Pregledni rad
Primljen: 7. 2. 2017.
Revidirana verzija: 30. 3. 2017.
Prihvaćen: 6. 12. 2017.

UDK: 351.746.2
007:004.056
doi: 10.5937/nabepo22-13128

OPŠTI ASPEKTI APLIKATIVNE IT BEZBEDNOSTI

Petar Čisar[1]
Kriminalističko-policijska akademija, Beograd

Sažetak: Za postizanje zadovoljavajućeg nivoa bezbednosti jednog informacionog sistema primenjuju se sistemske i aplikativne mere. Rad je fokusiran na opšte aspekte aplikativne IT bezbednosti, uz pregled bezbednosnih metoda primenjenih na veb i mobilne aplikacije. U skladu sa izveštajem OWASP, od veb-ranjivosti izdvojeni su, kao najčešći, napadi tipa *SQL Injection* i *Cross-site Scripting*. U radu je istaknuta i uloga alata za analizu koda, koji doprinose detekciji bezbednosnih propusta analizirane aplikacije. U kontekstu mobilnih aplikacija, posebno je izdvojen operativni sistem Android, kao jedan od najčešće korišćenih. Elaborirani su neophodni alati i okruženja za ispitivanje bezbednosti Android aplikacija, istaknute su ranjivosti i dat je veći broj bezbednosnih preporuka. U domenu aplikativne bezbednosti prikazana su i neka od novijih rešenja, kao što je pristup RASP. U radu je posebno istaknut značaj testiranja bezbednosti aplikacija, s akcentom na faze testiranja. Na kraju je, pored prethodno objašnjenih aplikativnih bezbednosnih metoda, dat i pregled metoda zaštite opšteg karaktera.

Ključne reči: veb-aplikacije, mobilne aplikacije, bezbednost, Android, OWASP, RASP, alati za analizu koda, testiranje bezbednosti.

1 Vanredni profesor, petar.cisar@kpa.edu.rs

Uvod

Informacioni sistem, imajući u vidu njegov strateški značaj, mora imati visok nivo bezbednosti podataka s kojima radi. Odgovarajući nivo bezbednosti postiže se primenom adekvatnih i pouzdanih metoda. U zavisnosti od predmeta zaštite, mere koje se primenjuju za ostvarenje IT bezbednosti mogu se podeliti na: *sistemske* (štite informacioni sistem u celini, npr. računarska LAN/WAN mreža) i *aplikativne* (štite jednu ili nekoliko aplikacija).

Na nivou sistema, bezbednost se ostvaruje primenom različitih metoda:

- kreiranje demilitarizovane zone (DMZ) – bezbednosna mrežna topologija realizovana adekvatnom upotrebom mrežne barijere (hardverske i/ili softverske) i određenim brojem servera;
- upotreba sistema za detekciju (IDS – *intrusion detection systems*) i prevenciju upada (IPS – *intrusion prevention systems*);
- ispitivanje (skeniranje) stepena ranjivosti (*vulnerability*) računara ili mreže, upotrebom onlajn ili licenciranog softvera;
- simulacija napada – testiranje sistema (npr. penetracioni test);
- antimalver paketi (antivirus, antispaj, antispam...).

U cilju postizanja što višeg nivoa bezbednosti sistema, preporučuje se primena većeg broja gorepomenutih metoda.

Imajući u vidu okruženje u kome je određena aplikacija realizovana, bezbednosne metode obuhvataju: veb ili mobilne aplikacije (servisi) i primenu novih tehnologija (npr. RASP – *Runtime Application Self-Protection Technology* i dr.).

1. Veb-aplikacije

Veb-aplikacije su aplikacije kojima korisnici pristupaju preko nekog oblika mreže: preko interneta – javne mreže bazirane na TCP (*Transmission Control Protocol*) / IP (*Internet Protocol*) protokolu, ili intraneta – privatne/interne mreže neke institucije ili firme, u kojoj se za prenos podataka takođe koristi TCP/IP protokol. Takođe, veb-aplikacija se može shvatiti i kao softver koji se nalazi na veb-serveru i kome se pristupa preko veb-pretraživača.

Međunarodna onlajn organizacija OWASP[2] (*Open Web Application Security Project*), koja se bavi bezbednosnim aspektima veb i mobilnih aplikacija, jednom godišnje objavljuje spisak deset najzastupljenijih ranjivosti. Pod ranjivošću se smatra slabost u dizajnu, implementaciji, korišćenju i menadžmentu predmetnog informacionog sistema. Za 2016. godinu, ovaj spisak čine:

1) Injection (*SQL, LDAP, XPATH*);

2 https://www.owasp.org

2) prenos napadačke skripte kroz sajt i njeno izvršavanje (*Cross-site Scripting – XSS*);
3) slaba autentifikacija i menadžment sesije;
4) direktan pristup neadekvatno zaštićenim objektima;
5) Cross-Site Request Forgery (*CSRF*);
6) nedovoljna konfiguracija;
7) nebezbedan kriptografski smeštaj;
8) nebezbedan direktan URL pristup;
9) neadekvatna zaštita transportnog nivoa;
10) nebezbedne redirekcije i prosleđivanja.

Dva najčešća vida ranjivosti, prema podacima OWASP-a, jesu *SQL Injection* i izvršavanje napadačkog koda preko sajta (*Cross-site Scripting*). U narednom tekstu biće dat njihov kraći prikaz.

SQL Injection je zlonamerna aktivnost kod koje napadač upisuje string preko SQL upita, koji u slučaju loše izvedene validacije aplikacija prosleđuje interpreteru i tako stiče pristup poverljivim podacima iz baze podataka. Ranjivosti ovog tipa su posledica neadekvatne provere ulaznih parametara koji u aplikaciju dolaze putem POST, GET ili nekog drugog zahteva.

Primer:

Pretpostavimo da veb-aplikacija sadrži sledeći kod:[3]

```
String SQLQuery = "SELECT Username FROM Users WHERE Username = '" +
username + "' AND Password = '" + password + "'".
```

Uzimaju se korisnički uneti podaci iz autentifikacionog obrasca i direktno se ubacuju u SQL izjavu (varijable username i password). Ako napadač za korisničko ime i lozinku upiše:

Username: ' OR ''=' i Password: ' OR ''='

tada će SQL izjava izgledati ovako:

SELECT Username FROM Users WHERE Username = '' OR"="

AND Password = '' OR ''='

Umesto upoređivanja podataka koje je uneo korisnik s podacima iz njegove tabele (*Users*), upoređivaće se ' ' (prazan niz) sa ' ' (prazan niz). Rezultat ove SQL izjave uvek će biti istinit, pa će napadač tako uspeti da se uloguje kao legitiman korisnik.

Zaštita od napada ovog tipa obuhvata upotrebu gotovih ORM (*Object-Relational Mapping*) paketa – *Entity Framework*, *Hibernate* i dr. – i pripremljene naredbe (*Prepared Statements*). Primena tih metoda je dobra, ali nije dovoljna. Za potpuniju zaštitu potrebno je paralelno primenjivati i tzv. dubinsku valida-

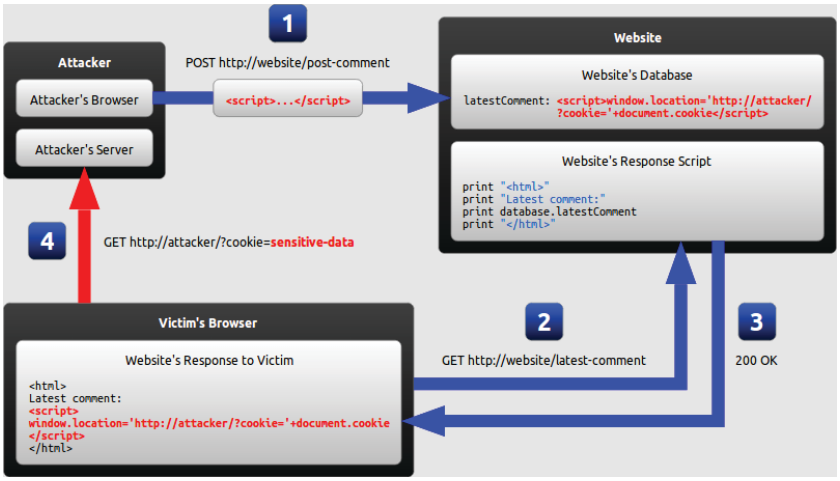
3 www.zemris.fer.hr

ciju unosa (*Defense in Depth*): osigurati pravilnu vrstu zapisa (ako aplikacija traži od korisnika da unese broj, onda treba proveriti da li je stvarno upisan broj) i pravilnu dužinu ulaznih parametara, filtrirati sve parametre koji se upisuju u bazu podataka i sve parametre koji se ispisuju u HTML dokumentu.

Cross-site scripting (XSS) jeste napadačka aktivnost kojom se postiže da veb-aplikacija prosledi napadačevu skriptu korisniku. U slučaju da je korisnik pozove, ona se učitava i izvršava u njegovom pretraživaču. To se događa zbog toga što aplikacija nema nadzor nad korisnikovim pretraživačem.

Kada napadač uspe da natera korisnički pretraživač na izvršavanje poslate skripte (koda), napadački kod će moći da čita, menja ili prosleđuje osetljive podatke koji su dostupni pretraživaču. Skripta može biti napisana u HTML-u, javaskriptu, flešu ili nekom drugom programskom jeziku koji podržava klijentov pretraživač.

Princip funkcionisanja XSS napada može se u opštem slučaju prikazati sledećom primerom.



Slika 1: *Primer XSS-a*

1) Napadač koristi jednu od formi veb-sajta da ubaci (pomoću naredbe POST - post-comment) zlonamerni string – traženje kolačića i njihovo slanje na adresu napadača (http://attacker/) u bazu podataka sajta, u kategoriji komentara (latestComment).

2) Žrtva traži neku stranicu sa sajta (latest-comment, pomoću GET-a).

3) Sajt uključuje u odgovor ubačeni string iz baze i šalje ga žrtvi.

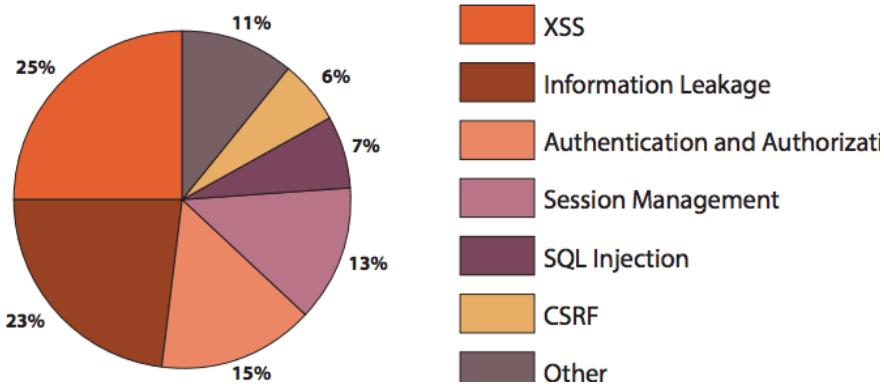
4) Žrtvin pretraživač izvršava zlonamerni skript u sklopu odgovora i šalje žrtvine kolačiće napadačevom serveru.

Za zaštitu od XSS napada koriste se različite metode:

- sprečavanje izvršenja skripti u korisnikovom pretraživaču kada nisu potrebne;
- filtriranje korisničkih zahteva – veb-aplikacija proverava korisničke zahteve i filtrira metaznakove definisane HTML specifikacijom kako bi ustanovila da li korisnički zahtev sadrži skriptu; ukoliko zahtev sadrži skriptu, veb-aplikacija sprečava prikazivanje HTML dokumenta u korisničkom pretraživaču[4];
- kodiranje stranica – veb-provajder kodira stranice sajta kako bi onemogućio nenamerno izvršavanje skripti;
- blokiranje IP adrese korisnika ukoliko pokuša da unese maliciozni kod više puta; novije verzije pretraživača imaju ugrađene filtere koji automatski prekidaju izvođenje bilo kakvih malicioznih skripti koje se nalaze na internetu.[5]

Primeri zaštite od XSS: *AngularJS* (*JavaScript framework*) „iskejpuje" (transformiše) karaktere koje je korisnik uneo u HTML entitete, koje većina brauzera ne interpretira kao HTML tag, što znači da se ulazni podaci tretiraju kao nepouzdani (*untrusted*). *.NET* platforma i *ASP.NET framework* koriste biblioteke za kodiranje (npr. *Microsoft Anti-Cross Site Scripting Library*). Java takođe koristi biblioteku tzv. *Escaping Library*.

XSS vs. SQL Injection. Različiti oblici veb-ranjivosti prikazani su na sledećoj slici. Može se zapaziti da je XSS ranjivost znatno češće zastupljena (25%) od *SQL Injection* (7%).



Slika 2: *XSS vs. SQL Injection*
(Izvor: https://geekflare.com/online-scan-website-security-vulnerabilities/)

4 www.zemris.fer.hr
5 www.veleri.hr

2. Alati za analizu koda

Alati za analizu koda (*Application Security Testing*) vrše statičku i dinamičku analizu koda – izvornog (*source*), objektnog i izvršnog – otkrivajući programerske nedostatke i ranjivosti. Bezbednosni propusti u aplikacijama mogu se detektovati na dva načina: statičkom analizom izvornog koda i dinamičkom analizom izvornog koda. Pod statičkom analizom izvornog koda podrazumeva se analiziranje izvornog koda aplikacije pre njenog puštanja u rad. Dinamička analiza se obavlja nakon puštanja same aplikacije u rad. Često se, radi što bolje detekcije bezbednosnih propusta, koristi kombinacija obe vrste analize, pri čemu rezultati statičke analize izvornog koda pomažu prilikom dinamičke analize.[6]

Primeri (*open source*) alata za analizu koda, u zavisnosti od programskog jezika posmatrane aplikacije: java (*FindBugs*, *PMD*, *VisualCodeGrepper*), rubi (*Brakeman*), C i C++ (*Flawfinder*, *CPPCheck*), PHP (*RIPS*, *DevBug*, *VisualCodeGrepper*).

Alati za analizu koda imaju određene nedostatke, od kojih je najvažnija nedovoljna pouzdanost: dosta ranjivosti ne nalaze automatski, generišu relativno visok broj lažnih pozitiva (detektuju postojanje problema koji to zapravo nije), često ne pronalaze konfiguracione probleme.

3. Mobilne aplikacije

Mobilne aplikacije su vrsta softvera za mobilne uređaje: pametne telefone (*smartphone*), tablet računare i dr. Dizajnirane su uzimajući u obzir zahteve i ograničenja mobilnih uređaja, kao i sa namerom da iskoriste svaku specijalizovanu mogućnost koju oni imaju.

Slično kao za veb-aplikacije, međunarodna onlajn organizacija OWASP objavila je spisak deset najkritičnijih rizika koji se odnose na mobilne aplikacije u 2016. godini.

Na slici 3 se može zapaziti da je najčešća ranjivost mobilnih aplikacija neadekvatna upotreba mogućnosti mobilne platforme ili nekorišćenje bezbednosnih kontrola (M1).

6 www.infoteh.rs.ba

Slika 3: *OWASP 2016 – mobilne aplikacije*
(Izvor: https://www.linkedin.com/pulse/owasp-mobile-top-10-2016-candidate-release-milan-singh-thakur)

4. Ispitivanje bezbednosti Android aplikacija

U današnje vreme najčešće korišćeni operativni sistem za razvoj mobilnih aplikacija jeste Android. U cilju ispitivanja bezbednosti Android aplikacija potrebno je obezbediti sledeće komponente: operativni sistem Vindous ili Linuks (potrebno korišćenje Android emulatora), Java 7, Android studio (razvojno okruženje) i SDK (*Software Development Kit*). Postupci za ispitivanje bezbednosti podrazumevaju:

- pokretanje simulatora mobilnog uređaja (AVD – *Android virtual device*);
- korišćenje posrednika (*Fiddler* ili *Paros*) ako aplikacija koristi HTTPS ili je veb-stranica koja se ispituje na Android pretraživaču (posrednik dekriptuje i omogućava modifikaciju HTTPS saobraćaja);
- korišćenje alata za proveru bezbednosti – SDK dolazi sa Manifest eksplorerom (*Manifest Explorer*, definiše sistemske i aplikativne dozvole, kao i brojne zaštite), Intent sniferom (*Intent Sniffer*, vrši monitoring *runtime* rutiranih intenta poslatih između aplikacija), Intent fazerom (*Intent Fuzzer*, testni alat za generisanje neočekivanog i neregularnog ulaza u aplikaciju, sa ciljem njenog obaranja) i Strejsom (*Strace*, alat za otkrivanje grešaka);
- korišćenje alata za komuniciranje sa simulatorom (*Android Debug Bridge – ADB*).

Prevođenje (dekompajliranje) aplikacija može da predstavlja bezbednosnu pretnju. Naime, moguće je preimenovati nastavak fajla .apk (*Android Application Package*) u .zip, otvoriti ga nekim ZIP alatom i proveriti njegov sadržaj. Zlonamerni programeri mogu da prevedu aplikaciju, izmene izvorni program ili ubace maliciozni kod, ponovno zapakuju aplikaciju i distribuiraju je kao originalnu.

Kako bi se povećao stepen aplikativne zaštite, u Androidu se koriste dozvole za datoteku. To znači da svaki APK fajl instaliran u simulatoru ima jedinstveni ID, što sprečava aplikaciju da pristupi podacima druge aplikacije.

5. Bezbednosni problemi Androida

Android, kao jedan od najpopularnijih operativnih sistema za mobilne uređaje, zbog otvorenog koda nosi niz potencijalnih bezbednosnih problema, koji se, u skladu sa donjom slikom, mogu formulisati u nekoliko kategorija.

Slika 4: *Ranjivosti Androida*
(Izvor: http://www.appmarsh.com/Android-security-vulnerabilities-every-developer-should-know/)

Otvoreni kod (open source). Pošto je ceo izvorni kod lako dostupan, ljudi sa zlim namerama mogu da traže mogućnost za stvaranje bezbednosnih problema.

Fragmentacija. Sa stotinama proizvođača koji svi daju svoj doprinos razvoju operativnog sistema, virusi i malveri novih generacija lako eksploatišu slabosti kojih nema kod stok (*stock*) Androida.

Spori razvoj novih verzija (Slowmoving Upgrades). Korisnici Androida su poznati po kašnjenju u usvajanju novih zakrpa i verzija ili prelasku na novu verziju operativnog sistema. Koristeći zastarelu verziju operativnog sistema, oni postaju sve više ranjivi na krađe podataka i identiteta.

Povezivanje javaskripta preko HTTP-a (JavaScript binding over). Programeri mogu javaskript metod za povezivanje prihvatiti kao najlakši način za učitavanje veb-sadržaja u Android aplikaciju, ali to može da bude izvor velikih bezbednosnih problema. Otvaranjem mogućnosti za veb-prikaz (*web view*) upotrebom HTTP-a, otvara se i mogućnost da napadači u potpunosti otmu HTTP podatke ili evente daljinskim izvršavanjem kodova aplikacije.

Prodavnica aplikacija treće strane (Third Party App Stores). Iako ovo ne mora da bude bezbednosni problem u direktnoj vezi sa Androidom, ipak zaslužuje određenu pažnju. Činjenica je da ove prodavnice sadrže više skrivenog malvera nego u Gugl plej (*Google Play*) prodavnici. Iako nisu sve prodavnice aplikacija treće strane loše, ipak je potrebno pažljivo potražiti one u kojima će aplikacija biti postavljena (dostupna). Aplikacija koja je postavljena u ove prodavnice ne mora nužno da ima bezbednosne pretnje, ali može da stvori lošu reputaciju ako su korisnici iz nekog razloga nezadovoljni njima.

6. Android – bezbednosne preporuke

Imajući u vidu elaborirane potencijalne bezbednosne probleme kod Android platforme, mogu se formulisati neke generalne bezbednosne preporuke za Android aplikacije.

Ako je u pitanju smeštaj podataka koji se koriste, preporuka je da se koriste interno skladište (*default*) ili dobavljači sadržaja (uslovno preporučljivo), dok eksterno skladište nije preporučljivo. Potrebno je minimizovati broj dozvola (zahtevi i kreiranje) koje aplikacije traže. Kod korišćenja mreže, u slučaju IP mreže treba koristiti protokol za bezbedan saobraćaj (HttpsURLConnection), a u slučaju telefonske mreže umesto SMS-a bolje je koristiti *Google Cloud Messaging* (GCM). Primenjivati validaciju ulaznih podataka, jer su kod string jezika (javaskript, SQL) mogući validacioni problemi. U toku rukovanja sa korisničkim podacima minimizovati korišćenje API-ja za pristup osetljivim i ličnim podacima. Iz bezbednosnih razloga preporučljivo je koristiti heš formu podataka i ne koristiti telefonske identifikatore (telefonski broj ili IMEI). Validacijom certifikata, uz bezbednosnu garanciju od strane CA (*Certification Authority*), sprečava se *man-in-the middle* napad.

Ako se u toku razvoja aplikacije koristi veb-prikaz, odnosno veb-sadržaj (HTML, javaskript), postoji opasnost od XSS-a, pa se zato uobičajeno (po

defoltu) ne izvršava javaskript. Smanjenjem traženja korisničkih kredencijala smanjuje se verovatnoća fišing napada, dok se istovremeno za autorizaciju sugeriše upotreba tokena. Za bezbedan saobraćaj primenjuju se kriptografske metode: HTTPS, tunelovanje (SSLSocket), AES, RSA algoritmi i dr. Za interprocesnu komunikaciju treba koristiti ugrađene sistemske funkcionalnosti (Intent, Binder, Messenger). Dinamičko učitavanje koda izvršava se sa istim bezbednosnim dozvolama kao i aplikacija, pa se ne preporučuje učitavanje koda izvan aplikacije APK. Što se tiče bezbednosti u virtuelnoj mašini (Dalvik – Androidova virtuelna mašina), nema neposrednih pretnji, jer se aplikacije izvršavaju u tzv. sendboks (*sandbox*) okruženju. U cilju postizanja bezbednosti u izvornom kodu, uputno je koristiti SDK, jer se sve aplikacije izvršavaju u *Application Sandbox*-u. Postavljanjem *Session Timeouts* definiše se vremensko ograničenje za unos PIN-a ili lozinke, čime se smanjuje verovatnoća pogađanja istih.

7. RASP

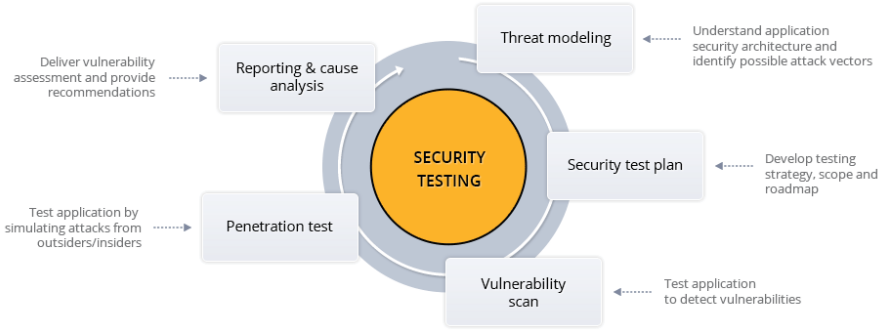
RASP (*Runtime application self-protection*) jeste bezbednosna tehnologija koja je ugrađena u aplikaciju ili u izvršno aplikaciono okruženje, ili je povezana s njima i može da kontroliše izvršenje aplikacije, kao i da detektuje i sprečava napade u realnom vremenu [6]. RASP sprečava napade samozaštitnom akcijom ili automatskom rekonfiguracijom bez učešća čoveka, kao odgovor na specifične mrežne situacije (pretnje, greške itd.).

RASP postaje aktivan u toku izvršenja aplikacije, uzrokujući da pokrenuti program nadzire sebe i detektuje sumnjive ulaze i ponašanja. RASP u realnom vremenu procesuira kako ponašanje aplikacije, tako i kontekst ponašanja. Na ovaj način je postignuta kontinualna bezbednosna analiza, sa sistemom koji trenutno reaguje na prepoznate napade [7]. Veb-aplikacije su tipična meta i napadači ih često koriste za prodor u sistem (mrežu). Više od 80% napada u današnje vreme dešava se na aplikacionom nivou [8].

U ovom momentu, RASP rešenja postoje za java virtuelnu mašinu (*Java Virtual Machine* – JVM) i .NET *Common Language Runtime* (CLR). Java i .NET su programske platforme kreirane za izvršenje nepouzdanih programa sa bezbednosnim ograničenjima. Nasuprot činjenici da imaju uporedive ciljeve i sličan dizajn u mnogim aspektima, postoje i značajne razlike u domenu bezbednosnih ranjivosti.

8. Testiranje bezbednosti

Testiranje bezbednosti predstavlja važan postupak u obezbeđivanju aplikacije. U opštem slučaju sastoji se od više faza (slika dole). Modelovanje pretnje (*threat modeling*) odnosi se na razumevanje bezbednosne arhitekture aplikacije i identifikacije mogućih napadačkih vektora. Planiranje bezbednosnog testa (*security test plan*) podrazumeva razvoj strategije testiranja, ciljeva koji se žele postići i mape puta za realizaciju. Skeniranje ranjivosti (*vulnerability scan*) uključuje testiranje aplikacije s ciljem detekcije mogućih ranjivosti. Penetracioni test (*penetration test*) ili etičko hakovanje (*ethical hacking*) podrazumeva testiranje aplikacije simuliranjem napada od strane spoljašnjih korisnika (nemaju ovlašćen pristup sistemu) ili insajdera (imaju određeni stepen ovlašćenog pristupa). Na kraju testiranja, generiše se izveštaj i analiziraju uzroci (*reporting and cause analysis*), odnosno izvode zaključci o stepenu ranjivosti i daju preporuke za unapređenje bezbednosti



Slika 5: *Testiranje bezbednosti*
(Izvor: http://www.qahelps.com/different-testing-types/security-testing-its-basic-terms/)

9. Mere zaštite

Pored prethodno opisanih mera aplikativne zaštite, mogu se formulisati i druge mere IT zaštite opšteg karaktera (nezavisno od toga da li se radi o veb ili mobilnoj aplikaciji).

Zaštita od napada uskraćivanjem usluge (DoS – Denial of Service). DoS napad se svodi na zauzimanje slobodnih resursa servera. Kada su njegovi kapaciteti u potpunosti zauzeti, korisnici više ne mogu da pristupe željenoj aplikaciji. Zaštita obuhvata distribuiranje aplikacije se na više servera, omogućavanje

samo onih procesa koji su neophodni za normalan rad servera i filtriranje paketa (npr. IP filter) u cilju provere IP adrese i porta.

Aplikacije za skrivanje podataka o serveru. Svrha im je da prikriju ili daju lažne podatke o serveru kako napadač ne bi znao koji operativni sistem napada i koje su njegove ranjivosti. Pitanje je koliko su takve aplikacije pouzdane, jer postoje slične aplikacije koje na osnovu oblika odgovora znaju da prikupe informaciju o serveru i njegovoj verziji.[7]

Bezbednosne provere u javaskriptu. Problem leži u činjenici da se kôd u potpunosti izvršava na strani klijenta (u veb-pretraživaču). U javaskript kôd ne bi smeo da bude implementiran nijedan bezbednosni mehanizam jer se on, zbog prirode izvršavanja koda, može utvrditi i zaobići.

Za bezbedno skladištenje lozinki preporučuje se upotreba kriptografskih heš algoritama (SHA1 – SHA512), kao i kriptografskih protokola tokom i nakon prijave korisnika na sistem (SSL, TLS).

Prilikom izgradnje i održavanja bezbednosti računarskog sistema od značaja je korišćenje mrežnih barijera (*firewall*) i lično postavljenih lozinki umesto uobičajenih (*default*). Za aspekt bezbednosti važno je i održavanje i upravljanje aplikacijom, odnosno zaštita svih elemenata aplikacije redovnim ažuriranjem. Uspostavljanjem pravila za pristup podacima dodatno se povećava nivo bezbednosti. Redovno nadgledanje aplikacije daje informacije o tome šta se s njom događa i ko je i kako koristi, a redovno testiranje dovodi do otkrivanja eventualnih ranjivosti. U kontekstu bezbednosti, važnu ulogu ima i održavanje bezbednosne politike preduzeća, koja podrazumeva postavljanje i sprovođenje pravila kojih se svi zaposleni moraju pridržavati.

Zaključak

U radu su elaborirane bezbednosne metode za veb i mobilne aplikacije, kao i primena novih bezbednosnih trendova na aplikativnom nivou (RASP). S obzirom na veliku različitost aplikacija i karakteristika okruženja u kojima se izvršavaju, velika je i razlika u metodama obezbeđenja zadovoljavajućeg nivoa bezbednosti. Nijedna pomenuta metoda ne omogućava univerzalnu bezbednost. Generalni zaključak je da se jedino istovremenom i promišljenom primenom većeg broja bezbednosnih preporuka, metoda fizičko-tehničke zaštite i odgovarajuće korporativne legislative, može postići prihvatljiv nivo bezbednosti aplikacija.

7 www.veleri.hr.

Literatura

1. Lazarevic, A; Kumar, V; Srivastava, J; *Managing Cyber Threats: Issues, Approaches and Challenges*, Kluwer Academic Publishers, Boston, 2005.
2. Pleskonjić, D; Đorđević, B; Maček, N; Carić, M; *Sigurnost računarskih mreža*. Viša elektrotehnička škola, Beograd, 2006.
3. Lazarevic, A; Ertoz, L; Ozgur, A; Srivastava, J; Kumar, V; *A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection*, Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, 2003, pp. 25–36.
4. Ranđelović, D; Đorđević, V; A Test of IDS Application Open Source and Commercial Source, *NBP – Journal of Criminalistics and Law*, Kriminalističko-policijska Akademija, Beograd, 2011, pp. 45–65.
5. Čisar, P; Maravić Čisar, S; *The Framework of Runtime Application Self-Protection Technology*, 17th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2016, Budapest, Hungary – Proceedings, 2016, pp. 81–85.
6. Gartner; *IT Glossary*, http://www.gartner.com/it-glossary/runtime-application-self-protection-rasp/.
7. *Veracode*, https://www.veracode.com/security/runtime-application-self-protection-rasp.
8. *Waratek*, http://www.waratek.com/runtime-application-self-protection/.
9. SANS Institute; *Protection from the Inside*, http://www.sans.org/reading-room/whitepapers/analyst/protection-inside-application-security-methodologies-compared-35917.

GENERAL ASPECTS OF APLICATION IT SECURITY

Petar Čisar
Academy of Criminalistic and Police Studies, Belgrade

Abstract: To achieve a satisfactory level of security of an information system, different system and application methods are applied. The paper has a focus on general aspects of application IT security, thereby giving an overview of security methods applied to the web and mobile applications. In accordance with the OWASP report, out of web vulnerabilities the most common include SQL Injection and Cross-site Scripting type of attacks. The paper also emphasizes the role of code analysis tools, which contribute to the detection of vulnerabilities of analyzed application. In the context of mobile applications, Android operating system is especially featured, as one of the most commonly used. The necessary environment and tools for testing the security of Android applications are elaborate, vulnerabilities highlighted and a greater number of security recommendations are offered. In the field of application security, some of the newer solutions are shown, such as RASP approach. The paper particularly emphasizes the importance of security testing of applications, with accent on testing phase. Finally, in addition to the previously explained application of security methods, an overview of security methods of a general character is given.