CARNet
HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Sigurnost AJAX tehnologije

CCERT-PUBDOC-2008-04-224

+CERT.hr u suradnji s LSS

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi od 1996. godine.

Rezultat toga rada je i ovaj dokument, koji je nastao suradnjom CARNet CERT–a i LS&S–a, a za koji se nadamo se da će Vam koristiti u boljem razumijevanju informacijske sigurnosti i poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr

Nacionalno središte za sigurnost računalnih mreža i sustava.

LS&S, www.LSS.hr

Laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu bavi se informacijskom sigurnošću od 1995. godine.

Sadržaj

1. Uvod

U posljednje vrijeme sve je češća upotreba AJAX (eng. Asynchronous JavaScript and XML) tehnologija u web aplikacijama. AJAX tehnologija osigurava pojačanu interaktivnost web stranica, a popularnost tehnologije je porasla i pojavom „Google Sugest“ i „Google Maps“ aplikacija. AJAX predstavlja nov način upotrebe postojećih tehnologija i standarda kod kojeg se stavlja naglasak na dinamičnost i asinkronost prijenosa podataka između klijenta i poslužitelja. Međutim nova primjena starih tehnologija donosi i potrebu za primjenom poboljšanih sigurnosnih mjera. Iako se ne javljaju novi sigurnosni propusti, napadači mogu otkriti nove metode iskorištavanja propusta. Napredak u primjeni i kombinaciji tehnologija kao što su programski jezik JavaScript, XHR (XMLHttpRequest) objekti i XML format zapisa omogućuje stvaranje zanimljivih, atraktivnih i intuitivnih web aplikacija.

Ovaj dokument opisuje moguće sigurnosne propuste u AJAX aplikacijama te utjecaj primjene AJAX tehnologija na razvoj modernih web aplikacija, metode testiranja takvih aplikacija te preporučene metode sigurnosnih provjera.

2. Osnove AJAX tehnologije

AJAX (eng. Asynchronous JavaScript and XML) je skup tehnologija za razvoj dinamičkih interaktivnih web aplikacija. AJAX kao skup razvojnih tehnologija uključuje prezentacijski sloj web aplikacije stvoren upotrebnom programskih jezika XHTML (eng. Extensible HyperText Markup Language) i CSS (eng. Cascading Style Sheets), dinamički prikaz i interakciju upotrebom DOM (eng. Document Object Model) modela, mogućnost izmjene podataka i manipulaciju sadržajem korištenjem programskog jezika XML (eng. Extensible Markup Language), asinkroni prihvat podataka upotrebom XMLHttpRequest objekta te programski jezik JavaScript kojim se povezuju sve prethodno spomenute tehnologije. Glavne karakteristike web aplikacija razvijenih AJAX tehnologijama su interaktivnost, brzina, iskoristivost i funkcionalnost.

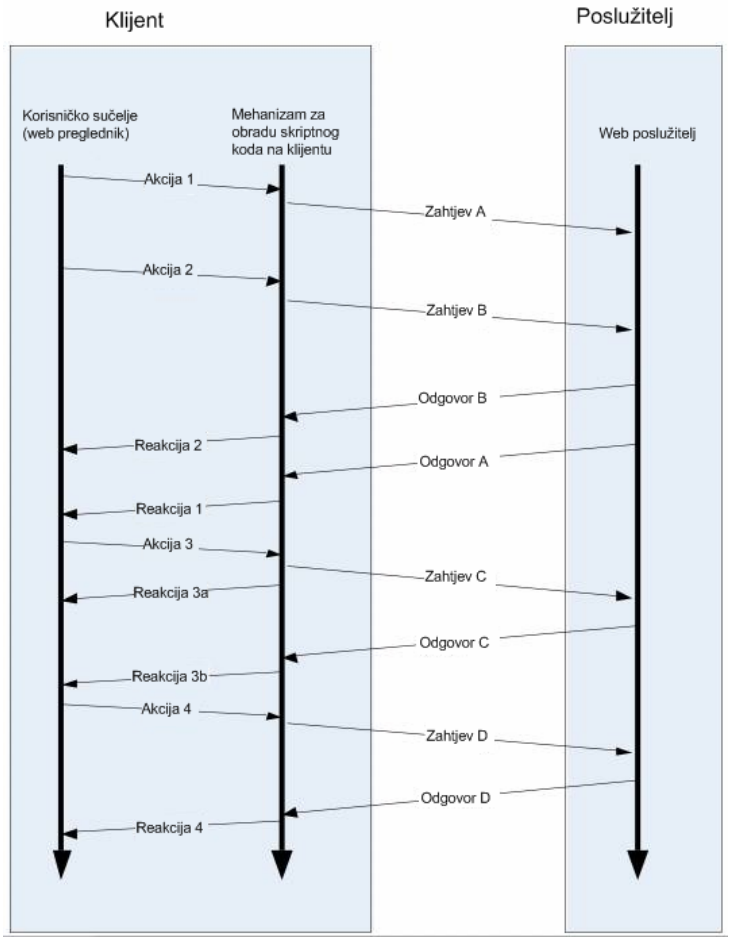
U prošlosti su web aplikacije bile isključivo statične(u idućim poglavljima će se takve aplikacije nazivati običnim web aplikacijama) što znači da su sadržavale i ograničenja kao na primjer, smanjenu interaktivnost zbog nemogućnosti dohvata samo dijela podataka s poslužitelja. 1995. godine krenuo je razvoj programskog okružja na strani klijenta te su nastali programski jezici JavaScript, VBScript, kao i Java Applet tehnologija i ActiveX kontrole uz još nekoliko drugih tehnologija. Razvoj spomenutih tehnologija kretao se u smjeru prijenosa aktivnosti s poslužitelja na klijenta. Iako je razvoj započeo već 1995. godine, javili su se problemi u radu s klijentskim tehnologijama te se velik dio aktivnosti vratio natrag na poslužitelj. Termin „AJAX“ skovao je u veljači 2005. godine Jesse James Garrett, osnivač Adaptive Path poduzeća koje se bavi razvojem informacijske arhitekture.

U posljednje vrijeme sve je češća upotreba AJAX tehnologija te većina web aplikacija koristi jedan od oblika klijentskih tehnologija u jednakom opsegu kao i poslužiteljske tehnologije, odnosno obrada podataka i funkcionalnosti web aplikacija nisu koncentrirani isključivo na poslužitelju. Najčešće korištena klijentska tehnologija je programski jezik JavaScript.

AJAX tehnologija omogućuje razmjenu samo nekih dijelova podataka između klijenta i poslužitelja, za razliku od statičnih web aplikacija kod kojih se treba ponovno učitati cijela web stranica kako bi se prikazala promjena na strani klijenta.

Web aplikacije koje se ne temelje na AJAX tehnologiji funkcioniraju prema sinkronom modelu gdje za svaki zahtjev prema poslužitelju slijedi odgovor koji je popraćen određenom akcijom u prezentacijskom sloju ISO/OSI modela. Na primjer, ukoliko na web stranici postoji tipka *submit* pritiskom na nju šalje se zahtjev poslužitelju zajedno s određenim parametrima (mogu uključivati i podatke koje je korisnik unio u aplikaciju). Ovo je uobičajeno ponašanje „klikni i čekaj“ koje ograničava interaktivnost aplikacije. Znači korisnik ne dobije povratnu informaciju od poslužitelja odmah, već mora čekati da se ponovno učita cijela web stranica. Spomenuti problem moguće je riješiti integracijom AJAX tehnologija u izradu stranica. AJAX omogućuje asinkronu komunikaciju s poslužiteljem bez potrebe za ponovnim učitavanjem cijele web stranice na klijentskoj strani. Ovakvu interakciju omogućuju tri komponente, skriptni jezik čiji se kod izvodi na klijentskoj strani, XmlHttpRequest (XHR) objekt i XML format za enkapsulaciju podataka. Ukratko, skriptni jezik na klijentskoj strani koristi se kod inicijalizacije poziva poslužitelju te za programski pristup i ažuriranje DOM objekta u klijentskom web pregledniku. Najpopularniji skriptni jezik je JavaSctipt koji je svuda prisutan i podržava ga većina modernih web preglednika. JavaScript koristi XHR objekt za slanje zahtjeva poslužitelju putem HTTP (eng. Hyper Text Transfer Protocol) protokola. Programski jezik XML se koristi kao format zapisa podataka za poruke koje se razmjenjuju između klijenta i poslužitelja. Međutim, za postizanje asinkronosti sustava nije potrebno koristiti XML kao format zapisa podataka. Mnoge web stranice upotrebljavaju JSON (eng. JavaScript Object Notation) format umjesto XML formata jer ga je lakše obraditi. Također, mnogo stranica ne koristi niti XML niti JSON kao format zapisa podataka za prijenos. Umjesto spomenutih formata koristi se HTML format (eng. HyperText Markup Language), čiji se dijelovi dinamički umetnu u kod stranice.

Dakle, AJAX nije skup novih tehnologija, već kombinacija postojećih stvorena za razvoj interaktivnih i atraktivnih web aplikacija. Razvojni su programeri otkrili mnogo načina upotrebe AJAX tehnologije, neki od njih su implementacija sugestivnih polja za unos podataka (Google Suggest) i popisa podataka koji se sami osvježavaju. Sve XHR zahtjeve obrađuju tipične poslužiteljske tehnologije kao što su J2EE, .NET, PHP. Slijedeća slika ilustrira asinkronu prirodu AJAX aplikacija:



Slika 1. Asinkrona komunikacija klijenta i poslužitelja kod primjene AJAX tehnologije

Web stranice temeljene na AJAX tehnologiji namijenjene su većem broju korisnika, a njihov je sadržaj interaktivan i često uključuje kombiniranje sadržaja s više izvora, aspekte mrežnog druženja (kao što su socijalne mreže (eng. social network).

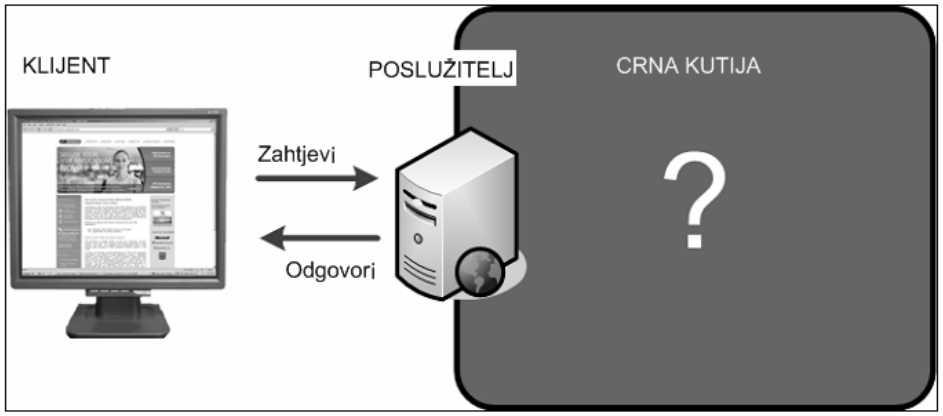
3. AJAX tehnologija i sigurnost

Primjena AJAX tehnologija u kreiranju web aplikacija ne donosi nove sigurnosne ranjivosti. Aplikacije temeljene na AJAX tehnologijama suočavaju se s tipičnim sigurnosnim problemima kao i aplikacije koje ne koriste AJAX tehnologije. Još uvijek nije standardizirana primjena AJAX tehnologija te ne postoje preporučene metode implementacije što ostavlja napadačima prostor za primjenu različitih načina zlouporabe sigurnosnih propusta. Programeri koji stvaraju web aplikaciju primjenom AJAX tehnologija trebaju primijeniti sigurne metode autentikacije, autorizacije, kontrole pristupa i provjere ulaznih podataka. U slijedećim poglavljima opisane su neki najčešći sigurnosni problemi koji se javljaju prilikom kreiranja AJAX aplikacija.

3.1. Povećanje izložene površine

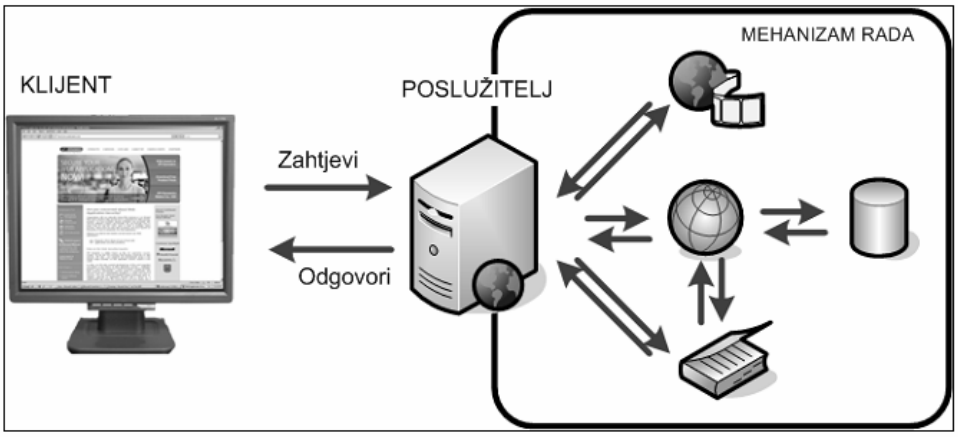
Primjena AJAX tehnologija znači i povećanje kompleksnosti sustava klijent-poslužitelj. Često je u razvoju AJAX aplikacija potrebno ostvariti posebne stranice na poslužitelju koje implementiraju određenu funkcionalnost, kao što je automatsko nadopunjavanje riječi u tražilici. Takve stranice mogu biti meta napadača i predstavljati još jedan dodatni pristup ranjivom sustavu te ih programer treba zaštititi od zlouporabe. Koncept AJAX aplikacije može se usporediti sa sigurnosnim konceptom višestrukih pristupa u kuću i poteškoće koje nastaju kod ostvarivanja osiguranja za takav entitet. Dakle, osiguravanje web stranica analogno je osiguravanju kuće koja ima deset ili više ulaznih vrata. Aplikacije koje napadaču omogućuju da sazna više informacija o njenom internom radu smatraju se aplikacijama s povećanom izloženom površinom.

Web aplikacije koje se ne temelje na AJAX tehnologiji moguće je zamisliti kao crnu kutiju. Većina korisnika ne zna kako funkcionira web aplikacija. Korisnik unosi podatke u polja predviđena za to na web stranici i dobiva zatražene podatke od poslužitelja. Način i logiku vraćanja podataka korisnik ne može saznati. Prema tome, web aplikacije koje ne koriste AJAX tehnologiju su zatvoreni sustavi, odnosno predstavljaju crnu kutiju. Za razliku od običnih web aplikacija, one temeljene na AJAX tehnologiji su transparentne za korisnika što znači da su korisniku dostupni neki podaci o radu web aplikacije. Kao i kod aplikacija koje ne koriste AJAX tehnologiju, korisnik unosi podatke i dobiva povratnu informaciju, ali sada ima mogućnost saznati mehanizme dohvata podataka. Slijedeća slika prikazuje sustav klijent-poslužitelj kod web aplikacija koje ne koriste AJAX tehnologije.



Slika 2 Sustav kod običnih aplikacija kao crna kutija

Kod AJAX aplikacija interni mehanizam rada aplikacije nije u potpunosti sakriven od korisnika, već je transparentan, kao što je prikazano na slijedećoj slici.



Slika 3. Transparentnost AJAX sustava

Ako AJAX aplikacija koristi JavaScript programski kod za komunikaciju s poslužiteljem, kod se mora prenijeti s poslužitelja na klijenta u svom originalnom (nekriptiranom) obliku. Dakle, sve što korisnik treba napraviti je pogledati izvorni kod web stranice i vidjet će kako aplikacija funkcionira, odnosno kako se dohvaćaju podaci. Napadači mogu iskoristiti transparentnost AJAX aplikacija za dobivanje informacija o načinu na koji funkcionira određeni dio aplikacije. Moguće je zamaskirati JavaScript kod tako da je na prvi pogled nečitljiv. Međutim, napadači uvijek mogu otkriti metode odmaskiranja podataka, obzirom da ti podaci nisu kriptirani već samo zamaskirani na neki način. Kako bi napadač mogao izvesti uspješan napad mora znati kako aplikacija funkcionira te kako se odvija komunikacija između poslužitelja i klijenta. Dakle, transparentnost aplikacije pogoduje napadaču u smislu dobivanja informacija o načinu rada i logici koja se krije iza sustava.

Slijedeća tablica prikazuje usporedbu podataka koje napadač može otkriti u slučaju kada aplikacija koristi AJAX tehnologiju i u slučaju kada ne koristi.

Dostupne informacije	Obična aplikacija	AJAX aplikacija
Jezik izvornog koda	Da	Da
Poslužitelj	Da	Da
Operacijski sustav poslužitelja	Da	Da
Dodatne podkomponente	Ne	Da
Karakteristike metoda	Ne	Da
Tipovi parametara	Ne	Da

Tablica 1 1. Usporedba podataka dostupnih napadaču

Napadač može upotrijebiti različite metode analize prometa između poslužitelja i klijenta za prikupljanje informacija.

3.2. Problemi s programskim kodom koji se izvodi na klijentskoj strani

Ovisnost o izvođenju dijela programskog koda na klijentskoj strani zahtijeva primjenu prikladnih sigurnosnih mjera. Kao što je opisano u prethodnom poglavlju web aplikacije temeljene na AJAX tehnologiji su transparentne u smislu da otkrivaju način funkcioniranja stranica. Razvojni programeri trebaju posvetiti posebnu pažnju kod implementacije klijentskih kontrola i njihove sigurnosti. Na primjer, neka postoji, kao dio web aplikacije, stranica za administraciju kojoj smiju pristupati samo autorizirani korisnici. Programeri često posvećuju mnogo pažnje kod kreiranja različitih sigurnosnih ograničenja za pristup spomenutoj stranici. Međutim, otvorenost AJAX aplikacija omogućuje napadaču zaobilaženje implementiranih sigurnosnih mjera. Napadač može analiziranjem prometa između poslužitelja i klijenta saznati koje funkcije koristi administratorska stranica i pozvati ih izravno bez pristupanja stranici za administraciju. Najčešće se propusti javljaju kod aplikacija kojima su AJAX funkcionalnosti dodane naknadno. Programeri kod takvih stranica često implementiraju prikupljanje podataka AJAX pozivima bez ostvarenja sigurnosnih provjera na poslužitelju. Ako ne postoji autorizacijski dio koda na poslužiteljskom računalu, napadač može jednostavno dodati nove korisnike te izmijeniti bitne podatke postojećih korisnika, kao što su zaporke. Dakle, za svaki zahtjev za podacima treba postojati prikladna sigurnosna provjera na poslužiteljskom računalu. Web aplikacije nikad ne bi trebale vjerovati zahtjevima klijenata, a programeri bi uvijek trebali računati na mogućnost pristizanja zlonamjernih zahtjeva poslužitelju. Prema tome, autentikacija i primjerena provjera ulaznih podataka je obvezna na poslužitelju.

3.3. Nove mogućnosti XSS napada

Moguća je pojava sigurnosnih problema kod transformacije podataka za prikaz HTML kodom. Bilo koji podaci pročitani iz nekog izvora podataka (npr. XML dokument, baza podataka, binarna datoteka) trebaju se pretvoriti u format koji ljudi mogu pročitati. U tradicionalnim web aplikacijama ta se pretvorba obavlja na poslužitelju, no kod AJAX aplikacija ona se često obavlja na klijentu. Prednost pretvorbe podataka na klijentskom računalu je smanjenje opterećenja poslužitelja, a nedostatak je pojava mogućnosti ubacivanja proizvoljnog programskog koda.

Kod običnih web aplikacija napadač je koristio vrijeme između obnavljanja stranica te zlouporabio sigurnosne propuste za izvođenje XSS (eng. Cross-site scripting) napada. Vrijeme čekanja omogućilo je napadačima korištenje redova (programskih struktura u kojima se čuva redoslijed dolaska podataka, odnosno FIFO (eng. First-In-First-Out) struktura) za pokretanje napada. Uvođenjem AJAX tehnologija u web aplikacije otvorile su se nove mogućnosti zlouporabe XSS ranjivosti. Međutim upotreba XHR objekata sprečava mogućnost XSS napada jer dozvoljava kontakt samo s izvornim poslužiteljem. To ograničenje je namjerno ugrađeno kako bi se spriječila pojava XSS ranjivosti.

JavaScript podržava objektno orijentirane tehnike programiranja te sadrži mnogo ugrađenih objekata, ali i dozvoljava stvaranje korisničkih objekata. Slijedeći kod prikazuje stvaranje novog objekta:

```
New Object()
ili
message = {from : "john@example.com",to : "jerry@victim.com",subject
:   "I   am   fine",body   :   "Long   message   here",showsubject   :
function(){document.write(this.subject)}};
```

Ovaj kod služi za stvaranje poruke elektroničke pošte koja sadrži sva potrebna polja. Objekt se može serijaliziratiza upotrebu u JavaScript kodu te ga programer može dodijeliti novoj varijabli koju će procesirati ili obaviti određene operacije nad njome te ispisati rezultat. Pod serijalizacijom se podrazumijeva preuzimanje objekata i njihovog sadržaja te njihovo pretvaranje u linearni tok podataka koji će se pohraniti u neki medij za spremanje podataka, kao što su datoteka, međuspremnik i slično. Serijalizacijom objetka pojednostavljuje se komunikacija između različitih tehnologija koje koristi AJAX aplikacija. No u isto vrijeme serijalizacijom objekata neki podaci o implementaciji postaju dostupni napadaču jer tipovi podataka više ne moraju biti apstraktni. Ako napadač umetne zlonamjerni kod u polje subject i posalje takvu poruku tada primatelj poruke postaje žrtva XSS napada. JavaScript objekt može sadržati podatke, ali i metode. Nepravilnom serijalizacijom JavaScript objekata može se stvoriti sigurnosna ranjivost koju napadač može zlouporabiti vještim umetanjem proizvoljnog programskog koda.

Također, i upotreba JSON komponente može napadaču omogućiti iskorištavanje različitih ranjivosti, kao što je krivotvorenje zahtjeva (CSRF – Cross-site request forgery). JSON se katkad koristi umjesto XML entiteta za komunikaciju između klijenta i poslužitelja. Poslužitelj odgovara na zahtjev JavaScript kodom kojeg odmah evaluira web preglednik. Ovaj scenarij podrazumijeva povjerenje da kod koji je poslan klijentu nije izmijenjen na zlonamjeran način. Zbog postojanja mogućnosti krivotvorenja zahtjeva preporuča se upotreba XML umjesto JSON objekata.

Primjer zlouporabe JSON formata razmjene podataka.

```
"bookmarks"{"bookmarks":[{"Link":"www.example.com","Desc":"Interestin
g link"}]}
```

Bookmarks je JSON objekt s parom ime-vrijednost. Napadač može umetnuti zlonamjerni programski kod u polje *Link* ili *Desc*. Ako se spomenuti objekt umetne u DOM i pokrene, napadač je uspješno zlouporabio XSS ranjivost.

Još jedan primjer XSS napada je izmjena JavaScript polja, ukoliko se ne obavi prikladna sigurnosna provjera podataka.

Još jedan oblik XSS napada je manipulacija XML tokom podataka. AJAX aplikacija može koristiti XML podatke s raznih lokacija, a prosljeđuju se aplikaciji upotrebom SOAP, REST ili XML-RPC tehnologija. Ukoliko napadač uspije presresti podatke, tada ih može i izmijeniti te umetnuti programski kod po volji. Web preglednik koji obrađuje dohvaćeni XML tok podataka može sadržavati različite ranjivosti pa je potrebno obaviti valjanu provjeru XML podataka.

XSS ranjivost se može pojaviti zbog problema sa serijalizacijom podataka. Kada je serijalizirani tok objekta dohvaćen u preglednik, JavaScript izvodi pozive metoda na DOM-u da bi izvršio izmjene i umetnuo novi sadržaj. To se može izvršiti pozivanjem funkcije eval(). Ukoliko se poziv izvrši nad nepouzdanim tokom podataka, preglednik postaje ranjiv na ubacivanje u DOM. Također, postoji nekoliko *document.*()* poziva koji se mogu iskoristiti da bi se u DOM ubacio novi sadržaj.

4. AJAX tehnologija i ispitivanje sigurnosti

Kod ispitivanja sigurnosti običnih web aplikacija, ispitivanje počinje prikupljanjem tragova koje aplikacija ostavlja za sobom. Prikupljanje tragova je faza ispitivanja u kojoj se dohvaćaju zahtjevi klijenta i odgovori poslužitelja u svrhu shvaćanja načina komunikacije aplikacije s poslužiteljem. Kod ispitivanja sigurnosti web aplikacija (ili izvršenja napada) moguće je iskoristiti komunikaciju između klijenta i poslužitelja. Ispitivač može presretati podatke koji se šalju u komunikaciji. Za takvo ispitivanje obično se koristi posredno računalo na kojem se nalaze posebni programi za testiranje ranjivosti. Javno su dostupni besplatni programi kao što su Burp i Paros koji predstavljaju interaktivne HTTP/S posredne poslužitelje za testiranje web aplikacija te izvođenje napada. Moglo bi se reći da ispitivač glumi napadača jer obavlja sve radnje koje bi i napadač obavio. Te radnje su koncentrirane na prikupljanje informacija o internom mehanizmu rada aplikacija. Napad koji uključuje opisani scenarij je oblik napada s "ubacivanjem posrednika" (eng. man-in-the-middle). Obično se računalo koje se koristi za ispitivanje ili napad postavlja između krajnjeg korisničkog web preglednika i pripadnog poslužitelja. Ako je na računalu za testiranje postavljen programski paket Burp te je postavljen kao posredni poslužitelj, tada ispitivač (ili napadač) može presretati, ispitati i mijenjati podatke koji se razmjenjuju između web aplikacije i poslužitelja. Paros je besplatna aplikacija pisana u programskom jeziku Java te također služi za analizu prometa između web aplikacije i njenog poslužitelja. Tokom faze prikupljanja tragova potrebno je posvetiti posebnu pažnju da se zabilježe zahtjevi svim stranicama koje web aplikacija koristi. Nakon spomenute faze slijedi proces metodičkog umetanja programskog koda kako bi se testirala osjetljivost aplikacije na prenesene parametre između web aplikacije i poslužitelja i obratno. Primjena AJAX tehnologije

komplicira opisanu metodu sigurnosnog testiranja zbog svoje asinkrone prirode. Kod AJAX aplikacija promet između aplikacije i poslužitelja je gust u odnosu na obične web aplikacije. Aplikacija može slati mnogo zahtjeva u pozadini, iako se korisniku čini da je statična. Kod ispitivanja sigurnosti web aplikacija treba pripaziti na nekoliko situacija koje mogu prouzročiti poteškoće u samom procesu testiranja. Te su situacije opisane u slijedećim poglavljima.

Kod testiranja web aplikacija potrebno je provjeriti da razvojni programeri nisu odstupali od sigurne arhitekture. U sigurnom sustavu, sigurnosne kontrole se implementiraju u okružju nad kojim krajnji korisnik nema nadzor. Kod revidiranja programskog koda potrebno je podrobno pregledati klijentski kod i odrediti mijenja li na neki način stanje varijabli (cookie datoteke, FORM parametre, GET parametre) prije predaje entiteta poslužitelju. Svaki puta kada se prenose spomenuti parametri treba analizirati JavaScript kod. Kao i obične aplikacije i AJAX aplikacije treba provjeriti sigurnost autorizacije.

4.1. Problem gubitka pojma "stanja"

Kod običnih aplikacija stanje aplikacije je dobro definirano. Sve što se nalazi u DOM komponenti smatra se trenutnim stanjem stranice. Ako se stanje treba promijeniti, šalje se zahtjev poslužitelju i njegov odgovor određuje kako se stanje mijenja. Za razliku od običnih aplikacija, kod AJAX aplikacija stvari su mnogo suptilnije. Aplikacija može generirati različite tipove zahtjeva u ovisnosti o trenutnom stanju stranice. Zahtjev koji se stvara kada korisnik klikne na polje za unos podataka može biti različit od zahtjeva koji se stvara kada korisnik prije spomenutog klika selektira nešto na stranici. Uz to, odgovor poslužitelja može obnoviti samo dio stranice te na primjer korisniku se mogu prikazati nove poveznice (eng. link) ili nove kontrole na stranici. Tokom obavljanja testiranja sigurnosti treba obratiti pozornost na opisano ponašanje jer je mnogo teže odrediti jesu li predviđeni svi tipovi zahtjeva koje web aplikacija može generirati.

4.2. Događaji pokrenuti vremenskim okidačima

Događaji pokrenuti vremenskim okidačima vezani su uz korisničko sučelje kod kojeg nema interakcije s korisnikom. Aplikacije mogu u određenim vremenskim intervalima slati zahtjeve poslužiteljima u smislu obnavljanja informacija prikazanih na web stranici. Na primjer, aplikacija za rukovanje financijama može koristiti XHR objekt za osvježavanje dijelova stranice koji prikazuju trenutno stanje dionica. Kod ispitivanja sigurnosti web aplikacije lako je previdjeti da se taj proces zbiva u pozadini. Kako ne postoje korisničke kontrole, zahtjev za osvježavanjem podataka treba uhvatiti u točno određeno vrijeme.

4.3. Dynamic DOM

AJAX odgovori mogu sadržavati dijelove JavaScript koda kojeg web aplikacija treba obraditi i prikazatina korisničkom sučelju. Dijelovi JavaScript koda mogu sadržavati nove poveznice, pristup novim dokumentima na poslužitelju itd. Jedan od načina postizanja evaluacije je upotreba funkcije *eval()* koja prima jedan parametar, niz znakova, i izvodi ga kao da je dio programa, odnosno obavi određene operacije nad tim nizom znakova i pretvori ih u format kojim se prikazuje sadržaj web stranice. Dobar primjer ovog scenarija je aplikacija „Google Suggest“ gdje web stranica prima odsječke JavaScript koda koji se izvode i prikazuju kao moguće sugestije za dopunjavanje upita unesenog u tražilicu. Ponašanje aplikacije u ovom slučaju može stvarati probleme kod ispitivanja sigurnosti, a napadač može funkciju *eval()* iskoristiti za presretanje odgovora poslužitelja. U svakom slučaju potrebno je shvatiti kontekst upotrebe JavaScript koda u web aplikaciji. Kod ispitivanja treba posvetiti posebnu pažnju kada se šalje odgovor na zahtijevani parametar koji se evaluira na klijentskoj strani. Situacija predstavlja tipičnu XSS ranjivost.

4.4. XML Fuzzing

AJAX tehnologija se može koristiti za slanje zahtjeva i primanje odgovora u XML formatu. Jednostavni automatizirani alati analiziraju GET i POST metode, ali ne i način na koji su informacije enkapsulirane u XML format.

Fuzzing je metoda testiranja aplikacija na ranjivosti umetanja zlonamjerno oblikovanih podataka. Ispitivač glumi napadača te analizira rezultate izvođenja programa s umetnutim zlonamjerno oblikovanim podacima. Ovakvim ispitivanjem moguće je otkriti propuste kod obrade podataka (kao što je upotreba neprimjerenog formata zapisa).

Ispitivanje se jednostavno ostvaruje u četiri koraka:

- priprema ulaznih podataka
- zamjena dijelova podataka sa slučajnim podacima
- otvaranje datoteke
- analiza

Moguće je promijeniti dijelove XML datoteke, umetnuti slučajne podatke i analizirati što se događa prilikom obrade tako izmijenjene datoteke.

Temeljno pravilo sigurnog koda kaže da se nikada ne smiju prihvatiti podaci u aplikaciju bez prethodnih provjera konzistentnosti (korektnosti, dosljednosti, usklađenosti) i valjanosti podataka. Osim toga, treba obratiti pozornost i na oporavak od pogrešaka. Ističu se tri provjere koje uvijek treba koristiti:

- checksum
- provjera XML formata
- verifikacija programskog koda

Fuzzing je jednostavna tehnika kojom se mogu otkriti razne pogreške i sigurnosni propusti u aplikacijama.

5. Zaključak

Primjena AJAX tehnologije nije unijela nove sigurnosne ranjivosti u razvoj i ostvarenje web aplikacija. AJAX aplikacije imaju povećanu izloženost napadima (izloženu površinu) zbog transparentnosti komunikacije između klijenta i poslužitelja što napadači mogu iskoristiti za prikupljanje informacija o implementiranim mehanizmima web aplikacije. Također, asinkronost i interaktivnost AJAX aplikacija povećava gustoću prometa, odnosno izmjene informacija između klijentskog i poslužiteljskog računala te je potrebno implementirati dodatne sigurnosne provjere na poslužitelju. Potrebno je izvesti opsežna testiranja sigurnosti AJAX aplikacija kako bi se uklonila mogućnost zlouporabe određenih ranjivosti. Programeri trebaju implementirati brojne sigurnosne provjere podataka, kao i provjere konzistentnosti i valjanosti podataka.

6. Reference

[1] Ajax Security Basics, http://www.securityfocus.com/infocus/1868/2#ref6, Securityfocus 2006

[2] Ranjivosti AJAX aplikacija http://www.spidynamic.com/assets/documents/AJAXdangers.pdf

[3] Sigurnost AJAX aplikacija http://media.techtarget.com/searchSoftwareQuality/downloads/Ajax_Security_CH_6.pdf

[4] Sigurnost AJAX aplikacija http://docs.backbase.com/docs/Backbase_Whitepaper_Ajax_Security.pdf

[5] Općenito o AJAX tehnologijama http://www.tizag.com/ajaxTutorial/