Gap analiza između implementiranog sistema i OWASP Top 10 propusta

Tim 9

Uvod

Za izradu aplikacije EnterprisePayments korišćen je *Play Framework* – RESTful web framework koji stavlja akcenat na produktivnost developera. Olakšava i ubrzava izradu aplikacije time što koristi *convention over configuration* princip, koji nalaže da je potrebno specificirati samo nekonvencionalne aspekte aplikacije. Takođe, nudi olakšice poput prikaza greške u browseru i reloadovanja aplikacije osvežavanjem stranice.

Uporedo se razvijaju verzije 1.x i 2.x, gde su verzije 1.x za Javu, a 2.x za Scalu. Ono po čemu je specifičan jeste *Play templating system*. Tempate je tekstualni fajl, čiji delovi imaju placeholdere za dinamičko generisanje sadržaja. Template engine koristi Groovy jezik. Moguće je pisati izraze, koristiti dekoratore, tagove, pristupiti raznim objektima (npr sesiji)… Templejti se mogu nasleđivati.

A1. Injection

Injection napadi se dešavaju kada nepouzdani podaci stignu do interpretera. Na taj način, interpreter se može prevariti kako bi izvršio neželjene komande ili pristupio podacima bez adekvatne autorizacije.

S obzirom da su korišćeni samo predefinisani upiti, aplikacija je zaštićena od SQL injection napada. Ne postoje *custom query*-ji koji bi bili podložni SQL injection napadu. Ukoliko se ne koriste predefinisani upiti (*save*, *delete*, *findBy*...), već se kreiraju, pravilan način bi bio da se koriste parametrizovani upiti. U slučaju Play frameworka, pravilan način za kreiranje custom upita bio bi korišćenje placeholdera, na sledeći način:

```
createQuery("SELECT * from Stuff WHERE type= ?1").setParameter(1,
theType);
```

Loš primer:

```
createQuery("SELECT * from Stuff WHERE type=" + theType);
```

Takođe, dodatni nivo zaštite obezbeđuje Play template engine, koji automatski escapeuje sve stringove koji potiču od korisničkih unosa. (Više o tome u odeljku o XSS-u)

S obzirom na to da aplikacija šalje i prima XML dokumente, podložna je XML External Entity Processing napadu (XXE injection). Ovakav napad bi se mogao desiti u slučaju da loše konfigurisan XML parser obrađuje XML ulaz koji sadrži referencu ka nekom spoljašnjem entitetu. Iz tog razloga, potrebno je koristiti XML šemu za validaciju XML dokumenata. Na slici je prikazan deo koda koji validira primljeni XML dokument spram odgovarajuće šeme.

```
Schema schema = schemaFactory.newSchema(schemaFile);
Validator validator = schema.newValidator();
validator.validate(xmlDocument);
```

Validacija naloga za plaćanje spram šeme:
https://github.com/tamarakatic/NationalBank/blob/master/Bank/src/main/java/com/example/Bank/service/SecurityServiceImpl.java#L67

Validacija fakture spram šeme:
https://github.com/tamarakatic/NationalBank/blob/master/Company/src/main/java/com/example/Company/service/ValidateXMLAndSaveToDB.java#L53

A2 – Broken Authentication and Session Management

Korisnici se na sistem prijavljuju pomoću korisničkog imena i lozinke. Ukoliko se unese pogrešni kredencijali, poruka koju korisnik dobija je da su netačni korisničko ime ILI lozinka. Na taj način, u slučaju napada, ukoliko napadač pogodi korisničko ime, neće dobiti informaciju o tome.

Za autentifikaciju korišćen je Play Secure modul. Anotacijom @With iznad Application klase dodaje se secure default login stranica. Da bismo prilagodili mehanizam autentifikacije, redefinisali smo metodu *authenticate* klase Security.

```
@With(Secure.class)
public class Application extends Controller {

    public static void index() {
        render();
    }
}
```

Prilikom registracije korisnika na sistem, od njih se traži da unesu lozinku, koja mora biti kompleksna da bi se otežao *brute force* napad. Kompleksnost se ogleda u tome da mora sadržati najmanje 8 karaktera, najmanje jedno malo, jedno veliko slovo i barem jedan broj. Takođe, prilikom registracije, korisnik mora uneti dvaput lozinku, da bi je potvrdio.

Lozinke se ne čuvaju u bazi podataka u otvorenom obliku, već se koristi Hash&Salt mehanizam. Za lozinke se koristi sporoheširajuća funkcija, PBKDF2, koja se vrši u 10000 iteracija. *Hash length* je 128, a *salt length* 64B.

U slučaju da korisnik zaboravi lozinku, postoji opcija da je promeni preko e-maila. Klikom na dugme, korisnik će dobiti mejl u kojem je token generisan od strane aplikacije. Nakon sat vremena, token ističe i postaje neupotrebljiv. Od korisnika se traži da unese token i novu lozinku. Ukoliko je token ispravan, i još nije istekao, promena lozinke je uspešna. Klasa `ChangePassword`

U aplikaciji se koristi HTTPS protokol, čime je obezbeđena tajnost lozinki u tranzitu.

Sesija nam je potrebna da bismo određene podatke povezali sa korisnikom – bez nje, korisnik bi morao da unosi svoje kredencijale prilikom slanja svakog zahteva. Zbog podataka koji se za nju vezuju, smatramo je bitnim resursom. Ukoliko se session management ne implementira korektno, napadač može doći u posed kredencijala, tokena, i sličnih stvari koje mu mogu pomoći u krađi identiteta registrovanih korisnika.

U Play Frameworku, ograničenje veličine podataka koji se mogu čuvati na sesiji je 4KB. Nije predviđeno da sesija služi kao cache, već sa to postoje *cache* i *flash scope*. Na sesiji mogu se čuvati samo stringovi. Izbegavano je čuvanje podataka na sesiji čija je poverljivost.

Generisali smo application secret koji se koristi za potpisivanje cookie-ja i CSRF tokena. Bitno je da ona ostane tajna. Dok god je očuvana poverljivost application secreta, nemoguće je da napadač lažira sesiju. Tajna se čuva u conf/application.conf datoteci. Kada bi se aplikacija pustila u produkciju, loša praksa bi bila da se ova datoteka nalazi u javnom repozitorijumu, te je pri puštanju u produkciju ne treba commitovati.

Da bi se smanjila verovatnoća eksploatacije sesije korisnika koji je ostao ulogovan na sistem, sesija se poništava zatvaranjem browsera. Ukoliko je potrebno da sesija traje određeni period vremena, moguće je definisati maxAge varijablu u application.conf datoteci, koja predstavlja maksimalan životni vek cookie-ja.

```
application.session.maxAge=1h
```
https://github.com/tamarakatic/EnterprisePayments/blob/master/conf/application.conf

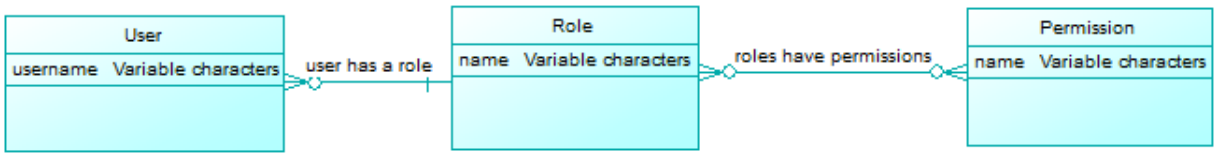
A3. Cross Site Scripting - XSS

Kao što je već napomenuto, template engine Play-a automatski escape-uje stringove. Ako nam je u template-u potreban HTML koji nije escape-ovan, moguće je koristiti raw() Java ekstenziju na stringu. Ali, ako string potiče od korisničkog unosa, potrebno je proveriti da li je bezbedan. Za takvu proveru, uvek je bolje koristiti whitelist (listu dozvoljenih tagova) nego blacklist (listu nedozvoljenih tagova). Posmatrana aplikacija je, samim korišćenjem Play frameworka, zaštićena od napada poput Cross Site Scriptinga. Template engine neće dozvoliti ubacivanje malicioznih skripti escape-ovanjem korisničkih unosa.

Ukoliko bi bilo potrebno imati npr sekciju sa komentarima, gde je poželjno da korisnici koriste HTML tagove za stilizovanje svojih komentara, a ne želimo da ih opterećujemo učenjem novih tagova, mogu se koristiti eksterne biblioteke za sanitizovanje unosa, poput JSOUP biblioteke. Ona bi bila zgodna u slučaju korišćenja nekog drugog frameworka, koji ne nudi ovakvu zaštitu od injection napada. JSOUP je Java HTML Parser koji nudi whitelistu HTML tagova. Na primer, ukoliko imamo metodu kontrolera koja prima parametre forme, možemo pozvati *clean* metodu, koja prima whitelistu i HTML koji treba isparsirati i sanitizovati. Moguće je birati između nekoliko različitih whitelisti (npr basic, relaxed..), dodavati tagove na listu... Na ovaj način bismo sprečili da se maliciozni unos izvrši u našoj aplikaciji.

A4 – Broken Access Control

Pravilno implementirana kontrola pristupa treba da spreči da korisnici vide/modifikuju podatke i pristupe funkcijama za koje nisu ovlašćeni. U posmatranoj aplikaciji, kontrola pristupa odrađena je po RBAC modelu, a konceptualni dijagram prikazan je na slici. Definisano je pet različitih rola – poslovni partner, računovođa, banka, menadžer i administrator, sa različitim permisijama.



Korišćen je princip najmanje privilegije, odnosno, vođeno je računa da se nijednoj ulozi ne dodeli više permisija nego što treba da ima. U klasi Application nalazi se funkcija koja vrši autorizaciju. Ona se poziva na početku svake metode kontrolera. Prvo se proveri da li postoji ulogovan korisnik, zatim, ukoliko postoji proverava se njegova uloga. Poredi se spisak permisija te uloge sa zadatom šifrom operacije koju metoda kontrolera vrši. Ukoliko se ona ne nalazi među permisijama posmatrane uloge, korisniku se ne dozvoljava pristup. Korisnik dobija poruku *401 – Unauthorized*.

Na slici ispod je deo kontrole pristupa u frontendu uz pomoć Play tagova – dugmad koja vode ka funkcijama za koje nemaju permisiju korisnicima neće biti prikazana.

```
172  #{if (session.get("role") == "administrator" || session.get("role") == "accountant")}
173    <li><a href="@{Invoices.show("add")}" id = "add"><img src="@{'/public/images/add.gif'}"/></a></
174    <li><a href="#" id = "remove"><img src="@{'/public/images/remove.gif'}"/></a></li>
175    <li><a href="#" id = "nextFormInvoice"><img src="@{'/public/images/nextform.gif'}"/></a></li>
176    <li><a href="#" id = "export_to_xml"><img src="@{'/public/images/xml.png'}"/></a></li>
177    <li><a href="#" id = "invoiceReport"><img src="@{'/public/images/reporta.png'}"/></a></li>
178    <li><a href="#" id = "dateReport" data-toggle="modal" data-target="#dateReportModal"><img src="
179    #{/if}
180  </ul>
```

A5 – Security Misconfiguration

Aplikacija, web server, server baze podataka itd treba da budu pravilno i bezbedno konfigurisani, s obzirom na to da su podrazumevana podešavanja često nebezbedna. Softver, DBMS, komponente i biblioteke bi trebalo redovno ažurirati. Potrebno je proći kroz postupak *system hardeninga* i smanjiti površinu za napad. To podrazumeva uklanjanje predefinisanih naloga koji su korišćeni u fazi implementacije i testiranja, isključivanje portova i uklanjanje servisa koji nisu neophodni, brisanje nepotrebnih stranica...

U slučaju greške, *stack trace* ne bi trebao da bude prikazan korisniku, kao ni preterano informativni opisi greške. Time se mogu otkriti bezbednosne mane sistema koje se mogu eksploatisati, poput verzija frameworka za koje je poznato da su ranjive.

A5 – Sensitive Data Exposure

Potrebno je identifikovati koji su podaci u sistemu osetljivi, a zatim ih zaštititi prilikom skladištenja i prilikom tranzita.

Podaci su zaštićeni u tranzitu korišćenjem *HTTPS*-a, kako bi se obezbedila njihova enkripcija. HTTPS ima za cilj da zaštiti i šifruje vezu za prenos podataka preko HTTP protokola pomoću TLS-a ili SSL-a. Ovaj protokol osigurava integritet i poverljivost podataka. Bezbednosni kanal je dvosmerna šifrovana komunikacija sa simetričnim algoritmom (npr. AES). HTTPS ima 2 glavna procesa: rukovanje i bezbednu komunikaciju. U procesu rukovanja koja se dogodi neposredno pre kreiranje sesije, obe strane se dogovore oko algoritma za šifrovanje i oko ključa sesije, što je najbitnije. Ovaj ključ je tajni ključ koji se koristi za šifrovanje sigurnosnog kanala u toku ove sesije. Web Server poseduje par ključeva: javnih i privatnih. Javni ključ se koristi za šifrovanje podataka a privatni ključ za dešifrovanje. Pored toga, privatni ključ se koristi da potpiše podatke a javni ključ se koristi da proveri potpis. Web serveri traže od CA-a zahtev za izdavanje sertifikata na osnovu njihovih javnih ključeva. CA izdaje sertifikat potpisan od strane svojih ključeva. Web server koristi ovaj sertifikat tokom procesa rukovanja. Potvrda identiteta i tačnost web servera vrši se proverom sertifikata i potpisa izdavaoca od strane korisnika(npr. unutar pretraživača).

https://github.com/tamarakatic/NationalBank/blob/master/Company/src/main/java/com/example/Company/HttpConfiguration.java

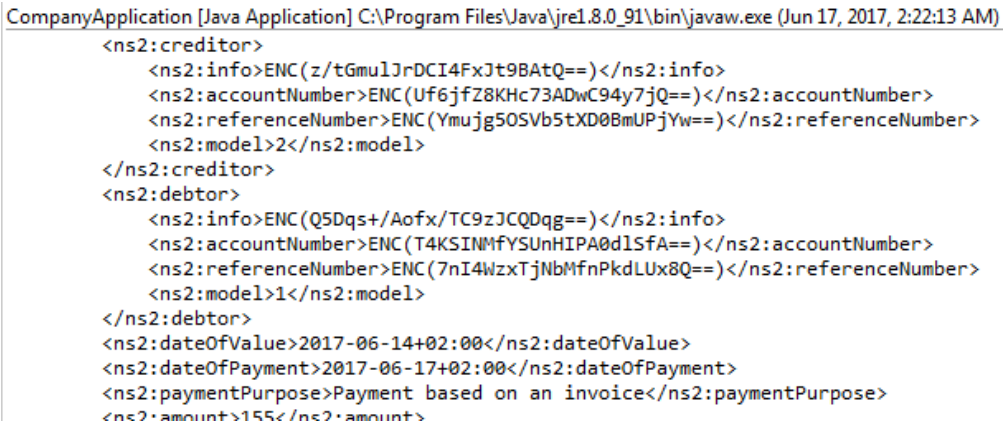
U ovom projektu, korišćen self-signed sertifikate bez podešavanja CA-a. Ukoliko bi aplikacija radila u produkciji bilo bi neophodno koristiti sertifikat, izdat od poverljivog CA-a. Za generisanje self-signed sertifikata korišćen je alat Keytool i Openssl. Komanda: ($ keytool -genkey -alias tomcat -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 4000) . Ključevi se čuvaju korišćenjem PKCS12 fajla za čuvanje privatnih i javnih ključeva sertifikata. Koristeći alat Openssl, postave se osnovni podaci korisnika nakon čega se prikaže RSA privatni ključ i sertifikat. Nakon ovoga se podesi u Spring Boot aplikaciji da koristi HTTPS, odnosno u *application.properties* fajlu se postavi port, keystore fajl, keystore šifra, keystore tip i alias.

https://github.com/tamarakatic/NationalBank/blob/master/Company/src/main/resources/application.properties#L22

Slanje naloga za plaćanje - SOAP

Prilikom slanja naloga za plaćanje od firme ka banci (komunikacija između Spring aplikacija *Company* i *Bank*), bilo je potrebno zaštititi osetljive podatke u tranzitu. Za implementaciju SOAPa iskorišćen je JAXB, pa je u skladu sa tim, za enkripciju XMLa iskorišćen XML Adapter. Njegova implementacija u klasi `EncryptedStringXmlAdapter` će prilikom marshallinga šifrovati podatke, i dešifrovati prilikom unmashallinga. To će se raditi simetričnom šifrom, uz pomoć passworda koji se zada. U korišćenoj verziji (1.9.2), podrazumevani algoritam je *PBEWITHMD5ANDDES*. Pre slanja XMLa, ključ za simetričnu šifru se mora razmeniti između firme i banke, putem REST poziva. Ključ ne sme putovati u otvorenom obliku, i za njegovu zaštitu korišćena je asimetrična šifra. Kada obe strane imaju ključ, vrši se potpisivanje naloga za plaćanje i njegovo slanje. Nakon prijema naloga, banka vrši dešifrovanje, proveru potpisa i validaciju šemom. Kada se pošalje nalog za plaćanje, ključ ističe i ne može se više koristiti.

- Firma kreira session key. Koristi se AES algotitam i dužina ključa od 192b. Učitava se sertifikat banke, i session key se šifruje javnim ključ. Korišćeni algoritam za šifrovanje je `RSA/ECB/OAEPWithSHA1AndMGF1Padding.` Metoda `generateAndEncryptSessionKey()`
- Firma kreira nalog za plaćanje i potpisuje ga privatnim ključem svog sertifikata; Metoda `signWithCert`
- Ključ se šalje banci; Metoda `sendSessionKey`
- Odmah zatim sledi slanje naloga za plaćanje; Prilikom marshallnga vrši se enkripcija; Metoda `sendPaymentOrder`
- Po prijemu naloga za plaćanje, banka proverava potpis na nalogu uz pomoć X509KeySelectora; Ukoliko potpis nije validan, obrada se prekida. Metoda `validateSignature`
- Prilikom umarshallinga se dešifruju podaci koristeći primljeni ključ; Prethodno je ključ dešifrovan; Metoda `decryptSessionKey`
- Banka validira primljeni xml šemom; Ukoliko validacija ne uspe, obrada se prekida. Metoda `validateWithSchema`



Na slici iznad je prikazan izgled naloga za plaćanje kojem su šifrovani podaci o dužniku i poveriocu, a na slici ispod je prikazan izgled digitalnog potpisa koji se dodaje na nalog za plaćanje.

```
CompanyApplication [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (Jun 17, 2017, 2:22:13 AM)
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <DigestValue>aKuGdZUNEn36jBMmUMBbuXvw2Ug=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>a+zdV3wl0ELP8072oyWNB+uVrWmiA1ry0ba0wdsVvAzJ8WEe4iqdbT3BNkTrYByYFniPXBKamTz3
+vKArareXA==</SignatureValue>
        <KeyInfo>
            <KeyValue>
                <RSAKeyValue>
                    <Modulus>gKEpS4BxsrUN32NiurvFOjyxjlw9BqSuzHum5ARnloQLKtR0RjfsWKnFNxqTaKbG+XOwTNm0+NQr
DwGd/OJOlQ==</Modulus>
                    <Exponent>AQAB</Exponent>
                </RSAKeyValue>
            </KeyValue>
```

Slanje faktura - REST

XML dokument se formira koristeći DOM XML parser. Generiše se tajni ključ, koristeći simetričnu šifru – algoritam je Triple DES. Da bi se tajni ključ šifrovao, potrebno je koristiti sertifikat koji se učitava iz KeyStore-a, uz pomoć KeyStoreReader klase. Formirani XML se šifruje prethodno generisanim tajnim ključem. Tajni ključ se šifruje javnim ključem koristeći RSA algoritam. Javni ključ je preuzet sa sertifikata. U XML dokumentu se pronađe element koji sadrži osetljive podatke (u ovom slučaju, invoice element) i izvrši se šifrovanje tog sadržaja.

https://github.com/tamarakatic/NationalBank/blob/master/Company/src/main/java/com/example/Company/service/XMLsecurity/XMLEncryptionUtility.java

Sledeći korak je potpisivanje XML-a. Ovde se za signature objekat vezuje javni ključ koji će se koristiti za proveru digitalnog potpisa. Potpisivanje se obavlja privatnim ključem, koji je učitan iz KeyStore-a. Nakon potpisivanja i šifrovanja XML dokumenta, vrši se transfer tog dokumenta od firme A do firme B.

https://github.com/tamarakatic/NationalBank/blob/master/Company/src/main/java/com/example/Company/service/XMLsecurity/XMLSigningUtility.java

Po prijemu dokumenta, firma B proverava validnost digitalnog potpisa. Pronalazi se Signature element, pa se proverava da li postoji sertifikat. Ukoliko postoji, proverava se potpis. Ukoliko je potpis validan, dokument se dešifruje tajnim ključem. Ako je dokument uspešno dešifrovan, validira se XML šemom.

Osetljivi podaci se ne trebaju skladištiti u otvorenom obliku. Takođe, bitno je da se ne koriste stare ili slabe kriptografske funkcije. Prilikom skladištenja lozinki, potrebna je posebna zaštita, iskazana kroz Hash&Salt mehanizam, o kojem je bilo reči. U slučaju da je neophodno korisnicima obezbediti pretragu po osetljivim podacima, nikada ne treba omogućiti autocomplete polja za pretragu. Dodatno, može se isključiti keširanje stranica koje prikazuju osetljive podatke.

A7 – Insufficient Attack Protection

Potrebno je da aplikacija bude u stanju da detektuje, spreči i odgovori na napade. Aplikacija treba da detektuje ponašanje koje korisnik ne bi trebao da može da izazove, poput ponovljenih zahteva ili netipičnih unosa. Pored toga, aplikacija mora da bude u mogućnosti da što pre odgovori na napade. Za to su bitni log fajlovi.

Posmatrana aplikacija pravi unos u log fajl prilikom obavljanja svake operacije. Struktura logova je : timestamp, kod operacije, ishod operacije, id objekta nad kojim se obavlja operacija, korisničko ime onog koji obavlja operaciju, i ostali relevantni podaci, u zavisnosti od operacije. Pored ovakvih unosa, u log fajl unose se greške i pokušaji prijave na sistem. Prilikom prijavljvanja na sistem, kao što je prikazano na slici, loguje se IP adresa sa koje je došao zahtev, čime se omogućava praćenje zahteva i eventualno blokiranje IP adresa napadača.

```
static boolean authenticate(String username, String password) {
    Application.logIPToFile(request.remoteAddress);
    User user = User.find("byUsername", username).first();
```

https://github.com/tamarakatic/EnterprisePayments/blob/master/app/controllers/Application.java

A8 – Cross Site Request Forgery

CSRF napad primorava browser žrtve da šalje HTTP zahtev, koji uključuje žrtvin session cookie i ostale uključene podatke, ranjivoj web aplikaciji, koja pretpostavlja da su to legitimni zahtevi od strane žrtve. Ovakav napad može se sprečiti CSRF tokenima i pravilnom upotrebom GET i POST zahteva.

Vođeno je računa da se stanje aplikacije menja samo putem POST zahteva. U forme za unos podataka ugrađeno je nevidljivo polje koje sadrži CSRF token. Na početku svake metode kontrolera koja koristi podatke iz forme, pozivana je Play-eva metoda *checkAutenticity()* koja proverava validnost tokena.

```
<form action = "${action}" method = "POST">
        #{authenticityToken /}

        #{field 'invoice.id'}
            <input type = "hidden" name = "${field.name}" id = "${field.id}"/>
        #{/}
```

A9 – Using Components with Known Vulnerabilities

Ranjivosti eksternih biblioteka koje aplikacija koristi mogu pružiti pristup samoj aplikaciji. Nakon pokretanja OWASP Dependency Check-a na aplikaciji „EnterprisePayments“, rezultati su sledeći – ne postoji nijedna ranjiva komponenta. Kod sistema koji su pušteni u produkciju, periodično bi se trebao koristiti dependency check, a ranjive biblioteke zamenjivati verzijama u kojima ranjivosti ne postoje. Ukoliko nije moguće izbeći korišćenje ranjivih delova koda, trebali bi se napraviti wrapperi.

A10 – Underprotected APIs

Kao i većina modernih aplikacija, i posmatrana aplikacija uključuje APIje. JavaScript u browseru se povezuje na REST, za komunikaciju sa drugim firmama koristi se REST, a za komunikaciju sa bankom koristi se SOAP. Dokumenti koji se razmenjuju putem SOAPa su šifrovani i potpisani. SOAP Parser koristi šemu za validaciju, da bi se sprečili XML injection napadi. Preko SOAPa dokumente mogu slati samo autorizovani korisnici.