/what-is-an-exploit-kit. [Pristup: 20.5.2021].
[47] BeEF project, "The Browser Exploitation Framework Project," *BeEF*. [Online]. Dostupno: https://beefproject.com/. [Pristup: 20.5.2021].
[48] Insane Coder i TylerH, "Javascript - What is the custom function(p,a,c,k,e,d) used for?," *Stack Overflow*, 29.1.2014. [Online]. Dostupno: https://stackoverflow.com/questions/21423397/what-is-the-custom-functionp-a-c-k-e-d-used-for. [Pristup: 20.5.2021].
[49] Microsoft, Inc., "Microsoft," *Office 365 Login - Microsoft Office*. [Online]. Dostupno: https://www.office.com/. [Pristup: 10.4.2021].
[50] Meta-Chart, "Graphing/Charting and General Data Visualization App," *Meta-Chart*. [Online]. Dostupno: https://www.meta-chart.com/. [Pristup: 5.6.2021].

[51] themepunch, "Slider Revolution Responsive WordPress Plugin," CodeCanyon, 29.4.2021. [Online]. Dostupno: https://codecanyon.net/item/slider-revolution-responsive-wordpress-plugin/2751380. [Pristup: 30.5.2021].
[52] T. T. Thiện, "JavaScript Deobfuscator and Unpacker," *de4js*. [Online]. Dostupno: https://lelinhtinh.github.io/de4js/. [Pristup: 31.5.2021].

Detekcija ubačenih zlonamjernih HTML i JavaScript fragmenata u Web stranicama

Sažetak

Uz sve načine na koje Web olakšava prijenos informacija i povećava produktivnost svakodnevnih aktivnosti u modernom svijetu, zbog zlouporabe različitih zlonamjernih dionika donosi i brojne opasnosti te rizike za sve svoje korisnike. Napadači koriste Web kako bi u njega ubacili zlonamjerni HTML i JavaScript kod uz pomoć kojeg pokušavaju direktno ili indirektno ostvariti određenu zaradu. Pravovremena detekcija takvih stranica izuzetno je bitna kako bi se zaštitili svi oni koji pristupaju Webu uz očuvanje svih prednosti koje donosi Web tehnologija.

Detekcija ubačenih JavaScript i HTML fragmenata nije trivijalna te je mnoštvo predloženih metoda koje pokušavaju riješiti taj problem. Cilj svakog od predloženih algoritama je pronaći maksimalni broj zlonamjernih stranica uz minimalni broj lažno pozitivnih detekcija u minimalnom vremenu izvođenja i uz maksimalnu učinkovitost. Objasnit će se i implementirati odabrani načini detekcije ubačenih zlonamjernih HTML i JavaScript elemenata u Web stranice. Metode kojima će se rad baviti koristit će isključivo HTML Web stranice, odnosno neće imati mogućnost pristupa izvornom kodu Web aplikacije.

Niti jedan učinkoviti algoritam za detekciju zlonamjernih stranica nije potpuno pouzdan i točan, odnosno svaki od njih ima određeni udio lažno pozitivnih rezultata te propušta detekciju određenog broja zlonamjernih stranica. S druge strane, klasični pristup detekcija stranica korištenjem uređaja za detekciju visoke interaktivnosti i ručnim pregledom stručnjaka za sigurnost rezultira puno većim trajanjem provjere za svaku od Web stranica. Opisani algoritmi uvode kompromis između brzine izvođenja, odnosno učinkovitosti i pouzdanosti, odnosno broja propuštenih napada i lažno pozitivnih detekcija.

Algoritam zasnovan na asocijativnoj klasifikaciji (CBA) ima visoku točnost s vrlo niskom stopom lažno pozitivnih i lažno negativnih rezultata. Performanse tog algoritma uspoređene su s dobro poznatim klasifikacijskim algoritmima, uključujući *support vector machine* (SVM), naivni Bayes i logističku regresiju. Analiza pokazuje da je CBA metoda bolja od naivnog Bayesa s većom preciznošću, dok logistička regresija i SVM imaju visoku stopu negativnih rezultata u usporedbi sa stopom lažno negativnih rezultata koju ima CBA. CBA daje veliku učinkovitost u usporedbi s klasifikatorima dobro poznatih algoritama, a također omogućuje lakšu interpretaciju rezultata koja pomaže sigurnosnim analitičarima i krajnjim korisnicima da lakše razumiju evoluciju zlonamjernih URL-ova.

Rezultati eksperimenta korištenjem algoritma statičke heuristike daju stopu zlonamjernih detektiranih stranica od 6.83%, tako da se može zaključiti da je većina detekcija u ovom eksperimentu lažna i može se djelomično smanjiti uvođenjem određenih optimizacija.

Rezultati eksperimenta korištenjem algoritma zasnovanog na bodovanju daju stopu zlonamjernih detektiranih stranica od 11.34%. Zaključuje se da je stopa lažno pozitivnih rezultata ovog algoritma relativno visoka uz manje mogućnosti optimizacije nego kod

statičke heuristike. Algoritam stranice neće binarno klasificirati kao zlonamjerne i benigne, nego kao potencijalno zlonamjerne te ih proslijediti odgovarajućim uređajima za detekciju ili stručnjacima za sigurnost za ručnu analizu koji s tim algoritmom čine drugu komponentu hibridnog sustava za ispitivanje zlonamjernosti stranica.

Algoritam statičke heuristike i algoritam zasnovan na bodovanju ne mogu biti korišteni kao pouzdani alati za detekciju zlonamjernosti bez dodatnih provjera budući da imaju relativno visoku stopu lažno pozitivne detekcije pa se njihovi rezultati trebaju prosljeđivati pouzdanijim algoritmima ili mehanizmima koji će donijeti konačnu odluku. Postoji određena korelacija između detekcija algoritma statičke heuristike i algoritma zasnovanog na mehanizmu bodovanja, odnosno veliki broj stranica koje su klasificirane kao zlonamjerne algoritmom statičke heuristike također će biti tako klasificirane algoritmom zasnovanim na mehanizmu bodovanja.

Udio zlonamjernih stranica dobivenih analizom algoritmom Yara pravila iznosi 7.27%. Algoritam zasnovan na Yara pravilima ima malu stopu lažno pozitivne detekcije te se može smatrati boljim algoritmom.

Algoritam zasnovan na Yara pravilima općenito je pouzdaniji od ostalih dvaju algoritama jer ima malu stopu lažno pozitivnih rezultata, može preciznije otkriti konkretnije uzorke zlonamjernosti, biti lakše prilagođen za nove vrste napada i dati točniju dijagnozu u usporedbi s drugim dvjema algoritmima. Ipak, algoritam zasnovan na Yara pravilima ima slabosti svih algoritama koji su zasnovani na potpisu, a to je određena stopa lažno negativnih rezultata zbog mogućnosti da Yara pravila ne detektiraju izmijenjene zlonamjerne isječke koda koji i dalje imaju istu funkcionalnost i nakon izmjene.

Ključne riječi: Web; HTML; JavaScript; zlonamjerne stranice; detekcija; asocijativna klasifikacija; statička heuristika; mehanizam bodovanja; Yara pravila

Detecting injected malicious HTML and JavaScript fragments in Web pages

Abstract

Along with all ways the Web makes it easier to transmit information and increase the productivity of everyday activities in the modern world, due to misuse of the different stakeholders, it also introduces numerous dangers and risks for all its users. Attackers use the Web to inject malicious HTML JavaScript code which they use to directly or indirectly obtain certain profits. Timely detection of such Web pages is crucial to protect everyone who accesses the Web while keeping all advantages brought by the Web technology. The detection of the injected JavaScript and HTML fragments is not trivial and there are plenty of suggested methods trying to fix this problem. The purpose of each of the suggested algorithms is to find the maximum number of malicious websites with the minimum number of false-positive detections in the minimal execution time and the maximum efficiency. The selected ways of injected malicious HTML and JavaScript elements in the Web pages will be explained and implemented. The methods the thesis will cover will use exclusively HTML of the Web page and won't have the possibility to access the Web application source code. There is no effective algorithm of malicious webpages detection that is completely reliable and accurate, but each of them has a certain rate of false-positive results and misses the detection of a certain number of malicious websites. On the other hand, the classic approach to detecting the websites using high-interactivity test devices and manual examination by the security experts results in a much greater check duration for each of the Web pages. The described algorithms introduce the compromise between the execution speed or effectiveness and the reliability or the number of misses attacked and false-positive detections. The algorithm based on the associative classification (CBA) has high accuracy with a very low false-positive rate and false-negative rate. The performance of this algorithm has been compared with the well-known classification algorithms, including support vector machine (SVM), naive Bayes, and logistic regression. The analysis shows that the CBA method is better than the naive Bayes with its high precision, while the logistic regression and SVM have a high false-negative rate compared to the false-negative rate of the CBA algorithm. CBA reaches high effectiveness compared to the classificators of the well-known algorithms and it also enables an easy interpretation of the results that help the security analysts and end-users to more easily understand the evolution of the malicious URLs. The experiment results using the static heuristics algorithm give a malicious webpages rate of 6.83%, so it is possible to include that the majority of detection in this experiment is false, which can be partially reduced by introducing certain optimizations. The experiment results using the algorithm based on the scoring mechanism give a malicious webpages rate of 11.34%. It can be concluded that the false positive rate of this algorithm is relatively high with a smaller possibility of optimization than for the static heuristics. This algorithm won't make a binary classification of the webpages as malicious or benign, but it will mark them as potentially malicious and forward them to the proper detection devices or security experts for manual analysis that make up the second component of the hybrid website malice testing system

along with this algorithm. Static heuristics algorithm and the algorithm based on the scoring mechanism cannot be used as reliable tools for malice detection without the extra checks, because they have a relatively high false-positive rate so their results should be forwarded to more reliable algorithms and mechanisms that will make the final decision. There is a certain correlation between the detections of the static heuristics algorithm and the algorithm based on the scoring mechanism, which means that a certain number of Web pages that have been classified as malicious by the static heuristics algorithm will also be classified as malicious by the algorithm based on the scoring mechanism. The malicious webpages rate obtained by the Yara rules algorithms analysis is 7.27%. The algorithm based on Yara rules has a lower false detection rate in general and can be considered a better algorithm. The algorithm based on Yara rules is more reliable than the other two algorithms in general because it has a low false-positive results rate, it can detect specific malicious samples and be more easily customized for new types of attacks compared and provide a more accurate diagnosis compared to the other two algorithms. Still, the algorithm based on Yara rules has the weaknesses of all signature-based algorithms, and that is a certain false-negative rate due to the possibility that Yara rules fail to detect the altered malicious code fragments that still have the same functionality even after their alteration.

Keywords: Web; HTML; JavaScript; malicious webpages; detection; associative classification; static heuristics; scoring mechanism; Yara rules