Aplikacija za demonstraciju *XSS* sigurnosnih propusta

Katarina Simić, Žarko Stanisavljević

***Apstrakt* — *XSS* (eng. *Cross-site scripting*) je jedna od najčešćih ranjivosti veb aplikacija uprkos tome što postoji veliki broj različitih mehanizama zaštite. U ovom radu prikazana je implementacija jedne ranjive aplikacije u okviru koje je moguće demonstrirati različite tipove XSS sigurnosnih propusta, kao i načina njihove zloupotrebe, ali i eliminisanja. Aplikacija se može koristiti kao edukativno sredstvo za praktičnu obuku softverskih inženjera u zatvorenom i bezbednom okruženju.**

***Ključne reči* — XSS, OWASP top 10, sigurnosni propusti.**

I. Uvod

Važnost veb aplikacija posebno je došla do izražaja u trenutku pandemije koronavirusa koja je počela 2019. godine i još uvek traje. Od tada ljudi sve više završavaju svoje poslove i obavljaju određene aktivnosti, poput online kupovine ili korišćenja društvenih mreža, uz pomoć veb aplikacija. Na ovaj način korisnici na različitim mestima ostavljaju svoje poverljive podatke, verujući veb aplikacijama da oni neće pasti u pogrešne ruke. Iz ovog razloga je veoma bitno da svaka aplikacija u svakom trenutku bude zaštićena od različitih tipova napada i pokušaja krađe osetljivih podataka, i na taj način zadobije i zadrži poverenje svojih korisnika.

Važan deo bezbednosti veb aplikacija jeste koncept polise zajedničkog porekla (eng. *same-origin policy*, *SOP*) [1]. Zahvaljujući ovom mehanizmu, skripte jedne veb stranice mogu da pristupe podacima druge veb stranice samo ako su istog porekla. Dva *URL*-a su istog porekla ako su im protokol, host i port identični. Na ovaj način sprečeno je da napadači preko svojih zlonamernih veb aplikacija dođu u posed osetljivih podataka smeštenih na nekom drugom veb sajtu. Zbog toga su napadači morali da osmisle nove načine kako mogu doći do korisničkih podataka, a da pritom zaobiđu polisu zajedničkog porekla.

XSS [2][3] je jedan od bezbednosnih propusta koji zaobilazi polisu zajedničkog porekla. Jedan je od retkih napada koji se iznova nalaze na *OWASP*-ovoj godišnjoj listi top 10 bezbednosnih propusta [4] i gotovo da ne postoji veliki veb sajt koji u nekom trenutku nije bio ranjiv na ovaj napad. *XSS* podrazumeva umetanje klijentskih skripti u ranjivu aplikaciju, koje su kasnije dostupne korisniku nakon učitavanja određenih veb stranica te aplikacije. *XSS* bezbednosni propust je i dalje popularan i zastupljen na velikom broju veb aplikacija. Razlog tome često može biti neiskustvo, neupućenost i neobazrivost programera koji izrađuju veb aplikacije, kao i nedostatak testiranja aplikacija na propuste prilikom svake velike izmene ili nadogradnje aplikacije.

U ovom radu prikazana je implementacija i način korišćenja ranjive veb aplikacije, na kojoj je moguće na određenim mestima umetnuti zlonamerne skripte i izvršiti neku od zlonamernih akcija na štetu regularnog korisnika. Cilj ove aplikacije je jednostavna demonstracija nekih od najčešće primenjenih i praktičnih *XSS* napada, koja bi na taj način pomogla korisniku da bolje razume kada i kako ti napadi mogu da se dese, kao i na kojim mestima u aplikaciji. Nakon korišćenja aplikacije, korisnik bi trebao da razume osnovne koncepte *XSS* napada, kao i da bude u stanju da primeni odgovarajuće mere zaštite, koristeći naučeno, prilikom izrade sopstvene aplikacije.

U drugom poglavlju se opisuju detalji *XSS* sigurnosnog propusta. U trećem poglavlju je prikazan razvoj aplikacije koji se koristi u demonstrativne svrhe *XSS* napada, uz detaljan opis korišćenih tehnologija. U četvrtom poglavlju je opisan rad aplikacije i način korišćenja aplikacije. U petom poglavlju je dat zaključak.

II. *XSS*

Pojavom *JavaScript* programskog jezika sredinom devedesetih godina prošlog veka omogućen je veliki napredak u izradi veb aplikacija, koje su sada mogle biti i interaktivne. Ali, pored svih dobrih i interesantnih mogućnosti koje su sada bile dostupne, pojavile su se i one loše koje mogu uticati negativno po korisnika, poput *XSS* napada. Prvobitno se *XSS* napadu nije pridavalo mnogo pažnje, jer su serveri bili izazovnija i interesantnija meta napadačima. Ali tokom godina situacija se preokrenula. Serveri su vremenom postajali mnogo zaštićeniji nego ranije i bilo je sve teže probiti njihovu zaštitu. Uvidelo se i da serveri nisu bili neophodni za izvršavanje napada sa klijentske strane. Pojavljivali su se različiti pretraživači koji izvršavaju klijentski kod, svaki sa svojim propustima u zavisnosti od verzije, što je programerima dodatno otežavalo posao zaštite. Sa druge strane, programeri zbog manjka vremena ili budžeta, kao i manjka iskustva i znanja, ne posvećuju dovoljno pažnje bezbednosnim propustima, te ih je veoma lako i napraviti. Zbog svega ovoga se *XSS* danas smatra za jedan od najopasnijih i najučestalijih napada. Dve trećine svih veb aplikacija imaju *XSS* propuste u sebi, i svaka velika i popularna aplikacija je u nekom trenutku imala ovaj propust.

Primarni cilj napadača jesu korisnici ranjivih aplikacija. Napadi najčešće podrazumevaju krađu sesije, preuzimanje osetljivih podataka, izvršavanje nedozvoljenih akcija u ime korisnika, dostavljanje zlonamernih softvera korisniku (eng. *malware*), pa čak i narušavanje zaštite aplikacije od drugih napada. Osim što ovi napadi oštećuju same korisnike, mogu

Katarina Simić je student master studija na Elektrotehničkom fakultetu, Univerziteta u Beogradu, Bul. kralja Aleksandra 73, 11120 Beograd, Srbija (e-mail: sk193473m@etf.bg.ac.rs).

Žarko Stanisavljević radi na Elektrotehničkom fakultetu, Univerziteta u Beogradu, Bul. kralja Aleksandra 73, 11120 Beograd, Srbija (telefon: +381-11-3218-484; e-mail: zarko.stanisavljevic@etf.bg.ac.rs).

da imaju i destruktivne posledice po samu aplikaciju, ponajviše zbog gubitka poverenja od strane korisnika ako aplikacija zahteva visok nivo bezbednosti zbog veoma osetljivih i bitnih korisnikovih podataka podeljenih sa tom aplikacijom.

Postoji nekoliko varijacija *XSS* napada, koje se mogu podeliti u tri glavna tipa: reflektujući, snimljeni i *DOM* bazirani *XSS* napad.

A. Reflektujući XSS napad

Reflektujući *XSS* napad (eng. *Reflected XSS attack*) [5] je najzastupljeniji od sva tri tipa. Pod ovim napadom se podrazumeva da se umetnuta, zlonamerna skripta pošalje na server kao deo zahteva i zatim odmah reflektuje u korisnikovom pretraživaču u vidu odgovora koji sadrži tu skriptu. Dakle, podatak poslat serveru je vraćen i prikazan na stranici bez ikakve prethodne provere tog podatka. Da bi uspešno sproveo ovaj napad, napadač prvo mora da osmisli *URL* koji će sadržati zlonamernu skriptu. Zatim će taj *URL* proslediti korisnicima na lukav način. Ako neko od korisnika ništa ne posumnja, zahtevaće *URL* od aplikacije i server će vratiti odgovor koji će sadržati i napadačevu skriptu. Korisnikov pretraživač će sada, između ostalog, izvršiti i napadačevu skriptu i ukradeni podaci se šalju na napadačev server i postaju mu dostupni. Ovakva vrsta napada se izvršava samo ako korisnik otvori napadačev *URL*, te je tako ovaj napad često jednokratan. Da bi napadač naveo korisnika da nasedne i otvori sastavljen *URL*, mora da se posluži lukavim trikovima. Ako su mu meta individualne osobe, napadač će im zamaskirani *URL* upotpunjen uverljivom porukom poslati direkno (npr. preko *e-mail* poruke), tako da korisnik poželi da taj *URL* i otvori. Ako mu je meta veći broj ljudi, zamaskirani *URL* će postaviti na nekim drugim veb stranicama u vidu linka, te će čekati da neko taj link i otvori.

B. Snimljeni XSS napad

Snimljeni XSS napad (eng. *Stored XSS attack*) [6] je najopasniji tip *XSS* napada, zato što može da ima značajnije posledice po veći broj korisnika. Za razliku od reflektujućeg napada, zlonamerna skripta se čuva na serveru, tako da će se svakom korisniku, koji od servera bude zahtevao stranicu sa umetnutom skriptom, u pretraživaču ta skripta i izvršiti. Ovi napadi su najčešći na sajtovima gde korisnici imaju vid međusobne komunikacije (forumi, komentari, pitanja korisnika, itd.). Za izvršavanje ovog napada napadač ne mora da podmeće korisnicima direktno sastavljen *URL*, već je dovoljno da sam umetne skriptu u aplikaciji. Pošto će ona biti sačuvana na serveru, biće dostupna svakom korisniku te stranice. Ovaj napad će najverovatnije dati bolje rezultate u odnosu na reflektujući, jer da bi napadač ukrao korisnikove osetljive podatke, korisnik u najvećem broju slučajeva mora biti ulogovan, što će verovatno i biti slučaj prilikom izvršavanja snimljenog *XSS* napada, a manje verovatan slučaj prilikom reflektujućeg. Snimljenim napadom je veća i verovatnoća da napadačeva žrtva bude administrator napadnute veb aplikacije, što znači da cela aplikacija može biti komprimitovana i ugrožena.

C. DOM bazirani XSS napad

Za razliku od reflektujućeg i snimljenog *XSS* napada, za čije je izvršavanje neophodno vraćanje zlonamerne skripte sa servera, *DOM* (eng. *Document Object Model*) bazirani *XSS* napad [7] će se izvršiti bez da server vrati skriptu prosleđenu u *URL*-u. Ovo je moguće zato što *JavaScript* može pristupiti *DOM*-u i samim tim može dohvatiti i parametre *URL*-a. To znači da će zlonamerni kod biti preuzet sa *URL*-a i obrađen u *JavaScript* kodu. Ovakva vrsta napada ima više sličnosti sa reflektujućim *XSS* napadom nego snimljenim, jer zahteva od napadača da sastavljen *URL* na razne načine podmetne korisnicima, ali je zbog svoje prirode znatno opasniji. Razlika u odnosu na reflektujući *XSS* napad jeste što će *JavaScript* kod procesirati napadačev *URL*, pa samim tim i napadačevu skriptu umetnutu u *URL*, tako da bilo šta što će server vratiti kao odgovor nije važno prilikom ovog napada. Najveći problem kod ovog propusta jeste naći uzrok zbog kojeg nastaje, a pošto se taj uzrok može naći bilo gde u klijentskom kodu, programer bi morao dobro da poznaje projekat prilikom istrage.

D. Zaštita od XSS napada

Nakon upoznavanja sa XSS propustima i uviđanja njihovih mogućnosti i posledica po korisnike, naredni korak za programere bi bio da svoje aplikacije od istih i zaštite. S obzirom da je malo potrebno da se propusti naprave, neophodno je redovno testirati aplikacije na njih prilikom svake nadogradnje koda. Radi efikasnije zaštite aplikacija preporučljivo je primeniti metode poput validacije *input*-a [8], validacije *output*-a [9], konfiguracije aplikacije tako da vraća *Content-Security-Policy* zaglavlje [10], zabrane unosa korisničkih podataka na potencijalno opasnim mestima u okviru aplikacije [11], kao i korišćenja odgovarajućih zaglavlja odgovora koji bi mogli da detektuju *HTML* ili *JavaScript* kod u *HTTP* odgovorima, i samim tim spreče njihovo ubrizgavanje, poput *X-XSS-Protection* zaglavlja [12].

III. Implementacija aplikacije

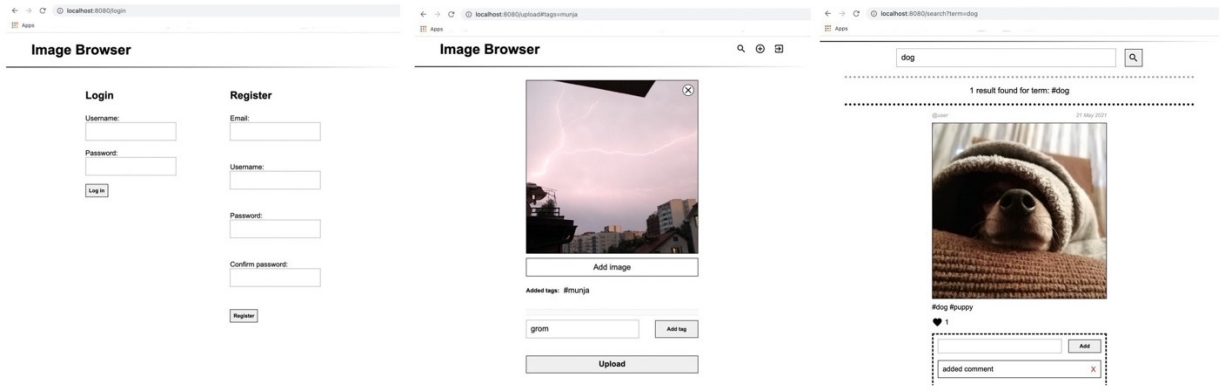
Implementirana aplikacija se sastoji iz dva dela. Prvi deo čini jednostavna, ali ranjiva veb aplikacija, koja služi za isprobavanje različitih *XSS* napada na različitim mestima u okviru te aplikacije. Drugi deo čini napadačev server, na koji pristižu ukradeni podaci nakon uspešno izvršenih napada.

A. Implementacija ranjive aplikacije

Prvi deo alata za učenje predstavlja jednostavnu veb aplikaciju za pretraživanje i dodavanje slika pod nazivom *ImageBrowser* (Sl. 1).

Na početku se od korisnika traži da se registruje ili uloguje na aplikaciju. Nakon što se korisnik uloguje, prikazuje mu se stranica sa porukom dobrodošlice. Nakon toga korisniku su dostupne dve različite stranice sa akcijama. Na prvoj stranici korisnik može da dodaje svoje slike uz koje ostavlja i određene tagove – reči koje služe za opisivanje slike. Na drugoj stranici korisnik može da, uz pomoć *input* polja, pretražuje slike na osnovu postojećih tagova, da pretražuje druge korisnike aplikacije dodajući simbol @ ispred korisničkog imena, kao i da prikazane slike komentariše i na njih reaguje.

Sama aplikacija ima minimalan skup potrebnih funkcionalnosti, ali i namerno napravljene *XSS* bezbednosne propuste na više mesta radi demonstracije nekih od *XSS* napada.

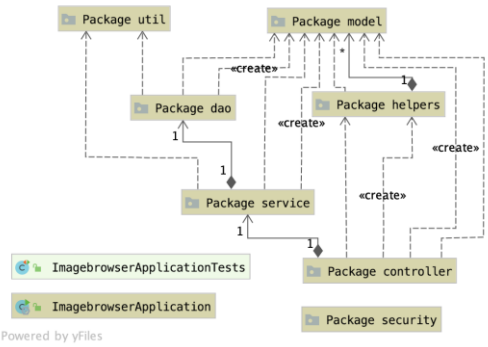


a) b) c)

Sl. 1. Izgled ranjive aplikacije u pretraživaču: a) početna stranica, b) stranica za dodavanje slika i c) stranica za pretraživanje slika

Da bi korisnik mogao da dodaje i pretražuje slike, kao i da ostavlja komentare i reakcije, određeni podaci moraju da se čuvaju u bazi podataka. Aplikacija koristi *H2 Java in-memory* bazu [13], koja omogućava da se prilikom svakog pokretanja aplikacije koristi inicijalno stanje podataka baze i da se sve dotadašnje izmene gube, što omogućava lakše testiranje aplikacije.

Za implementaciju servera je korišćen razvojni okvir *Spring* [14], kao i *SpringBoot* [15] koji koristi *Spring* kao podlogu. U projektu se koristi i *Apache Maven* [16], alat za izvršavanje `compile` i `build` naredbi *Java* koda. *Spring Initializer* [17] je projekat otvorenog koda (eng. *open source*), koji omogućava generisanje konfigurisanog *Spring* projekta uz odabir potrebnih zavisnosti (eng. *dependencies*). *Java* kod je grupisan po paketima, od kojih je okružujući *application.imagebrowser* u okviru koga se pored drugih nalazi i *ImagebrowserApplication,* glavna klasa aplikacije (Sl. 2).



Sl. 2. *UML* dijagram paketa unutar *application.imagebrowser* okružujućeg paketa

Unutar ovog paketa se nalazi nekoliko drugih paketa, koji sadrže sav potreban kod za uspostavljanje komunikacije i ispravan rad servera sa bazom podataka, obradu pristiglih zahteva na serveru i vraćanje odgovarajućih stranica i podataka koji se učitavaju na stranici sa kojima će korisnik interagovati.

Klijentska strana sadrži kod koji se izvršava u pretraživaču, i sadrži sve što korisnik može da vidi i sa čime može da interaguje. Najbitnije i najosnovnije tehnologije koje su korišćene prilikom izrade klijentskog dela aplikacije su *HTML* (eng. *Hyper Text Markup Language*), *CSS* (eng. *Cascading Style Sheets*) i *JavaScript*. U okviru posmatrane aplikacije se koriste *JSP* (eng. *Java Server Pages*) stranice [18] koje podržavaju dinamički sadržaj, što omogućava umetanje *Java* koda unutar *HTML* koda uz pomoć specijalnih *JSP* tagova. *JSP* komponenta predstavlja *servlet* koji ispunjava ulogu korisničkog interfejsa za *Java* veb aplikaciju. Radi olakšanog i preglednijeg fajla, uz *JSP* se koristi i biblioteka *JSTL* (eng. *The JSP Standard Template Library*) [19], koja omogućava da se *Java* kod zameni tagovima koji će raditi identičan posao. Fragmenti *JSP* koda koji se mogu koristiti na više mesta su smešteni u zasebne fajlove, koji se uz pomoć tagova učitavaju na određenoj *JSP* stranici. *JSP* fajlovi u aplikaciji koji čine stranice su:

- *login.jsp,* gde korisnik može da se uloguje ili registruje,
- *index.jsp*, koji predstavlja glavnu stranicu prikazanu korisniku nakon što se uloguje,
- *upload.jsp*, gde korisnik može da dodaje slike i tagove,
- *search.jsp*, gde korisnik može da pretražuje slike po tagovima ili drugim korisnicima, kao i
- *searchResults.jsp* i *noResults.jsp*, dve stranice koje server vraća kao rezultat *AJAX* [20] poziva, prva sa rezultatima, i druga sa informacijom da rezultata nema.

B. Implementacija napadačevog servera

Drugi deo aplikacije čini napadačev server, na kojem će se obrađivati pristigli zahtevi, tačnije prethodno dohvaćeni osetljivi podaci korisnika. Da bi zahtevi uopšte mogli da pristignu na server, napadač mora da sastavi *URL* koji će u sebi sadržati zlonamerni kod za dohvatanje podataka i za redirekciju. Nakon dohvatanja podataka, napadač može da ponovo izvrši redirekciju nazad ka napadnutoj aplikaciji tako da korisnik ni ne posumnja da je bio napadnut.

U okviru ove aplikacije je napadačev server implementiran u *Node.js* [21] platformi koja predstavlja asinhrono *runtime* okruženje za *JavaScript* jezik, i koja omogućava stvaranje skalabilnih veb aplikacija, samim tim i izvršavanje *JavaScript* koda van pretraživača.

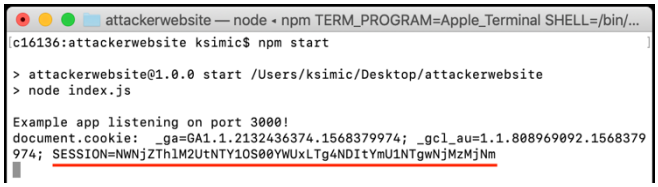
U slučaju napadačevog projekta je instaliran paket *Express* [22], koji predstavlja radni okvir za organizaciju aplikacije prema *MVC* arhitekturi. Pomoću *Express*-a se mogu na jednostavan način obrađivati pristigli zahtevi. Napravljen je jedan *JavaScript* fajl, *index.js*, u kojem se, prilikom izvršavanja koda, pokreće server koji osluškivanjem čeka na zahteve i obrađuje one koji su pristigli. Svaki zahtev se obrađuje tako da se u terminalu napadača gde je pokrenuta skripta ispišu pristigli podaci, a zatim po potrebi izvrši i redirekcija. Dokle god napadačev server radi, moći će da obrađuje pristigle zahteve.

IV. Način korišćenja aplikacije

U ovom poglavlju je dat prikaz nekoliko tipičnih scenarija *XSS* napada, gde je prvo objašnjen cilj napada, uz priložene zlonamerne skripte za ispunjenje tog cilja, a zatim je na kraju svakog primera dat savet za sprečavanje tog napada. Bitno je napomenuti da su ovakve vrste napada kažnjive zakonom svuda u svetu (npr. u Srbiji prema Krivičnom zakoniku Republike Srbije (Članovi 298 do 304a)) ukoliko se sprovode prema aplikacijama fizičkih i pravnih lica koja nisu upoznata i saglasna sa aktivnostima na proveri ranjivosti.

A. Krađa korisnikove sesije

Nakon što se korisnik uspešno uloguje na aplikaciju, server će poslati kolačić sesije preko *Set-Cookie* zaglavlja. Taj kolačić će se sada slati ka serveru uz svaki korisnikov zahtev. Zbog toga je kolačić sesije izuzetno osetljiv podatak, jer ako napadač nekako uspe da dođe do njegove vrednosti moći će da šalje zahteve ka serveru u ime oštećenog korisnika. *HttpOnly* predstavlja deo *Set-Cookie* zaglavlja u vidu *flag*-a. Ako je taj *flag* postavljen, to će sprečiti klijentske skripte da pristupe vrednostima kolačića. U slučaju da taj *flag* nije postavljen, krađu sesije je moguće izvesti. U slučaju ove aplikacije *flag* nije postavljen, tako da je moguće pristupiti vrednosti kolačića sesije u *JavaScript*-u preko *document.cookie* atributa (Sl. 3).



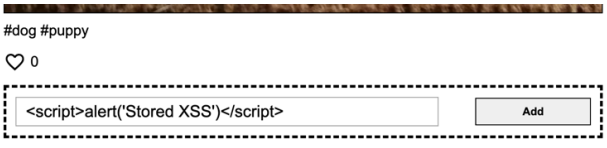
Sl. 3. Prikaz ukradenog kolačića sesije u terminalu napadača

1) Krađa kolačića sesije - reflektujući XSS napad

Na *search* stranici aplikacije postoji reflektujući *XSS* propust. Prilikom pretrage slika, uneti termin postaje *URL* parametar, i prilikom vraćanja rezultata od strane servera se vraća i pretražen termin koji se dodaje na stranicu. Validacija na tim osetljivim tačkama nije realizovana, samim tim korisnik umesto termina može da ukuca skriptu unutar `<script>` taga. Napadač takođe proverava na svojoj mašini da li *document.cookie* vraća njegovu tekuću sesiju. Nakon što utvrdi da vraća, napadač može da sastavi *URL* koji sadrži zlonamerni kod.

2) Krađa kolačića sesije - snimljeni XSS napad

U aplikaciji postoji i snimljeni *XSS* propust, tako da napadač može u komentare da ubacuje zlonamerni kod. Ako je situacija ista kao kod reflektujućeg *XSS* propusta, napadač sada može isti zlonamerni kod da doda kao komentar (Sl. 4). To znači da će svakom korisniku kojem se taj komentar bude učitao na stranici biti ukraden kolačić sesije. Ovo je suptilniji način za prevaru korisnika, te je veća verovatnoća da će korisnici biti prevareni ovom metodom nego da su kliknuli na *URL* primljen od napadača u slučaju reflektujućeg *XSS* napada.



Sl. 4. Trenutak dodavanja napadačeve skripte u komentar

3) Krađa kolačića sesije - DOM bazirani XSS napad

Na *upload* stranici aplikacije postoji *DOM* bazirani *XSS* propust. Korisnik može da dodaje tagove koji se pridodaju slici, i kako se neki tag doda on postaje deo *URL*-a u vidu `href` parametra. Problem je što se taj deo *URL*-a nikada ne šalje na server i ne obrađuje, ali se prilikom učitavanja stranice sa takvim *URL*-om tagovi automatski dodaju. To znači da se taj deo *URL*-a obradio negde u *JavaScript* kodu.

4) Način sprečavanja napada

Najlakši i najefikasniji način sprečavanja ovog napada jeste jednostavno podesiti *HttpOnly flag*, koji u tom slučaju sprečava klijentske skripte da dohvate podatke o kolačićima (Sl. 5). Na ovaj način *document.cookie* će uvek vratiti praznu vrednost i kolačić sesije će ostati bezbedan. Iako ovaj mehanizam odbrane od krađe kolačića funcioniše, to ne znači da napadač na istom mestu ne može da izvrši druge zlonamerne akcije. Ovaj *flag* se setuje na različite načine, u zavisnosti od korišćenog programskog jezika.

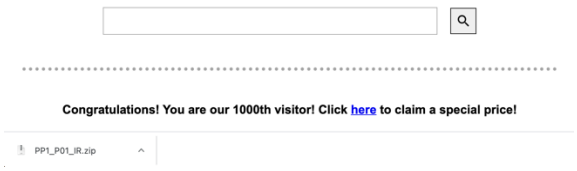
Sl. 5. Postavljanje *HTTPOnly flag*-a radi sprečavanja krađe kolačića sesije

B. Umetanje napadačevog koda na stranicu

U prethodnim primerima je data jednostavna skripta koja izvršava redirekciju i prosleđuje kolačić sesije napadaču, ali ako je primenjena navedena tehnika zaštite od tog napada, napadač mora da nađe drugi način da naškodi korisniku. Još jedna tehnika jeste umetanje *HTML* koda na stranicu, kojem se dodeljuju stilovi tako da izgleda kao da je zapravo deo stranice. Ako je kod dovoljno uverljiv, može da uveri korisnika da uradi određene akcije vođene tim kodom. U naredna dva primera se može videti kako umetanjem *HTML* koda napadač može da ukrade kredencijale korisnika, i kako može da navede korisnika da preuzme sumnjiv sadržaj na svoju mašinu.

1) Umetanje koda za preuzimanje sumnjivih fajlova

Ponekad napadaču nije samo cilj da ukrade korisnikove podatke, već i da ga navede da preuzme određeni fajl. To postiže umetanjem linka, na čiji klik se započinje preuzimanje nekog fajla. U zavisnosti od toga šta je cilj napadača, taj fajl može da ima nikakav ili razoran uticaj na korisnikovu mašinu. Dovoljno je samo sastaviti taj link dovoljno uverljivim da navede korisnika da klikne na njega, tako da će napadač i ovde uneti *inline* stilove, kao i uverljiv tekst (Sl. 6).

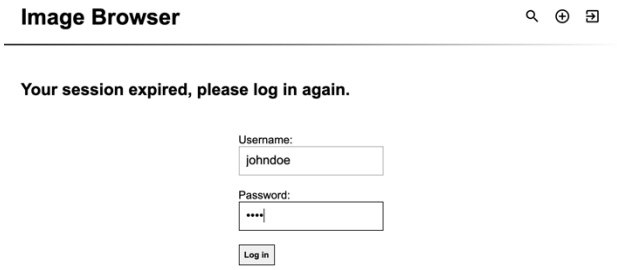


Sl. 6. Prikaz stranice za pretraživanje slike sa umetnutim linkom za preuzimanje sumnjivih fajlova

2) Phishing tehnika

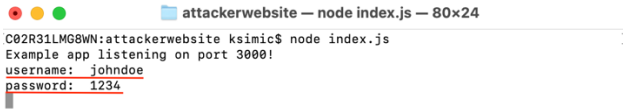
Iako kolačići nakon primenjene zaštite ne mogu biti ukradeni, *XSS* propust i dalje postoji na istim mestima. Napadač uviđa da može da umetne skriptu koja sa stranice

briše ceo *HTML* kod i zameni ga svojim. U ovom primeru je za brisanje i dodavanje koda korišćen *Jquery* (tačnije *.empty* i *.append* funkcije). Kod koji se dodaje predstavlja lažnu formu za unos korisničkog imena i šifre. Ta taktika umetanja ovakve vrste koda gde korisnik „dobrovoljno“ ostavlja lične podatke napadaču se zove *phishing*. Napadač dodaje tekst da uveri korisnika da treba tu formu da popuni, na primer saopšti korisniku da mu je istekla sesija i da mora ponovo da se uloguje (Sl. 7).



Sl. 7. Prikaz stranice za pretraživanje slike sa umetnutom formom

Ako korisnik ništa ne posumnja, može uneti svoje lične podatke. Klikom na dugme za logovanje se zapravo izvršava redirekcija ka napadačevom serveru i štampaju se podaci u napadačevu konzolu i na kraju ponovo dešava redirekcija nazad ka aplikaciji. Napadač sad ima korisnikovo ime i šifru, što može da zloupotrebi na sličan način kao i prilikom krađe sesije (Sl. 8).



Sl. 8. Prikaz ukradenih kredencijala u terminalu napadača pomoću *phishing* tehnike

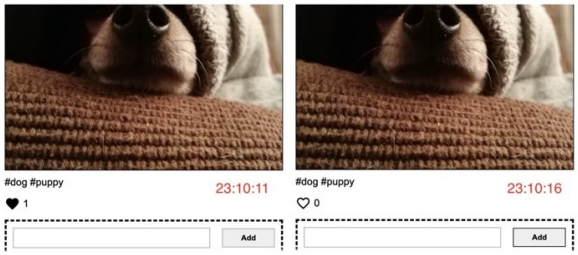
3) *Način sprečavanja napada*

U prethodnim primerima nije bilo potrebno raditi dodatnu validaciju zbog prirode napada. Ali u ovom slučaju bi bilo poželjno uraditi validaciju i *input*-a i *output*-a, na klijentskoj i na serverskoj strani. Prilikom pretrage se dešava *AJAX* poziv, pa je poželjno da se pre toga uradi sanitizacija *input*-a, najverovatnije uz pomoć regularnog izraza. U slučaju da *input* ne ispunjava zahteve, *AJAX* poziv se neće izvršiti i korisniku se ostavlja poruka da zna da je *input* polje bilo neispravno popunjeno. Sledeći korak je uraditi validaciju u kontroleru prilikom obrade zahteva i vraćanja odgovarajuće stranice. Tu bi najbolje rešenje bilo korišćenje gotovih metoda za zamenu (*escaping*) *HTML* koda. Ako je ipak potrebno dozvoliti određene tagove ili specijalne karaktere, metode moraju ručno da se pišu uz veliki oprez. Na primer, ako korisnik zameni `<script>` tag praznim stringom bez rekurzije, napadač može da sastavi skriptu sa *script* tagom `<scr<script>ipt>`. Takođe mora da se vodi računa da se obrade i *uppercase* i *lowercase* karakteri. Bitno je određene karaktere i stringove menjati u celoj skripti, a ne samo po prvom pojavljivanju. Zlonameran kod može biti prosleđen i kao atribut nekog dozvoljenog taga, pa je i za to potrebna validacija. Nakon što programer utvrdi sve šta mu je potrebno za validaciju na serveru, može da uradi i validaciju *output*-a. Najlakši način uraditi to u okviru ove aplikacije jeste koristeći *JSTL* tag `<c:out>`, jer ovaj tag omogućava *escaping* vraćenog koda. Na kraju se uz ove tehnike postiže visok nivo zaštite od ovakvog *XSS* napada.

C. *Izvršavanje nedozvoljenih radnji u ime korisnika*

Osim krađe podataka, napadač može pomoću zlonamernog koda i da izvrši neku radnju na stranici u ime korisnika. U tom slučaju nema potrebe da išta radi na svom serveru, već samo da pripremi kod koji će se izvršiti. Ako je cilj napadača da što više korisnika ošteti, to bi najbolje postigao uz snimljen *XSS* napad.

Skripta za ovaj primer je sastavljena tako da se u određenom vremenskom intervalu daju ili sklanjaju korisnikove reakcije na slike (Sl. 9). Iako ovaj napad nema veće posledice po korisnika, sam napad može da izazove nelagodnost i zbunjenost. Ovo je samo jedan primer izvršavanja nedozvoljenih akcija u skladu sa datim alatom, ali i u ovom slučaju zlonamerne skripte mogu da izazovu i znatno veće posledice, posebno ako korisnik nije ni svestan da se nešto desilo.



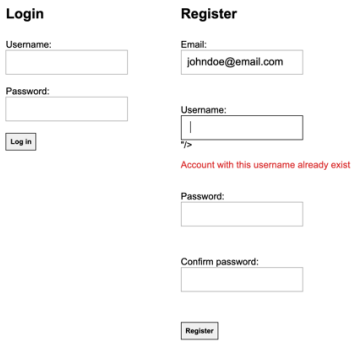
Sl. 9. Prikaz izvršavanja napadačeve skripte u ime korisnika u toku određenog vremenskog intervala

Iako je cilj napadača različit u odnosu na prethodni primer, metoda sprečavanja od *XSS* napada je ista. Potrebno je validirati *input* i *output*, sa klijentske i serverske strane. Poželjno je raditi validaciju na svim osetljivim mestima da bi se smanjio rizik od napada.

D. *Keylogger*

Još jedan način na koji napadač može da dođe do osetljivih korisnikovih podataka jeste da umetne skriptu koja napadačevom serveru prosleđuje karaktere koje korisnik unosi, karakter po karakter uz tačno vreme unosa. Ova tehnika nadgledanja pojedinačnih karaktera koje korisnik unosi se naziva *keylogger*, i najefikasnija je na stranicama na kojima se unose poverljivi podaci, poput broja kreditne kartice ili kredencijala. Skripta radi tako što pravi ograničen niz karaktera koji se šalje u određenom vremenskom intervalu samo ako je taj niz popunjen.

Na stranici za registraciju namerno je napravljen bezbednosni propust. Kada korisnik prilikom registracije unese već postojeće korisničko ime ili *e-mail*, prilikom osvežavanja stranice će ta informacija biti prisutna u *URL*-u i iz nje biti ispisana u odgovarajuće *input* polje (Sl. 10).



Sl. 10. Prikaz početne stranice u toku izvršavanja *keylogger XSS* napada

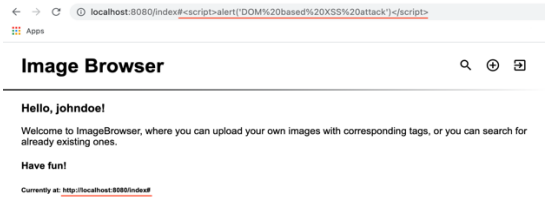
Napadač je otkrio da može svojim zlonamernim kodom da zatvori tag tog *input* polja, i dalje samo izvrši svoju skriptu. *Chrome*, *Safari* i *IE* pretraživači su se u ovom slučaju pokazali otpornim na napad zahvaljujući *X-XSS-Protection* zaglavlju odgovora. Svi korisnici koji se nalaze na ostalim pretraživačima će biti ranjivi. Napadač na ovaj način može da sazna kredencijale korisnika kroz individualne karaktere (Sl. 11).

Sl. 11. Prikaz ukradenih informacija u terminalu napadača tokom izvršavanja *keylogger XSS* napada

E. Jednostavan primer DOM baziranog XSS propusta

Preporučljivo je zaobići manipulaciju *DOM*-a u klijentskom kodu koliko god je to moguće da bi se smanjile šanse za *DOM* bazirani *XSS* napad. U ovom primeru se namerno na dnu svake stranice nalazi dekodovani *URL* tekuće stranice, koji se i dohvata i dekoduje u *JavaScript* kodu (Sl. 12). Ovde napadač takođe može da umetne skriptu koja će potpuno zaobići server i manipulisati kod koji se izvršava u *JavaScript*-u.

Dekodovani *URL* tekuće stranice se dohvata prilikom učitavanja te stranice u *JavaScript* kodu uz pomoć svojstva *document.URL*. Na ovaj način se vrši manipulacija *DOM* podataka, što otvara mogućnost da stranica bude ranjiva na *DOM* bazirani *XSS* napad. Eliminacijom tog koda problem bi u potpunosti nestao, ali ako je ipak potrebno taj kod i izvršiti, neophodno je uraditi validaciju. Tokom izvršavanja atributa *document.URL*, on se i dekoduje, te je preporučljivo ukloniti tu funkcionalnost. Ako informacija treba da bude dekodovana, treba izvršiti zamenu (eng. *escaping*) nepoželjnih karaktera koji se mogu naći u skripti sa odgovarajućom interpretacijom tog karaktera. Na ovaj način je sprečeno izvršavanje zlonamerne skripte na svim stranicama aplikacije.



Sl. 12. Prikaz ranjivosti stranice na kojoj se nalazi dekodovani *URL*

V. Zaključak

U ovom radu predstavljena je jedna ranjiva aplikacija pomoću koje se može izučavati *XSS* sigurnosni propust. Aplikacijom su pokriveni različiti tipovi *XSS* napada. Korišćenjem aplikacije moguće je demonstrirati neke interesantne zloupotrebe ovih propusta. Najvažnije je da korisnici mogu na siguran način u zatvorenom okruženju detektovati propuste, a zatim koristeći tehnike zaštite ispraviti propuste i uveriti se da njihova rešenja ispravno rade. Aplikaciju je takođe moguće nadograditi po potrebi u budućnosti, radi dodavanja novih primera i propusta koji bi korisnicima dodatno omogućili testiranje i učenje o XSS napadu. Aplikacija je korišćena za izvođenje laboratorijskih vežbi na predmetu Zaštita računarskih sistema i mreža na Elektrotehničkom fakultetu u Beogradu, ali efekti korišćenja nisu izmereni usled uslova izvođenja nastave izazvanih pandemijom koronavirusa.

Literatura

[1] Same origin policy, pristupano 21.05.2021, https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[2] D. Stuttard, M. Pinto, "*The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*", second edition, John Wiley & Sons Inc, 2011

[3] Cross-site Scripting, pristupano 21.05.2021, https://owasp.org/www-community/attacks/xss/

[4] OWASP Top Ten Project, pristupano 21.05.2021, https://owasp.org/www-project-top-ten/

[5] Reflected XSS Attacks, pristupano 21.05.2021, https://owasp.org/www-community/attacks/xss/#reflected-xss-attacks

[6] Stored XSS Attacks, pristupano 21.05.2021, https://owasp.org/www-community/attacks/xss/#stored-xss-attacks

[7] DOM Based XSS, pristupano 21.05.2021, https://owasp.org/www-community/attacks/DOM_Based_XSS

[8] Input Validation Cheat Sheet, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[9] XSS Prevention Rules, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-1-html-encode-before-inserting-untrusted-data-into-html-element-content

[10] Content Security Policy (CSP), pristupano 21.05.2021, https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[11] XSS Prevention Rules, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-0-never-insert-untrusted-data-except-in-allowed-locations

[12] X-XSS-Protection Header, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#x-xss-protection-header

[13] H2 Database, pristupano 21.05.2021, https://www.h2database.com/html/main.html

[14] Spring, pristupano 21.05.2021, https://spring.io/

[15] Spring Boot, pristupano 21.05.2021, https://spring.io/projects/spring-boot

[16] Apache Maven, pristupano 21.05.2021, https://maven.apache.org/

[17] Spring Initializr, pristupano 21.05.2021, https://start.spring.io/

[18] JSP Tutorial, pristupano 21.05.2021, https://www.javatpoint.com/jsp-tutorial

[19] JSTL (JSP Standard Tag Library), pristupano 21.05.2021, https://www.javatpoint.com/jstl

[20] AJAX, pristupano 21.05.2021, https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

[21] Node.js, pristupano 21.05.2021, https://nodejs.org/en/

[22] Express, pristupano 21.05.2021, https://expressjs.com/

Abstract

XSS (Cross-site scripting) is one of the most common vulnerabilities in web applications, despite the fact that there are many defense mechanisms against it that are available. This paper presents the implementation of a vulnerable application in which different types of XSS vulnerability can be demonstrated, along with the ways they can be misused, but also the ways they can be eliminated. The application can be used as an educational tool for software developer practical training in a closed and safe environment.

AN APPLICATION FOR DEMONSTRATION OF XSS VULNERABILITY

Katarina Simic, Zarko Stanisavljevic