SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 707

SUSTAV ZA PRIKUPLJANJE PODATAKA O PONAŠANJU WEB PREGLEDNIKA I NJIHOVU ANALIZU STROJNIM UČENJEM

Filip Jukić

Zagreb, lipanj 2014.

Sadržaj

1. Uvod

2014. godine na svijetu se nalazi preko 2.9 milijarde korisnika Interneta [1]. No, s dobrim stranama takvog naglog rasta broja korisnika dolaze i one loše. Sve je veći broj napada na Internetu usmjeren prema krajnjim korisnicima od kojih napadači pokušavaju ukrasti privatne i povjerljive podatke, a u nekim slučajevima ostvariti i materijalnu korist na štetu žrtve. Veliki broj korisnika posjeduje nisku razinu informatičke pismenosti što ih čini idealnim ciljem za napadače koji raznim napadima pokušavaju pokrenuti neovlaštene aplikacije na računalima korisnika i na taj način ostvariti nekakvu korist.

Da bi bilo moguće spriječiti takve potencijalne napade, potrebno je biti u mogućnosti detektirati ih na vrijeme. Prikupljanjem i analizom velike količine podataka o ponašanju kako normalnih, tako i zlođudnih stranica, moguće je uočiti neke karakteristike specifične za zlođudne stranice i tako u budućnosti automatski detektirati zlođudne stranice i napade na korisnika. U sklopu ovog diplomskog rada planira se razviti sustav koji će pratiti različite parametre unutar web preglednika Google Chrome tijekom njegova rada te omogućiti da se ti podaci prikupljaju na poslužitelju i obrađuju alatima strojnog učenja.

U poglavlju 2 navedeni su najčešći napadi na Internetu, mehanizmi njihovog funkcioniranja kao i načini za potencijalno sprječavanje takvih napada. U poglavlju 3 opisana je struktura i način funkcioniranja ekstenzije za preglednik Google Chrome te dostupni API-ji koje ekstenzija može koristiti. U poglavlju 4 opisan je sustav za prikupljanje podataka razvijen u sklopu ovog diplomskog rada, upute za instalaciju i pokretanje te opis korisničkog sučelja. U poglavlju 5 nalazi se zaključak diplomskog rada.

2. Prijetnje na Internetu

Obični korisnici Interneta su u današnje vrijeme izloženi mnogim napadima i pokušajima da zlćudni korisnici od njih ukradu razne privatne informacije ili na neki drugi način ostvare osobnu korist na njihovu štetu. U ovom poglavlju bit će opisane neke od najčešćih prijetnji s kojima se susreću obični korisnici te metode potencijalne detekcije i sprječavanja takvih napada.

2.1. Cross-site scripting

Cross-site scripting (*XSS*) je sigurnosni propust koji se nalazi u web aplikacijama i omogućava zlonamjernim korisnicima ubacivanje vlastitih skripti na web stranice kojima pristupaju drugi korisnici. *Open Web Application Security Project* (*OWASP*) je 2013. godine stavio XSS napad na 3. mjesto svoje godišnje top liste najopasnijih i najčešćih propusta u web aplikacijama [7].

Da bi napad bilo moguće izvesti, web aplikacija mora biti ranjiva na taj tip napada na način da ne provjerava i osigurava ulazne podatke koji dolaze od korisnika koji se zatim koriste i prikazuju na nekom dijelu stranice ostalim korisnicima. Kako je moguće u stranicu ubaciti bilo kakav kod, tako je moguće učitati vlastitu zlćudnu stranicu preko stranice koju korisnik posjećuje bez da on to primjeti i na taj način skupljati privatne podatke od korisnika.

Postoje dvije glavne vrste XSS napada:

1. Privremeni (engl. *non-persistent/reflected*)
2. Trajni (engl. *persistent/stored*)

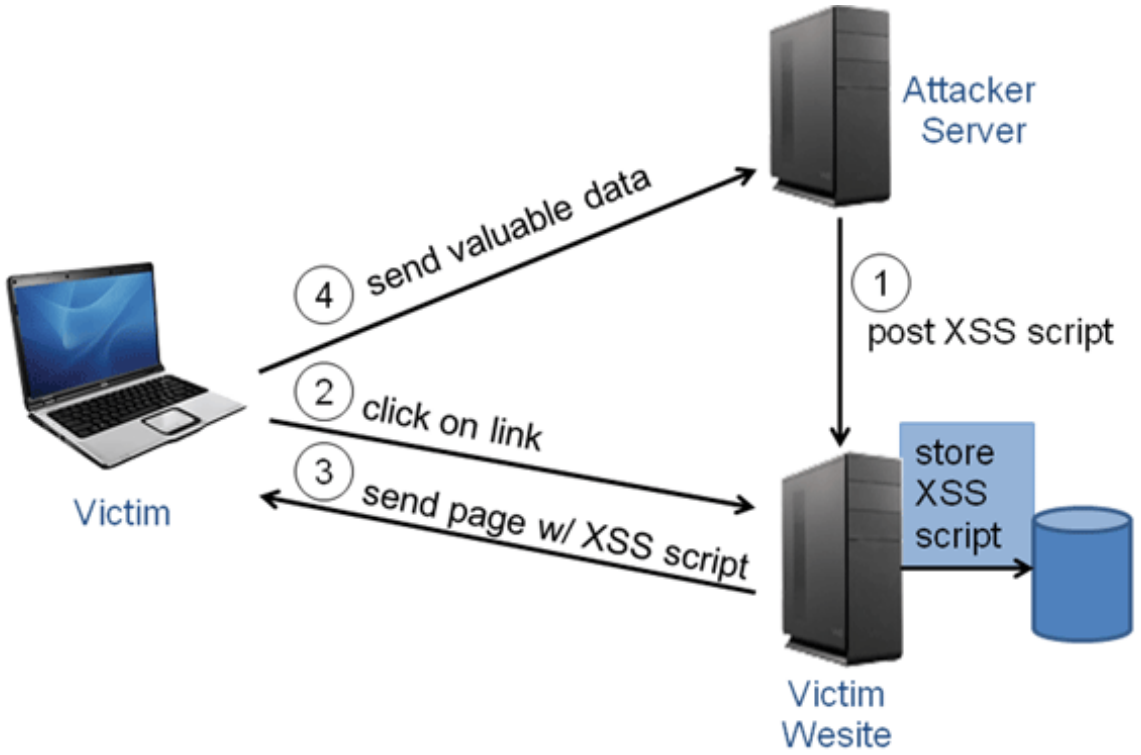
U privremenim napadima, koji su češći oblik XSS napada, najčešće se podaci koje korisnik šalje na poslužitelj, npr. u obliku HTTP GET parametara, prikazuju odmah na stranici koja se servira korisniku. Poveznica (engl. *link*) u čijem je URL-u ubačen zlćudni kod putem GET HTTP parametara šalje se potencijalnim žrtvama preko e-maila ili nekog drugog oblika komunikacije. Kada žrtva klikne na takvu poveznicu, otvara se stranica na kojoj se izvršava zlćudni kod koji je napadač ubacio. Kada se stranica otvori, korisnicima je takav napad teško otkriti jer, kada pogledaju URL u web pregledniku, čini im se da se nalaze na sigurnoj stranici i ne sumnjaju na napad.

Trajni napadi slično se izvršavaju, samo se podaci koje napadač šalje na poslužitelj. npr. na nekoj formi koja šalje POST zahtjev, trajno spremaju na poslužitelju. Zlćudni kod se tako izvršava prilikom svakog prikaza stranice koja pristupa spremljenim korisnikovim podacima (Slika 1):

1. napadač putem HTML forme šalje zlćudni kod na ranjivu stranicu gdje se trajno sprema u bazu podataka
2. žrtva otvara web stranicu sa zlćudnim kodom
3. sadržaj stranice se šalje žrtvi, uključujući i zlćudni kod
4. zlćudni kod se izvršava u web pregledniku i žrtvine privatne podatke šalje na poslužitelj koji kontrolira napadač

Primjer toga su Internet forumi i razne forme za ostavljanje komentara na web stranicama. Trajni napadi opasniji su od privremenih jer napadač ne mora individualno slati poveznicu svakoj potencijalnoj žrtvi nego se zlćudni kod izvršava čim netko pristupi opasnoj web

stranici.



Slika 1: Dijagram XSS napada [8]

XSS napadi se primarno sprječavaju na poslužiteljskoj strani gdje se ispravnom obradom podataka koje korisnik šalje na poslužitelj neutralizira prijetnja te se zlćudni kod ne izvršava prilikom prikaza web stranice. Ipak, dodatna pomoć u sprječavanju ovakvih napada se može izvesti u ekstenziji, pažljivim praćenjem web zahtjeva koje preglednik šalje na razne poslužitelje te se tako mogu detektirati «sumnjivi» zahtjevi koji se izvršavaju kao posljedica XSS napada i zaustaviti njihovo slanje.

2.2. Cross-site Request Forgery

Za razliku od prethodno opisanog XSS napada u kojemu se iskorištava povjerenje koje korisnik ima u neku stranicu, *Cross-site Request Forgery (CSRF)* napad iskorištava povjerenje koje neka stranica ima u korisnikove akcije.

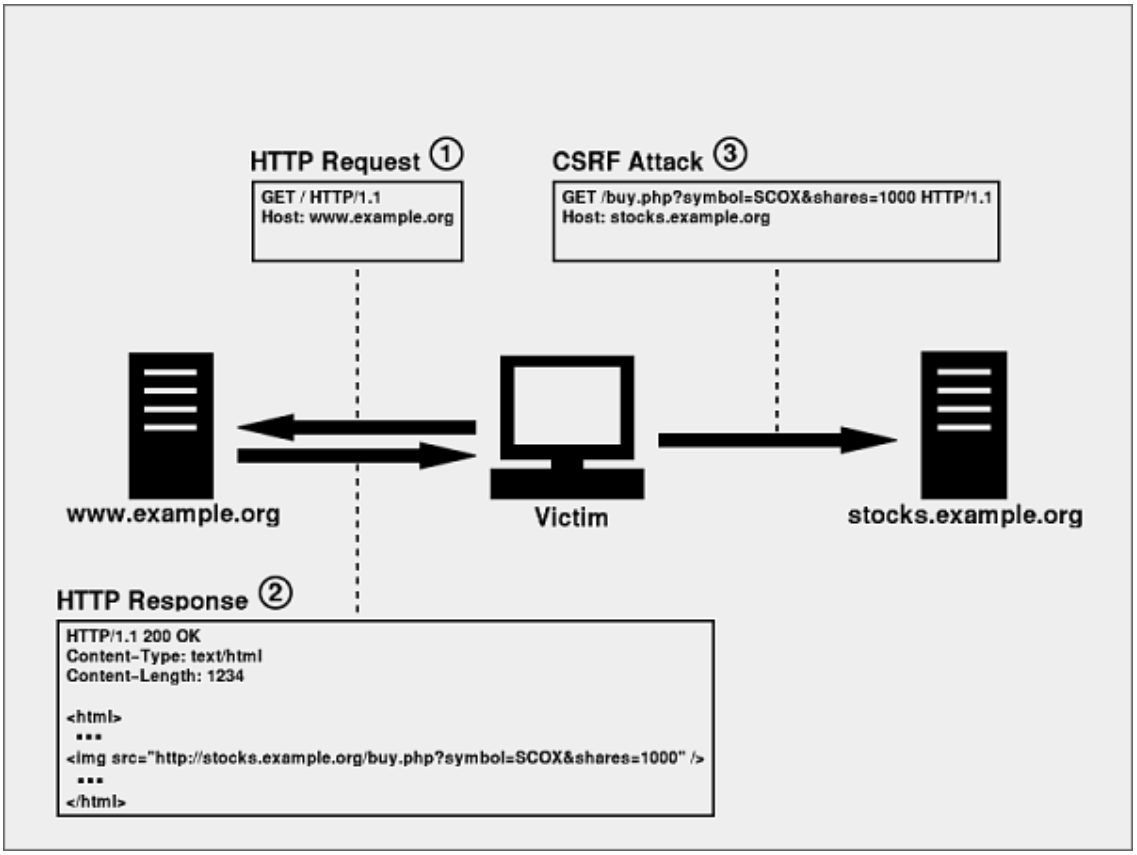
Napad se izvodi tako da se na neku web stranicu postavi element, vrlo često slika, koji šalje zahtjev na neku web stanicu na kojoj je korisnik koji otvara sliku prijavljen (engl. *logged in*). Ako web stranica na kojoj je korisnik prijavljen ne provjerava odakle dolazi web zahtjev i napadač zna točan URL pomoću kojeg je moguće izvršiti neke radnje na toj stranici, otvaranjem ranjive web stranice i slanjem zlćudnog zahtjeva na tu stranicu moguće je na njoj izvršiti radnje koje štete korisniku.

Klasičan primjer mete ovakvog napada je bankarski sustav ranjiv na CSRF napad (Slika 2). Napadač na neku web stranicu (npr. Internet forum) stavlja sliku koja dohvaća URL `http://bank.com/money_transfer?to_account=12345` ranjive stranice. Žrtva zatim posjećuje stranicu gdje je ostavljen zlćudni kod i:

1. šalje HTTP zahtjev koji dohvaća kod stranice
2. dobija HTTP odgovor u kojem se nalazi i zlćudni URL

3. web preglednik automatski šalje HTTP zahtjev na zlćudni URL i, ako je na toj ranjivoj stranici prijavljen, izvršava se radnja naznačena u URL-u

Napadač to može iskoristiti da bi izvršio slanje novca s korisnikovog računa na svoj račun. Bitno je napomenuti da za slanje ovakvog zlćudnog zahtjeva nije potrebno iskoristiti XSS napad jer je moguće na raznim forumima postavljati slike koje dohvaćaju sadržaj sa zadanog URL-a.



Slika 2: Dijagram CSRF napada [9]

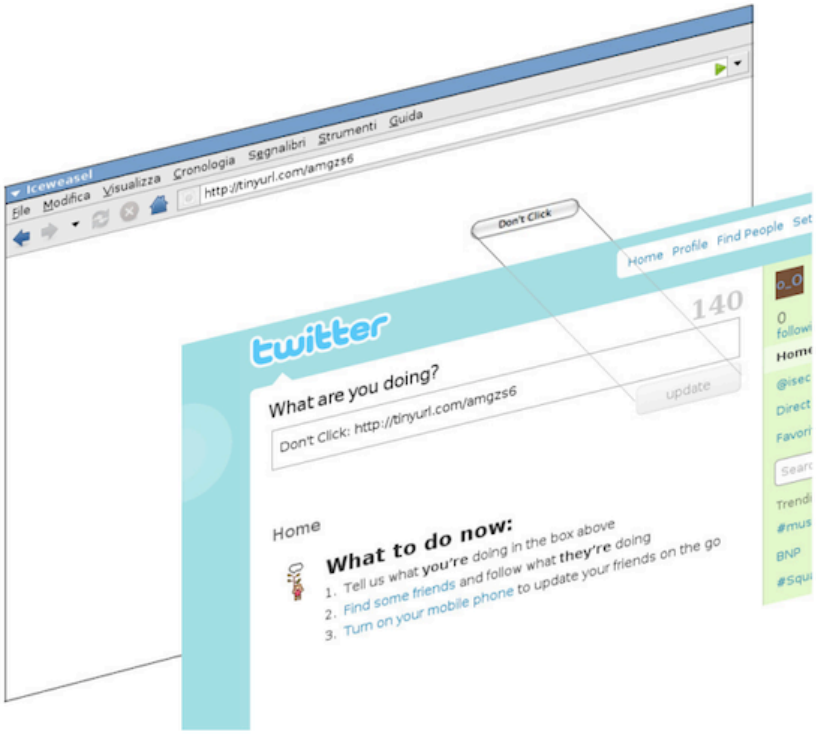
Ovakvi napadi najčešće se rješavaju na poslužiteljskoj strani provjerom odakle dolaze zahtjevi, onemogućavanjem izvršavanja raznih akcija na poslužitelju putem GET HTTP zahtjeva te zahtijevanjem tajnog jednokratnog koda (engl. *token*) vezanog na korisnika koji napadač u trenutku slanja napada ne zna. Ipak, moguće je dodatno onemogućiti ovakve napade u ekstenziji provjerom URL-a na koji se šalju zahtjevi te tako spriječiti slanje zahtjeva na domene koje u tom trenutku djeluju sumnjivo.

2.3. Otimanje klikova

Otimanje klikova (engl. *Clickjacking*) je napad u kojem se korisnika koji pristupa nekoj web stranici pokušava zavarati i natjerati ga da klikne na dio web stranice koji zapravo nije ono na što korisnik misli da klikće. Tako napadač potencijalno dolazi do povjerljivih korisnikovih podataka.

Napad se izvodi tako da se na web stranicu, pomoću ubačenog iframe HTML elementa ili nekog koda, učitava zlćudna web stranica te se ta stranica postavlja kao prozirna (skrivena) stranica iznad postojeće (Slika 3). Korisnik u svom radu na web stranici misli da vrši interakciju sa stranicom koju je posjetio, međutim u stvarnosti klikće po zlćudnoj

stranici te tako izvršava zloćudne radnje koje je napadač omogućio.



Slika 3: Clickjacking napad [25]

Napad je moguće detektirati i spriječiti u ekstenziji, tako da se provjerava postojanje iframe elemenata u DOM stablu, te izvršiti detekciju skrivenih elemenata na stranici, izmjeriti njihovu veličinu te eventualno preklapanje s postojećim elementima na stranici. Također, u slučaju da korisnik pokrene zloćudnu akciju, potencijalno je moguće spriječiti npr. slanje web zahtjeva ili izvršavanje neke skripte.

2.4. Napadi na privatnost

2.4.1. Kolačići za praćenje

U posljednjih desetak godina, s naglim razvojem Interneta, nastankom i popularizacijom mnogih društvenih mreža, u medijima se sve više spominje pitanje privatnosti na Internetu. Privatnost na Internetu može se definirati kao pravo pojedinca na privatnost i sigurnost osobnih informacija koje se prenose putem Interneta [26]. Kao što se može vidjeti na primjeru oglašivačkih kompanija koje prate korisnikovo korištenje Interneta, kao i na primjeru nedavnog skandala prisluškivanja ljudi na Internetu od strane američke Agencije za nacionalnu sigurnost (engl. *National Security Agency* - *NSA*), privatnost na Internetu je ozbiljno ugrožena.
Jedan od najčešćih načina na koji kompanije prate korisnikovo ponašanje na Internetu i stranice koje posjećuje je putem kolačića za praćenje (engl. *tracking cookie*). Kolačići su mali skup podataka koji neka stranica pohranjuje u web preglednik korisnika prilikom posjeta toj stranici te se šalje natrag na poslužitelj prilikom svakog sljedećeg učitavanja web stranice. Time se omogućuje spremanje informacija o korisniku u kolačić i njegovo praćenje tijekom daljnjeg korištenja web stranice.

Još veći udar na privatnost su kolačići koje postavljaju dodatne stranice kojima se pristupa prilikom otvaranja stranice (engl. *third-party cookies*). Iako kolačiće primarno postavlja i

prima web stranica koju korisnik posjećuje, ta stranica može sadržavati razne elemente poput slika ili skripti koje se učitavaju s neke druge web stranice, tj. domene, pa se i s te domene mogu postaviti kolačići u korisnikov web preglednik. Tako u teoriji postoji mogućnost da korisnik posjeti neku web stranicu i pritom dobije kolačiće s desetak različitih domena. Tim pristupom najčešće se služe oglašavačke i analitičke kompanije na Internetu koje postavljaju svoje elemente (poput oglasa) na brojne web stranice, postavljaju i primaju kolačiće te tako imaju mogućnost pratiti koje stranice korisnik posjećuje i koristiti to za ostvarivanje vlastite dobiti. Srećom, u zadnjih nekoliko godina ostvareni su pozitivni pomaci u smjeru ograničavanja korištenja takvih kolačića i pravnim putem poput direktive Europske Unije o e-privatnosti [10] i tehnološkim rješenjima poput predloženog HTTP zaglavlja (engl. *header*) *Do Not Track* (*DNT*) koji od web stranice zahtijeva prestanak bilo kakvog praćenja korisnika [11].

U ekstenziji koja se izrađuje u sklopu ovog diplomskog rada, pomoću API-ja koje Google Chrome nudi, moguće je reagirati prilikom postavljanja svakog kolačića u korisnikov web preglednik, pratiti tu aktivnost i eventualno na temelju liste zlođudnih domena zabraniti postavljanje takvih kolačića.

2.4.2. Prisluškivanje mrežnog prometa

Iznimno je bitno spomenuti još jednu vrstu napada na privatnost korisnika, a to je prisluškivanje njegovog Internet prometa. Pomoću takvog napada napadač može pratiti svaku aktivnost korisnika na Internetu, doći do njegovih privatnih podataka i lozinki te mu tako nanijeti veliku štetu.

Internetski promet moguće je prisluškivati kada se privatne i osjetljive stvari prenose putem protokola HTTP koji ne štiti promet korištenjem kriptografskih metoda. Ako napadač uhvati bilo koji paket mrežnog prometa prilikom prijave na neku web stranicu, jasno će mu biti vidljivo korisničko ime i lozinka. Također, prisluškivanjem prometa napadač može prikupiti druge privatne stvari u običnom tekstualnom formatu poput liste web stranica koje korisnik posjećuje. Da bi se ovakav napad izbjegao, web stranica koju korisnik posjećuje mora imati mogućnost korištenja šifriranog HTTPS protokola koji onemogućuje napadaču prisluškivanje mrežnog prometa. HTTPS sam po sebi nije protokol, već slojevito korištenje HTTP protokola preko SSL/TLS protokola stoga sigurnost komunikacije ovisi o sigurnosti SSL/TLS sloja. Iako su se u posljednjih par godina pojavili učestali napadi na SSL/TLS protokol (*BEAST [12]* i *BREACH* [13] napadi, *Heartbleed bug [14]*), on se uz pravilnu konfiguraciju još uvijek smatra sigurnim te ga je iz tog razloga preporučljivo uvijek koristiti prilikom prenošenja osjetljivih informacija putem Interneta. Iz tog razloga, i ekstenzija će koristiti šifrirani HTTP promet prema poslužiteljskoj strani koja prikuplja podatke da bi se izbjegla mogućnost da napadač presretne podatke koje ekstenzija šalje i tako ugrozi privatnost korisnika.

3. Google Chrome ekstenzije

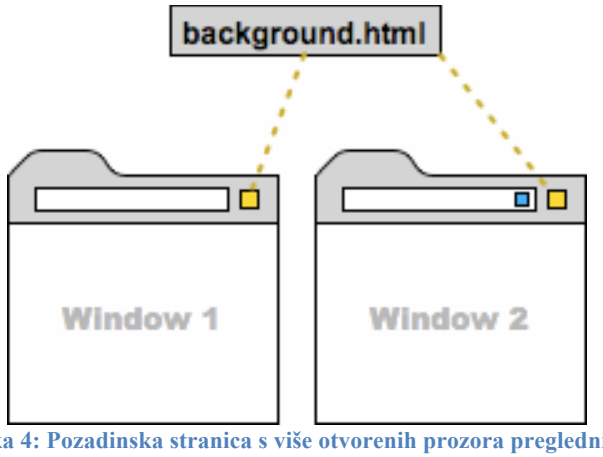
3.1. Općenito

Ekstenzija za preglednik Google Chrome u suštini je skup datoteka (HTML, CSS, Javascript, slike) zapakiran u arhivu koje omogućuju dodatnu funkcionalnost unutar web preglednika. Ekstenzije se ponašaju kao web stranice te kao takve imaju pristup svim API-jima kojima i obične web stranice imaju pristup unutar preglednika. Radi sigurnosti, svaka ekstenzija je izolirana od drugih ekstenzija i web stranica kojima preglednik pristupa te se skripte unutar njih izvršavaju u zasebnom kontekstu. Ipak, postoje mehanizmi koji na siguran način izmjenom poruka omogućuju međusobnu komunikaciju između dvije ekstenzije ili ekstenzije i stranice kojoj preglednik pristupa. Najčešće se sastoje od tri glavna dijela koji će biti objašnjeni u nastavku:

1. Pozadinska stranica (engl. *background page*)
2. Stranica korisničkog sučelja (engl. *UI page*)
3. Sadržajna skripta (engl. *content script*)

3.1.1. Pozadinska stranica

Pozadinska stranica je nevidljiva stranica koja sadrži glavnu programsku logiku ekstenzije te je centralno mjesto iz kojeg se upravlja i komunicira s ostatkom ekstenzije, korisničkim sučeljem i stranicama kojima korisnik pristupa. Budući da je pozadinska stranica obična HTML stranica, većina programske logike se nalazi unutar skripte koju poziva pozadinska stranica, što uključuje i pristup svim API-jima koje Google Chrome nudi. DOM stablu korisničkog sučelja ekstenzija ima potpun pristup, kao što je slučaj i s drugim stranicama unutar ekstenzije. Sve stranice unutar ekstenzije povezane su s jednom instancom pozadinske stranice (Slika 4).



Slika 4: Pozadinska stranica s više otvorenih prozora preglednika

No, ipak pozadinska stranica ne može čitati ili mijenjati DOM stablo web stranica koje korisnik posjećuje bez korištenja sadržajnih skripti (engl. *content script*) s kojima pozadinska stranica komunicira porukama.

3.1.2. Stranica korisničkog sučelja

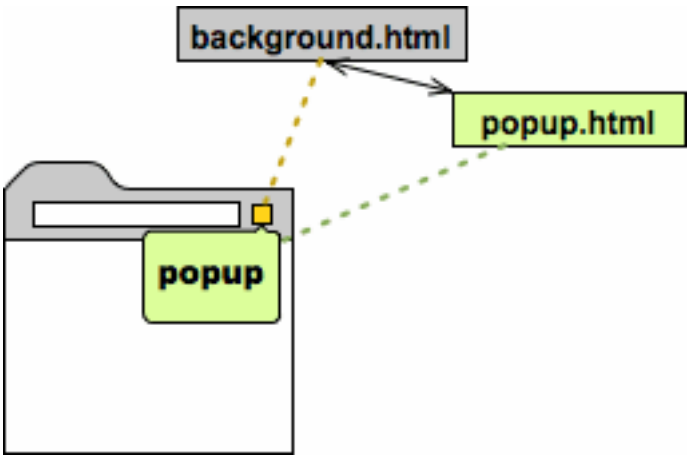
Ukoliko unutar ekstenzije želimo ekran s opcijama za prilagođavanje rada ekstenzije ili samo za praćenje posla koji se obavlja, ekstenzija može, ali i ne mora, sadržavati dodatne

HTML stranice koje predstavljaju njeno korisničko sučelje. Postoje dva glavna načina na koji ekstenzije dodaju korisničko sučelje u preglednik:

1. Akcija preglednika (engl. *Browser action*)
2. Akcija stranice (engl. *Page action*)

Akcija preglednika se koristi u slučaju kada se ekstenzija koristi na većini ili na svim web stranicama koje korisnik posjećuje te se ikona ekstenzije postavlja u traku s alatima. Akcija stranice se koristi kada želimo da se korisničko sučelje pojavljuje samo na određenim web stranicama (npr. na svim stranicama s domenom *.google.com). Ikona se tad pojavljuje unutar trake s adresom (engl. *address bar*).

Najčešće se klikom na ikonu ekstenzije otvara prozor s dodatnim sadržajem (engl. *pop-up*). Sve stranice unutar ekstenzije se izvršavaju unutar istog procesa te tako prozor s dodatnim sadržajem može pozivati funkcije i pristupati DOM stablu pozadinske stranice, čime se izbjegava nepotrebno dupliciranje koda i postiže kvalitetna separacija prezentacijskog i upravljačkog dijela ekstenzije (Slika 5).

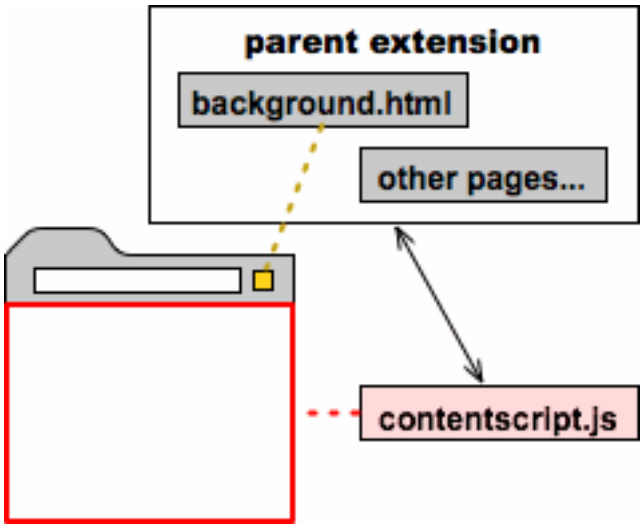


Slika 5: Povezanost korisničkog sučelja i pozadinske stranice

3.1.3. Sadržajna skripta

Ako je potrebno da ekstenzija čita ili mijenja web stranice koje korisnik posjećuje, potrebno je koristiti sadržajnu skriptu. Sadržajna skripta je skripta pisana u programskom jeziku Javascript koja se izvršava u kontekstu stranice koja je otvorena u pregledniku. S te strane, sadržajna skripta se ponaša više kao skripta koja je dio učitane stranice, a manje kao dio same ekstenzije.

Sadržajna skripta otvara velik broj mogućnosti poput čitanja i mijenjanja DOM stabla učitane stranice te omogućava reagiranje na događaje (engl. *event*) unutar stranice. Za razliku od ostalih datoteka u ekstenziji, sadržajna skripta ne može pristupati DOM stablu pozadinske stranice niti Chrome Extension API-jima. Ipak, sadržajna skripta nije u potpunosti odsječena od ostatka ekstenzije jer ima mogućnost slanja i primanja poruka ostalim stranicama u ekstenziji koristeći *chrome.runtime.sendMessage* API. Sadržajna skripta tako najčešće prikuplja podatke sa stranica koje korisnik posjećuje i putem poruka šalje te informacije pozadinskoj stranici koja ih obrađuje te ih eventualno pokazuje u korisničkom sučelju ekstenzije (Slika 6).

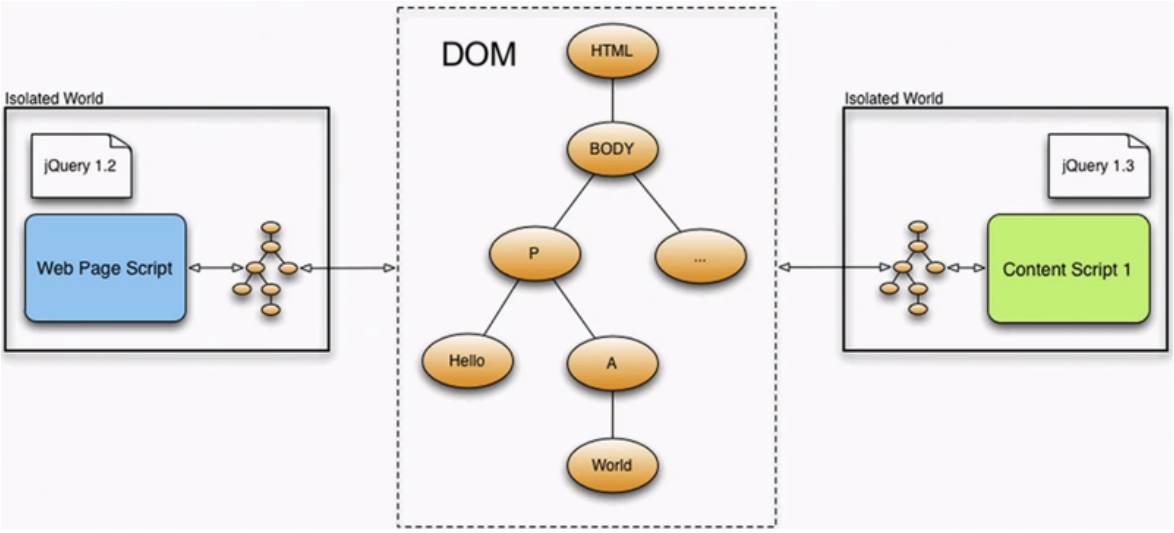


Slika 6: Komunikacija porukama sadržajne skripte

Kao što je ranije napomenuto, sadržajna skripta se izvršava u kontekstu web stranice koju korisnik posjećuje. Glavna razlika u odnosu na ostale skripte koje se nalaze na toj stranici je to da se sadržajna skripta izvršava u posebnoj okolini nazvanoj *izolirani svijet* (engl. *isolated world*). Tako skripta ima pristup DOM stablu stranice gdje se izvršava, ali ne i svim Javascript varijablama i funkcijama na toj stranici. Ukratko - sadržajna skripta ne može pristupati Javascript kodu koji se izvršava na toj stranici, ali ni Javascript kod ne može pristupati sadržajnoj skripti čime se dosta dobiva na sigurnosti ekstenzije (Slika 7). Iako ovo štiti ekstenziju od zlođudnih web stranica, isto tako i ograničava što ekstenzija može raditi. Da bi se to zaobišlo, moguće je postaviti kod koji reagira na događaje (engl. *event listener*) na stranicu koja se izvršava ili ubaciti Javascript kod direktno na stranicu.

3.2. Dostupni API-ji

3.2.1. Upravljanje procesima



Slika 7: Izolirani svijet sadržajne skripte

Pomoću chrome.processes API-ja moguće je pristupati informacijama o procesima koji se izvršavaju unutar Chrome preglednika te se tako za svaki proces može saznati zauzeće

procesora, količina rezervirane memorije, količina mrežnog prometa ili veličina međuspremnika koje skripte koriste.

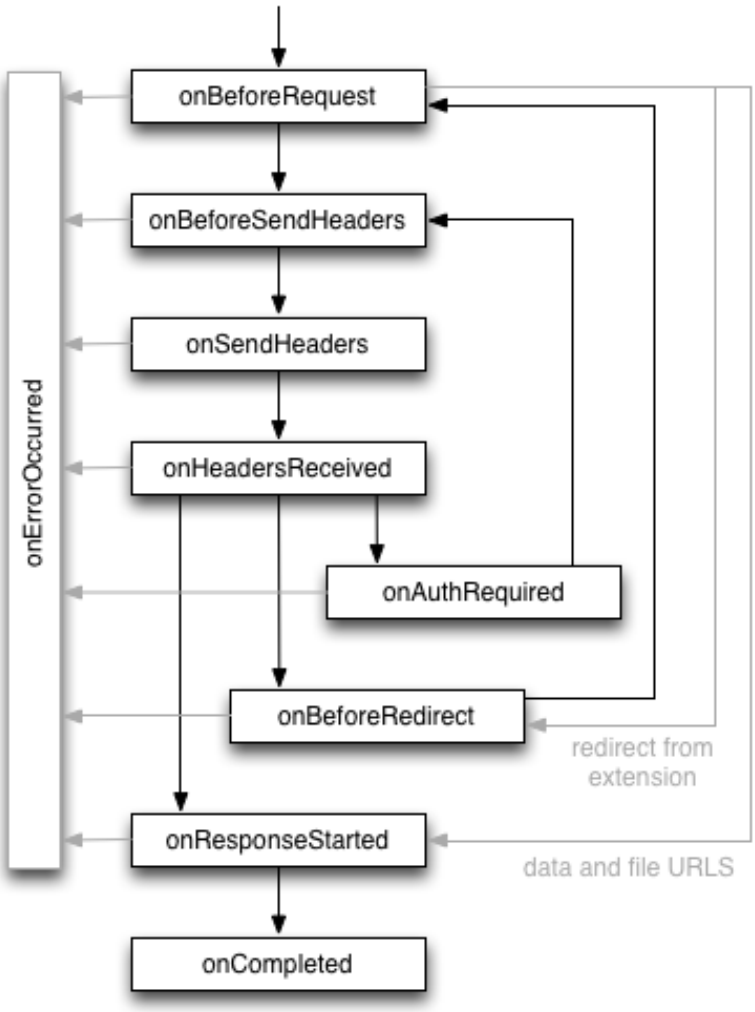
Iako je moguće eksplicitno tražiti podatke o pojedinom procesu pomoću njegove identifikacijske oznake, najčešće će se koristiti osluškivači događaja `onUpdated` ili `onUpdatedWithMemory` koji se pokreću svaki put kada upravljač zadacima (engl. *Task Manager*) unutar preglednika osvježi podatke o svim procesima. Procesu koji osluškuje događaje se tada predaje lista koja sadrži podatke o svim procesima unutar preglednika koji se trenutno izvršavaju.

Ovaj API je koristan u slučajevima kada zlćudna skripta izvršava napad da bi usporila rad preglednika na način da pokušava opteretiti procesor složenim kalkulacijama ili pokušava ispuniti što više memorije unutar procesa. Konstantnim praćenjem tih vrijednosti, takav napad bi se mogao detektirati te bi se po potrebi mogao ugasiti proces unutar kojega se izvršava zlćudni kod.

3.2.2. Upravljanje web zahtjevima

webRequest API omogućava promatranje i analizu mrežnog prometa, kao i mogućnost presretanja, blokiranja ili izmjene web zahtjeva u stvarnom vremenu. API definira skup događaja koji prate životni ciklus web zahtjeva.

Definiranjem koda koji se izvršava u slučaju pojave odgovarajućih događaja, moguće je pristupati i mijenjati zahtjev u bilo kojem trenutku unutar njegovog životnog ciklusa, npr. prije otvaranja TCP konekcije, prije HTTP preusmjeravanja ili nakon što je primljen HTTP odgovor na zahtjev (Slika 8).



Slika 8: Životni ciklus web zahtjeva

Ovo je možda i najvažniji API u izradi ove ekstenzije jer nam omogućuje detaljno praćenje svih web zahtjeva koje šalju stranice otvorene u pregledniku, prikupljanje statističkih podataka o najčešće posjećenim stranicama, ali i mogućnost blokiranja sumnjivih zahtjeva uz pomoć predefinirane liste zloćudnih domena ili algoritama koji prate ponašanje neke web stranice i ocjenjuju koliko je ona potencijalno opasna.

U sljedećem primjeru koda, definira se kod koji reagira na događaj prije samog slanja HTTP zahtjeva na neku web stranicu. Prilikom svakog pokretanja, uspoređuje se URL zahtjeva s listom URL-ova koja je predana funkciji i, ako se URL nalazi u toj listi, pokreće se funkcija koja u ovom slučaju prekida slanje zahtjeva. U ovom primjeru blokiraju se svi zahtjevi koji se šalju na domenu evil.com:

```
chrome.webRequest.onBeforeRequest.addListener(
    function(details) {
        return {cancel: true};
    },
    {urls: ["*://www.evil.com/*"]},
    ["blocking"]
);
```

U funkciji koja se izvršava u slučaju pojave odgovarajućih događaja moguće je definirati složenije uvjete te je tako moguće blokirati zahtjeve ovisno o tome što se nalazi u tijelu samog zahtjeva, je li on poslan s glavne stranice ili pak iz iframe elementa te da li zahtjev pokušava dohvatiti skriptu, sliku ili se radi o AJAX zahtjevu.

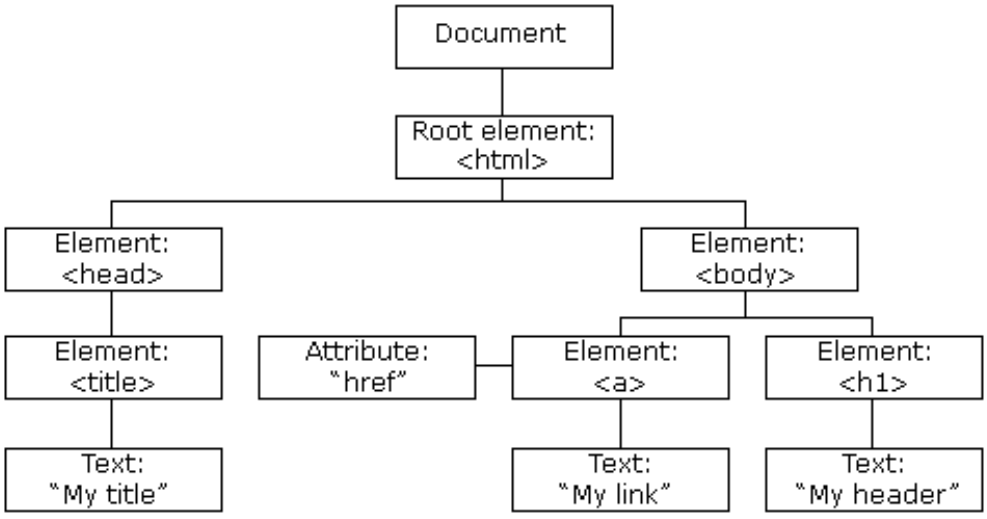
3.2.3. Upravljanje HTTP kolačićima

Jedan od potencijalno korisnih API-ja je i Chromeov API za upravljanje HTTP kolačićima. Taj API omogućuje dohvat, postavljanje i brisanje kolačića, ali ono što je najbitnije je mogućnost reakcije na svaku promjenu kolačića koju neka web stranica pokušava izvesti. To je omogućeno postavljanjem koda koji se izvršava u slučaju pojave događaja `onChanged` koji se aktivira prilikom svakog postavljanja, izmjene ili brisanja kolačića. Kodu koji se izvršava u slučaju pojave odgovarajućih događaja se predaje kolačić i razlog promjene kolačića - je li kolačiću automatski istekao rok ili se radi o eksplicitnom zahtjevu za promjenu kolačića.

Ovaj API moguće je iskoristiti na način da se prilikom svakog postavljanja novog kolačića provjerava za koju se domenu on pokušava postaviti. Ukoliko se ta domena nalazi u listi sumnjivih domena ili pak sadrži neke ključne riječi u domeni, moguće je spriječiti postavljanje takvog kolačića. Takvim pristupom u teoriji je moguće samo donekle spriječiti napade na privatnost korisnika postavljanjem kolačića za praćenje, kao što je opisano u ranijem poglavlju.

3.2.4. Objekti model dokumenta

Objektni model dokumenta (engl. *Document Object Model - DOM*) je strukturirani prikaz HTML ili XML dokumenta u obliku stabla gdje čvorovi predstavljaju HTML elemente iz dokumenta. Korijenski element je html koji se zatim grana na zaglavlje (engl. *head*) i tijelo HTML dokumenta (engl. *body*). Unutar svakog se nalaze drugi HTML elementi te se tako stablo sve više i više grana (Slika 9). Također, DOM pruža i programsko sučelje pomoću kojeg programski jezici (uglavnom Javascript) mogu pristupati tom stablu, pretraživati ga i izmjenjivati.



Slika 9: Grafički prikaz DOM stabla

Putem ovog API-ja, ekstenzija ima pristup sadržaju stranica koje korisnik posjećuje te je

moguće vršiti analizu DOM stabla, npr. pratiti broj pojedinih elemenata na stranici i njihove atribute ili pratiti eventualne izmjene stabla od strane neke druge skripte koja se izvršava. Zbog lakšeg pristupa stablu i njegovog pretraživanja, koristit će se popularna Javascript biblioteka jQuery [15].

U sljedećem primjeru koda pretražuje se HTML dokument da bi se saznalo koliko se iframe elemenata nalazi u njemu te ispis URL-a svakog od njih. Prvo se pomoću jQuery selektora $ dohvaća lista svih iframe elemenata unutar web stranice. Potom se u konzolu web preglednika ispisuje dužina te liste, tj. ukupni broj iframe elemenata. Zatim se pomoću metode `each` izvršava petlja u kojoj se pristupa svakom pojedinom iframe elementu u listi te se u konzolu web preglednika ispisuje njegove src atribute tj. URL.

```
var $elementi = $("iframe")
console.log($elementi.length) // ispis broja iframe
elemenata
$elementi.each(function() {
    console.log($(this).src)) // ispis src atributa (URL)


> 1
>http://rj3.net/code/projects/jquery-
postmessage/examples/iframe...
```

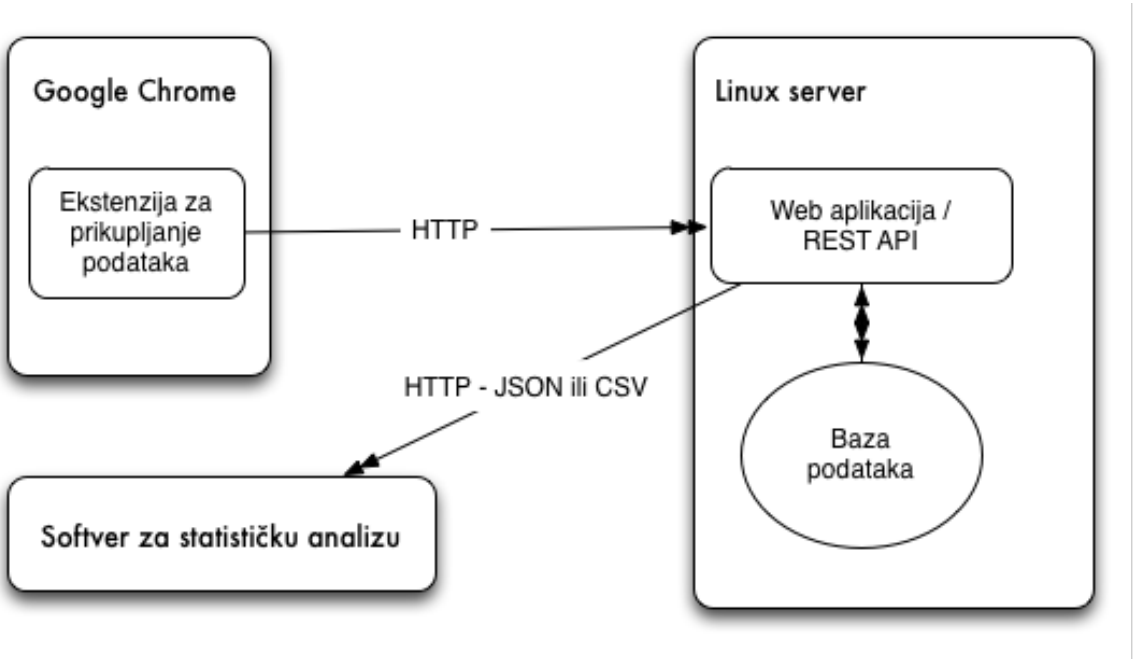
Ovo je vrlo važno za izradu ekstenzije jer omogućuje praćenje i broj pojedinih HTML elemenata u DOM stablu i njihovih atributa. Pretpostavka je da postoji velika korelacija između broja pojedinih elemenata i nekih čestih napada - tako je razumno za pretpostaviti da će se u slučaju napada otimanjem klikova na stranici nalaziti iframe element.

Velika snaga DOM stabla su događaji (engl. *event*). Događaji su pojava koja nastaje kao reakcija na neku korisnikovu akciju ili neku promjenu u DOM stablu stranice. Standardni skup događaja definiran je W3C specifikacijom [16], ali je moguće definirati i vlastite događaje i pratiti njihovu pojavu. Unaprijed je moguće definirati vlastitu funkciju koja će se izvršavati prilikom pojave nekog događaja (engl. *event handler*), moguće je definirati reakciju na točan tip događaja i mjesto pojave - hoće li se funkcija izvršavati prilikom pojave događaja u cijelom DOM stablu ili samo na nekim određenim elementima. To nam omogućuje koncept propagacije događaja unutar DOM stabla - svaki događaj počinje u stablu na elementu gdje se događaj zbio (npr. klik na poveznicu), provjerava se je li definiran kod koji se izvršava u slučaju pojave odgovarajućih događaja za taj element i, ako nije, ista provjera se ponavlja na elementu roditelju sve dok se ne nađe definiran kod koji se izvršava u slučaju pojave odgovarajućih događaja ili se dođe do korijena DOM stabla. To omogućuje granulaciju u reakciji na događaje pa je moguće ignorirati događaje koji nam nisu bitni i reagirati samo na one koji nas zanimaju.

Koristeći DOM događaje može se definirati ponašanje u slučaju ubacivanja novih elemenata u DOM stablo. U slučaju da se npr. pomoću XSS napada pokušava ubaciti `<script>` element u DOM stablo može se zabilježiti takvo ponašanje i spriječiti takav pokušaj da se zloćudna skripta ne bi izvršila. Nedostatak ovog pristupa je to što potencijalno može negativno djelovati na performanse zbog velikog broja događaja koji nastaju prilikom korištenja web preglednika i reakcije na svaki od njih što troši dragocjeno procesorsko vrijeme.

4. Sustav za prikupljanje podataka

Sustav za prikupljanje podataka zamišljen je kao softver koji prati korisnikovu upotrebu web preglednika, skuplja i pohranjuje relevantne podatke i eventualno sprječava neke jednostavnije napade. Sastoji se od klijentske strane, u obliku ekstenzije za Google Chrome preglednik, i poslužiteljske strane. Poslužiteljska strana sastoji se od web aplikacije s REST API-jem i baze podataka kamo se šalju i pohranjuju podaci s klijentske strane. Na slici 10 prikazana je komunikacija ekstenzije za web preglednik s poslužiteljskom stranom putem HTTP protokola. Na isti način, putem HTTP protokola, softver za statističku analizu može pristupati spremljenim podacima na poslužitelju te ih preuzeti u formatima JSON ili CSV.



Slika 10: Organizacija sustava za prikupljanje podataka

4.1. Upute za instalaciju

Da bi se ovaj sustav mogao uspješno početi koristiti, potrebno je prvo instalirati web aplikaciju na poslužiteljskoj strani da bi taj dio sustava za prikupljanje podataka bio spreman prije instalacije ekstenzije za Chrome.

4.1.1. Poslužiteljska aplikacija

Da bi se uspješno instalirali i konfigurirali poslužiteljska aplikacija, baza podataka i svi popratni programski paketi, dostupne su skripte za automatiziranu instalaciju svega potrebnog koje koriste programski paket Ansible. Postoje dva glavna načina za instalaciju poslužiteljske aplikacije: unutar virtualnog stroja na lokalnom računalu ili instalacija na postojeći Linux poslužitelj. U nastavku su opisane obje metode. Izvorni kod aplikacije nalazi se u obliku Git repozitorija [17].

Napomena: upute za instalaciju prilagođene su operacijskom sustavu Ubuntu Linux 14.04 LTS. Da bi se upute koristile na drugim operacijskim sustavima, vjerojatno su potrebne manje preinake koje neće biti navedene u ovom tekstu.

Virtualni stroj

Metoda instalacije unutar virtualnog stroja jednostavnija je jer korisnik ne mora imati postojeći Linux poslužitelj nego se kreira novi virtualni stroj unutar kojeg se sve instalira. Da bi bilo moguće koristiti ovu metodu, potrebno je instalirati sljedeće programske pakete:

- Git [18]
- Oracle VM VirtualBox [19]
- Vagrant [20]
- Ansible [21]

Virtualni stroj kreira se pomoću programskog paketa VirtualBox, dok se programski paketi Vagrant i Ansible brinu za upravljanje virtualnim strojem, automatskom instalacijom i konfiguracijom svega što je potrebno da bi aplikacija radila. Za početak, instalirat će se programski paketi Git i Ansible:

```
apt-get update
apt-get install git-core python python-setuptools
easy_install pip
pip install -U ansible
```

Programske pakete VirtualBox i Vagrant potrebno je skinuti s gore navedenih adresa i instalirati svaki paket dvoklikom na preuzetu datoteku. Nakon toga, potrebno je pozicionirati se u direktorij gdje želimo dohvatiti Git repozitorij u kojem se nalaze instalacijske skripte:

```
cd /direktorij/gdje/zelimo/spremiti/skripte
git clone \
https://github.com/bezidejni/master_thesis_backend.git
```

Nakon toga, potrebno je ući u direktorij koji smo upravo kreirali i pokrenuti proces stvaranja virtualnog stroja i konfiguriranja aplikacije:

```
cd master_thesis_backend/deploy
vagrant up
```

Ovaj proces će vjerojatno trajati oko nekoliko minuta. Kada proces završi, potrebno je u web pregledniku otići na adresu http://192.168.33.10 na kojoj virtualni stroj automatski poslužuje stranicu da bi se provjerilo je li sve uspješno instalirano.

Ako je potrebno pristupiti virtualnom stroju putem konzole radi eventualne daljnje konfiguracije, to je moguće ostvariti naredbom `vagrant ssh`. Nakon što se završi s radom, virtualni stroj potrebno je ugasiti naredbom `vagrant halt`.

Postojeći poslužitelj

Ova metoda instalacije podrazumijeva da korisnik već ima postojeći Ubuntu Linux 14.04 poslužitelj u koji se može ulogirati i izvršavati naredbe kao *root* korisnik. Nakon toga, potrebno je instalirati programski paket Git:

```
ssh username@ip.adresa.poslužitelja
sudo su
apt-get install git-core
```

Zatim je potrebno dohvatiti Git repozitorij sa skriptama za instalaciju te potom pokrenuti

instalacijsku skriptu:

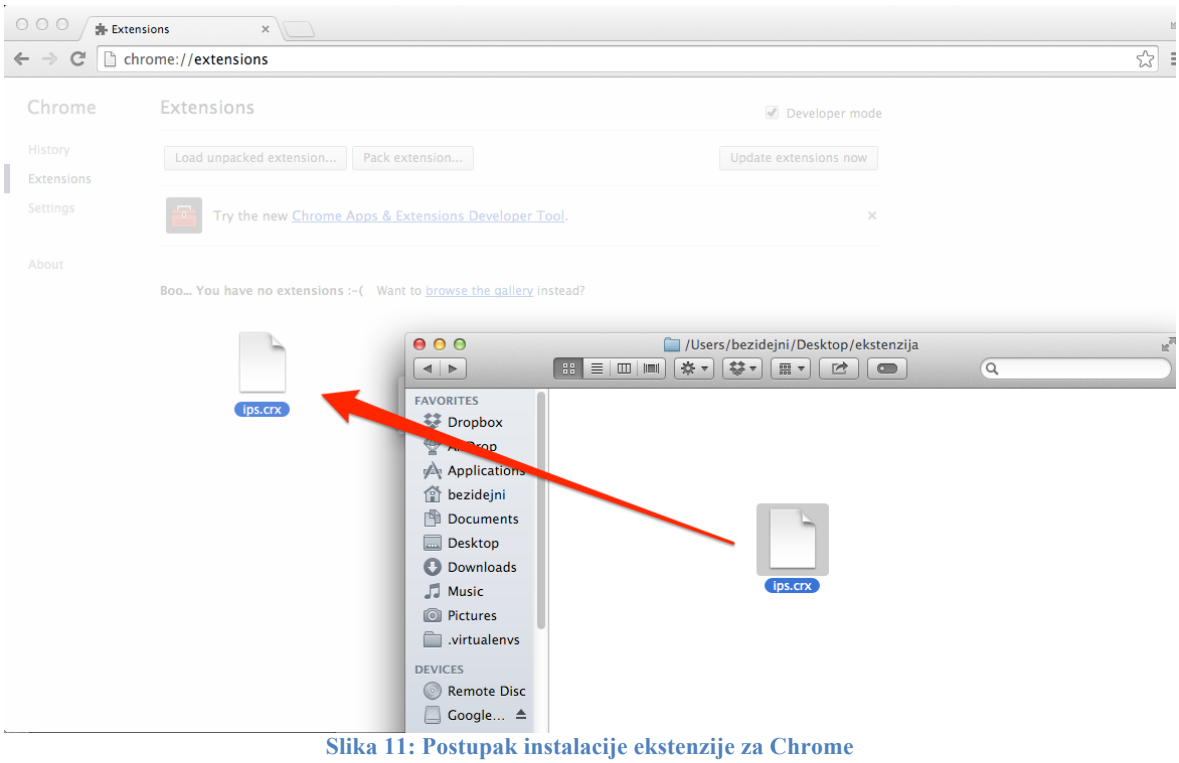
```
cd /direktorij/gdje/zelimo/spremiti/skripte
git clone \
https://github.com/bezidejni/master_thesis_backend.git
cd master_thesis_backend/deploy
chmod +x deploy.sh
./deploy.sh
```

Proces instalacije najvjerojatnije će trajati nekoliko minuta. Nakon instalacije, potrebno je provjeriti je li sve uspješno instalirano odlaskom na adresu http://ip.adresa.poslužitelja . Trebala bi se prikazati prazna početna stranica s tablicama u kojima će se prikazivati prikupljeni podaci.

4.1.2. Chrome ekstenzija

Za instalaciju Chrome ekstenzije, potrebno je imati testnu verziju Chrome preglednika nazvanu Chrome Canary [22]. Testna verzija je potrebna jer su u njoj dostupni neki API-ji koji trenutno nisu dostupni u standardnoj verziji Chrome preglednika, no to će se vjerojatno s vremenom promijeniti. Nakon instalacije Chrome Canary preglednika, potrebno je pratiti sljedeće korake:

1. Preuzeti binarnu verziju ekstenzije [23]
2. U Chrome Canary pregledniku kliknuti na ikonicu menija: 
3. Odabrati **Tools -> Extensions**
4. Odvući datoteku s ektenzijom na trenutno otvoreni prozor u Chromeu (Slika 11)
5. Pritisnuti gumb **Install** u prozoru koji se pojavi



Slika 11: Postupak instalacije ekstenzije za Chrome

4.2. Poslužiteljska strana

Poslužiteljsku stranu ovog sustava čini web aplikacija pisana u programskom jeziku Python uz pomoć korištenja Django web frameworka. Sastoji se od tri glavna dijela koja će biti opisana u nastavku poglavlja:

1. Sloj pohrane podataka (modeli)
2. Programsko sučelje (API)
3. Korisničko i administratorsko sučelje

Nakon pokretanja instalacijskih skripti, na način opisan u poglavlju 4.1.1, izvorni kod aplikacije nalazi se u direktoriju `/opt/ips/`. Običaj je da se pripadni programski paketi koje programski jezik Python zahtijeva, poput Djanga, instaliraju u virtualno okruženje da ne bi zagađivali globalni prostor imena. Virtualno okruženje nalazi se u direktoriju `/opt/venv`, a popis programskih paketa u datoteci `/opt/ips/requirements.txt`.

4.2.1. Sloj pohrane podataka (modeli)

Podaci koje aplikacija dobiva s klijentske strane i koje pohranjuje u bazu podataka oblikovani su uz pomoć klasa u Pythonu, konkretnije naslijeđivanjem ugrađene Django klase `django.db.models.Model` te se stoga nazivaju modelima. Time je omogućeno definiranje tipa atributa koje želimo spremati na samoj klasi (tekstualno polje, polje za datum, cjelobrojno polje), kao i vlastito ponašanje za svaki od tipova podataka koji se primaju s klijentske strane. Uz pomoć ugrađenog objektno-relacijskog mapera (*Object-Relational Mappera - ORM*), iz tih se klasa automatski generiraju potrebne tablice u bazi podataka te je spremanje podataka iz tih klasa u bazu moguće jednostavnim pozivanjem metode `save()` na željenoj klasi. Svaki zapis u bazi podataka predstavljen je objektom koji nasljeđuje klasu `Model`. Korištenjem ORM-a moguće je dohvatiti podskup ukupnog broja zapisa u bazi podataka i tako efektivno filtrirati podatke po željenom parametru poput datuma ili vrijednosti.

Svaki od tipova podataka koji dolaze s klijentske strane pripada jednom od definiranih modela (Slika 12):

1. CPUInfo – sprema vrijednosti opterećenja procesora u nekom trenutku
2. NetworkInfo – sprema podatke o mrežnom prometu tj. informacije o HTTP zahtjevima koji se šalju
3. DOMElementCount – sprema broj elemenata u DOM stablu za određeni URL
4. Cookie – sprema podatke o kolačićima koje stranice pokušavaju postaviti u web preglednik

Slika 12: Grafički prikaz modela u aplikaciji

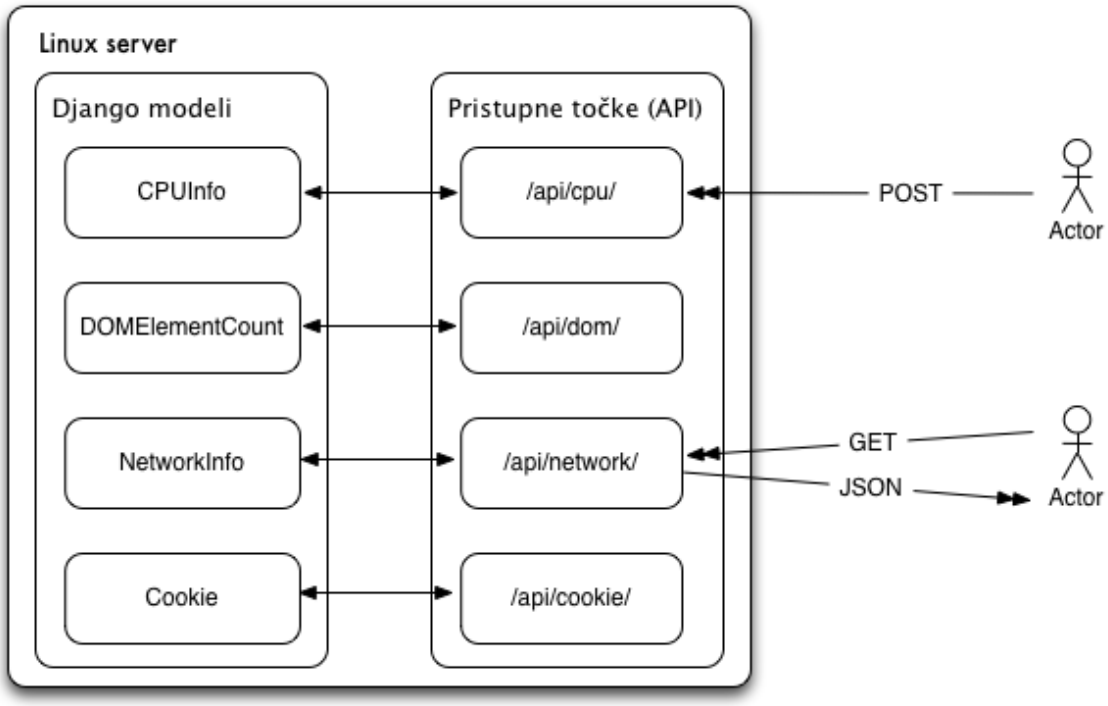
Modele je moguće jednostavno nadograđivati prilikom budućih radova na ovoj ekstenziji novim atributima, tako da se željeni atribut doda na model i pokrenu se sljedeće naredbe:

```
python manage.py schemamigration --auto stats
python manage.py migrate stats
```

Nakon toga, tablice u bazi će biti automatski izmijenjene i novi stupac (atribut) će biti dodan.

4.2.2. Programsko sučelje

Programsko sučelje aplikacije (API) je sloj koji služi kao veza između klijentske strane i sloja pohrane podataka na poslužitelju. Ostvareno je u obliku *Representational State Transfer* (*REST*) web servisa kojem se pristupa putem HTTP protokola. U REST arhitekturi svaki Django model predstavljen je resursom - objektom koji ima svoj tip, pripadne podatke i odnose s drugim resursima. U praksi to znači da se svakom tipu resursa (modelu) pristupa putem zasebnog URL-a (pristupne točke) i nad njim se vrše radnje uz pomoć standardnih HTTP glagola - GET, POST, PUT i DELETE. Moguće je pristupati individualnim resursima pomoću njihovog identifikatora ili skupovima resursa ako identifikator nije naveden (Slika 13).



Slika 13: Veza između API-ja i modela

Kada klijentska strana prikupi nove podatke, šalje ih na pristupnu točku na poslužitelju u obliku HTTP POST zahtjeva u čijem se tijelu nalaze skupljeni podaci u JSON formatu. Na sličan način funkcionira i dohvat podataka, da bi ih bilo moguće obraditi, npr. u nekom programu za statističku obradu, korisnik šalje GET zahtjev na pristupnu točku na poslužitelju i kao odgovor dobiva JSON dokument u kojemu se nalaze traženi podaci (Slika 14). Na slici je pokazana povezanost između modela i pristupnih točaka API-ja gdje se svakom modelu pristupa preko druge pristupne točke.

Funkcije koje obavljaju obradu podataka i pretvaranje iz Python klasa u JSON format nalaze se u datotekama `ips/stats/views.py` i `ips/stats/serializers.py`.

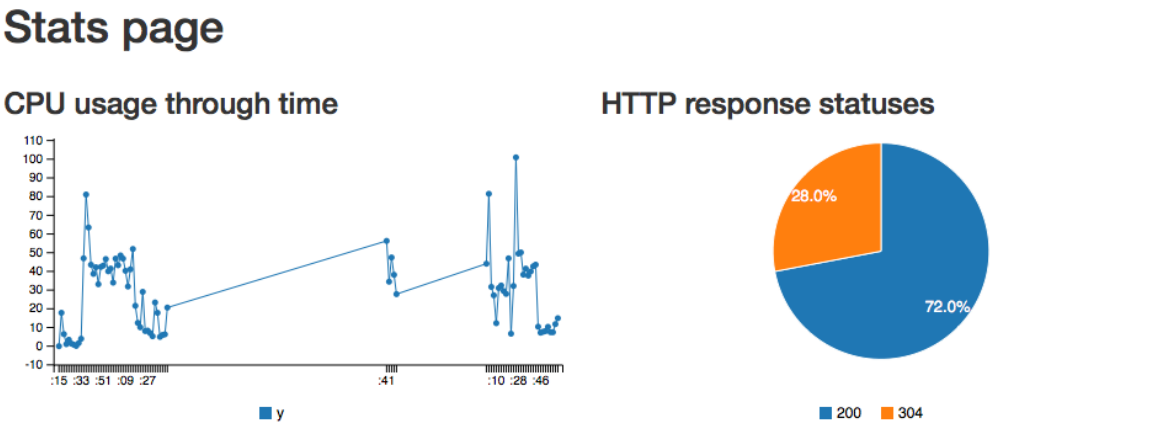
```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, POST, HEAD, OPTIONS

{
    "count": 25,
    "next": "http://127.0.0.1:8000/api/network/?limit=2&page=2",
    "previous": null,
    "results": [
        {
            "timestamp": "2014-04-16T02:11:41.943Z",
            "id": 1,
            "source_url": "https://news.ycombinator.com/",
            "destination_url": "http://blog.samaltman.com/the-worst-part-of-yc",
            "method": "GET",
            "http_status": "200",
            "type": "main_frame"
        },
        {
            "timestamp": "2014-04-16T02:11:41.996Z",
            "id": 2,
            "source_url": "http://blog.samaltman.com/the-worst-part-of-yc",
            "destination_url": "http://ajax.googleapis.com/ajax/libs/swfobject/2.2/swfobject.js",
            "method": "GET",
            "http_status": "200",
            "type": "script"
        }
    ]
}
```


Slika 14: Primjer API odgovora u JSON formatu

4.2.3. Korisničko i administratorsko sučelje

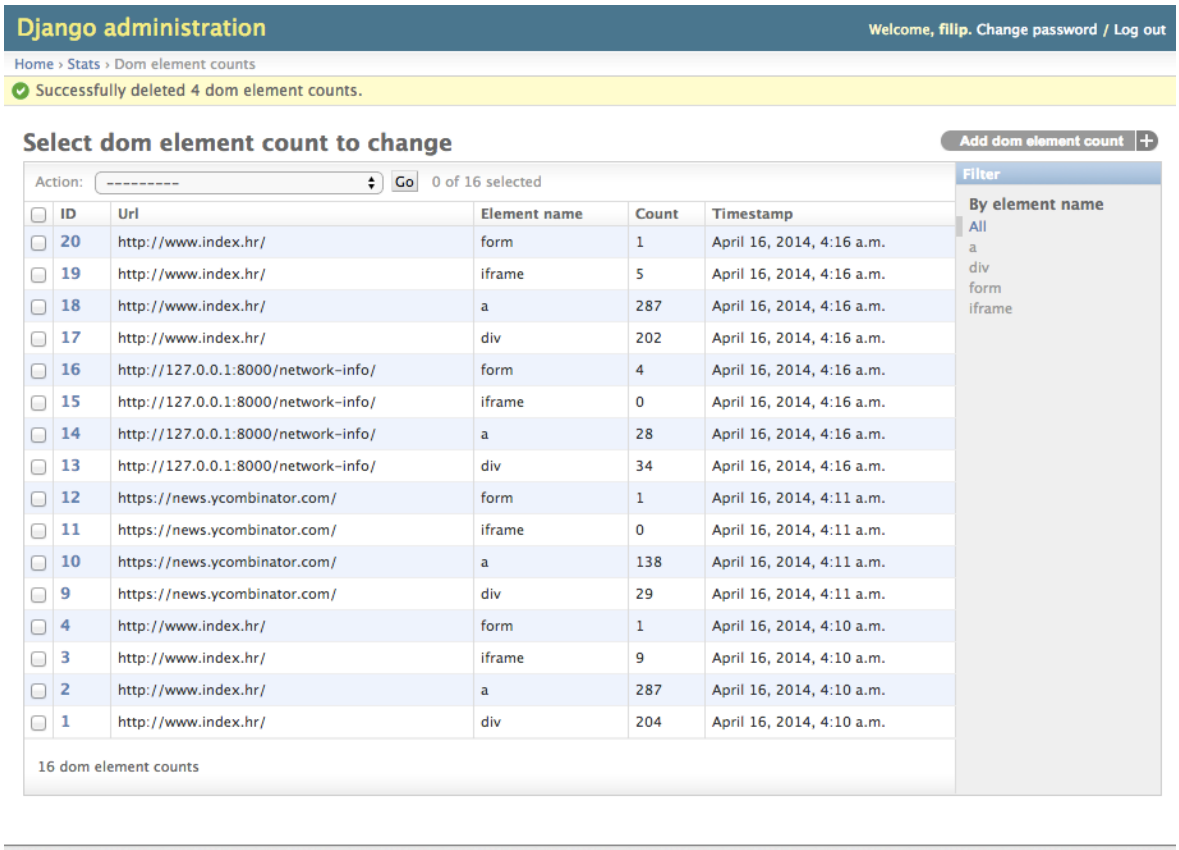
Korisničkom sučelju pristupa se pomoću web preglednika na adresi http://192.168.33.10 u slučaju instalacije u virtualnoj stroju ili preko IP adrese već postojećeg poslužitelja. Na glavnoj stranici korisničkog sučelja može se vidjeti tablični prikaz zadnje primljenih podataka s klijentske strane podijeljen po modelima, kao i grafički prikaz pojedinih vrijednosti u vremenu. U ovom slučaju na slici, to je grafički prikaz iskorištenosti procesora kroz vrijeme i podjela HTTP odgovora na temelju status koda odgovora (Slika 15).



Slika 15: Korisničko sučelje

Izmjenama u datoteci `ips/stats/models.py` moguće je za svaki model posebno regulirati količinu podataka koja će se prikazivati na korisničkom sučelju. U budućnosti, ako se budu dodavali novi modeli, vrlo je lako je izmjenama u datotekama `ips/stats/models.py` i `ips/templates/home.html` ostvariti njihov prikaz na korisničkom sučelju, kao i dodavanje novih grafova za te modele.

Osim korisničkog sučelja, tu je i administratorsko sučelje putem kojeg je moguće pregledati sve dosad prikupljene podatke, sortirati ih, filtrirati po određenom parametru, mijenjati i brisati (Slika 16). Iako administratorsko sučelje uz programsko i korisničko sučelje vjerojatno neće biti toliko često korišteno, ipak je koristan dodatak kada se želi vidjeti s kojim točno podacima sustav raspolaže.

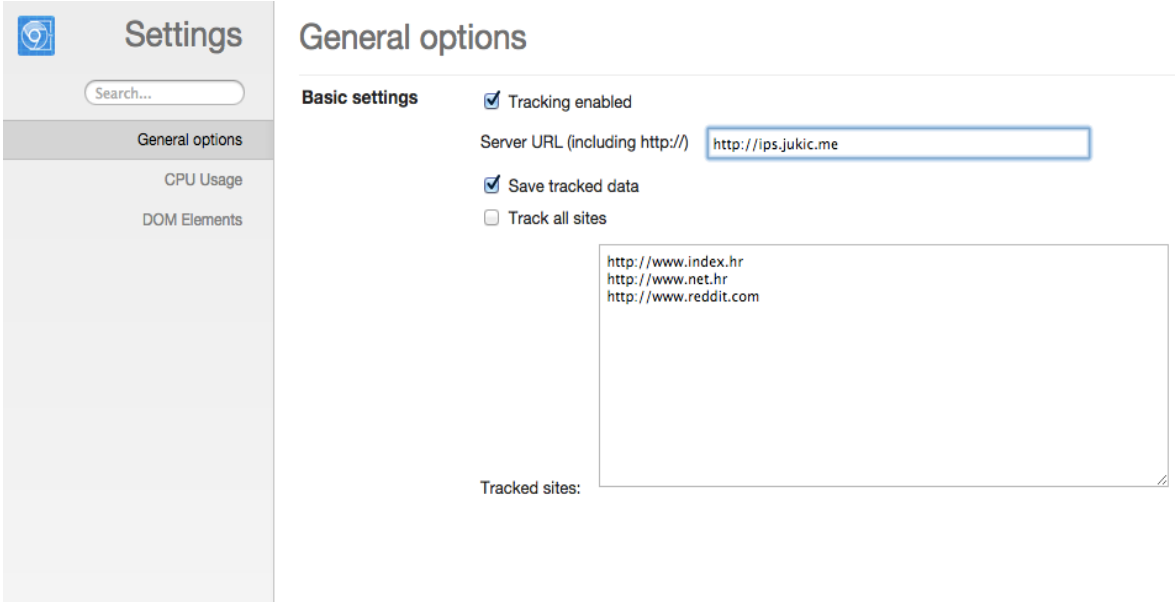


Slika 16: Administratorsko sučelje

Prikaz pojedinih parametara, određivanje filtera i polja koja se mogu pretraživati te redoslijed sortiranja mogu se definirati u datoteci `ips/stats/admin.py`.

4.3. Klijentska strana

Klijentska strana ovog sustava sastoji se od ekstenzije za web preglednik Google Chrome koja u realnom vremenu prati stranice koje korisnik posjećuje i na poslužitelj šalje relevantne informacije. Ekstenzija u sebi sadrži ekrane za podešavanje parametara praćenja do kojih se dolazi pritiskom na ikonicu  , zatim izborom **Tools -> Extensions** i na kraju pritiskom na link **Options** pored ekstenzije u popisu ekstenzija. To otvara ekran za podešavanje na kojemu se mogu podešavati opće postavke ekstenzije, ali i postavke za svaki pojedini modul. Tako je, na primjer, moguće definirati adresu poslužitelja na koju će se slati skupljeni podaci, popis stranica na kojima će se podaci skupljati ili DOM elementi koji nas zanimaju i koje ćemo pratiti (Slika 17).



Slika 17: Ekran s postavkama ekstenzije

Nakon što spremimo postavke, ekstenzija je spremna za rad. Surfanjem po Internetu ekstenzija automatski skuplja podatke i šalje ih na poslužitelj.

Prilikom otvaranja web preglednika, ekstenzija se automatski inicijalizira i pokreće se skripta `background.js` gdje se prvo dohvaćaju parametri koje smo konfigurirali na ekranu s postavkama:

```
var serverURL = store.get("store.settings.serverURL")
|| "http://ips.jukic.me";
var trackingEnabled =
store.get("store.settings.trackingEnabled") ||
"false";
var trackAllSites =
store.get("store.settings.trackAllSites");
var trackedSites = localStorage['trackedSites'] || "";
var pollingFrequency =
store.get("store.settings.pollingFrequency") * 1000;
```

Nakon toga, dodaje se kod koji se izvršava u slučaju pojave odgovarajućih događaja koje generiraju nama bitni ugrađeni Chrome API-ji:

```
chrome.runtime.onMessage.addListener(handleDOMElementC
ount);
chrome.webRequest.onResponseStarted.addListener(
        handleNetworkResponse,
        {urls: ["*://*.com/*"], types: ['main_frame',
'sub_frame', 'script']}
);
chrome.processes.onUpdated.addListener(handleCpuUpdate
);
```

Zadaća funkcija koje reagiraju na događaje je ta da od događaja preuzmu sve podatke koji su zanimljivi, oblikuju ih na način koji API na poslužitelju može prihvatiti i stavlja ih u red čekanja. Ovdje je primjer koda funkcije koja reagira na uspješno primljen odgovor na HTTP zahtjev:

```
function handleNetworkResponse(details) {
    chrome.tabs.get(details.tabId, function(tab) {

networkEventsWaiting[details.requestId].source_url =
tab.url;

networkEventsComplete.push(networkEventsWaiting[detail
s.requestId]);
        delete
networkEventsWaiting[details.requestId];

    });
    networkEventsWaiting[details.requestId] = {
        destination_url: details.url,
        method: details.method,
        http_status: details.statusCode,
        type: details.type,
        timestamp: details.timeStamp
    };
};
```

Kao što je ranije navedeno, funkcija stavlja gotov skup podataka u red čekanja. Korištenje reda čekanja je optimizacija čiji je cilj smanjiti opterećenje poslužitelja da ga se ne bi preopteretilo HTTP zahtjevima koje ne bi stigao obraditi, a potencijalno ih ima jako puno. Iz tog razloga, podaci se na poslužitelj šalju u grupi (engl. *batch send*) u određenim vremenskim intervalima. Osim što to kao posljedicu ima smanjenje opterećenje poslužitelja, povećavaju se i performanse ekstenzije i cijelog web preglednika jer nije potrebno uspostavljati veliki broj HTTP veza svake sekunde. Ta odluka doduše ima kao posljedicu povećano korištenje memorije, ali u normalnim okvirima to i ne bi trebalo predstavljati prepreku normalnom radu. Na sljedećem primjeru koda vidi se primjer funkcije koja na poslužitelj šalje podatke o mrežnom prometu koje je skupila gore spomenuta funkcija. Funkcija se automatski izvršava prilikom učitavanja pozadinske stranice i šalje AJAX POST zahtjev na poslužitelj u kojemu se nalazi lista skupljenih podataka o mrežnom prometu u JSON formatu. Kada se funkcija izvrši, prazni se lista skupljenih podataka i zadaje se sljedeće pokretanje funkcije za 10 sekundi.

```
(function sendNetworkDataHome() {
    $.ajax({
        url: serverURL + '/api/network/',
        data: JSON.stringify(networkEventsComplete),
        processData: false,
        contentType: 'application/json',
        type: 'POST',
        complete: function() {
```

```
            // Schedule the next request when the
current one's complete
            setTimeout(sendNetworkDataHome, 10000);
            networkEventsComplete = [];

        }
    });
})();
```

Interval u kojem će se funkcija izvršavati moguće je podesiti, u ovom slučaju se radi o intervalu od 10 sekundi.

5. Zaključak

Napadi i prijetnje na internetu brojni su te se mijenjanju i poboljšavaju iz dana u dan. Samo konstantnim praćenjem i analizom metoda napada moguće je potencijalne napade na vrijeme prepoznati i tako zaštititi običnog korisnika. Da bi to bilo moguće, potrebno je konstantno pratiti i prikupljati nove podatke o tome što se sve događa prilikom uobičajenog korisnikovog surfanja Internetom, ali i što se događa kada korisnik posjeti zloćudnu, tj. zaraženu, web stranicu.

Analizom tako prikupljenih podataka statističkim metodama ili mehanizmima strojnog učenja moguće je prepoznati neke karakteristike napada na korisnike i iskoristiti tako stečeno znanje za izradu mehanizama za sprječavanje takvih napada.

U sklopu ovog diplomskog rada izrađen je sustav za prikupljanje podataka u obliku ekstenzije za Google Chrome preglednik koja prati rad web preglednika i poslužiteljskog dijela gdje se spremaju prikupljeni podaci. Te podatke moguće je putem API-ja preuzeti i koristiti u daljnjoj statističkoj obradi.

U budućnosti, sustav je moguće proširiti novim vrstama podataka koji se prikupljaju, kada se u Chrome pregledniku pojave API-ji za prikupljanje tih podataka. Također, nakon izvršene analize prikupljenih podataka, moguće je u ekstenziji postaviti pravila koja bi detektirala neke napade i sprječavala ih po potrebi.

6. Literatura

1. *Internet live stats: Internet users*. 2014. http://www.internetlivestats.com/internet-users/
2. Google. *Overview - Google Chrome*. https://developer.chrome.com/extensions/overview
3. Mozilla Foundation. *DOM Reference*. 2014. https://developer.mozilla.org/en-US/docs/DOM/DOM_Reference
4. Zalewski, M. *Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks*. No Starch Press, 2012
5. Zalewski, M. *The Tangled Web*. No Starch Press, 2012
6. Erickson, J. *Hacking: The Art of Exploitation* (2. izdanje). No Starch Press, 2008
7. OWASP Foundation. 2013. *Top 10 2013*. https://www.owasp.org/index.php/Top_10_2013-Top_10
8. Hwang, D. 2013. *Secure Web Development*, http://hwang.cisdept.csupomona.edu/swanew/Code.aspx?m=XSS
9. Shiflett C. 2003. *Foiling Cross-Site Attacks*. http://shiflett.org/articles/foiling-cross-site-attacks
10. *Directive on Privacy and Electronic Communications*. https://en.wikipedia.org/wiki/Directive_on_Privacy_and_Electronic_Communications
11. *Do Not Track - Universal Web Tracking Opt Out*. http://donottrack.us/
12. *Server Technologies - HTTPS BEAST Attack*. http://www.contextis.com/blog/server-technologies-https-beast-attack/
13. *SSL, Gone in 30 seconds*. 2012. http://breachattack.com/
14. *The Heartbleed Bug*. 2014. http://heartbleed.com/
15. *jQuery*. 2014. https://jquery.com/
16. W3C. 2009. *Document Object Model (DOM)*. http://www.w3.org/DOM/
17. Jukić, F. 2014. *Github: Master thesis backend*. https://github.com/bezidejni/master_thesis_backend.
18. *Git - Downloads*. 2014. http://git-scm.com/downloads
19. *Download VirtualBox*. 2014. https://www.virtualbox.org/wiki/Downloads
20. *Download Vagrant*. 2014. https://www.vagrantup.com/downloads.html
21. *Ansible*. 2014. http://www.ansible.com/home
22. *Chrome Canary Browser*. 2014. https://www.google.com/intl/en/chrome/browser/canary.html
23. Jukić, F. *Chrome ekstenzija*. 2014. https://github.com/bezidejni/master_thesis/blob/master/extension_binary/ips.crx?raw=true
24. Stuttard, D. i Pinto, M. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. Wiley, 2011.
25. *How to defend clickjacking?*. 2014. http://www.encoders.co.in/blog/how-to-defend-clickjacking
26. *What is internet privacy?*. http://www.encoders.co.in/blog/how-to-defend-clickjacking

7. Sažetak

Naslov rada: Sustav za prikupljanje podataka o ponašanju web preglednika i njihovu analizu strojnim učenjem

Sažetak: Opisane su najčešće prijetnje i napadi s kojima se susreću korisnici na Internetu. Opisana je programska struktura ekstenzije za Google Chrome preglednik i neki od dostupnih API-ja. Razvijen je i dokumentiran softver za prikupljanje podataka o ponašanju web preglednika koji se sastoji od ekstenzije za Chrome preglednik i web aplikacije na poslužitelju gdje se prikupljaju podaci.

Ključne riječi: Google Chrome, ekstenzija za preglednik, prikupljanje podataka, napadi na Internetu, XSS, DOM

8. Abstract

Thesis title: System for collecting data on web browser behaviour and its analysis using machine learning

Summary: Most common threats and attacks that Internet users encounter were described. The structure of a Google Chrome extension and some of the available APIs were described. A system for collecting data on web browser behaviour was developed and documented, containing the Chrome extension and a server side web application where the data is collected.

Keywords: Google Chrome, browser extension, data collection, Internet threats, XSS, DOM