SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1173

Sustav za generiranje skupa podataka obfusciranog JavaScript koda

Bojan Puvača

Zagreb, lipanj 2023.

SADRŽAJ

1. Uvod

Internetski je prostor otvoren i pristupačan svima, zbog čega je idealan za komunikaciju i dijeljenje sadržaja među korisnicima. Međutim, to isto svojstvo čini ga i idealnim za kibernetičke kriminalce koji motivirani raznim ciljevima, od financijske do političke prirode, pokušavaju iskoristiti slobodu koju internet pruža kako bi ih ostvarili.

Zloćudni se kod na internetu najčešće pojavljuje u obliku JavaScript skripti koje se pokreću pri pokretanju web stranice ili pri interakciji korisnika s web stranicom. Najčešći oblici napada su *phishing* napadi u kojima se stranica korisniku predstavlja kao neka druga stranica kako bi se došlo do korisničkih osobnih ili financijskih podataka, napadi u kojima se na korisničko računalo pokušavaju prenijeti zloćudni programi koji će se onda izvršavati na žrtvinom računalu te skripte koje koriste računalo žrtve za rudarenje kriptovaluta bez znanja korisnika [11].

Ipak, mana korištenja zloćudnih JavaScript skripti na klijentskoj strani je što je njihov kod vidljiv svakom korisniku te se može ručno ili automatski analizirati te otkriti prijetnja. Zbog toga neki napadači pokušavaju proces analize koda otežati procesom obfuskacije. Obfuskacija je postupak kojim se programski kod preoblikuje s ciljem da bude teško razumljiv ljudima ili računalima [8].

U prijašnjim se radovima istraživala detekcija zloćudnih web stranica koristeći metode strojnog učenja [14] [6] kao i drugim metodama [9] [13]. Također se istraživala i statička detekcija obfusciranog JavaScript koda [10], no područje otkrivanja obfusciranog koda koristeći strojno učenje još uvijek nije u velikoj mjeri istraženo.

Cilj je ovog rada olakšati daljnji razvoj metoda strojnog učenja za detekciju obfusciranog koda tako što se stvori sustav koji će generirati proizvoljno velik skup podataka obfusciranog JavaScript koda koji će se moći koristiti za treniranje i testiranje modela strojnog učenja u nekom od daljnjih radova. U ovom se radu tom problemu pristupa koristeći Selenium, alat za automatizaciju web preglednika, zbog čega je i dodatni cilj sagledati mogućnosti, prednosti i mane tog alata u kontekstu izrade takvog sustava.

Selenium je alat za automatizaciju web preglednika koji omogućuje programsko upravljanje web preglednicima kao što su Google Chrome, Mozilla Firefox i drugi. On

omogućava automatiziranu interakciju s web stranicama zbog čega se može koristiti za testiranje web stranica, prikupljanje podataka s web stranica, ili u ovom slučaju za prikupljanje JavaScript kodova s web stranica te za obfuskaciju tih kodova koristeći servise koji nemaju javno dostupna programska sučelja.

Sustav za generiranje obfusciranog JavaScript koda implementiran je kao skripta u programskom jeziku Python te se sastoji od dva dijela. Sustav prvo prikuplja JavaScript kodove koristeći popis web adresa u obliku *.txt* datoteke. Svakoj se web lokaciji pristupa koristeći Selenium te se čitajući izvorni kod izdvaja isključivo JavaScript kod web lokacije. Zatim se Seleniumom pristupa različitim servisima za obfuskaciju JavaScript koda te se na web stranici svakog od servisa umeće svaki dohvaćeni kod te se on obfuscira te se pohranjuje kao JavaScript datoteka.

U nastavku slijede dva teorijska poglavlja posvećena obfuskaciji te alatima za automatizaciju web preglednika s naglaskom na Selenium. Četvrto poglavlje odnosi se na arhitekturu, implementaciju, rezultate i moguće nadogradnje sustava za generiranje obfusciranog JavaScript koda te na kraju slijedi zaključak.

2. Obfuskacija

2.1. Definicija obfuskacije

Obfuskacija u kontekstu razvoja softvera je postupak kojim se stvara programski kod s namjerom da bude teško razumljiv ljudima ili računalima [8].

Osim skrivanja zloćudnog koda, obfuscirati se može i iz defenzivnih razloga, primjerice ako se želi zaštititi kod kao intelektualno vlasništvo tako što se otežava njegovo reverzno inženjerstvo ili ako se žele sakriti potencijalne ranjivosti u sustavu od napadača. To znači da web stranica koja ima obfusciran kod ili dio koda ne sadrži nužno zloćudan sadržaj.

Proces suprotan obfuskaciji je deobfuskacija [8]. Deobfuskacijom se obfuscirani kod pokušava vratiti što je bliže moguće izvornom obliku.

2.2. Mjerenje uspješnosti obfuskacije

Neki od kriterija obfuskacije kojima se može mjeriti njena uspješnost su: jačina, izmjenjenost, složenost i trošak [8].

Jačina mjeri otpornost automatizirane deobfuskacije koda. Što je obfuskacija jača, to je više vremena i resursa potrebno za deobfuskaciju koda. Izmjenjenost se odnosi na stupanj različitosti izvornog i obfusciranog koda. Neke metode obfuskacije mogu značajno smanjiti brzinu izvođenja koda, povećati njegovu veličinu ili memorijski utisak. Tu cijenu obfuskacije nazivamo troškom. Složenost se odnosi na broj metoda i slojeva obfuskacije. Što je obfuskacija slojevitija, to je teže razumijeti i obfuscirati kod.

2.3. Minifikacija

Minifikacija je postupak kojim se smanjuje veličina izvornog koda tako što se uklanjaju nepotrebni znakovi i skraćuju identifikatori u kodu [3]. Iako se minifikacijom također

smanjuje čitljivost izvornog koda, ona se razlikuje od obfuskacije po motivaciji iza postupka kao i po rezultatnom kodu.

Minifikacijom se kod ne može zašititi od reverznog inženjerstva jer se kod jednostavno može transformirati nazad u izvorni oblik preslikavanjem skraćenih naziva varijabli i funkcija u njihove originalne nazive. Takvo trivijalno preslikavanje nije moguće za obfuscirani kod jer se obfuskacijom mijenja i struktura koda. Još je jedna razlika između oba postupka ta da se minifikacijom uvijek smanjuje veličina koda, dok se obfuskacijom veličina koda može i povećati, osobito korištenjem tehnike umetanja mrtvog koda.

Budući da minifikaciju nema smisla koristiti u maliciozne svrhe, u ovom se radu ona neće razmatrati, iako bi idealno model strojnog učenja koji bi koristio podatke nastale sustavom opisanim u ovom radu trebao moći razlikovati minificirani kod od obfusciranog.

2.4. Alati za obfuskaciju

Alati za obfuskaciju ili obfuskatori, programi su koji automatski obfusciraju predani izvorni kod ovisno o nekim parametrima. Obfuskatori se razlikuju ovisno o programskom jeziku koda kojeg obfusciraju, o tome jesu li otvorenog koda ili ne, o načinu pristupa njihovom sučelju itd.

U kontekstu ovog rada bavimo se obfuskatorima otvorenog koda za programski jezik JavaScript i to onima s grafičkim web sučeljem. Takvih obfuskatora ima mnogo, a neki od njih su:

- *obfuscator.io*
- *freejsobfuscator.com*
- *htmlstrip.com/javascript-obfuscator*
- *codebeautify.org/javascript-obfuscator*

Obfuscator.io temelji se na *javascript-obfuscator* [4] obfuskatoru otvorenog koda, koji nudi sučelje iz naredbenog retka kao i *NodeJS* knjižnicu. Taj obfuskator nudi mnoštvo parametara i metoda obfuskacije te sadrži detaljnu dokumentaciju, za razliku od drugih gore navedenih obfuskatora koji su manje dokumentirani, nude manje metoda obfuskacije te nemaju opciju obfuskacije iz naredbenog retka ili programske knjižnice. Slika 2.1 prikazuje web sučelje obfuscator.io obfuskatora.

2.5. Metode obfuskacije

Obfuskacija nije jedan strogo definiran postupak, već se postiže slojevitom primjenom raznih metoda obfuskacije kako bi kod bio što manje čitljiv i razumljiv. U nastavku su navedene neke od najčešće korištenih metoda obfuskacije za programski jezik JavaScript.

2.5.1. Umetanje mrtvog koda

Umetanje mrtvog koda metoda je obfuskacije kojom se u izvorni kod umeću dijelovi koda koji se nikad ne izvršavaju ili ne utječu na ostatak koda [4]. To se primjerice može postići dodavanjem naredbi grananja s tautološkim ili kontradiktornim uvjetima, ovisno o tome želimo li da se neki blok koda izvrši ili ne. Ako je evaluacija tih uvjeta trivijalna, postupak deobfuskacije postaje jednostavan, zbog čega je za učinkovitost ove metode potrebno da uvjeti budu složeni i da se mogu evaluirati tek za vrijeme izvođenja programa. Neki drugi načini umetanja mrtvog koda su dodavanje funkcija koje se nikad ne pozivaju, naredbi koje ništa korisno ne rade i sl.

2.5.2. Transformacija identifikatora

Tranformacija identifikatora zajednička je metoda obfuskacije i minifikacije, ali s različitom svrhom, a potencijalno i implementacijom. Tipična tranformacija identifikatora će maksimalno smanjiti duljinu identifikatora pa će se identifikator funkcije npr. *parseDataFromJSON* skratiti u *a*. Iako je ovo idealno za minifikaciju, obfuskatori će najčešće podržavati i druge načine transformacije. Primjerice, zadana transformacija za *obfuscator.io* [5] će zamijeniti identifikatore nasumičnim hexadecimalnim nizovima znakova npr. u *_0x1ef459*, čime se dodatno otežava razlikovanje identifikatora od konstanti.

Transformacija identifikatora u kombinaciji s metodama kojima se mijenja struktura koda značajno otežava ručnu deobfuskaciju koda jer postaje teško preslikati obfuscirane dijelove koda u njihove originalne oblike.

2.5.3. String array

String array metodom se transformiraju konstantni znakovni nizovi tako što se podnizovi korištenih nizova pohranjuju u polje, te se u kod umjesto izvornih konstanti umeću vezani elementi tog polja. Tako se primjerice za korišteni konstantni znakovni

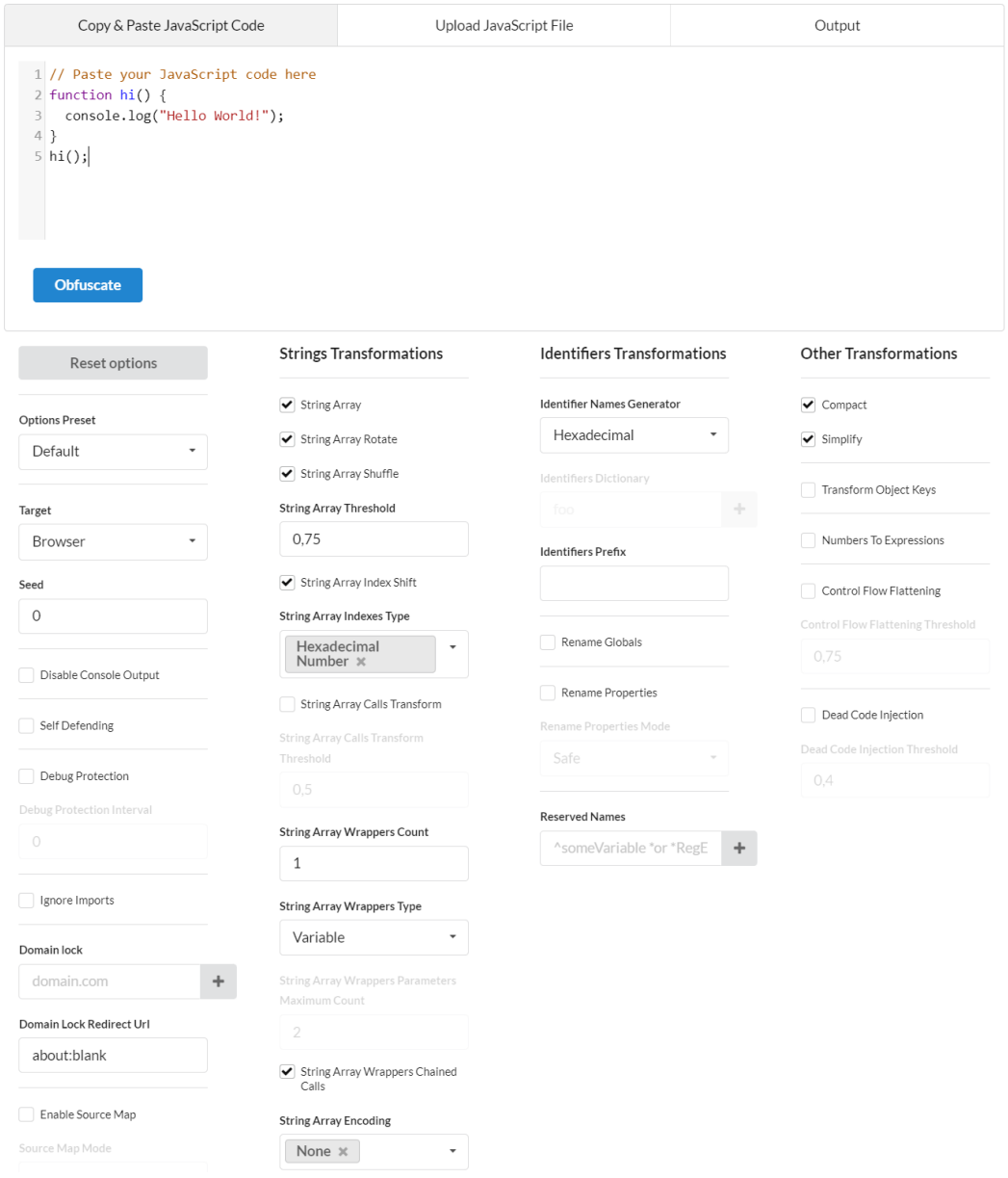
niz `"Hello world"` može definirati *String array* polje naredbom `let strArr = ["wo", "Hel", "lo ", "rld"]`, te se na mjesto korištenja originalnog znakovnog niza umetne `strArr[1]+strArr[2]+strArr[0]+strArr[3]`.

Ova se metoda može koristiti i za obfuskaciju identifikatora, tako da se u polje pohrane podnizovi identifikatora te se koristeći JavaScript funkciju `eval(script)` pozove funkcija ili pristupi elementu čiji se naziv dobije vezanjem podnizova iz polja. Ipak, ovaj postupak je redundantan ako se već koristi transformacija identifikatora.

2.5.4. Debug protection

Zaštita od otklanjanja pogrešaka (engl. *debug protection*) metoda je obfuskacije kojom se sprječava praćenje tijeka programa koristeći alate za razvoj [2]. Konkretno za JavaScript, alati kao što su *Chrome DevTools* i *Firebug* omogućuju korisniku da unutar browsera prati tijek programa postavljajući lomnu točku (engl. *breakpoint*) ispred bilo koje linije koda, čime se pauzira izvođenje programa prije te linije koda, što može značajno olakšati razumijevanje koda pri ručnoj deobfuskaciji. Ova metoda obfuskacije pokušava spriječiti korištenje lomnih točaka ili korištenje tih alata općenito.

Jedan od načina na koji se to može postići je umetajući ključnu riječ `debugger` unutar koda u beskonačnoj petlji čime se na otvaranje nekog od alata za razvoj neprekidno pauzira izvođenje programa [2]. Neke druge *debug protection* metode temelje se na računanju razlike u vremenu između izvršavanja dviju naredbi ili razlike u veličini prozora web preglednika [2].



Slika 2.1: obfuscator.io web sučelje

3. Automatizacija web preglednika i Selenium

Sustav za generiranje obfusciranog JavaScript koda trebao bi koristiti različite obfuskatore kako bi dobiveni kodovi bolje predstavljali stvarne obfuscirane kodove na webu, zbog čega nije dovoljno samo koristiti *javascript-obfuscator* obfuskator, iako je on najlakši za programsku implementaciju zbog dostupnosti NodeJS knjižnice. Kako bi sustav na uniforman način pristupao raznim JavaScript obfuskatorima kojima je jedini zajednički način pristupa web sučelje, predlaže se korištenje nekog od alata za automatizaciju web preglednika.

Automatizacija web preglednika je proces kojim se programski kontrolira web preglednik, što omogućuje automatizaciju raznih zadataka koji bi se inače morali obavljati ručno. Neke od primjena automatizacije web preglednika su testiranje web aplikacija i prikupljanje podataka s web stranica [7], ali se može koristiti i za automatizirano korištenje alata na webu kojima je dostupno samo web sučelje.

3.1. Alati za automatizaciju web preglednika

Postoje brojni alati za automatizaciju web preglednika koji se razlikuju u načinu korištenja, podržanim programskim jezicima i ciljanoj publici. Neki od najpopularnijih alata su:

- *Selenium*
- *Puppeteer*
- *Playwright*

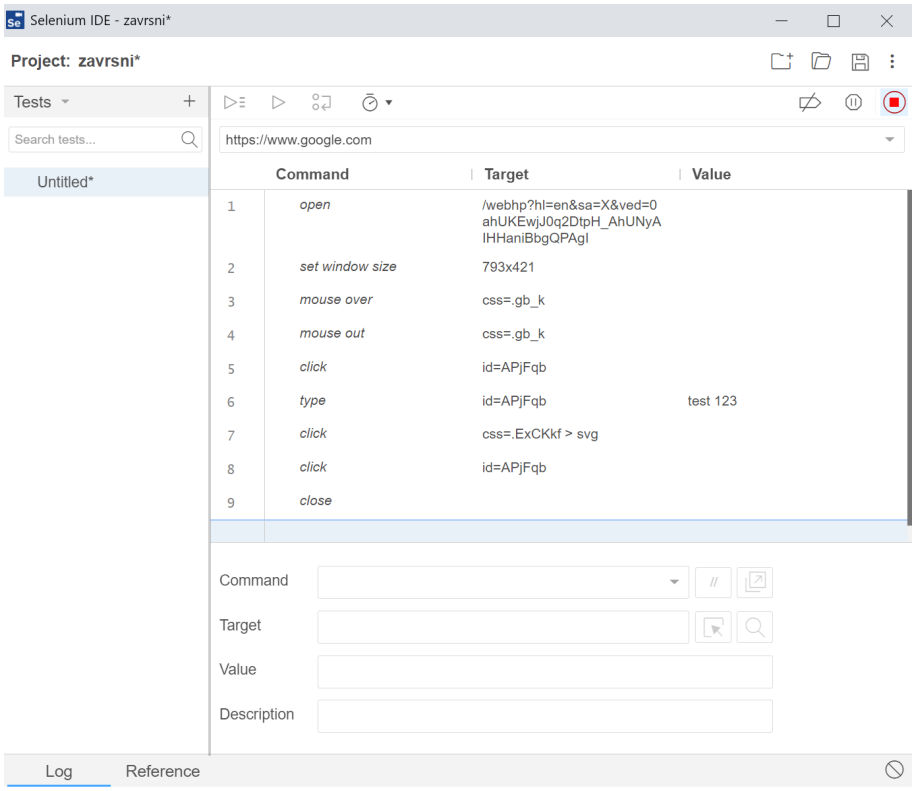
U ovom radu koristiti će se *Selenium* [12] jer je otvorenog koda i jer omogućuje programsko korištenje u programskom jeziku Pythonu koristeći programsko sučelje *Selenium WebDriver*.

3.2. Selenium

Selenium je alat otvorenog koda za automatizaciju web preglednika te se zapravo sastoji od triju alata koji se razlikuju po načinu uporabe: *Selenium IDE*, *Selenium WebDriver* i *Selenium Grid* [12].

3.2.1. Selenium IDE

Selenium IDE je dodatak dostupan za preglednike *Google Chrome* i *Firefox* koji omogućuje snimanje, reprodukciju, uređivanje, spremanje i izvoz testova za web aplikacije preko grafičkog sučelja. Na slici 3.1 prikazan je izgled jednog testa u grafičkom sučelju dodatka *Selenium IDE*.



Slika 3.1: Grafičko sučelje dodatka *Selenium IDE*

3.2.2. Selenium WebDriver

Selenium WebDriver je programsko sučelje koje omogućuje programsku automatizaciju web preglednika u programskim jezicima Java, C#, Ruby, Python, JavaScript i PHP [12]. Kompatibilan je s *Selenium IDE* dodatkom na način da se testovi snimljeni

u dodatku mogu izvesti kao programski kod u nekom od podržanih programskih jezika. Za programski jezik Python, *Selenium WebDriver* dostupan je kao Python paket koji se može instalirati pomoću alata *pip*.

Uporaba počinje stvaranjem objekta klase `WebDriver` koristeći metodu tvornicu za željeni web preglednik iz paketa `webdriver`. Nakon stvaranja objekta, koristeći njegove metode može se upravljati web preglednikom. Neke od najkorisnijih metoda su:

- `get(url)` - metoda kojom se otvara web stranica na zadanom *URL*-u
- `execute_script(script)` - metoda kojom se u web pregledniku izvršava zadani JavaScript kod
- `find_element(by, value)` - metoda kojom se pronalazi element na stranici koristeći neku od njegovih svojstvenih oznaka
- `element.send_keys(keys)` - metoda kojom se u odabrani element šalju zadane tipke odnosno upisuje tekst
- `element.click()` - metoda kojom se klikne na odabrani element

Pri instanciranju objekta klase `WebDriver` moguće je odabrati opciju `headless` koja omogućuje pokretanje web preglednika u pozadini bez grafičkog sučelja. Iako je automatizacija s `headless` opcijom brža i učinkovitija, ona ima i svoje nedostatke. Web stranice mogu detektirati korištenje ove opcije i ugraditi mehanizme protiv ekstrakcije podataka koji mogu otežati njihovo korištenje. Također, neke akcije kao što je simulirano korištenje miša ili mijenjanje veličine prozora nisu moguće u `headless` načinu rada.

3.2.3. Primjer korištenja Selenium WebDrivera

U nastavku je prikazan jednostavan primjer korištenja *Selenium WebDrivera* u programskom jeziku Python. U primjeru se otvara web stranica *Google* te se u tražilicu upisuje riječ obfuscation i šalje se zahtjev za pretraživanje pritiskom tipke enter. Nakon toga se zatvara web preglednik. Potrebni uvozi su izostavljeni zbog preglednosti. Pretpostavlja se da je u radnom direktoriju izvršna datoteka `chromedriver.exe` koja je potrebna za komunikaciju *Seleniuma* s web preglednikom *Google Chrome*.

```
service = Service("chromedriver.exe")
driver = webdriver.Chrome(service=service)
driver.get("https://www.google.com")
search_input = driver.find_element_by_name("q")
```

```
search_input.send_keys("obfuscation")
search_input.send_keys(Keys.ENTER)
driver.quit()
```

3.2.4. Selenium Grid

Selenium Grid je alat za pokretanje testova paralelno na više različitih web preglednika i udaljenih računala. Svaka različita kombinacija web preglednika i stroja naziva se čvorom [12] te se paralelno izvode testovi na svakom od čvorova. Paralelno izvođenje testova nije potrebno za ovaj rad te se neće koristiti.

3.2.5. Nedostatci korištenja Seleniuma

Alati za automatizaciju web preglednika poput *Seleniuma* primarno su namijenjeni testiranju klijentske strane web aplikacija, te su za tu svrhu optimizirani. Iz tog razloga postoje nedostaci *Seleniuma* kad se koristi za pristup web sučeljima.

Jedan od nedostataka je nemogućnost pokretanja skripti u pozadini bez da se pokreće web preglednik ako se ne koristi `headless` opcija. To može uzrokovati ovisnost o specifičnom okruženju te učiniti skriptu manje prenosivom [12].

Ako se na kontroliranoj web aplikaciji izvode neke asinkrone ili dugotrajne akcije, to može dovesti do sinkronizacijskih problema. Moguće je neke dijelove skripte izvršiti prerano ili prekasno, što može dovesti do nekonzistentnosti rezultata. Za takve slučajeve postoje mehanizmi čekanja u *Seleniumu* kojima se može odgoditi izvršavanje skripte dok nije zadovoljen neki uvjet. Ipak, ukoliko se radi o kompleksnim uvjetima, smisliti način na koji će se skripta sinkronizirati s preglednikom može biti zahtjevno.

Još jedan nedostatak odnosi se na neizbježnu ovisnost o specifičnostima web aplikacije. I najmanja promjena na web aplikaciji može uzrokovati neupotrebljivost skripte. U slučaju testiranja aplikacije ovo nije problem jer su promjene na aplikaciji interno poznate i dokumentirane, ali ako se radi o web sučelju za koje ne znamo hoće li se i kada će se promijeniti, ne možemo znati kada će skripta prestati funkcionirati.

3.2.6. Alternative automatizaciji web preglednika

Alati za automatizaciju web preglednika pružaju generalni pristup grafičkim sučeljima web aplikacija. Za svako sučelje kojemu želimo pristupiti potrebno je napisati skriptu koja će obavljati akcije nad njim te će ispravna skripta neovisno o specifičnostima web

aplikacije davati ispravne rezultate. Ipak, ta generalnost dolazi s nedostacima navedenim u odlomku 3.2.5. Ovisno o detaljima web sučelja koje želimo automatizirati, mogući su i drugi pristupi.

Ako se radi o aplikaciji čija se funkcionalnost koja nas zanima obavlja na poslužitelju, moguće je slanjem HTTP zahtjeva dobiti podatke koje želimo. Na taj način dobivamo jednostavnije i brže rješenje koje se ne oslanja na specifičnosti klijentske strane aplikacije, iako ipak ovisimo o specifičnostima poslužiteljske strane. Ovakav pristup ne funkcionira ako se dio funkcionalnosti obavlja na klijentskoj strani, kao što je to slučaj s nekim od aplikacija korištenih u ovom radu.

U slučaju da se funkcionalnost koju želimo automatizirati obavlja na klijentskoj strani, moguće je inspekcijom izvornog koda web stranice pronaći dio koda koji obavlja tu funkcionalnost. Ako se želi izbjeći korištenje alata za automatizaciju web preglednika, željeni se kod može izdvojiti i izvršiti izravno. Ovakav pristup može dobro funkcionirati za jednostavnije funkcionalnosti, ali ako kod sadrži puno međuovisnosti, proces izdvajanja koda može biti zahtjevan. Također, ovakav pristup ne funkcionira ako se dio funkcionalnosti obavlja na poslužiteljskoj strani.

Analizom web aplikacije moguće je utvrditi da se za traženu funkcionalnost koristi neka javno dostupna knjižnica, u kojem je slučaju moguće koristiti tu knjižnicu izravno. Ako se pak knjižnica koristi na poslužiteljskoj strani, teško je utvrditi o kojoj se točno knjižnici i kojoj verziji radi. Također može biti teško utvrditi na koji se ona točno način koristi. Ovakva analiza svejedno može biti korisna ako primjerice utvrdimo da različiti obfuskatori na webu zapravo koriste istu javno dostupnu knjižnicu. U tom slučaju ne moramo zasebno promatrati svaki od tih obfuskatora već ih možemo grupirati.

4. Sustav za generiranje obfusciranog JavaScript koda

4.1. Opis sustava

Sustav za generiranje obfusciranog JavaScript koda implementiran je kao Python skripta koja koristi *Selenium WebDriver* za dohvaćanje JavaScript kodova sa zadanih domena te za pristup trima različitima obfuskatorima JavaScript koda. Za svaki od obfuskatora koristimo sve dostupne tehnike obfuskacije pod pretpostavkom da nema razloga da netko tko koristi obfuskaciju da prikrije zlćudni kod ne koristi sve moguće tehnike obfuskacije.

Ulaz u sustav je popis domena s kojih će se dohvatiti JavaScript kodovi zadan kao .txt datoteka, a izlaz čine obfuscirani JavaScript kodovi u obliku datoteka pohranjenih u direktorij *obfuscated_codes*, grupirani prema domeni i korištenom obfuskatoru.

4.2. Inicijalizacija sustava i čitanje ulazne datoteke

Potrebne knjižnice uvedene za sustav su `requests` koji služi za slanje HTTP zahtjeva te `selenium` koji nam omogućuje inicijalizaciju *Selenium WebDrivera*.

Funkcija `instantiateDriver(headless:bool)` omotava inicijalizaciju *Selenium WebDrivera*. Parametar `headless` određuje hoće li se inicijalizacija obaviti u *headless* načinu rada. Sustav koristi web preglednik za testiranje od *Google Chrome-a* pa je potrebno da se *ChromeDriver* izvršna datoteka nalazi u radnom direktoriju. Dodan je argument `-ignore-certificate-errors` kako bi se izbjegle greške vezane uz certifikate koje otežavaju automatizaciju.

```
def instantiateDriver(headless:bool = False):
    options = Options()
    options.add_argument('--ignore-certificate-errors')
```

```
    if headless:
        options.add_argument('headless')
    return webdriver.Chrome(service=
        Service("chromedriver.exe"), options= options)
```

Pomoćna funkcija `readURLs()` čita ulaznu datoteku *urls.txt* i vraća listu domena.

```
def readURLs():
    urls = []
    with open('urls.txt', 'r') as file:
        for line in file:
            urls.append(line.strip())
    return urls
```

4.3. Dobivanje JavaScript kodova

Neobfuscirani se kodovi dohvaćaju funkcijom `fetchJSFromURL(URL)` koja za jedan URL dohvaća sve JavaScript kodove koji se koriste na toj stranici koristeći *Selenium WebDriver* za dohvaćanje HTML koda stranice te *requests* knjižnice za dohvaćanje .js datoteka. Funkcija vraća listu JavaScript kodova.

```
    def fetchJSFromURL(URL):
        driver = instantiateDriver(headless=True)
        driver.get(URL)
        js_urls = driver.execute_script(get_urls_script)
        codes = driver.execute_script(get_codes_script)
        for js_url in js_urls:
            try:
                response = requests.get(js_url)
            except requests.exceptions.MissingSchema:
                continue
            if response.status_code == 200:
            codes.append(response.content)
        driver.quit()
        return codes
```

Funkcija `fetchJSFromURL(URL)` stvara instancu *Selenium WebDrivera* u *headless* načinu rada te se povezuje na web lokaciju zadanu parametrom `URL`.

Zatim se na lokaciji pokreću dvije JavaScript skripte: `get_codes_script` i `get_urls_script` koje iz HTML koda stranice dohvaćaju JavaScript kodove i URL-ove JavaScript datoteka iz `<script>` *tagova*. Pomoću *requests* biblioteke, polje kodova `codes` se nadopunjava kodovima dohvaćenim HTTP zahtjevima prema URL-ovima. Funkcija zatim vraća to polje codes i zatvara web preglednik.

4.4. Obfuskacija kodova

Sustav obfuscira JavaScript kodove koristeći funkciju `obfuscate(codes, site)` koja prima listu kodova `codes` i enumeraciju `site` koja definira obfuskator koji će se koristiti. Ovisno o enumeraciji, funkcija delegira postupak obfuskacije funkciji odgovarajućeg obfuskatora.

```
def obfuscate(codes, site: Website):
    if site == Website.CODE_BEAUTIFY:
        return obfuscateWithCodeBeautify(codes)
    elif site == Website.OBFUSCATOR_IO:
        return obfuscateWithObfuscatorIO(codes)
    elif site == Website.HTML_STRIP:
        return obfuscateWithHtmlStrip(codes)
```

Sustav podržava tri različita obfuskatora: *obfuscator.io*, *htmlstrip.com/javascript-obfuscator* i *codebeautify.org/javascript-obfuscator*. Svaka od funkcija za obfuskaciju prima listu kodova `codes` i vraća listu obfusciranih kodova.

Budući da svaka funkcija za obfuskaciju ima sličnu strukturu, u nastavku je objašnjena samo `obfuscateWithObfuscatorIO(codes)`.

```
def obfuscateWithObfuscatorIO(codes):
driver = instantiateDriver(headless=True)
driver.get("https://obfuscator.io/")
obfuscated_codes = []

for code in codes:
    if len(code) > 5000:
        continue

    driver.find_element(By.CSS_SELECTOR, "#root > form
        > div > div:nth-child(1) > div > div:nth-child
```

```
            (3) > div").click()
        time.sleep(0.1)
        driver.find_element(By.CSS_SELECTOR, "#root > form
            > div > div:nth-child(1) > div > div:nth-child
            (3) > div > div.visible.menu.transition > div:
            nth-child(4)").click()

        driver.find_element(By.CSS_SELECTOR, ".CodeMirror-
            scroll").click()
        driver.find_element(By.CSS_SELECTOR, ".bottom").
            click()
        driver.find_element(By.CSS_SELECTOR, "textarea").
            send_keys(Keys.CONTROL + "a")
        driver.find_element(By.CSS_SELECTOR, "textarea").
            send_keys(Keys.DELETE)

        for i in range(0, len(code), 500):
            chunk = code[i:i+500]
            if len(chunk) == 0:
                break
            driver.find_element(By.CSS_SELECTOR, "textarea")
                .send_keys(chunk)

        driver.find_element(By.CSS_SELECTOR, ".primary").
            click()
        time.sleep(0.7)
        obfuscated_codes.append(driver.find_element(By.
            CSS_SELECTOR, ".field > textarea").text)
        driver.find_element(By.CSS_SELECTOR, ".stackable >
            .item:nth-child(1)").click()

    driver.quit()
    return obfuscated_codes
```

Funkcija stvara instancu *Selenium WebDrivera* u *headless* načinu rada te se povezuje na web lokaciju *obfuscator.io*.

Zatim se za svaki kod u listi briše postojeći kod u tekstnom polju za unos neobfusciranog koda. Kod se dio po dio unosi u tekstno polje kako bi se izbjegli problemi s prekoračenjem veličine spremnika. Kodovi dulji od 5000 znakova se izostavljaju zbog vremenske uštede. Nakon unosa cijelog koda, klikom na gumb za obfuskaciju obfuscira se kod te se rezultat pojavljuje u tekstnom polju za obfuscirani kod. Taj proces u prosjeku traje manje od polovice sekunde, zbog čega je potrebno čekati prije čitanja koda iz odgovarajućeg tekstnog polja. Nakon dohvaćanja obfusciranog koda, on se dodaje u listu `obfuscated_codes`.

Svi se elementi dohvaćaju pomoću CSS selektora dobivenih koristeći *Selenium IDE*.

Funkcija vraća listu obfusciranih kodova te se zatvara web preglednik. Ostale funkcije za obfuskaciju imaju vrlo sličnu strukturu te se razlikuju samo u detaljima implementacije.

4.5. Pohrana obfusciranih kodova

Nakon dohvaćanja obfusciranih kodova, oni se pohranjuju u obliku datoteka s ekstenzijom `.js` u direktorij `obfuscated_codes`. Naziv datoteke odgovara URL-u s kojeg je dohvaćen kod na koji je nadodan redni broj koda s tog URL-a. Ta se funkcionalnost postiže funkcijom `createJSFiles(baseName, folderName, codes)` koja prima osnovni naziv datoteke, u ovom slučaju URL, naziv direktorija te listu kodova.

```
def createJSFiles(baseName:str, folderName:str, codes)
    :
    os.makedirs(folderName, exist_ok=True)
    for i in range(len(codes)):
        code = codes[i]
        name = folderName + "/" + baseName + str(i) + ".
            js"
        with open(name, 'w') as file:
            file.write(code)
```

Funkcija stvara direktorij naziva *folderName* ukoliko on ne postoji te za svaki kod u listi stvara datoteku s odgovarajućim nazivom, sadržajem i rednim brojem. Ova se funkcija također može koristiti i za pohranu odgovarajućih neobfusciranih kodova uz manje izmjene u glavnoj funkciji.

4.6. Glavna funkcija

```
def main():
    URLs = readURLs()
    for url in URLs:
        codes = fetchJSFromURL(url)
        obfuscated_codes = []
        for site in Website:
            obfuscated_codes.append(obfuscate(codes, site
                ))
        createJSFiles(url_to_filename(url), "
            obfuscated_codes", obfuscated_codes)
```

Glavna funkcija ima jednostavnu funkcionalnost čitanja URL-ova iz datoteke te kreiranje obfusciranih datoteka za svaki URL i svaki obfuskator. Za pretvorbu URL-a u valjan naziv datoteke koristi se funkcija `url_to_filename(url)` koja iz URL-a izdvaja isključivo domenu te znakove koji nisu valjani za naziv datoteke pretvara u donje crte.

4.7. Moguće nadogradnje sustava

Izrađen sustav za generiranje skupa podataka obfusciranog JavaScript koda funkcionira ispravno, no moguće ga je u budućnosti nadograditi na više načina.

4.7.1. Dodavanje novih obfuskatora

Dodavanje novih obfuskatora je poželjno ukoliko se utvrdi da postoje obfuskatori čiji rezultatni kodovi nisu slični generiranim kodovima nijednog od podržanih obfuskatora, što bi značilo da naš skup podataka ne predstavlja dobro obfuscirane kodove koji se mogu naći na internetu.

U tom bi se slučaju morala dodati nova funkcija za obfuskaciju te implementirati dohvaćanje obfusciranog koda, bilo to koristeći *Selenium WebDriver* ili neki od alternativnih načina.

4.7.2. Kontejnerizacija sustava

Sustav je moguće kontejnerizirati koristeći *Docker* kako bi se omogućilo jednostavnije pokretanje sustava neovisno o platformi. *Kontejnerizacija* je postupak kojim se apli-

kacija ili servis pakira zajedno sa svim potrebnim ovisnostima u izolirano okruženje koje se može pokrenuti na bilo kojem računalu [1]. *Docker* je jedan od najpopularnijih alata za kontejnerizaciju koji omogućuje kreiranje, pokretanje i dijeljenje kontejnera [1].

Kontejnerizacija bi bila korisna jer bi tako sustav postao lako prenosiv i pokretljiv na svim sustavima.

4.7.3. Uporaba baze podataka

Ukoliko bi se sustav koristio za generiranje velikog skupa podataka, bilo bi poželjno koristiti bazu podataka za pohranu obfusciranih kodova umjesto jednostavnog datotečnog pristupa. Također, ako bi se neobfuscirani kodovi nalazili u nekoj bazi podataka, ne bi se trebao koristiti *Selenium* za dohvaćanje kodova.

4.8. Rezultat

Rezultat je skripta koja generira proizvoljnu količinu obfusciranih JavaScript datoteka koristeći tri različita obfuskatora te kodove s proizvoljnih web stranica. Skripta se jednostavno pokreće i generira proizvoljno velik skup podataka koji se može koristiti za daljnja istraživanja.

Selenium se pokazao kao solidan pristup ovom problemu. Budući da su gotovo svi JavaScript obfuskatori dostupni kao grafičko web sučelje, moguće je za gotovo svaki obfuskator napisati jednostavnu skriptu koja će obfuscirati kodove i vratiti obfuscirani kod.

Glavni nedostaci ovog pristupa su brzina izvođenja te ovisnost o speficičnostima web sučelja obfuskatora. Potonje je teško izbjeći ukoliko obfuskator nema dostupno svoje programsko sučelje, ali se brzina izvođenja može poboljšati koristeći alternativne metode dohvaćanja obfusciranog koda navedene u odlomku 3.2.6. Problem također predstavlja ovisnost o specifičnom web pregledniku odnosno njegovom *WebDriveru*. Ako se verzija *drivera* ne poklapa s verzijom odgovarajućeg web preglednika instaliranog na lokalnom računalu, mogu se pojaviti problemi pri pokretanju skripte, zbog čega bi bilo poželjno za buduće primjene kontejnizirati sustav.

5. Zaključak

Obfuskacija je postupak kojim se programski kod mijenja kako bi se učinio manje razumljivim i čitljivim, a da pritom zadržava početnu funkcionalnost. Postupak obfuskacije se može koristiti u zloćudne svrhe, kako bi se prikrio maliciozni kod.

Kako bi se u budućnosti mogao razviti sustav koji bi detektirao obfuscirane JavaScript kodove na internetu koristeći modele strojnog učenja, napravljen je sustav za generiranje skupa podataka obfusciranih JavaScript kodova u obliku Python skripte.

Skripta koristi *Selenium*, alat za automatizaciju web preglednika s kojim se mogu automatski otvarati i koristiti web stranice te dohvaćati njihov sadržaj. Razlog korištenja tog alata je nedostupnost programskih sučelja za veliki dio javno dostupnih obfuskatora na internetu. Skripta uzima neobfuscirane JavaScript kodove s različitih zadanih web stranica te ih obfuscira koristeći tri različita javno dostupna obfuskatora. Obfuscirani kodovi se zatim spremaju kao datoteke.

Selenium se pokazao kao solidan pristup ovom problemu, jer rješava problem dohvaćanja neobfusciranih kodova s web stranica, kao i obfusciranja istih koristeći grafička web sučelja raznih obfuskatora. Iz tog je razloga sustav jednostavan za uporabu i lako proširiv, uz cijenu nešto sporijeg izvođenja zbog pozadinskog korištenja web preglednika.

LITERATURA

[1] Docker. What is a container?, 2023. URL https://www.docker.com/resources/what-container. 2023-05-25.

[2] J. M. Fernández. Javascript antidebugging tricks, 2021. URL https://x-c3ll.github.io/posts/javascript-antidebugging/. 2023-05-22.

[3] Imperva. Minification, 2023. URL https://www.imperva.com/learn/performance/minification. 2023-05-21.

[4] Timofey Kachalov. Javascript obfuscator, 2023. URL https://github.com/javascript-obfuscator/javascript-obfuscator. 2023-05-21.

[5] Timofey Kachalov. Obfuscator.io, 2023. URL https://obfuscator.io/. 2023-05-21.

[6] M. Kathiravan, V. Rajasekar, Shaik Javed Parvez, V. Sathya Durga, M. Meenakshi, i S. Gowsalya. Detecting phishing websites using machine learning algorithm. U *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*, stranice 270–275, 2023. doi: 10.1109/ICCMC56507.2023.10083999.

[7] LambdaTest. Web automation, 2023. URL https://www.lambdatest.com/learning-hub/web-automation. 2023-05-22.

[8] Ben Lutkevich. What is obfuscation, 2021. URL https://www.techtarget.com/searchsecurity/definition/obfuscation. 2023-05-20.

[9] Rima Masri i Monther Aldwairi. Automated malicious advertisement detection using virustotal, urlvoid, and trendmicro. U *2017 8th International Conference*

on Information and Communication Systems (ICICS), stranice 336–341, 2017. doi: 10.1109/IACS.2017.7921994.

[10] Marvin Moog, Markus Demmel, Michael Backes, i Aurore Fass. Statically detecting javascript obfuscation and minification techniques in the wild. U *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, stranice 569–580, 2021. doi: 10.1109/DSN48987.2021.00065.

[11] Amaya Paucek. Most dangerous websites you should avoid, 2022. URL `https://secureblitz.com/most-dangerous-websites/`. 2023-05-20.

[12] Selenium. The selenium browser automation project, 2023. URL `https://www.selenium.dev/documentation`. 2023-05-20.

[13] Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, i Takeshi Yada. Detecting malicious websites by integrating malicious, benign, and compromised redirection subgraph similarities. U *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, svezak 1, stranice 655–664, 2017. doi: 10.1109/COMPSAC.2017.105.

[14] A K Singh i Navneet Goyal. A comparison of machine learning attributes for detecting malicious websites. U *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*, stranice 352–358, 2019. doi: 10.1109/COMSNETS.2019.8711133.

Sustav za generiranje skupa podataka obfusciranog JavaScript koda

Sažetak

Obfuskacija je postupak kojim se programski kod mijenja kako bi se učinio manje razumljivim i čitljivim, a da pritom zadržava početnu funkcionalnost. Zbog svoje prirode, obfuskacija se može koristiti u zloćudne svrhe, kako bi se prikrio maliciozni kod. Zbog potrebe za skupom podataka obfusciranih JavaScript kodova na kojem bi se učili modeli strojnog učenja, napravljen je sustav za generiranje takvog skupa podataka u obliku Python skripte.

Zbog nedostupnosti programskog sučelja većine alata za obfuskaciju JavaScripta, za obfuskaciju kodova koristi se Selenium, alat za automatizaciju web preglednika. Skripta koristi Selenium kako bi dohvatila neobfuscirane JavaScript kodove sa zadanih domena, te kako bi ih obfuscirala koristeći različite javno dostupne obfuskatore tako što programski upravlja njihovim grafičkim web sučeljima.

Pokretanjem skripte stvara se proizvoljan broj obfusciranih JavaScript datoteka koje se mogu koristiti za daljnja istraživanja.

Ključne riječi: kibernetička sigurnost, obfuskacija, JavaScript, Selenium, automatizacija web preglednika

System for Generating Obfuscated JavaScript Code Dataset

Abstract

Obfuscation is a transformation of program code in order to make it less understandable and readable, while retaining the initial functionality. Because of its nature, obfuscation can be used for malicious purposes, in order to hide malicious code. Because of the need for a dataset of obfuscated JavaScript codes on which machine learning models would be trained, a system for generating such a dataset in the form of a Python script was made.

Because most tools for JavaScript obfuscation do not have a programming interface, Selenium, a tool for automating web browsers, was used to access and use those tools. The script uses Selenium to retrieve unobfuscated JavaScript codes from the specified domains, and to obfuscate them using different publicly available obfuscators. This is done by programmatically controlling their graphical web interfaces.

Running the script creates an arbitrary number of obfuscated JavaScript files that can be used for further research.

Keywords: cyber security, obfuscation, JavaScript, Selenium, web browser automation