SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 763

DETEKCIJA PROMJENE WEB-STRANICE TEMELJEM ANALIZE NJENE STRUKTURE

Fran Domović

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022

ZAVRŠNI ZADATAK br. 763

Pristupnik: **Fran Domović (0036526632)**

Studij: Elektrotehnika i informacijska tehnologija i Računarstvo

Modul: Računarstvo

Mentor: doc. dr. sc. Stjepan Groš

Zadatak: **Detekcija promjene Web-stranice temeljem analize njene strukture**

Opis zadatka:

Na Internetu je aktivno stotine milijuna web stranica. One olakšavaju razmjenu informacija i omogućuju korištenje brojnih usluga. Zbog broja korisnika koji redovito posjećuju web stranice i putem njih razmjenjuju podatke te web tražilica koje ih indeksiraju, napadačima je u interesu kompromitirati web stranice te u njih ugraditi vlastite komponente. Te komponente mogu, između ostalog, služiti za maliciozno oglašavanje, širenje zloćudnog koda i krađu osobnih podataka. Zbog količine web stranica na Internetu i kreativnih načina kojima napadači prikrivaju ubačene komponente, vrlo je zahtjevno otkrivati i tražiti takve komponente ručno. Također, kod web pauka (engl. crawlera) bi rješenje obrnutog problema, tj. otkrivanje stranica koje ne sadrže promjene strukture, omogućilo izbjegavanje suvišnog dohvaćanja web stranica i smanjio potrebu za diskovnim prostorom. U sklopu završnog rada potrebno je istražiti kako se struktura HTML-a može koristiti za detekciju promjene web stranica. Potrebno je razviti rješenje koji će na temelju dvaju zadanih struktura HTML dokumenata uspoređivati i detektirati razlike te utvrditi je li web stranica promijenjena. Nadalje, potrebno je istražiti kako se i na koji način HTML dokument može filtrirati (odstraniti atribute, sadržaje teksta i sl.) kako bi se pospješili rezultati otkrivanja promjena. Naposljetku, potrebno je testirati rješenje nad podacima dohvaćenih web-paukom, među kojima se nalaze i označeni primjeri web stranica za koje je poznato da sadrže zlonamjerno ubačene HTML komponente. Na temelju rezultata potrebno je odgovoriti na pitanje može li se struktura HTML dokumenta koristiti za detekciju promjene stranica te može li se na temelju takve detekcije utvrditi maliciozna promjena. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 10. lipnja 2022

Zahvaljujem se mentoru doc. dr. sc. Stjepanu Grošu, mag. ing. Bruni Skendroviću i mag. ing. Ivanu Kovačeviću na stručnim savjetima i pomoći oko izvedbe ovoga rada.

Sadržaj

Uvod

Internet je tehnologija koja nam omogućava prijenos podataka i informacija te omogućuje prikaz različitog sadržaja velikom broju korisnika. Prijenos informacija odvija se putem *web* stranica koje omogućuju korištenje brojnih usluga, a na internetu ih je u svakom trenutku aktivno nekoliko stotina milijuna [1] . Razvojem internet tehnologije i njegovim uključenjem u svakodnevni život, došlo je do povećanja kibernetičkog kriminala i broja napadača kojima je cilj ostvariti neku vrstu zarade. Zbog broja korisnika koji redovito posjećuju *web* stranice, cilj je napadača kompromitirati sigurnost *web* stranica te u njih ugraditi vlastite komponente.

Načini kompromitiranja i ubacivanja vlastitog koda napadači provode na razne načine, a ubačene komponente mogu biti *HTML* strukture i zloćudne skripte napisane u *JavaScript*-u. Komponente mogu, između ostalog, služiti za maliciozno oglašavanje, širenje zloćudnog koda i krađu osobnih podataka. Zbog količine *web* stranica na internetu i kreativnih načina kojima napadači prikrivaju ubačene komponente, proces otkrivanja i traženja takvih komponenata je zahtjevan. Kod *web* pauka (eng. *web crawler*) bi rješenje obrnutog problema, tj. otkrivanje stranica koje ne sadrže promjene strukture, omogućilo izbjegavanje suvišnog dohvaćanja web stranica i smanjio potrebu za diskovnim prostorom. Važna je brza detekcija zloćudnog koda radi sigurnosti web prostora, a počinjena šteta, koja uključuje krađu korisničkih podataka, svedena na minimum.

U sklopu završnog rada istraženo je kako se struktura *HTML*-a može koristiti za detekciju promjena *web* stranica. Razvijeno je rješenje koje će na temelju dvaju zadanih struktura *HTML* dokumenata uspoređivati i detektirati razlike te utvrditi je li *web* stranica promijenjena. Nadalje, istraženo je kako se i na koji način *HTML* dokument može filtrirati u smislu izbacivanja atributa, sadržaja slika i teksta. Naposljetku, potrebno je bilo testirati rješenje nad podacima dohvaćenim *web* paukom, među kojima se nalaze i označeni primjeri *web* stranica za koje je poznato da sadrže zlonamjerno ubačene *HTML* strukture. U sklopu detekcije zlonamjerno ubačenih *HTML* struktura, istražena je uloga *CSS* stiliziranja komponenata, kojima se postiže prikrivanje na web stranici. Na temelju rezultata, potrebno je odgovoriti na pitanje može li se struktura *HTML* dokumenta koristiti

za detekciju promjene *web* stranice te može li se na temelju takve detekcije utvrditi maliciozna promjena.

U prvom poglavlju završnog rada objašnjen je *HTML* i njegova uloga u prikazu web stranica te načini vađenja oznaka i usporedba dvaju *HTML* dokumenata te uloga *CSS* stiliziranja *HTML* oznaka (eng. *HTML tag*).

U drugom poglavlju naglasak će biti stavljen na parsere *web* stranica i programske podrške za ostvarenje parsiranja. Bit će objašnjena pozadina trovanja *web* tražilica (eng. *search engine poisoning*) i svrha ubacivanja zlonamjernog koda.

U trećem poglavlju rada opisan je algoritam za pronalazak novo dodanih *HTML* struktura koje su prisutne u novoj verziji stranice te će se objasniti pronalazak zlonamjernog koda na temelju *HTML* oznaka te njihovih atributa i vrijednosti.

U zadnjem poglavlju dan je prijedlog načina poboljšanja postojeće skripte koja koristi prethodno opisane metode. Bit će prikazani rezultati eksperimenta te navedene prednosti i mane ovog pristupa.

1. HTML prezentacijski jezik i njegova uloga u izradi modernih web stranica

HTML (engl. *HyperText Markup Language*) je najpopularniji prezentacijski jezik koji se koristi prilikom izrade *web* stranica za prikaz na *web* preglednicima. Definira *HTML* dokument (hipertekst) te daje upute *web* pregledniku kako prikazati DOM (engl. *Document Object Model*) strukturu odnosno stablo [2] U DOM-u, dokumenti imaju logičku strukturu koja omogućava manipulaciju i pristup određenim strukturama. *HTML* dokumenti grade se uz pomoć: oznaka koji grade strukturu dokumenta, programskog jezika *JavaScript* koji služi za manipulacijom DOM stabla i podacima prikazanim na stranici te *CSS* opisni jezik koji služi za stiliziranje prikazanih *HTML* oznaka.

Hipertekstovi označavaju poveznicu unutar samog *HTML* dokumenta koja odvodi na neki drugi tekst unutar istog *HTML* dokumenta ili dokumenta druge web stranice. Svaka stranica koja sadrži poveznicu na neku drugu stranicu bit će uhvaćena i indeksirana na veću poziciju unutar *web* tražilice, pomoću *web* pauka.

HTML oznake služe za anotiranje teksta, slika ili drugog sadržaja za prikazivanje na *web* stranicama. Primjeri *HTML* oznaka koje dokument mogu sadržavati su*:* `<head>`, `<body>`, `<title>`, `<header>`, `<footer>`. *HTML* oznake mogu sadržavati i atribute koji imaju svoje pripadajuće vrijednosti i koji omogućuju dodatne opcije. Atributi se zapisuju u obliku `<a href="www.website.com" hidden>` , gdje je href ime atributa, a navedena *URL* poveznica, njegova vrijednost. U ovom završnom radu naglasak se stavlja na parsiranje *HTML* dokumenta, analizu *HTML* oznaka i njihovih atributa dobivenih parsiranjem te otkrivanje novo ubačenih struktura.

1.1. JavaScript jezik u HTML-u te načini ubacivanja zloćudnog koda

JavaScript je skriptni jezik koji omogućava velik broj kompleksnih funkcija u sklopu *web* stranica. Te funkcije mogu raditi manipulaciju DOM stabla odnosno dodavati u stablo,

modificirati i brisati oznake *HTML* dokumenta [3] Jedna od značajka *JavaScript*-a je mogućnost primanja informacija od korisnika i slanje istih na druge *web* lokacije ili servere za pohranu podataka. *JavaScript* je jezik koji može biti korišten u maliciozne svrhe zbog različitih načina kojima se može ostvariti narušavanje sigurnosti *web* prostora. Jedan od načina zlonamjernog korištenja *JavaSript*-a od strane korisnika je *Cross-Site Scripting* (kraće *XSS*) napad kojim se narušava sigurnost napadnute web stranice.

1.1.1. Cross-Site Scripting (XSS) napad i ubacivanje HTML struktura u postojeću web stranicu

Klijentske skripte su skripte korištene od strane modernih pretraživača te imaju jednostavne funkcije kao što su formatiranje teksta ili manipulacija podacima na stranici. *Cross-Site Scripting* (kraće *XSS*) omogućava klijentu da ubaci skriptu u zahtjev koji se šalje serveru koji kao odgovor vraća skriptu u klijent [4] Vraćena skripta će izvršiti određene funkcije pri svakom pristupu korisnika na stranicu. *XSS* napadi su opasni zato što mogu vrlo brzo zahvatiti velik broj korisnika koji pristupi web stranici kojoj je narušena sigurnost. Napadači ovakvom vrstom napada pokušavaju narušiti sigurnost iz zarade, a ona se može manifestirati u obliku ubacivanja vlastitog *HTML* koda ili kao preusmjeravanje podataka s *web* stranice koji u konačnici mogu završiti krađom korisničkih podataka.

Postoje mnogi sigurnosni propusti prilikom korištenja *JavaScript*-a te se oni mogu koristiti u zlonamjerne svrhe od strane napdača. Listing 1.1 je primjer *HTML* dokumenta gdje se zbog nesigurnog korištenja *JavaScript*-a, pomoću *URL* polja *web* preglednika, može ubaciti *HTML* struktura ili izvršiti kod iz *HTML* strukture ako je ona omotana *HTML* oznakom `<script>`.

Nad HTML dokumentom prikazanim u Listing 1.1 moguće izvršiti *XSS* napad jer oznaka `<script>` koristi *URL* vrijednost koju je moguće mijenjati preko *URL* polja *web* preglednika. Stranica na kojoj će se odvijati napad označena je *URL* poveznicom: „*http://www.vulnerable.site*“. Kod korišten u `<script>` oznaci može se promijeniti tako da se *URL web* stranice promjeni u oblik: „`http://www.vulnerable.site/-welcome.html?name=Joe`“, gdje *web* stranica iz *URL*-a pročita vrijednost *Joe* za varijablu *name* te ju zapiše na stranici. U *HTML* dokumentu prikazanim u Listing 1.1 nije implementirana sigurnost u sprječavanju *XSS* napada. Napad je moguće ostvariti

promjenom *URL* poveznice prikazane u Listing 1.2, gdje će kod *web* stranice pročitati vrijednost varijable *name* i kad dođe do izvršavanja koda, web preglednik će `<script>` tag ugraditi u stranicu te izvršiti njen kod. *XSS* napadom moguće je ubaciti maliciozne zloćudne strukture koje u sebi mogu sadržavati linkove kojima je cilj ostati sto duže skriven na web stranici i ostvariti trovanje web tražilice.

Listing 1.1 ubacivanje *HTML* strukture *XSS* napadom

```
<html>
    <title>Welcome!<title>
    <script>
    Var x=document.URL.indexOf('name')+5;
Document.write(document.URL.substring(x.document.URL.length)
    </script>
</html>
```

Listing 1.2 primjer ubacivanja *JavaScript* skripte putem *URL* polja

```
„http://www.vulnerable.site/welcome.html?name=<script>alert-
(document.cookie)</script>“
```

1.2. Trovanje web tražilica (eng. Search Engine Poisoning)

Web stranice u sebi imaju prethodno objašnjene hiperlinkove, kojima se *web* pauk (eng. *web crawler*) koristi i njima odlazi na druge *web* stranice. Tako *web* pauk indeksira stranice na viša mjesta u *web* tražilici. Trovanje *web* tražilica je napad kojim napadač pokušava svoju stranicu indeksirati na što višu poziciju prikazivanje prilikom korištenja tako da *URL* svoje stranice pokuša ubaciti u neku drugu stranicu koju posjećuje više ljudi. Napadač u ostvarenju svog cilja traži stranice na kojima postoje ranjivosti i na kojima se može izvesti *XSS* napad. On može ubaciti *HTML* strukturu koja sadrži *URL* poveznicu na njegovu stranicu, a svoju strukturu različitim metodama može sakriti od korisnika i

vlasnika stranice. Njegov glavni cilj je da ta struktura bude što duže nedetektirana jer, iako je skrivena, *web* pauk će ju indeksirati i pritom napadač ostvaruje svoj cilj.

HTML oznaka <a> iz Listing 1.3 sadrži poveznicu na drugu *web* stranicu koja može biti maliciozna. Struktura ostaje skrivena od korisnika zbog definiranog *inline CSS*-a koji tu *HTML* oznaku pozicionira izvan vidljivog prostora prikaza *web* stranice. U ovom radu istražen je i način filtriranja *CSS*-a u svrhu traženja poznatih kombinacija koje se obično koriste u ovakvoj vrsti napada.

Listing 1.3 HTML struktura za ostvarenje skrivenosti prilikom napada

```
<a href="https://malicious-website.com/" style="position:absolute;
left:9999px;"></a>
```

1.3. Cascading Style Sheets uloga u ubacivanju zloćudnog koda

Cascading Style Sheets (kraće *CSS*) je jezik korišten za opisivanje na koji se način dokumenti napisani u *HTML*-u ili *XML*-u prikazuju u *web* pregledniku [5] . *CSS* opisuje kako se svaki element treba prikazivati na ekranu. *CSS* može biti napisan u zasebnom dokumentu, koji se uključuje u pridružujući HTML dokument, ili može biti napisan unutar samih HTML oznaka u *style* atributu. *CSS* ima veliku ulogu u *XSS* napadima, gdje je svrha ostvariti trovanja tražilica, a to je ostvarivo skrivanjem ubačenog koda. Što maliciozni kod dulje ostane skriven na *web* stranici od korisnika i održavatelja stranice, to je mogućnost zarade napadača veća. U sklopu rada analizirani su *HTML* dokumenti *web* stranica za koje je poznato da sadrže maliciozne strukture. Istraženi su i načini kojima se *CSS*-om može ostvariti skrivenost takvih struktura te načini detektiranja takvih struktura unutar *HTML* dokumenata.

Analiziranjem velikog broja *web* stranica za koje je poznato da sadrže takve strukture stvorena je lista mogućih kombinacija kojima se ostvaruje skrivenost. Kombinacije *CSS*-a prikazane u Listing 1.4 mogu biti korištene za ostvarenje skrivenosti *HTML* struktura. Prva stavka ostvaruje skrivenost tako da strukturu pomakne iz vidljivog dijela ekrana. Treća stavka ostvaruje skrivenost tako da za prozirnost (eng. *opacity*) postavi vrijednost nula.

Peta stavka *z-index* govori o poziciji strukture gledane po z-osi koordinatnog sustava, omogućuje da se preko određene strukture prikazuje neki drugi dio *web* stranice dok je ona u pozadini skrivena. Pomoću *CSS*-a skrivenost strukture moguće je ostvariti na mnoge načine, a u sklopu rada su istražene kombinacije iz Listing 1.4. Navedene kombinacije su pronađene analizom poznatih stranica koje sadrže maliciozne strukture te su najčešće korištene na *web* stranicama iz *.hr* domene.

Listing 1.4 *CSS* kombinacije za skrivanje *HTML* struktura

```
1. position: absolute; left: -9999px;
2. visibility: hidden;
3. opacity: 0;
4. display: hidden;
5. z-index: 0;
6. filter: alpha(opacity)=0;
7. hidden
```

1.4. Programska potpora za izradu skripte i usporedbu HTML dokumenata

Pri izradi skripte korišten je programski jezik *Python* [6] te je u njemu implementirana usporedba dva *HTML* dokumenta koji su parsirani pomoću *Python* biblioteke Beautifulsoup 4 [7] . Parsiranje je proces koji se događa kad korisnik putem svog *web* preglednika pristupi nekoj *web* stranici te se kao posljedica toga izgrađuje stablasta struktura. Parsiranje je u programskom jeziku *Python* ostvareno pomoću *lxml* [8] parsera koji se temelji na C biblioteci i odabran je zbog svoje brzine. Parser *lxml* nema mogućnost ispravljanja nepravilnog *HTML* dokumenta, a kao posljedica toga prilikom parsiranja dolazi do ispadanja pojedinih *HTML* oznaka. Ispadanje pojedinih *HTML* oznaka može se zanemariti zbog veličine *HTML* dokumenata koji se parsiraju i zbog brzine parsiranja koja je potrebna.

U primjeru iz Listing 1.5 je prilikom parsiranja izbačena `<p>` oznaka, ali u slučaju da je *HTML* dokument pravilno oblikovan parsiranje neće napraviti promjene. Pravilno oblikovan dokument kod kojeg se neće dogoditi ispadanje je onaj koji za svaku otvarajuću *HTML* oznaku `<tag>` ima odgovarajuću zatvarajuću oznaku `</tag>`.

Listing 1.5 Parsiranje *HTML* dokumenta pomoću *lxml* parsera i *Beautifulsoup* 4 biblioteke

```
Lxml_structure = BeautifulSoup("<head><li></p>", "lxml")

<html><head><body><li></li></body></html>
```

Biblioteka *BeautifulSoup* 4 nakon parsiranja *HTML* dokumenta stvara objekt koji posjeduje funkcije za obradu podataka. Postoje funkcije pomoću kojih se mogu izvaditi pojedine oznake, filtrirati atributi *HTML* oznaka i njihovih vrijednosti. Kombinacijom funkcija biblioteke iz *HTML* dokumenta izvučene su samo oznake bez vrijednosti i teksta te je dobivena čista *HTML* struktura. Dvije strukture *HTML* dokumenata koje posjeduju samo oznake uspoređuju se pomoću Python biblioteke *Difflib* [9] korištenjem opcije *differ*. Biblioteka koristi "*SequenceMatcher*" koji prolazi po svim linijama oba dokumenta i vraća novi dokument u kojem su ispisane razlike. Razlike su označene tako da se dodaje simbol plusa (+) ispred retka koji se pojavljuje u novom dokumentu, a ne postoji u starom, te se dodaje simbol minusa (–) ispred retka koji ne postoji u novom dokumentu.
Primjer ispisanih razlika u Listing 1.6 je rezultat usporedbe dva *HTML* dokumenta. Novo dodane strukture su označavaju se simbolom plusa na početku retka, a one su: `<div>` i `<a>`. Linije koje su izbačene u novoj verziji *HTML* dokumenta označene su sa simbolom minusa na početku retka.

Listing 1.6 ispis *differ* funkcije nakon usporedbe dva *HTML* dokumenta

```
-<a></a>
+<div>
+<a></>
+</div>
```

2. Ocjena malicioznosti i pronalazak ubačene HTML strukture

Zbog prirode napada kojima se napadači služe kako bi ostvarili zaradu, većina maliciozno ubačenog koda je skrivena iz vidljivog dijela prikaza *web* stranice osim u slučaju kad se struktura namjerno ubaci u području oglasa na *web* stranici. Provjerom stranica kojima je narušena sigurnost, dolazi se do problema gdje je teško bez ručnog pregleda pronaći maliciozni kod na stranici. Pomoću ranije opisanih biblioteka i pristupa u poglavlju 1.4, lako se dolazi do novo dodanih *HTML* oznaka. Novo dodanih elemenata može biti nekoliko tisuća, a ručnim pregledom taj posao postaje vremenski zahtjevan. Radi automatizirane detekcije, uvodi se i pregled atributa unutar *HTML* oznaka te se gledaju vrijednosti zapisane unutar *style* atributa. Ranije opisanim metodama, ispitan je način pronalaska malicioznog koda, a kao rezultat usporedbe vraća samo se dio strukture *HTML* dokumenta za koji algoritam ocjeni da je maliciozan. Dodatno se ispisuje popis svih pronađenih *URL* poveznica na stranici. Ovakvim se pristupom smanjuje veliki broj lažno pozitivnih stranica te se proces ručnog ocjenjivanja značajno ubrzava.

2.1. Algoritam za ocjenu malicioznosti te otkrivanje ubačene HTML strukture

Za otkrivanje ubačene strukture i ocjenu malicioznosti, skripta pisana u programskom jeziku Python prvo dohvaća dva *HTML* dokumenta, najnoviju i drugu najnoviju verziju određene *web* stranice. Polazi se od pretpostavke da se u najnovijoj verziji pojavila neka promjena koju je potrebno detektirati te joj ocijeniti malicioznost. Detekcija promjene, odnosno novo dodanih *HTML* struktura, obavlja se pomoću *Python* biblioteke *Beautifulsoup* 4. Ocjena i detekcija malicioznosti provodi se osmišljenim algoritmom koji provjerava *CSS* kombinacije zapisane u *style* atributu unutar *HTML* oznake. Rezultat skripte stvara *HTML* grafički prikaz usporedbe nove i stare verzije radi lakšeg pronalaska zloćudnog koda u originalnom *HTML* dokumentu. Ispisuje se i isječak *HTML* dokumenta za koji se smatra da sadrži maliciozni kod.

Ocjena malicioznosti radi se pomoću komponente nazvane sigurnosni indeks (eng. *security index*), koja nam govori o pronađenim opasnim kombinacijama unutar novo dodanih *HTML* struktura. Svaka komponenta koja se provjerava, povećava sigurnosni indeks za jedan, osim u slučaju kad je unutar stranice dodana nova *URL* poveznica i u slučaju kad se koriste jako neuobičajene vrijednosti za pozicioniranje elemenata. U ta dva slučaja, sigurnosni indeks povećava se za dva. Sigurnosni indeks ima granicu (eng. *threshold*) četiri, što znači da ukupni zbroj pronađenih kombinacija omogućuje postizanje skrivenosti komponente. Granica sigurnosnog indeksa određena je višestrukim testiranjem nad podacima te vrijednost četiri ima najbolje rezultate. Kad se sigurnosni indeks smanji dolazi se do značajnog povećanja lažno pozitivnih rezultata, što otežava ručnu provjeru. Sigurnosni indeks kalibriran je algoritmom tako da može pronaći maliciozne strukture, a ako se on poveća, gubi se svojstvo pronalaska malicioznih struktura koje su prije bile pronađene. U slučaju povećanja sigurnosnog indeks, potrebno je promjeniti algoritam detekcije i zbrajanja zlođudnih kombinacija. Sigurnosni indeks je najbitnija komponenta ovog algoritma te će kombinacije biti dodatno objašnjene.

Nakon što se usporede dvije dvije verzije spremljenih *HTML* dokumenata, ponovo se parsiraju dobivene razlike koje se nalaze samo u najnovijoj verziji dokumenta. Prolaskom kroz parser, stranica se vraća u obliku u kojem se mogu čitati i izvlačiti sadržane *HTML* oznake te njihove atribute i vrijednosti. Svakoj novo dodanoj oznaci *web* stranice provjeravaju se njeni atributi. Ako oznaka sadrži atribute, provjerava se postoje li *URL* poveznice u vrijednosti atributa. Novo dodane *URL* poveznice spremaju se i kasnije se koristi prilikom ispisa rezultata. Iduće se provjerava prisutnost atributa hidden za koji se dodaje sigurnosni indeks.

Zadnja provjera vrši se nad atributom *style* čiji se *CSS* kod formatira i zapisuje ako ispunjuje određene uvjete. Formatirani kod prolazi kroz algoritam i uspoređuje se sa kombinacijama koje su korištene na *web* stranicama za koje je poznato da sadrže zlođudne strukture. Ako je kombinacija pronađena, uvjet zlođudnosti je ispunjen i HTML oznaka se zapisuje. Radi bržeg prolaska po *CSS*-u, dodana je provjera gdje se za svaki *CSS* atribut provjerava je li on u listi koja sadrži *CSS* elemente kojima je moguće ostvarit skrivanje *HTML* struktura. Sadržani *CSS* atributi su: `z-index`, `filter` koji koristi vrijednost `alpha(opacity)=0`, `letter-spacing`, `position` sa vrijednošću `absolute`, `height`, `width` te lista *CSS* vrijednosti korištenih za pozicioniranje: `left`, `right`, `bottom`, `top`.

Prolaskom *HTML* oznakama, uvodi se još jedan dodatni stilski sigurnosni indeks, koji se odnosi samo na kombinacije unutar *HTML* oznaka. Granica stilskog sigurnosnog indeksa je dva zbog toga što su dvije kombinacije dovoljne da *HTML* oznaka može zadovoljiti uvjet zloćudnosti.

Kombinacija prikazana u Listing 2.1 može se pojaviti unutar *style* atributa neke *HTML* oznake, a ona može zadovoljiti granicu potrebnu za ocjenu malicioznosti. Ako sigurnosni indeks `style` atributa dostigne granicu od dva, odgovarajuća oznaka i sve oznake ugnježđene u roditeljskoj oznaci, zapisuju se za kasniji ispis. Sigurnosni indeksi se broje samo jednom za svaki element dok se stilski sigurnosni indeks svaki put broji isponova i dodaju se novo pronađene oznake. Kad sigurnosni indeks prođe granicu od 4, *web* stranica se ocjenjuje kao maliciozna te se oznake u kojima je stilski sigurnosni indeks bio veći od 2 zapisuju u vanjski *HTML* dokument. U rezultatu se još zapisuju i sve pronađene *URL* poveznice na *web* stranici kako bi daljnja analiza i provjera neuhvaćenih poveznica bila efikasnija. Uz zapisani rezultat, zapisuje se i kombinacija koja je dovela do ocjene malicioznosti. Proces se ponavlja za svaku novo ubačenu *HTML* oznaku.

Listing 2.1 Kombinacija dva *CSS* pozicioniranja dovoljna za skrivanje strukture

```
position: absolute; left: -9999px
```

Rezultat prikazan u Listing 2.2 je ispis algoritma provedenog nad nekom *web* stranicom. *HTML* struktura pronađena algoritmom je ocjenjena kao maliciozna zbog sigurnosnih indeksa. Sigurnosni indeks povećale su *CSS* kombinacije, a one su *position* i *left* koja ima vrijednost kojoj je namjera sakriti nešto s vidljivog dijela ekrana.

Listing 2.2 Zloćudna struktura pronađena algoritmom

```
<div style=”position: absolute; left: -12339px;”>
      <p>
            <a href=”www.badcasino.com/”>
      </p>
</div>
```

2.2. CSS atributi za skrivanje struktura s vidljivog područja ekrana

U rada su istražene poznate kombinacije CSS atributa pomoću kojih se isječak *HTML* strukture može sakriti s vidljivog područja ekrana. Najčešće korištene kombinacije su one koje uključuju `position:aboslute` i neku od kombinacija *CSS*-a za pozicioniranje. Kombinacije za pozicioniranje su: `left`, `right` , `top`, `bottom` pomoću kojih se *HTML* struktura po stranici može pomicati lijevo, desno, gore i dolje za određene vrijednosti koje se mogu mjeriti pixelima (skraćeno *px*). Korištenjem pozicioniranja uz vrijednosti koje idu u plus ili minus, nekoliko tisuća, lako je sakriti *HTML* strukture koje su zloćudno ubačene i čiji je cilj ostvariti trovanje tražilica.

3. Eksperiment detekcije ubačenog HTML koda

Web pauk je alat koji prolazi po *web* stranicama tako da posjećuje svaku poveznicu koju pronađe u *HTML* dokumentu *web* stranice. *Web* pauk preuzima i indeksira stranice po cijelom internetskom prostoru. Cilj takvog *web* alata je taj da nauči o čemu je svaka *web* stranica na internetu te da se kasnije spremljeni podaci mogu dohvatiti [10] . Tu se nailazi na prvu problematiku već ranije opisanu u radu, *web* pauk odlazi i indeksira sve pronađene poveznice u nekoj *web* stranici, a to ne isključuje odlazak na zloćudno ubačene poveznice. Pauk prolaskom kroz stranicu sprema *HTML* dokument *web* stranice te se spremljeni dokument pretvara u kriptografski sažetak (*hash*) i sprema se u bazu podataka.

Provedeno testiranje skripte za detekciju promjene web stranice pomoću njene strukture te detekcija maliciozno ubačenog *HTML* koda testirana je nad podacima iz *MongoDB* [11] baze podataka. *MongoDB* baza podataka je korisna u radu s velikim brojem podataka te se u nju zapisuju podaci u obliku dokumenata. U dokumentima podaci se zapisuju u različitim oblicima. Dokumenti mogu imati više polja, tipova podataka te imaju mogućnost spremanja liste podataka. *MongoDB* baza podataka korištena za eksperiment pohranjuje *URL* poveznice i parsirane *HTML* dokumente preuzete odlaskom na *URL* poveznice. Eksperiment je proveden u programskom jeziku Python te je za spajanje na *MongoDB* bazu korištena *Python* biblioteka *PyMongo* [12] . Baza podataka MongoDB ima bazu podataka koja sadrži više kolekcija. Korištene su dvije kolekcije, jedna sprema *URL* poveznicu na *web* stranicu i kriptografsku funkciju sažetka dok druga sprema kriptografsku funkciju sažetka i *HTML* dokument *web* stranice, preuzete s *URL* poveznice prve kolekcije. Kriptografska funkcija sažetka je poveznica između dokumenata spremljenih u različitim kolekcijama. Baza podataka se ažurira na svakodnevnoj bazi.

Testiranje je provedeno u dva slučaja. U prvom slučaju provjeravala se detekcija promjene web stranice te detekcija zloćudnog koda u detektiranoj strukturi promjene između dvije spremljene verzije stranice. Drugo testiranje se provodilo nad svim spremljenim najnovijim verzijama stranica bez uspoređivanja s prethodnom verzijom. Potupci su slični, osim što se u drugom postupku ne koriste usporedbe dvaju verzijski susjednih *web* stranica te se postupak provodi nad cijelim *HTML* dokumentom.

Koraci provedenog istraživanja su sljedeći:

1. Spajanje na *MongoDB* bazu i iteriracija po kolekciji *crawled_data_urls_v0* koja sadrži spremljene *URL* poveznice. Ta kolekcija sadrži polje u kojoj su zapisane svi spremljeni *hash*-evi za tu stranicu. Ako postoje dvije spremljene verzije, za usporedbu se dohvaća najnovija i druga najnovija verzija *web* stranice, odnosno uzimaju se njihovi spremljeni *hash*-evi.

2. Za dohvaćeni par *hash*-eva dohvaćaju se njihovi odgovarajući *HTML* dokumenti koji su spremljeni u kolekciji *crawled_data_pages_v0*.

3. Dva dohvaćena *HTML* dokumenta se pomoću *Python* biblioteke *Beautifulsoup* 4 [7] pretvaraju u oblik iz kojeg je moguće vaditi *HTML* oznake. Pretvoreni oblik parsira se tako da se stvori oblik u kojem su sadržane samo oznake bez vrijednosti i atributa. Ako postoje razlike za dva tako parsirana dokumenta, novo dohvaćene strukture se spremaju.

4. Nakon što su izvučene novo dohvaćene strukture, one prolaze kroz algoritam za detekciju malicioznog koda. Rezultat se sprema u odvojenu datoteku, a zapisuju se *HTML* tagovi u kojima je pronađena zloćudnost i zapisuju se sve *URL* poveznice pronađeni na stranicama. Kao posljedica rezultata, stvara se i *HTML* grafički prikaz usporedbe iz prethodnog koraka, radi lakšeg pronalaska ubačene maliciozne strukture u originalnom *HTML* dokumentu.

5. Ako je ubačena struktura pronađena i u njoj je detektiran zloćudni kod, stranica se ručno pregledava i daje joj se ocjena malicioznosti.

U drugom testiranju, koje se provodi nad svim stranicama, izbacuje se treći korak usporedbe. Dohvaćena najnovija verzija *HTML* dokumenta šalje se u algoritam opisan u četvrtom koraku. Konačni rezultati su jednaki prvom testiranju osim što se kod drugog testiranja izbacuje grafički prikaz uspordbe.

3.1. Prednosti i nedostaci metode korištene u eksperimentu

Prednost metode provedene u ovom eksperimentu je automatizacija velikog dijela postupka koji se provodi skriptom. Parsiranje, uspoređivanje i ocjena malicioznosti isječka

HTML dokumenta provodi se automatizirano te značajno vremenski smanjuje kompleksnost ručne provjere i pronalaska struktura u originalnom *HTML* dokumentu. Nedostatak ove metode jest konačna ručna provjera maliciozno detektiranih struktura, koja nad velikim brojem podataka postaje vremenski zahtjevna. Zbog pristupa kojim se pronalaze maliciozne strukture dolazi do stvaranje velikog broja lažno pozitivnih rezultata. Dodatno ograničenje metode je to što se maliciozne strukture traže na temelju njihove skrivenosti na stranici.

Zloćudno ubačene strukture ne moraju biti skrivene na web stranici te će tako biti vrlo brzo detektirani od strane korisnika i održavatelja stranice. Ako se takva stranica spremi u bazu podataka u vremenu gdje održavatelj stranice još nije uklonio ubačenu strukturu, skripta koja će provesti detektiranje neće uhvatiti takvu strukturu. Takav nedostatak skripte može se poboljšati uvođenjem pretraživanja *URL* poveznica koje se nalaze u *HTML* dokumentu *web* stranice i na temelju tih poveznica provjeravati postoje li one u poznatoj bazi pronađenih zloćudnih poveznica.

U završnom radu detaljno je opisan način pronalaska ubačene maliciozne *HTML* strukture te je definiran pojam i način rada sigurnosnog indeksa. Korištenjem *CSS*-a postoji mnogo kombinacija koje nisu istražene, a mogu se koristiti u maliciozne svrhe. Poboljšanje metode može se provesti dodatnim istraživanjem kombinacija *CSS*-a te dodatna kalibracija sigurnosnog indeksa. Kalibracija sigurnosnog indeksa opisana je u poglavlju 2.1 , ali i dalje ostaje nedovoljno istražena.

3.2. Rezultati provedenog eksperimenta

Testiranje je provedeno nad podacima iz *MongoDB* baze podataka. Prvi testiranje provedeno je nad rezultatima usporedbe dvije verzije stranice, dok je drugo provedeno nad cijelim stranicama.

Rezultati prikazani u Tablica 3.1 dobiveni su testiranjem 120000 *URL* poveznica, od kojih je uspješno dohvaćeno 3862 parova *HTML* dokumenata koji su u bazi podataka imali dvije spremljene verzije stranice koje imaju različiti *hash*. U testnim podacima pronađeno je tri maliciozne strukture za koje se ručnom provjerom uspostavilo da su stvarno maliciozne.

Rezultati testiranja nad cijelim HTML dokumentima prikazano je u

Tablica 3.2, a detektirano 285 malicioznih *HTML* struktura, a ručnom provjerom je pronađeno petnaest stvarno malicioznih *HTML* struktura.

Eksperimentom je dokazano da se kibernetički napadi provode i na web stranicama iz *.hr* domene. Dolazi se do zaključka da *web* stranice koriste nesigurne prakse prilikom izrade te su podložne *XSS* napadima. Metoda se pokazala kao uspješna u detekciji promjena i u detekciji zloćudnog koda. Od 285 rezultata pronađenih algoritmom, ručnim je pregledom potvrđeno 15 (5.26%) stvarno zloćudnih *HTML* struktura. Uspoređivanjem dobivenih rezultata, dolazi se do zaključka da algoritam dobro filtrira podatke jer je broj lažno pozotivnih rezultata i dalje značajno manji od broja testiranih stranica. Glavni nedostatak eksperimenta je ručni pregled koji nad većim brojem podataka postaje vremenski zahtjevan.

Tablica 3.1 Rezultati postupka koji uspoređuje dvije verzije *HTML* dokumenta

Opis uvjeta	Vrijednosti
Ukupan broj *URL* poveznica za provjeru	120000
Broj testiranih *HTML* dokumenata u kojima su detektirane promjene	3862
Broj detektiranih malicioznih *HTML* struktura	25
Broj lažno pozitivnih malicioznih *HTML* struktura	22
Broj stvarnih malicizonih *HTML* struktura	3

Tablica 3.2 Rezultati postupka koji provodi testiranje nad cijelim *HTML* dokumentom

Opis uvjeta	Vrijednosti
Ukupan broj *URL* poveznica za provjeru	90000
Broj testiranih *HTML* dokumenata	54329
Broj detektiranih malicioznih *HTML* struktura	285
Broj lažno pozitivnih malicioznih *HTML* struktura	270
Broj stvarnih malicizonih *HTML* struktura	15

4. Načini poboljšanja

Algoritam ima svoje prednosti i mane, ali postoje načini njegovog poboljšanja. Algoritam se može poboljšati ako se dodatno istraže načini skrivanja i kombinacije *CSS*-a koji nisu korišteni u klasičnoj izradi *web* stranice. Algoritam se može dodatno poboljšati tako da se uz provjere novo ubačenih struktura doda i baza podataka zloćudno ubačenih *URL* poveznica koje su već prije pronađeni na stranicama. Na taj način bi se mogle detektirati i web stranice u kojima ubačene *HTML* strukture ne sadrže kombinacije *CSS*-a za skrivanje. Potrebno je dodatno istražiti priloženi kod izvornog algoritama u Listing 6.1 te osmisliti način na koji bi se sigurnosni indeks mogao poboljšati. Testiranjem više stranica uz kalibraciju sigurnosnog indeks doći će do bolje procjene granice sigurnosnog indeksa za koji bi neka struktura trebala biti ocjenjena kao maliciozna. Algoritam provjerava samo *HTML* oznake koje mogu sadržavati *URL* poveznice pa bi moguće poboljšanje bilo testirati sve oznake u *HTML* dokumentu.

5. Zaključak

U ovom radu istražena je metoda detekcije promjene *web* stranice temeljem *HTML* strukture i detekcija zloćudno ubačenog koda. Osmišljena je metoda koja pronalazi novo ubačene *HTML* strukture i metoda koja pomoću atributa i vrijednosti *HTML* oznaka detektira maliciozno ubačene strukture. Metoda uspoređuje dva *HTML* dokumenta, to jest dvije spremljene verzije *web* stranice preuzete iz baze podataka. Dokumenti u kojima je pronađena promjena, odnosno ubačene su nove *HTML* strukture, prolaze kroz osmišljeni algoritam koji uz pomoć parametra sigurnosnog indeksa i zbroja kombinacija *CSS* stiliziranja daje ocjenu malicioznosti. Sigurnosni indeks je parametar koji ima granicu četiri, a prelaskom ukupnog zbroja te granice *HTML* struktura ocjenjuje se kao maliciozna. Ovim metodama dobivene su strukture koje su ocijenjene kao maliciozne, a ručnom analizom je ocjena potvrđena. Dobiveni rezultati dovode do zaključka da se metoda dokazala kao uspješna te je detekcija promjena i detekcija malicioznog koda na *web* stranicama pomoću *HTML* strukture ostvariva.

Za pospješenje korištenih metoda dani su prijedlozi u poglavlju 4., te su dani i prijedlozi kako poboljšati algoritam detekcije malicioznog koda. Poboljšanim algoritmom može se ostvariti bolja detekcija i smanjeni broj lažno pozitivnih rezultata. Opisane su prednosti i nedostaci metoda koje se daljnjim istraživanjem mogu pospješiti.

LITERATURA

[1] Tennant, Roy. *Internet basics. ERIC Clearinghouse on Information and Technology*, 1992. Poveznica: https://edu.gcfglobal.org/en/internetbasics/what-is-the-internet/1/; pristupljeno, 1. lipnja 2022.

[2] Astari S. *What Is HTML? Hypertext Markup Language Basics Explained*, (2022.)

Poveznica: https://www.hostinger.com/tutorials/what-is-html/; pristupljeno, 1. lipnja 2022.

[3] Network, M. D. *What is JavaScript?*, (2021.) Poveznica: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps.What_is_JavaScript/; pristupljeno, 1. lipnja 2022.

[4] Gupta, S., & Sharma L. *Exploitation of cross-site scripting (XSS) vulnerability on real world web applications and its defense*, 2012.

[5] MDN contributors, *CSS: Cascading Style Sheets, (*2022.)

Poveznica: https://developer.mozilla.org/en-US/docs/Web/CSS; pristupljeno, 1. lipnja 2022.

[6] Python 3.10.4 documentation, Jun 2022.

Poveznica: https://docs.python.org/3/; pristupljeno, 5. lipnja 2022.

[7] Leonard Richardson, *Beautiful soup documentation* , (2019.) https://www.crummy.com/software/BeautifulSoup/bs4/doc/; pristupljeno, 7. lipnja 2022.

[8] Martijn F. *Parsing XML and HTML with lxml*, (2022, lipanj).

Poveznica: https://lxml.de/credits.html/;pristupljeno, 7. lipnja 2022.

[9] Julien P. difflib — Helpers for computing deltas, (2022, lipanj). Poveznica: https://docs.python.org/3/library/difflib.html/; pristupljeno, 7. lipnja 2022.

[10] *What is a web crawler? | How web spiders work*, (2022, lipanj). Poveznica: `https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/;` pristupljeno, 7. lipnja 2022.

[11] Pymongo 4.1.1 documentation, (2022, lipanj). Poveznica: `https://pymongo.readthedocs.io/en/stable/;` pristupljeno, 7. lipnja 2022.

[12] Mongodb documentation, (2022, lipanj). Poveznica: `https://docs.mongodb.com/;` pristupljeno, 7. lipnja 2022.

DETEKCIJA PROMJENE WEB-STRANICE TEMELJEM ANALIZE NJENE STRUKTURE

Sažetak

Ovaj završni rad je kao glavni cilj imao prepoznavanje promjene *web* stranica pomoću *HTML* strukture. Istražena je i detekcija malicioznog koda filtriranjem *HTML* oznaka i njihovih pripadajućih atributa. Opisana je metoda detekcije promjena na *web* stranici i način na koji se iz strukture vade novo ubačene *HTML* strukture. Metoda je ostvarena pomoću gotove programske potpore za parsiranje *HTML* dokumenata. Pronađena ubačena struktura testira se algoritmom za pronalazak maliciozne strukture. Algoritmom za detekciju malicioznosti, provodi se testiranje tako da se pronađenim *HTML* oznakama vrate atributi te se gledaju vrijednosti tih atributa. Provjerom vrijednosti atributa traže se kombinacije *CSS*-a kojima se strukture mogu sakriti iz vidljivog područja *web* preglednika. Kombinacije se zbrajaju u parametar sigurnosni indeks, koji nam govori koliko je uvjeta za zloćudnost ostvareno. Algoritam ima dvije verzije te se u jednoj verziji predaje najnovija verzije spremljenog *HTML* dokumenta i provjerava se cijeli dokument. U drugoj se verziji predaju dva *HTML* dokumenta, najnovija i druga najnovija spremljena verzija, a rezultat algoritma je ocjena malicioznosti te isječak *HTML* strukture u kojoj algoritam pronađe opasne *CSS* kombinacije.

Korištena metoda i algoritmi pokazali su se kao uspješni jer je detektiran određen broj malicioznih stranica za koje su ručno potvrđene kao maliciozne. Predložen je način poboljšanja algoritma koji uključuje podešavanje sigurnosnog indeksa i dodatno istraživanje kombinacija *CSS*-a koje se koristi za skrivanje ubačenih *HTML* struktura.

KLJUČNE RIJEČI: *HTML*, struktura, malicioznost, *web* , sigurnost

Abastract

The main goal of this final paper was to identify changes to web pages using HTML structure. As part of the main goal, the detection of malicious code by filtering HTML tags and their associated attributes was also researched. The method of detecting changes on a web page and the way in which newly inserted HTML structures are removed from the structure are described. The method was implemented using existing software support for parsing HTML documents. The found inserted structure is tested by an algorithm for finding a malicious structure. With the malware detection algorithm, testing is performed in such a way that the found HTML tags return the attributes and look at the values of these attributes. Checking attribute values looks for CSS combinations that can be used to hide structures from the screen. Combinations are added to the security index parameter, which tells us how much the condition for maliciousness has been met. The algorithm has two versions, and in one version the latest version of the equipped HTML document is submitted, and the entire document is checked. In the second version, two HTML documents are submitted, the latest and the second latest version of the website, and the result of the algorithm is an assessment of maliciousness and a part of the HTML structure in which the algorithm finds dangerous CSS combinations.

The method and algorithms used proved to be successful because several malicious pages were detected for which they were manually confirmed as malicious. A way to improve the algorithm is proposed, which includes setting the security index and further researching the CSS combinations used to hide inserted HTML structures.

KEYWORDS: *HTML*, *structure*, *malicious*, *web*, *security*

6. Privitak: izvorni kod algoritma

Privitak izvornog koda opisan je u poglavlju 2.1 i on služi za pronalazak maliciozno ubačene *HTML* strukture pomoću *CSS* kombinacija.

Listing 6.1 Izvorni kod algoritma

```
urls_on_page = set({})
security_index = 0
malicious_code = set()
c_sec1 = True
c_sec2 = True
c_sec3 = True
c_sec4 = True
c_sec5 = True
c_sec6 = True
c_sec7 = True
parsed = getParsedPageDBS(firstpage)
for tag in parsed.findAll({}):
    if(len(tag.attrs) > 0 ):
        for attribute_name in tag.attrs:
            inline_security_index = 0
            url_regex_lista = ["www.",".com",".org","https://","http://"]
            if any(x in tag.attrs[attribute_name] for x in url_regex_lista):
                if(tag.attrs[attribute_name] != "http://www.w3.org/1999/xhtml"
                                        and "@" not in tag.attrs[attribute_name]):
                    if(tag.name != "html"):
                        urls_on_page.add(tag.attrs[attribute_name])

            if(attribute_name == "hidden"):
                if(c_sec1):
                    security_index +=1
                    c_sec1 = False

            if(attribute_name == "style"):

                if(tag.name == "div" or tag.name == "p" or tag.name=="a"):

                    lista_css_atributa = tag.attrs[attribute_name].split(";")

                    for css_name_value in lista_css_atributa:

                        if(css_name_value != "" and ":" in css_name_value):

```

```
                                a = css_name_value.split(":")
                                if a[0].strip() in bad_css:
                                    if(a[0].strip() == "z-index"):
                                        if(int(a[1]) < 10):
                                            if(c_sec2):
                                                security_index +=1
                                                c_sec2 = False
                                            inline_security_index +=1

                                    elif a[0].strip() in bad_css_positioning:

                                        val = int(a[1].replace("px","")
                                                      .replace("vh","")
                                                      .replace("vw","")
                                                      .replace("rem","")
                                                      .replace("%",""))
                                        if( val < -500 or val > 2500):
                                            if(c_sec3):
                                                security_index +=2
                                                c_sec3 = False

                                            inline_security_index +=2
                                        elif (val == 0):
                                            if(c_sec4):
                                                security_index +=1
                                                c_sec4 = False
                                            inline_security_index +=1
                                    elif(a[0].strip() == "filter"):
                                        if(a[1] == "alpha(opacity=0)"):
                                            if(c_sec5):
                                                security_index +=1
                                                c_sec5 =False
                                            inline_security_index +=1
                                    elif(a[0].strip() == "opacity"):
                                        if(float(a[1]) < 1):
                                            if(c_sec6):
                                                security_index +=1
                                                c_sec6 =False
                                            inline_security_index +=1
                                    elif(a[0].strip() == "letter-spacing"):
                                        if(a[1].strip() == "0px"):
                                            if(c_sec7):
                                                security_index +=1
                                                c_sec7 = False
                                            inline_security_index +=1
            if(inline_security_index >= 2):
                malicious_code.add(tag)

if(len(urls_on_page) != 0):
```

```
87.         security_index += 2
88. if(security_index >= 4):
89.         //write saved data to file
```