TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut
Infosüsteemide õppetool

Turvalise veebirakenduse disain

Bakalaureusetöö
IDU40LT

Üliõpilane: Ivo Pure
Üliõpilaskood: 104269
Juhendaja: Raul Liivrand

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

(*kuupäev*)	(*allkiri*)

Annotatsioon

Töö eesmärk on tõsta esile turvalisuse tähtsus kogu selle arendustsükli jooksul kasutades selleks näidis veebirakendust, mis on arendatud kasutades läbistusteste rakenduse haavatavusi skaneerides. Tuua esile logide tähtsust ning analüüsida erinevaid logide kasutamisvõimalusi. Töös tegeletakse turvalisuse tähtsuse esile toomisega ning leitakse võimalusi erinevate rünnakute vastu võitlemiseks. Tulemuseks saab öelda, et ilma haavatavuste skaneerimisteta ei ole võimalik luua turvalist rakendust. Lisaks, logide puudumisel ei ole võimalik tuvastada pahalase tegusid ning rakendada aktiivset kaitset veebirakendusele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 39. leheküljel, 6 peatükki, 11 joonist, 6 tabelit.

Abstract

The purpose of this paper is to emphasize security during the entire software development cycle. Therefore a test web application is created and used to demonstrate how to secure it using vulnerability scanners. The second purpose is to emphasize the importance of application logging, log files and the benefit of both of them. Most of the work is done to emphasize security in general and find possibilities to secure a web application from hackers. The result of this paper aims to be that without using penetration testing tools to scan for vulnerabilities, there is no real way to build a secure web application. Furthermore, without the presence of log files there does not exist a way to do digital forensics and render through logging active defense mechanisms.

The thesis is in Estonian and contains 39 pages of text, 6 chapters, 11 figures, 6 tables.

Lühendite ja mõistete sõnastik

RFI	*Remote File Inclusion* Kaugfailisüsti rünnak.
AJAX	*Asynchronous JavaScript and XML* Asünkroonne JavaScripti- ja XML-põhine veebiarendustehnika.
CGI	*Common Gateway Interface* Ühine lüüsi liides ehk serveripoole andmete sisestus – väljastus koht
PHP	*Hypertext Preprocessor* Programmeerimiskeel.
MYSQL	*MY-Structured Query Language* Vabavaraline struktureeritud päringu keel.
JSON	*JavaScript Object Notation* Andmevahetusformaat.
ZAP	*Zed attack proxy* Läbistustestimis rakendus
OS	*Operating System* Operatsioonisüsteem.
SQL	*Structured Query Language* Struktureeritud päringu keel
ASP	*Active server pages* Programmeerimiskeel.
SQLi	*SQL Injection* SQL-süst rünnak
LFI	*Local file inclusion*

	Lähifailisüst rünnak
DT	*Directory Traversal* Kataloogihüppe rünnak.
XSS	*cross-site scripting* Skriptisüst rünnak.
HTTP	*HyperText Transfer Protocol* Hüperteksti edastuse protokoll
EmE	*Embedded media encryption* Krüpteeritud kandja manusrünnak
SPAM	*SPAM* Spämm. Soovimatu rämpspost.
SIEM	*security information and event management* Turvateabe ning sündmuste haldus seade või tarkvara
JS	*JavaScript* HTML-lähtekoodiga interakteeruv objektorienteeritud programmeerimiskeel
DOM	*Document Object Model* HTML Interaktiivne tehnoloogia
POST	*(HTTP) POST* Hüperteksti edastuse protokoll „post“ meetod
GET	*(HTTP) GET* Hüperteksti edastuse protokoll „get“ meetod
SNMP	*Simple network management protocol* Lihtne võrguhalduse protokoll võrguseadmete suhtluseks
IDS	*Intrusion Detection System* *sissetungi tuvastuse süsteem*

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

1.1 Taust ja probleem

Informaatika on info hankimist, struktuuri, töötlemist ning esitamist hõlmav teadusharu. Informaatika on lahutamatu osa igapäeva elust. Info on selle tuumaks ning see kuidas seda töödeldakse, edasi kantakse ning esitatakse on väga tähtis. Infotehnoloogia maailmas on informatsioon selle valuutaks. Seega, siit saab järeldada, et informatsiooni turvalisus on üks põhilistest temaatikatest antud valdkonnas. Veebitarkvara nõudluse kasvuga on kaasnenud vajadus rahuldada erinevate asutuste ning organisatsioonide nõudeid uue ning parema tarkvara järgi, mis on viinud tarkvara kvaliteedi ning kvantiteedi kaalust välja. Kuna turvalisus pole olnud algselt prioriteet, siis praod hakkavad ilmnema hiljem. Vundamenti on raske restaureerida kui maja juba püsti on.

1.2 Ülesande püstitus

Töö eesmärk on tõsta esile turvalisuse tähtsus kasutades selleks näidis veebirakendust. Sisuline eesmärk on tuua esile selle tähtsus juba arenduse algfaasides. Lisaks soovib autor lugejate sihtgrupile anda ülevaate suuremast pildist, miks peab loodud rakendus olema liidestav teiste rakendustega. Turvalisuse väljendamiseks esitatakse nii häid kui ka halbu näiteid, tutvustatakse nõrkuseotsingut ning analüüsitake seda tuginedes autori praktilisele lahendusele. Lugejaskonna sihtgrupiks on algajad arendajad ning töö loomuse eesmärk on õppematerjal.

1.3 Metoodika

Töös keskendub autor põhiliselt veebirakenduse serveri poole turvalisuse temaatikale. Arendatakse näidisrakendus, mille kliendi poolel kasutatakse AJAX arendusmeetodit. Serveri poolel CGI programmeerimiskeeleks on valitud PHP keel, mis on liidestatud MYSQL andmebaasiga. Andmevahetuseks üle arvutivõrgu kasutatakse JSON andmevormingut.

Veebirakenduse turbeanalüüsiks ning rünnete vastu kaitsemehhanismi arendamiseks kasutatakse ZAP rünnaku proksit.

1.4 Ülevaade tööst

Töö teises peatükis annab autor ülevaate töö sisust ning kirjeldab probleeme tuginedes statistikale. Lisaks kirjeldatakse peatükis ka töö nõudeid.

Töö kolmandas peatükis antakse ülevaade rakenduse arhitektuurist. Kirjeldatakse selle funktsionaalsust kliendi kui ka serveri poolelt. Parema ülevaate saamiseks kirjeldatakse arhitektuur jooniste ja skeemidega.

Töö neljandas peatükis kirjeldatakse näidete abil erinevad rünnakuid ning nende vektoreid. Näidatakse rakenduse funktsioonide abil, kuidas neid tõrjuda.

Töö viiendas peatükis kirjeldatakse lühidalt, kuidas ning miks võiks rakendust edasi arendada.

2. Probleem ning nõuded

2.1 Ülevaade

Veebirakendus on klient-server-arhitektuuriga rakendusprogramm, milles kliendiks on veebibrauser, serveriks veebiserver. Üldiselt andmeid hoitakse ja töödeldakse serveris ning andmevahetus toimub arvutivõrgu kaudu [1].

Veebirakendused on praeguseks hetkeks üheks levinumateks rakendusprogrammideks. Algselt olid veebilehed staatiliselt ning ainult sirvimiseks mõeldud. Nüüdseks on veeb arenenud dünaamiliseks andmete esitlus, töötlus ning transportimis keskkonnaks. Selle edu, võrreldes spetsiifiliste rakenduste platvormidega, tagab dünaamilisus. Sõltumata operatsioonisüsteemist (OS) või seadme tüübist on veebi sirvimine platvormist sõltumatu. Platvormid varieeruvad, alates väikse eraldusvõimega ning vähese jõudlusega mobiiltelefonidest kuni suurema eraldusvõimega ning suurema jõudlusega tahvelarvutiteni. Lisaks juba levinud sülearvutid ning lauaarvutid [2].

2.2 Probleemi kirjeldus

Elame ühiskonnas mille ajajärku nimetatakse informatsiooni ajastuks. Informatsioon on selle keskseks valuutaks. Kõikvõimalikud toimingud ning tegevused saab tänapäeval läbi veebirakenduste sooritada. Eesmärgiga pakkuda võimalikult palju funktsionaalsust ning teenida võimalikult palju kasumit asutusele on veebirakendused viidud väga kõrgele visuaalsele ning teenuse kvaliteedi tasemele. Problemaatiline selle juures on turvalisuse teisejärguliseks jätmine [6].

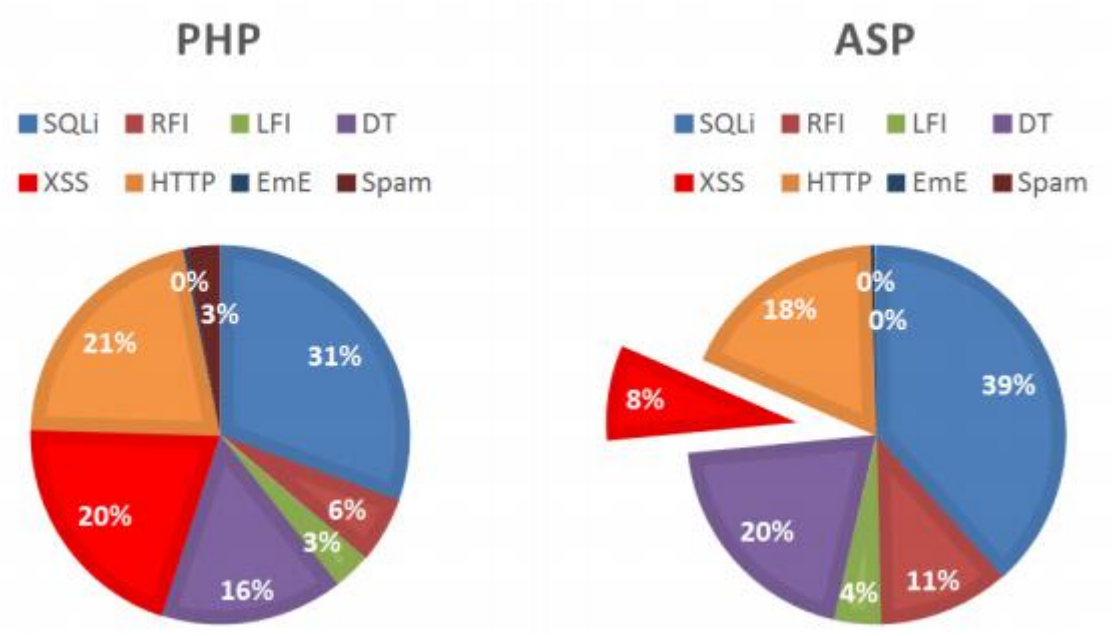
Veebiarendus on märkimisväärselt suur ning levinud äri. Arendusfirmad toodavad tarkvara projekti tähtaegade alusel ning prioriteediks on töötav ning jõudluse kasutamise aspektist kiire lahendus. Tähtis on tellija jaoks välimuselt meeldiv ning funktsionaalselt töötav lahendus, mis jõuab temani võimalikult kiirelt. Kuna konkurente on palju, siis arendusfirmad on sunnitud kirjeldatud viisil liikuma. Turvalisuse aspekte võetakse minimaalselt arvesse, kuna see nõuab lisaaega. Seda aega ei tule juurde ega ka kasumlikkuse hoidmiseks ei kulutata turvalisuse jaoks väga palju rohkem ressursse. Kliendid on sunnitud ostma turbeseadmeid,

mis nende asutuse veebiteenuseid kaitseksid. Kõik see on leevendus asjaolule, et veebirakendus on haavatav. Probleemile tuleb läheneda juurtest alates [16].

Koolides programmeerimist õpetades, tuleks igal võimalikul viisil kaasata turbe aspektid. Turvalise koodi tavade sisse harjutamine oleks juba eos tarvilik. Samuti peaks analoogne tava kajastuma iga firma programmeerimis etiketis. Kui turvalise koodi kirjutamise tavad on välja arendatud, tuleb neid regulatsioonide või nõuete alusel rakendada.

Koodi testimine peaks jagunema vähemalt kahte peamisse kategooriasse, funktsionaalsus ning koodi turvalisus. Need kaks testi faasi ei tohiks olla eraldatud, kuna rünnakute kaasmõjul võib programm käituda ettearvamatult. Esimene samm on kaasata turbetestimine koodi testimise plaanidesse. Testimine peaks algama funktsioonidest, liikuma klassideni ning sealt edasi teekideni [7].

Kuna PHP on kõige populaarsem veebiserveri arendus keel, siis ka selle vastu rünnakute vektorite hulk omab proportsionaalset suurust. „Imperva“ iga aastane veebirakenduste rünnete raport toob välja tähtsamad muutused küberrünnakute vektorites, intensiivsuses ning sihtmärkide valiku vahel. Võrreldes aastat 2014 aastaga 2013, tõusid SQL süstrünnakud 10 % ning RFI süstrünnakud 24%. Teiste vektorite muutus jäi üksikute protsentide juurde. Joonisel on võrdluseks võetud kaks äärmuslikku ning enim kasutatud raamistikku, PHP ja ASP [3].



Joonis 1. PHP ning ASP raamistiku võrdlus rünnakute suhtes [3].

- PHP rakendused kannatavad ligi kolm korda rohkem XSS rünnakute all kui ASP rakendused
- PHP rakendused kannatavad ligi kaks korda rohkem DT rünnakuid kui ASP rakendused
- PHP rakendused kannatavad ligi kaks korda vähem SQL süstrünnakute all kui ASP rakendused

Üldine pahatahtlik liiklus arvutivõrkudes kasvab aasta-aastaga, rünnakute kestvus ning intensiivsus koos sellega. Aasta 2013 statistika seisust kuni aasta 2014 seisuni on müügiga tegelevad asutuste veebilehed olnud suurimaks sihtgrupiks, millele on suunatud:

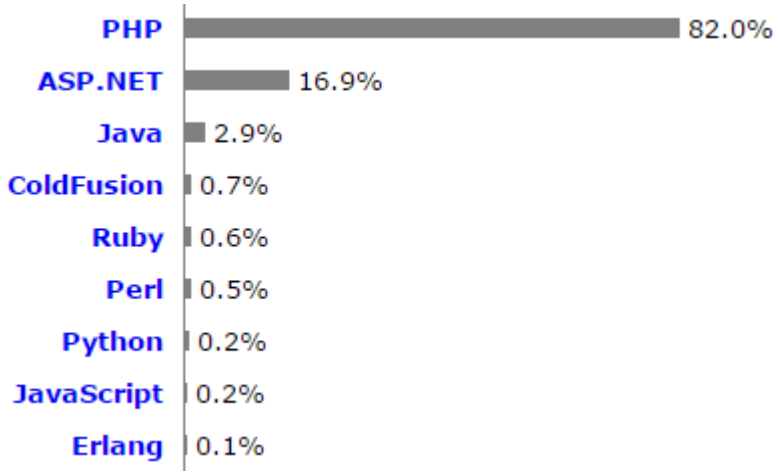
- 48,1% kogu rünnakukampaaniatest
- 40% kogu SQL süstrünnakutest
- 64% kogu halvaloomulisest HTTP liiklusest

Rünnakud on intensiivsed, laiemate vektoritega ning eesmärgipärased, millest tulenevalt tuleb veelgi suuremat rõhku panna veebirakenduste turvalisusele [3].

2.3 Nõuete kirjeldus

2.3.1 Asjakohasus

Autori eesmärk on esitada võimalikult paljudele lugejatele oma lahendust ning ideed. Selle teostamiseks on nõue kasutada ühte populaarsemat programmeerimiskeelt. Selleks on näidisrakenduse serveri pool disainitud PHP keeles. „w3techs“ hetke seisuga on kõige populaarsemaks serveri poolseks programmeerimiskeeleks PHP [5].



Joonis 2. Serveri poolne programmeerimis keelte kasutus [5].

2.3.2 Haavatavuste tuvastamine

Kuna ühel rünnakutüübil (nt SQL süst) on väga palju alamvektoreid, siis on nõue kindlustada rakendus haavatavuste skaneerimiste alusel. Ründeliigid ning nendest vähemalt üks vektor, mille vastu rakendus peab kaitstud olema, on välja toodud töö lõpus (vaata lisa 1) ning kirjeldatud teemas neli punkt kaks. ZAP ründe proksi on kergesti kasutatav ning integreeritav läbistustestimis rakendus haavatavuste avastamiseks veebirakendustes. Rakendus on disainitud infoturbespetsialistidele kuid omab ka juba seadistatud ründe komplekte, mida saavad kasutada tavalised arendajad ning tarkvaratestijad. Ründekomplektid koosnevad hetkel populaarsematest erinevatest ründevektoritest [9].

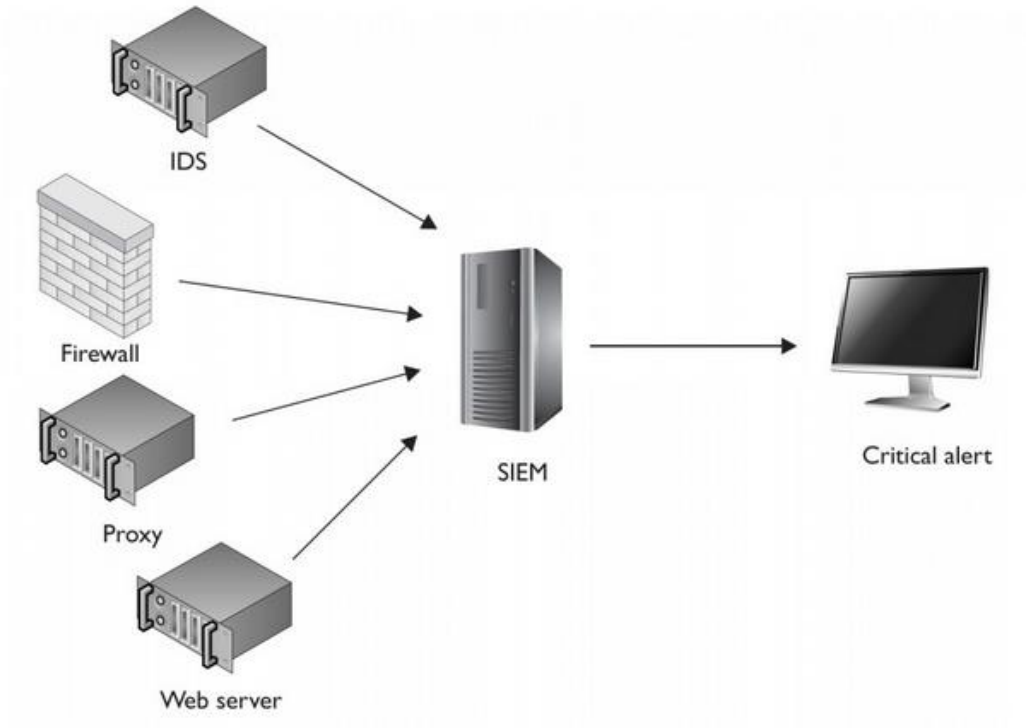
Autor kasutab töös ainult aktiivse skaneerimise moodulit, seega kirjeldatakse ainult seda funktsiooni. Aktiivne skaneerimine tähendab reaalsete rünnete sooritamist, seetõttu ei tohi ilma nõusolekuta ühegi sihtmärgi suunas läbistustesti sooritada. Juba töötavat veebirakenduse keskkonda rünnates võib see kokku joosta, andmeid lekitada ning seeläbi asutusele tekitada märkimisväärset rahalist või mainekahju. Aktiivne skaneerimine tuvastab ainult nõrkused haavatavuste vastu [10]. Loogilised süsteemivead ning muud rakenduse nõrkused jäävad tööskoobist välja.

2.3.3 Logihaldus

Sajandi alguses polnud tarkvara sündmuste logimine ning logide haldamine levinud tava. Logimine lülitati sisse, kui tekkis mingi probleem tarkvaraga või selle ühendusega

serveri suunal ehk tegu oli passiivse logimise vormiga. Seoses kogu infotehnoloogia valdkonna arengu ning kasutajate hulga suurenemisega jäi antud vorm logimisest jalgu. Vaja oli aktiivset pidevat logimist ning selle haldamist, ehk keskset logikollektorit. Pelgalt logide kogumine ei viinud kuhugi, tekkis vajadus organiseerituse ning keskse süsteemi järgi. Aastal 2009 jõudis infotehnoloogia maastikule SIEM, mis omab alljärgnevaid põhifunktsioone [4]:

- Erinevatest logiallikates info kogumine
- Korrelatsioon - erinevate allikate ühiste võrdsete omaduste sidumine
- Andmete koondamine valitud parameetri alusel
- Aktiivne hoiatus sündmuste künnise ületamisel [4]



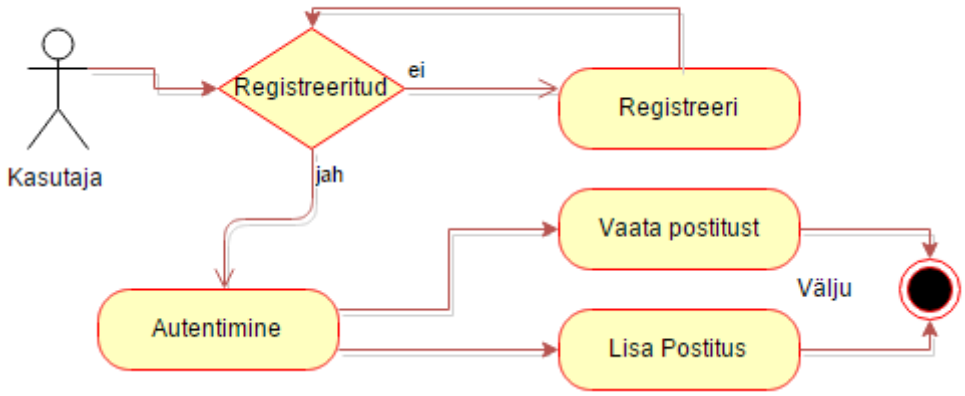
Joonis 3. Loogiline SIEM serveri paiknemine arvutivõrgus [8].

Kuna autor keskendub töös turvalisuse aspektidele, siis parametriseeritud turbelogide olemus on nõutud. Logide vorming ning struktuur peab võimaldama selle info saatmist SIEM rakendusse. Lisaks annab joonis ülevaate ühest võimalikust loogilisest võrgustruktuurist ning seadmete loogilistest asukohtadest arvutivõrgus.

3. Näidis Veebirakendus

3.1 Ülevaade loodud rakendusest

Rakenduse kliendi poolne keerukus on väga lihtne. Samuti on lihtsustatud andmete edastamine, kirjutamine ning lugemine andmebaasist. Tulenevalt püstitatud eesmärgist koostada õppematerjal algajale arendajale on põhirõhk siiski turbeelementide olemus. Lihtsa näite alusel keerukaimaid turbemeetmeid tutvustades suudab lugejate sihtgrupis olev isik väiksema vaevaga aru saada töö eesmärkidest. Näidisrakenduse funktsionaalsusest piisab kõikide töö eesmärgis nõutud rakenduste haavatavuste tutvustamiseks.



Joonis 4. Tegevusdiagramm.

Kasutaja saab ennast registreerida, autentida ning seejärel vaadata postitusi ning ka neid lisada.

Kasutusjuhud	Funktsionaalsus
Registreerimine	Kasutaja sisestab kasutajanime, parooli ning emaili. Päring saadetakse veebiserverile töötlemiseks. Kui kasutajanimi on saadaval, registreeritakse kasutaja.
Postituse lisamine	Kasutaja täidab nõutud väljad ning vajutab nupule postita. Päring saadetakse veebiserverile. Postitus lisatakse andmebaasi

	ning kuvatakse esilehel koos teiste juba lisatud postitustega
Postituste vaatamine	Kasutaja valib soovitud postituse ning vajutab selle identifitseerimis numbriga märgistatud nupule. Päring saadetakse veebiserverile töötlemiseks ning veebiserver tagastab informatsiooni postituse kohta
Autentimine	Kasutaja sisestab kasutajanime ning parooli. Päring saadetakse veebiserverile töötlemiseks. Korrektsete andmete korral logitakse kasutaja sisse.

Tabel 1. Kasutusjuhtude kirjeldus.

3.2 Kliendi poolne arhitektuur

Kliendi poolne lahendus on kirjutatud HTML kujunduskeeles ning JS programmeerimiskeeles. JS lihtsustamiseks kasutatakse serveriga suhtlemiseks „jQuery“ rakendusliidest. Lisaks kasutatakse visuaalsete efektide jaoks „Pop“ [11] „jQuery“ lisamoodulit. HTML ja JS on eraldatud loetavuse parandamiseks.

Enne andmete edastamist serveritele kontrollitakse need puhastusprotsessis. Esimene tase on HTML5 [12] omaduste kasutamine. Tühja sisendi vältimiseks kasutatakse atribuuti „required“ ning atribuuti tüüp, mis piirab sisendi formaati. Teine tase on JS skripti tasemel kontroll. JS ootab HTML vormi „submit“ sündmust, mis tähendab kasutaja mõistes vormi kinnitamist ja saatmist. Alustuseks kontrollitakse sisendite pikkust, vältimaks tühja sisendit. Järgmiseks kontrollitakse sisendeid regulaaravaldisega [13].

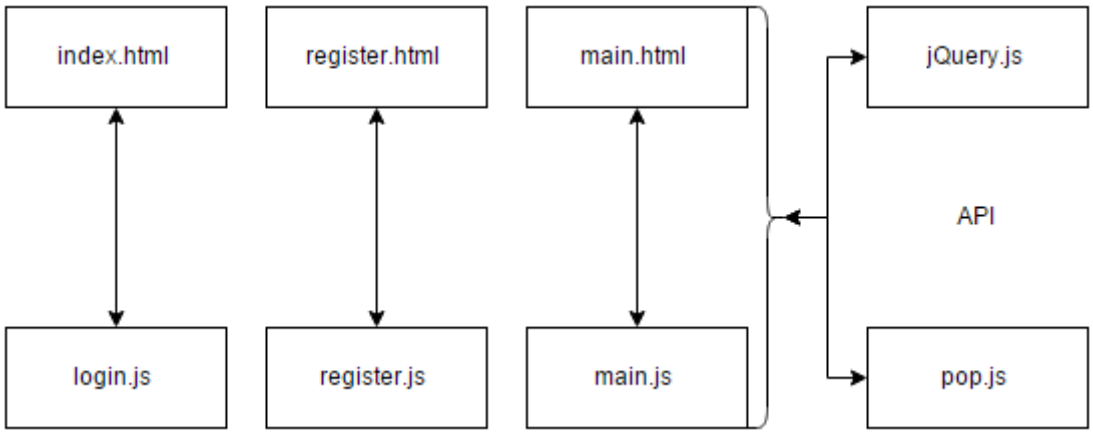
```
$('form.login-form').on('submit',function(e) {
    ...
    var name_regex = /^[a-zA-Z0-9]{3,20}$/;
    ...
//Checking the input against regular expression

if(!data['username'].match(name_regex)||!data['password'].match(name_regex)
){
```

```
        alert('Invalid username/password [a-z, A-Z, 0-9]');
        return false;
    }
```

Joonis 5. Sisendi kontroll kliendi poolel.

Peale andmete kontrolli edastatakse need läbi „jQuery“ AJAX funktsiooni serverisse. Andmed edastatakse serverisse kasutades JSON andmevormingut ning tagastatakse samas vormingus. Edasi manipuleeritakse „jQuery“ abil HTML DOM elemente, mis tagab veebirakenduses dünaamilise andmete esitamise.



Joonis 6. Funktsionaalsuse seosed klient poolel.

Faili nimi	Funktsioon	Funktsionaalsuse selgitus
login.js	form.login-form.on(submit)	HTML vormist andmete vormistamine andmemassiivi. Selle käigus väljade väärtuste kontroll regulaaravaldisega, tühja ning vigase sisendi suhtes. Tagasiside andmine vigase sisendi korral. Andmemassiivi teisendamine JSON andmeformaati. JSON andmete edastamine serverile AJAX päringuga. Vastuse ootamine serverilt. Tagasiside andmine ebaõnnestumise korral. Õnnestumise korral pealehele suunamine. Küpsise seadmine väärtustades selle kasutajanimega.
register.js	form.register-form.on(submit)	HTML vormist andmete vormistamine andmemassiivi. Selle käigus väljade väärtuste kontroll regulaaravaldisega, tühja ning vigase sisendi suhtes. Tagasiside andmine vigase sisendi korral. Andmemassiivi teisendamine JSON

		andmeformaati. JSON andmete edastamine serverile AJAX päringuga. Vastuse ootamine serverilt. Tagasiside andmine ebaõnnestumise korral. Õnnestumise korral autentimislehele suunamine.
main.js	document.ready()	Kasutajanime sisaldava küpsise kontroll. Ebaõnnestumise korral autentimiselehele suunamine. Õnnestumise korral AJAX päringu saatmine serverile. Päritakse kõiki sisestatud postitusi. JSON andmete teisendamine andmemassiivi. Õnnestumise korral nende väljastamine HTML-i.
main.js	logout()	AJAX päringu saatmine serverile sessiooni lõpetamiseks. Õnnestumise korral kasutajanime sisaldava küpsise kustutamine ning pealehele suunamine.
main.js	popAddPost()	DIV elemendi esile tõstmine koos alamelementidega.
main.js	openView(var nr)	Kasutaja nime sisaldava küpsise kontroll. Õnnestumise korral AJAX päringu saatmine serverile. Päritakse funktsiooni sisendparameetris sisaldavat postituse numbrit. JSON andmete teisendamine andmemassiivi. Õnnestumise korral selle sisu väljastamine HTML-i.
main.js	form.sendForm.on (submit)	HTML vormist andmete vormistamine andmemassiivi. Selle käigus väljade väärtuste kontroll regulaaravaldisega, tühja ning vigase sisendi suhtes. Tühikute eemaldamine sisendi lõpust ning algusest. Tagasiside andmine vigase sisendi korral. Andmemassiivi teisendamine JSON andmeformaati. JSON andmete edastamine serverile AJAX päringuga. Vastuse ootamine serverilt. Tagasiside andmine ebaõnnestumise korral. Õnnestumise korral pealehele suunamine.

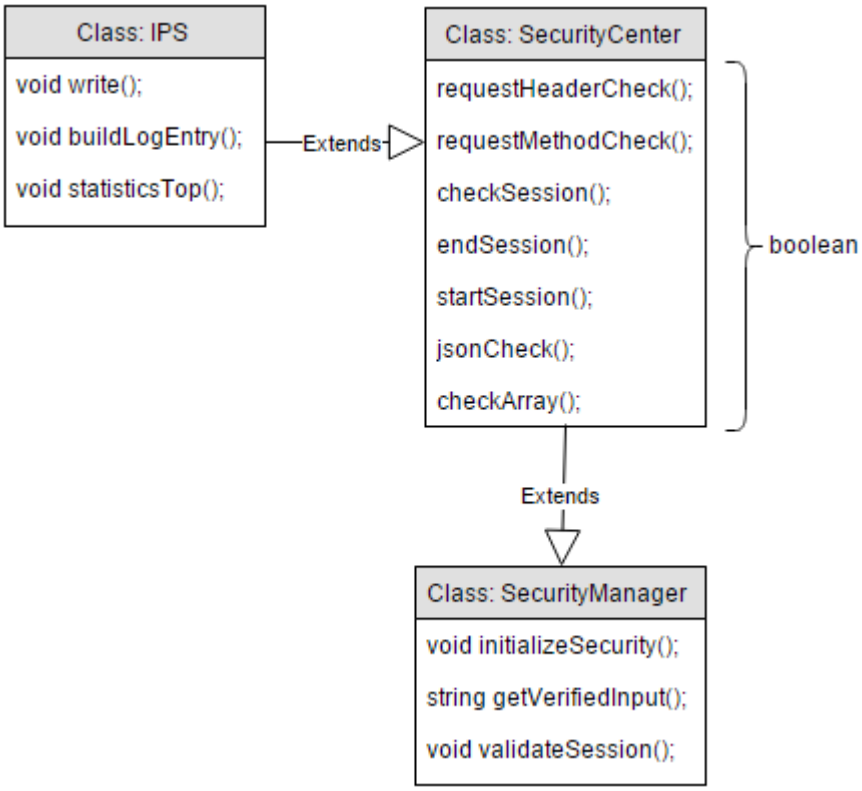
Tabel 2. Klient poole JS funktsioonide kirjeldus.

3.3 Serveri poolne arhitektuur

Serveri poolne arhitektuur on disainitud PHP programmeerimiskeeles. Iga kasutusjuhu jaoks on eraldi CGI funktsiooni fail. CGI funktsioonid on vahekihiks andmebaasi ning kliendi

vahel. Kui kliendi pool kontrolliti ainult andmete kehtivust ja õigsust, siis serveri pool teostatakse kogu päringu kontroll, sh päis ning selle last. Kirjeldatud tegevuse eest vastutab „SecurityManager“ klass, mis on disainitud tõrjuma ning logima rünnakuid. Kui päring on jõusolev ning kontrollitud, vastutavad PHP PDO [14] funktsioonid andmebaasiga suhtlemise eest.

„SecurityManager“ on väike raamistik, mis on arendatud kasutades ZAP ründe proksi haavatavuste teste. Testide sisuks on praegusel hetkel üldlevinud ründevektorid. Nimetatud raamistik suudab tuvastada sessiooni üle võtmisi, võrreldes iga päringu korral selle parameetri väärtusi kasutaja autentimise hetkest. HTTP GET päringute korral toimub parameetri ning selle väärtuse kontroll. POST meetodi korral on lubatud ainult staatilised päringud, mida arendaja saab seadistada. Lisaks tuvastatakse POST meetodist „referer“, „useragent“ ning „x-requested-with“ parameetrite vigased väärtused. Üldiselt kasutatakse esimese kahe parameetri väärtusi statistika kogumiseks ning viimast parameetri väärtust veendumaks, et tegu on AJAX päringuga. Kuna POST meetodi tähtsaim osa on siiski last, siis on ka sellele mõeldud autori poolt loodud raamistikus. Lastis kontrollitakse esmalt, kas tegu on JSON andmevahetus formaadiga, et selle struktuuri võltsitud pole. Lisaks sellele kontrollitakse võtme välja väärtuse ning sellele vastava andmevälja väärtuse vastavust tingimustele, mis normaalse ranguse astme korral lubavad numbreid, tähti ning mõningaid sümboleid. Karmima astme korral ainult number-tähe kombinatsioone. Selleks teisendatakse JSON andmevahetusformaadist informatsioon massiivi. Veel enam, halvaloomulise päringu tuvastamise korral logitakse kogu HTTP päis koos selle lastiga, kui tegu on POST meetodiga. Selle tegevuse eesmärk on salvestada pahatahtlik tegevus ning hoida andmeid rünnaku analüüsimiseks, mis hiljem annavad süsteemiadministraatorile võimaluse probleemiga tegeleda. Täienduseks eelnevale on logifailid struktureeritud kujul, mis võimaldab need näiteks SIEM saata, mis omakorda annavad süsteemiadministraatorile võimalusi neid korrelatsiooni viia teiste allikate logidega ning logi parameetreid koondada ühe parameetri alusel. Raamistik loob ka statistika faili, kus hoitakse edetabelit ründajate võrgu aadressidest rünnakute kordade alusel.



Joonis 7. „SecurityManager“ raamistiku Struktuur.

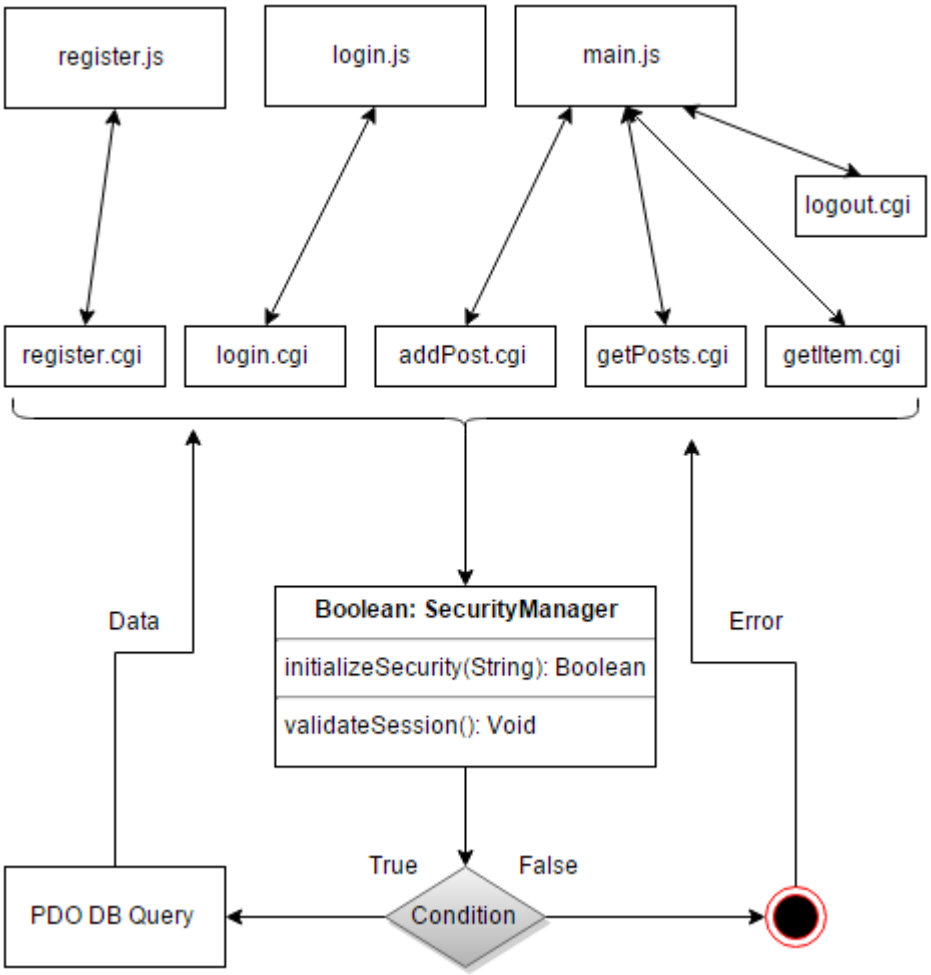
Kood	Klass	Funktsioon	Kirjeldus
[a1]	IPS	Write(string, string)	Kirjutab logirea logifaili. Esimene sisend on rünnet kirjeldav string. Teine sisend on valikuline, kui päisel on last, siis kirjutatakse ka selle vigane osa (muutuja või selle väärtus).
[a2]	IPS	buildLogEntry(string, string)	Koostab logirea, mis hiljem kirjutatakse logifaili. Esimene sisend on rünnet kirjeldav string. Teine sisend on valikuline, kui päisel on last, siis kirjutatakse ka selle vigane osa (muutuja või selle väärtus).
[a3]	IPS	statisticsTop()	Muudab statistika faili. Kui faili pole, luuakse fail. Statistika failis hoitakse edetabelit ründavatest IP aadressidest ning selle IP rünnete arvust.

[b1]	SecurityCenter	requestHeaderCheck()	Esimene kontroll on HTTP päise „referer“ välja väärtuse verifitseerimine. Kontroll toimub regulaaravaldisega, mis lubab ainult standardseid URL-e. Teiseks vaadatakse „requested_with“ välja väärtust. Kuna kliendi pool on AJAX funktsiooniga siis peab ka päises olema „xmlhttprequest“. Viimaseks vaadatakse välja „userAgent“. Selle väärtust kontrollitakse lubatud brauserite massiivist, kas seal on päringu saatja brauser olemas.
[b2]	SecurityCenter	requestMethodCheck(string, string)	Kontrollib GET ja POST päringuid. Esimest ja teist sisendi väärtust on vaja GET jaoks. Esimeseks sisendiks on päringus küsitava GET parameetri nimetus. Parameetri väärtuse pikkus peab olema vähemalt üks tähemärk ning selle väärtust kontrollitakse regulaaravaldisega. POST puhul kontrollitakse päringu URI väärtust lubatud päringute massiivist.
[b3]	SecurityCenter	checkSession()	Kontrollitakse sessiooni kehtivust. Kui päringu päistes on erinevusi sisse logimisest alates, on võimalus, et sessioon on üle võetud ning see katkestatakse päisest erinevuste leidmisel.
[b4]	SecurityCenter	endSession()	Lõpetatakse sessioon.
[b5]	SecurityCenter	startSession()	Algatatakse sessioon.
[b6]	SecurityCenter	jsonCheck(string)	Kontrollitakse JSON andmevormingu

			õigsust. Sisendiks oodatakse JSON stringi. Struktuuri võltsimisi kontrollitakse regulaaravaldisega.
[b7]	SecurityCenter	checkArray(string[], string)	Kontrollitakse massiivi võtme väärtusi kui ka võtmele vastava välja väärtusi. Esimene sisend on massiiv mida kontrollida ning teine sisend seab kontrollimise valiku (range, normaalne). Võtme kui ka sellele vastava välja väärtused kontrollitakse regulaaravaldisega.
[c1]	SecurityManager	initializeSecurity(string)	Sisendiks andmemassiivi kontrollimiseks ranguse aste (range = jah / normaalne = ei). Kutsutakse järjest välja „SecurityCenter“ klassi kontrollfunktsioonid. Ebaõnnestumisel sulgetakse ühendus.
[c2]	SecurityManager	getVerifiedInput()	Tagastatakse kehtiv ning kontrollitud sisend.
[c3]	SecurityManager	validateSession()	Kontrollitakse kehtivat sessioon. Ebaõnnestumisel sessioon lõpetatakse ning ühendus sulgetakse.

Tabel 3. „SecurityManager“ raamistiku funktsioonide kirjeldus.

CGI funktsionaalsus on hajutatud kujul, st iga toimingi jaoks on eraldi fail. Kihtide eraldamine ning eraldi failide olemasolu säästab arendajat „if“ ning „case“ lausete üleküllusest ning annab parema ülevaate. Lisaks on lihtsam vigu tuvastada ning eraldatus annab veebirakendusele suurema töökindluse. Ühes failis viga lõpetab parseri töö üle kogu selle faili.



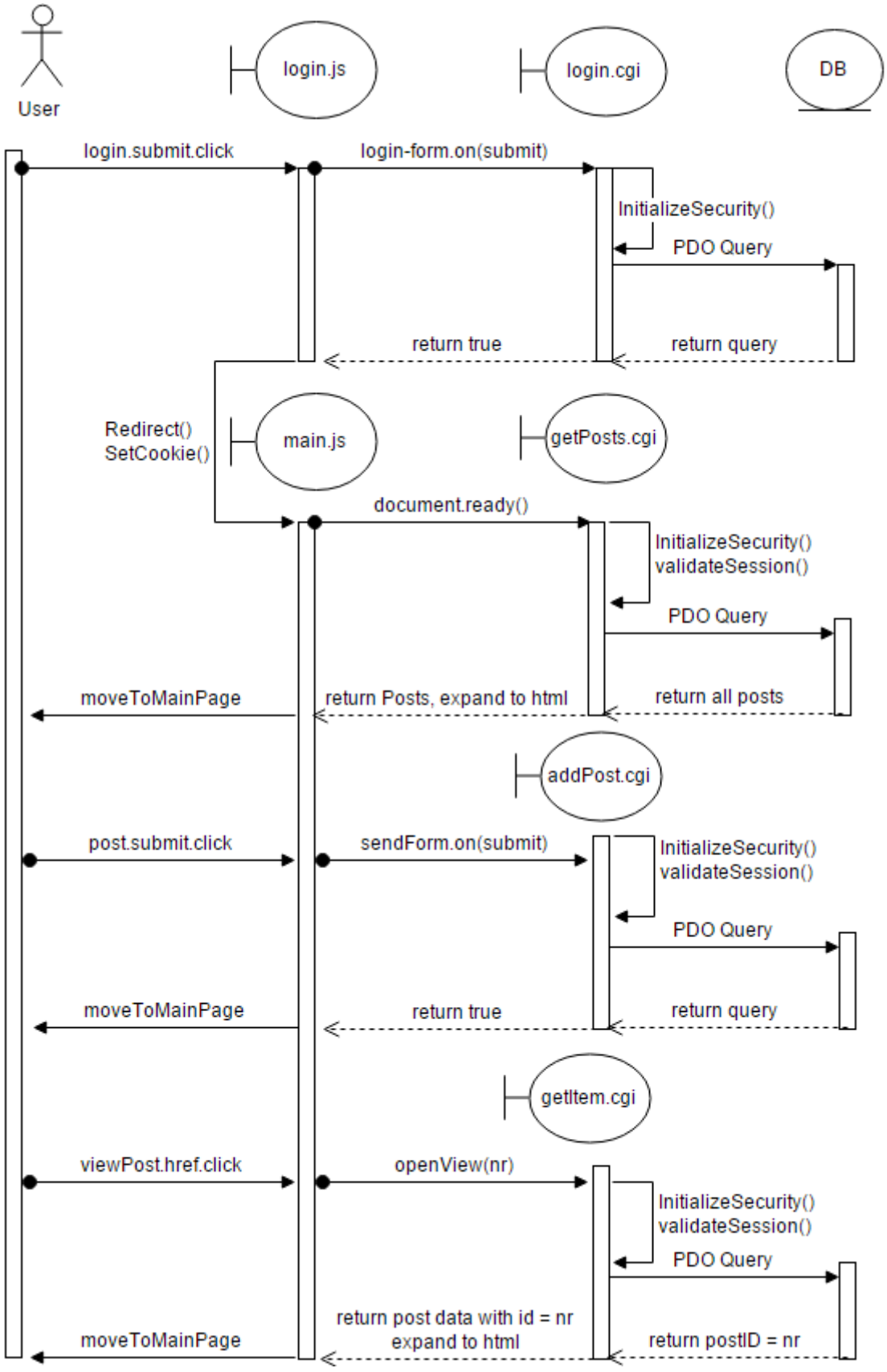
Joonis 8. Funktsionaalsuse seosed serveri poolel.

Faili nimi	Funktsionaalsus
Register.cgi	Kutsutakse välja „Securitymanager“ klassi põhifunktsioon. Korrektse päringu korral küsitakse sellelt verifitseeritud sisend. Kontrollitakse andmemassiivi väärtuste olemasolu. Paroolile lisatakse sool ning sellest kombinatsioonist võetakse räsi. Räsi ning kliendi poolt saadetud andmed kirjutatakse andmebaasi. Õnnestumise ning ebaõnnestumise korral antakse tagasiside.
addPost.cgi	Kutsutakse välja „Securitymanager“ klassi põhifunktsioon. Korrektse päringu korral küsitakse sellelt verifitseeritud sisend. Kontrollitakse andmemassiivi väärtuste ning sessiooni olemasolu. Kliendi poolt saadetud andmed kirjutatakse andmebaasi. Õnnestumise ning ebaõnnestumise korral antakse tagasiside.
Login.cgi	Kutsutakse välja „Securitymanager“ klassi põhifunktsioon. Korrektse

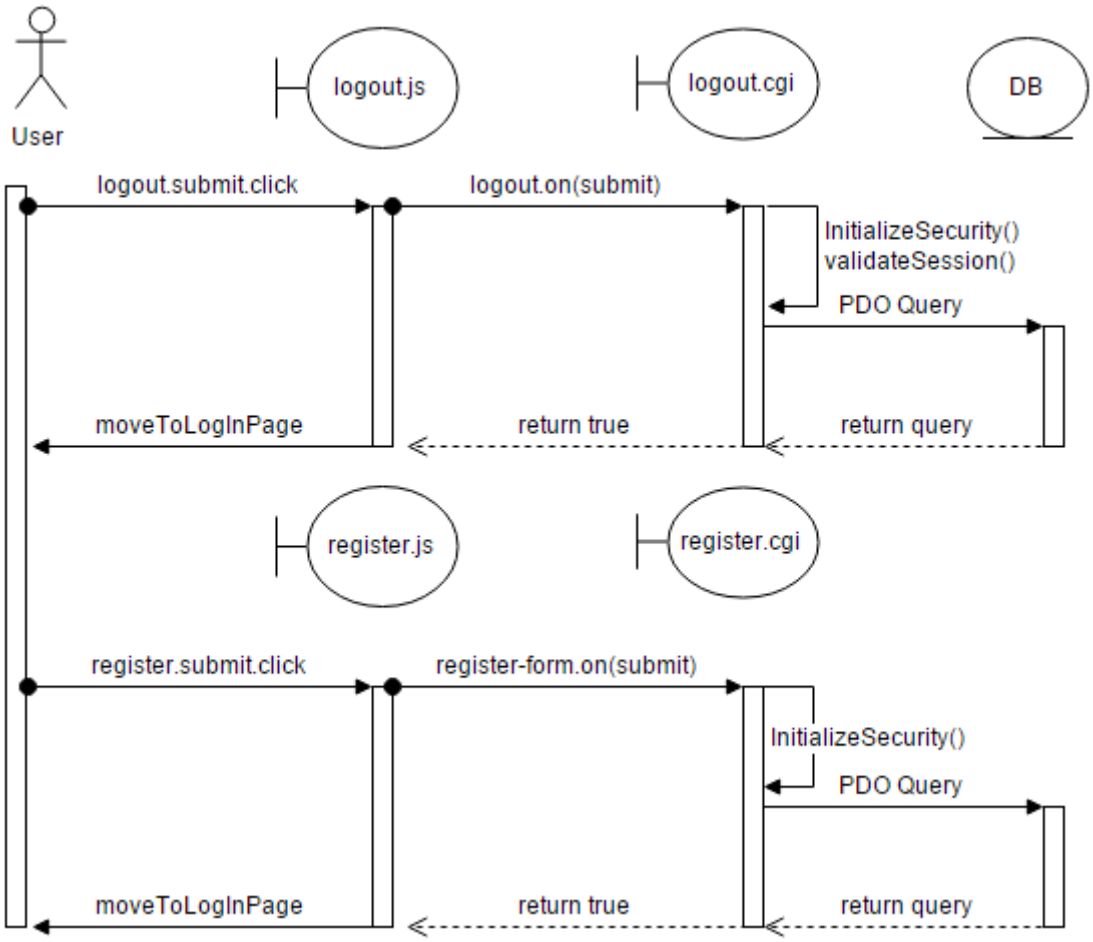
	päringu korral küsitakse sellelt verifitseeritud sisend. Kontrollitakse andmemassiivi väärtuste olemasolu. Paroolile lisatakse sool ning sellest kombinatsioonist võetakse räsi. Küsitakse andmebaasist seotud kasutajale vastav räsi. Räside võrdsuse korral algatatakse serveri poolne sessioon. Õnnestumise ning ebaõnnestumise korral antakse tagasiside.
Logout.cgi	Kutsutakse välja „Securitymanager“ klassi põhifunktsioon. Kontrollitakse päringu parameetri väärtuse korrektsust ning valideeritakse sessioon. Eelnevate tingimuste täitmisel lõpetatakse sessioon. Õnnestumise ning ebaõnnestumise korral antakse tagasiside.
getPosts.cgi	Kutsutakse välja „Securitymanager“ klassi põhifunktsioon. Kontrollitakse päringu parameetri väärtuse korrektsust ning valideeritakse sessioon. Eelnevate tingimuste täitmisel tehakse andmebaasipäring mille tulemuseks on kõik seal olevad postitused. Andmed tagastatakse kliendile. Õnnestumise ning ebaõnnestumise korral antakse tagasiside.
getItem.cgi	Kutsutakse välja „Securitymanager“ klassi põhifunktsioon. Kontrollitakse päringu parameetri väärtuse korrektsust ning valideeritakse sessioon. Eelnevate tingimuste täitmisel tehakse andmebaasipäring mille tulemuseks on numbrilise identifikaatori abil küsitud postituse andmed. Andmed tagastatakse kliendile. Õnnestumise ning ebaõnnestumise korral antakse tagasiside.

Tabel 4. Serveri poolsete funktsioonide kirjeldus.

3.4 Veebirakenduse jadadiagramm



Joonis 9. Jadadiagramm, esimene osa.



Joonis 10. Jadadiagramm, teine osa.

4. Rünnakute tõrjumine ja logimine

ZAP proksi haavatavuste test mängitakse läbi stsenaariumiga, eesmärgiga tuvastada erinevaid ründeid:

- Sessioon on üle võetud
- Ründajale on teada kehtiva päringu päis ja last, mida server ootab

```
POST http://www.localhost.local/cgi-bin/login.cgi HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/json; charset=utf-8
X-Requested-With: XMLHttpRequest
Referer: http://www.localhost.local/
Content-Length: 35
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Host: www.localhost.local
{"username":"ivo","password":"ivo"}
```

Joonis 11. Näide kehtivast päringust serveri suunal.

Sessiooni turvamine ning liikluse pealt kuulamine (kehtiva päringu saamiseks) jääb töö skoobist välja. Rünnakud ei õnnestuks juba „X-Requested-With“ välja puudumisel ning sessiooni olematuses.

4.1 Get päringud

GET päringutel puudub last, seega ründevektoriks saab olla ainult selle päise päringu sisu. Ebaõnnestunud rünnakud ning kaitsefunktsioonid on defineeritud koodidega (vaata lisa 1) ning (vaata tabel 3).

Ründe kood	Vektor	Kaitse kood	Kommentaar
2a	GET http://www.localhost.local/cgi-bin/getPosts.cgi?listItems=Set-cookie%3A+Tamper%3D8860ba2e-8762-47d0-8164-28da0160f134&_=1431597296007 HTTP/1.1	b2	Parameetri „listitems“ väärtuse eesmärk on sessiooni küpsist üle kirjutada.
2b	GET http://www.localhost.local/cgi-bin/getPosts.cgi?listItems=zApPX7sS&_=1431597296007 HTTP/1.1	b2	Parameetri „listitems“ väärtuse eesmärk on sisestada püsiv JS kood. Kuna see on kodeeritud number-tähed variatsiooni, siis regulaaravaldisest ei piisa. Rakendusel on teada oodatud URL string pikkus ning hetkel ületab see piirmäära.
2b	GET http://www.localhost.local/cgi-bin/getPosts.cgi?listItems=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E&_=1431603076223 HTTP/1.1	b2	Parameetri „listitems“ on lisatud kodeeritud skript sildid ning skripti sisu. Eesmärk on jooksutada JS kood.
2c	GET http://www.localhost.local/cgi-bin/getPosts.cgi?listItems=%2B&_=1431603076223 HTTP/1.1	b2	Parameetri „listitems“ väärtuse eesmärk on kodeeritud parameetri alusel infot vastusena saada.
2d	GET http://www.localhost.local/cgi-bin/getPosts.cgi?listItems=get%26type+%25SYSTEMROOT%25%5Cwin.ini&_=14316030	b2	Parameetri „listitems“ väärtuse eesmärk on serverit panna tagastama „win.ini“ OS süsteemi fail.

	76223 HTTP/1.1		
2e	GET http://www.localhost.local/cgi-bin/getPosts.cgi?listItems=%2Bresponse.write%28%7B0%7D*%7B1%7D%29%2B&_=1431603076223 HTTP/1.1	b2	Parameetri „listitems“ väärtuse eesmärk on serverit panna tagastama kogu vastuse sisu.
3a	GET http://www.localhost.local/cgi-bin/getPosts.cgi?listItems=get&_=%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5CWEB-INF%5Cweb.xml HTTP/1.1	b2	Parameetri „listitems“ väärtuse eesmärk on serveri kaustapuus leida lubatud koht, peale veebikataloogi enda.

Tabel 5. HTTP GET päringu võimalikke ründevektoreid.

Tabelis tuuakse välja üks näidisvektor igast ründegrupist, kuna ainuke mis muutub on ründevektori väärtus ning seetõttu kaitsefunktsioon reageerib üheselt. Teiste HTTP päise parameetrite kontroll ei ole tähtis, kuna reaalselt kasutatakse ainult URL parameetri väärtusi serverkoodis. HTTP GET päisest rünnatakse ka „cookie“ välja eelnevalt tabelis kirjeldatud halvaloomuliste sisenditega. Seetõttu on küpsiste kasutamine turvarisk. Soovitav on kasutada HTTP küpsist, millele on sätestatud lisaparameeter „http only“ [15], mis omakorda keelab brauseri poolse ligipääsu sellele.

Kokkuvõtvalt võib väita, et teades päringu pikkust ning lubades GET parameetrite väärtuseks ainult tähed ning numbrid, on võimalik suur osa rünnakuid ära hoida.

4.2 Post päringud

HTTP POST päringud on üldiselt infokandjad. Erinevalt GET meetodist, kannab POST endaga kaasas lasti, kus hoitakse andmeid. POST meetodi omapära avab selle ka

rohketele ründevektoritele. Seega, rünnatakse POST päise parameetri väärtusi enamate vektoritega kui GET meetodi päist. Lisaks on ka magusaks sihtmärgiks POST last, kuhu süstitakse rünnakuid. POST meetodi puhul saab samuti kasutada URL süstrünnakuid, mis on identsed GET URL süstrünnakutele, seega siin peatükis me neid eraldi välja ei too – kaitsemehhanism töötab identselt POST ja GET puhul. Ebaõnnestunud rünnakud ning kaitsefunktsioonid on defineeritud koodidega (vaata lisa 1) ning (vaata tabel 3).

Ründe kood	Vektor	Kaitse kood	Kommentaar
2a	{"username":"Set-cookie: Tamper=6de7550e-2d47-4e26-b429-d987565d570d","password":"ivo"}	b7	„username“ parameetrile vastava väärtuse abil üritatakse genereerida serveri poolne küpsis.
2a	{"username":"any?\r\nSet-cookie: Tamper=7c12027e-ada8-465e-979d-5ed16c7880d9\r\n","password":"ivo"}	b6	Parameetrite vahele süstitakse lisa string, mille eesmärk on serveri poolne sessiooni küpsis luua.
2b	User-Agent: "'<script>alert(1);</script>	b1	Läbi „user-agent“ parameetri väärtuse üritatakse andmebaasi süstida jääv XSS omane skript.
2c	{"\u0000":"ivo","password":"ivo"}	b7	Parameetri enda väärtus asendatakse lubamatu väärtusega, lootuses, et serveri poolel ei valideerita parameetri enda väärtusi.
2d	Referer: http://www.localhost.local/&sleep 5s&	b1	Saadetakse kauge OS käsk läbi „referer“ parameetri väärtuse, et serverit kontrollida.
2e	Referer: +response.write({0}*{1})+	b1	Eesmärk on koodi süstida jupp, et see tagastaks kõik andmebaasi tabelist.

2g	{"username":"ivo","password": "ivo UNION ALL select NULL -- "}	b7	Eesmärk on sooritada SQL süstrünnak läbi parooli parameetri väärtuse.
2g	Referer: http://www.localhost.local/' AND '1'='1	b1	Eesmärk on sooritada SQL süstrünnak läbi „referer“ parameetri väärtuse.
3a	/../Windows\system.ini	b6	Lisatakse lastiks serveris kaustapuud läbiv pahatahtlik käsk.

Tabel 6. HTTP POST päringu võimalikke ründevektoreid.

Kokkuvõtvalt võib väita, et POST päisel on palju parameetreid, millele väärtustada halvaloomuline väärtus, veel enam kogu POST päringu lasti sisule. Järeldades sellest, tuleks alati serveri poolsesse sessiooni salvestada võimalikult palju infot kasutajast. Iga väiksemgi erinevus päise parameetrites viitab muutusule isikus või konstrueeritud rünnakule. Rangelt soovituslik on hoida kinni andmevahetusformaatidest, kontrollida formaati ise ning selle sisu eraldi, eemaldades võimaluse mööda libiseda esimesest kontrollist. Kindlasti tuleb kontrollida ka parameetri enda väärtust, mitte ainult sellele vastavat väärtust. Kuigi see ei ohusta otseselt andmebaasi jms, vaid omab reaalselt ohtu serveri rakenduse kokku jooksmisel, kuna kood ei oska halvaloomulist sisendit parsida.

4.3 Logimine

Logimine on tähtis osa iga rakenduse puhul. See pole ainult vigade funktsionaalsete vigade tuvastamiseks vaid pahatahtliku sisendi, päringu või tegevuse hilisemaks tuvastamiseks. „SecurityManager“ raamistiku põhi klassiks on „IPS“. „IPS“ klass vastutab logi kirjete ja logifailide halduse eest. Iga turberaamistiku funktsioonil on kaks väljundit, tõene ning väär. Väär valiku tagastamine tähendab, et sisendi väärtus või päis on mittevastavuses funktsiooni nõuetega. Iga väära väljundi korral kirjutatakse kogu saabunud päis koos lastiga logifaili ning tuuakse välja, millest on tingitud viga. Peale vigase päringu või selle lasti osa tuvastamise, suudab raamistik välja tuua, kui tegu on ülevõetud sessiooniga ning lisab juurde lisaks eelnevale kasutaja ja sessiooni identifikaatori kombinatsiooni.

Raamistik hoiab ka edetabelit ründaja aadressi ja selle katsete arvust. Iga rünnak või sessiooni üle võtmine suurendab seda arvu.

Veebiserveri tehnilise osa eest vastutav isik või muu vastutav peaks pidevalt jälgima logisid. Ennetamine on parem kui tagajärgedele põhjuse leidmine. Aadressi põhise edetabeli põhjal on soovituslik aktiivsed ja intensiivsed ründajate IP aadressid blokeerida vältimaks nende rünnakute õnnestumist. Lisaks on rangelt soovituslik lülitada välja kasutaja, kelle sessioon on üle võetud. Võib eeldada, et tema autentimisanded on jõudnud siis juba ründajani. Esimesel võimalusel teavitada kasutajat ning uurida välja võimalik kahju. Üksikumad päise või selle lasti halvaloomulised sisendid tuleks korreleerida ning näiteks ründaja aadressi järgi koondada, mis annavad parema üldpildi ning mille alusel saab objektiivsemaid otsuseid langetada. Kuna logid on struktureeritud parameeter võtmevälja ning väärtuse alusel, on nad liidestavad logi kollektoritega. Sellest on kasu siis, kui tegu on suurema infosüsteemiga kus jookseb palju muid servereid või suurema veebirakendusega, kus rünnakute intensiivsus on kõrge.

Logide kogumine suurtes kogustes pole hea lahendus, kui soov on aktiivselt ning automaatselt nende alusel toiminguid teostada ehk logi kasutamine rünnete tuvastamiseks ning tõkestamiseks. Soovitav on rakendada SIEM, mis teeb logide otsimise, sidumise, koondamise ning korreleerimise automaatseks. Lisaks sellele, on võimalik seadistada teatud künnise ületamisel mingi spetsiifilise sisendi pideval ärakasutamisel SIEM saatma näiteks tulemüürile SNMP käsu, mille alusel tulemüür IP blokeerib. Sama loogikat saab rakendada ka kasutaja välja lülitamisel. Teatud veebirakenduse liidese suunal, saadetakse tavaline HTTP päring, mille sees on kasutaja identifikaator number. Teostatakse SQL päring, millega kasutaja suletakse. Sellele boonuseks, tuleks seadistada ka samale kasutajale e-kirja saatmine, kus on kirjeldatud põhjus ning lahendus.

Kokkuvõtvalt saab väita, et iga väiksemagi sündmuse või tegevuse logimine, koos kõikvõimaliku lisainfoga, võib suures pildis kaasa aidata pahalase tuvastamisel või lahenduse leidmisel.

5. Arendusvaade

Öeldakse, et tarkvara ei saa kunagi valmis, see vajab alati täiendusi ning uuendusi. Praegune näidisrakenduse turberaamistik on arendatud kasutades ZAP ründe proksi standardseid rünnakuid. See küll teeb praeguse rakenduse turvaliseks, kuid autor on arendanud enamus turvafunktsioone staatiliseks. Kui rakenduse suurus ning maht kasvab, muutub staatiliste klasside ning funktsioonide täiendamine tülikaks ning ajakulukaks. Selletõttu tuleks kogu turberaamistik arendada dünaamiliste funktsioonide alustel. Dünaamilisuse tagavad edukalt näiteks regulaaravaldiste alusel kontrolli teostavad funktsioonid. Regulaaravaldised töötavad kõige madalamal tarkvara tasandil ehk võrdlused toimuvad binaarsel tasemel, mis tagab nende usalduse. Regulaaravaldised on ka enamuste IDS/IPS signatuuride [17] osaks. Lisaks standardsetele haavatavuste tuvastamise testidele, tuleks neid muuta ning lisada oma poolseid vektoreid, täiendada neid ka teiste isikute soovituste ning testide alusel. Mida rohkem erinevaid rünnakuvektoreid läbi testitakse, seda turvalisem saab rakendus olema. Skeptilistes olukordades saab alati taanduda vanale heale staatilisele, luba ja ära luba listile.

Väiksemad organisatsioonid ning asutused hoiavad tavaliselt oma veebirakendusi avalikes teenusepakkujate serverites, kus puuduvad igasugused tulemüürid ja muud kaitseseadmed. Selleks saab antud rakendust täiendada, et neil tekiks primaarne ülevaade turbeintsidentidest nende rakenduses. Raportite genereerimine tsüklites, võttes aluseks mingi parameetri väärtuse vastu eksimise künnis ning saata need üle e-kirja asutuses tehnilisele vastutavale. Lisaks, arendada juurde automaatselt kasutajaid sulgev klass, mis korreleerib ise logisid ning saadab välja e-kirju. Teisisõnu programmeerida väikene SIEM moodul juurde.

Kokkuvõtteks tuleks rakenduse turbeosa täiendades liikuda dünaamiliste funktsioonide printsiibi suunas ning laiendada logihaldust.

6. Kokkuvõte

Töö eesmärk on esile tõsta turvalisuse tähtsust kogu tarkvara elutsükli protsessi jooksul. Seejuures näidata logimise vajalikkust. Oluliste tulemustena võib mainida haavatavuste skaneerimise tulemuste alusel loodud näidisrakenduse turberaamistik, mis suudab tõrjuda enamus levinud rünnakuid ning selle vektoreid. Lisaks suudab raamistik logida kogu informatsiooni rünnakust koos kõikvõimaliku lisainformatsiooniga. Turberaamistikku on võimalik ka siduda teiste sama tehnoloogia alusel loodud veebirakendustega ning on võimeline esitama logi kujul, mis omakorda võimaldab liidestavust logikollektoritega.

Kokkuvõtvalt järeldusena võib mainida, et ilma haavatavuste skaneerimisteta ei ole võimalik luua turvalist rakendust. Lisaks, logide puudumisel ei ole võimalik tuvastada pahalase tegusid ning rakendada aktiivset kaitset veebirakendusele.

Võib öelda, et autori eesmärk luua rakendus, mis on kindlustatud kasutades piiratud ründevektoreid, on rahuldatud. Autori eesmärkidest tuua esile turvalisuse tähtsus on ka rahuldatud – selle jaoks on toodud teadustööde alusel näiteid, statistikat ning esile toodud teostatud rünnakute näiteid. Veel enamgi, autori eesmärk tuua esile logi osatähtsus ning selle võimalikud rakendusviisid on ka rahuldatud.

Kasutatud kirjandus

[1] Andmekaitse ja infoturbe seletussõnastik http://akit.cyber.ee/

[2] Anthony C.W. Finkelstein *Ubiquitous Web Application Development*
http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/papers/uwa.pdf

[3] WEB APPLICATION ATTACK REPORT #5
http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed5.pdf (2014)

[4] SIEM http://aut.researchgateway.ac.nz/bitstream/handle/10292/7688/MaimboC.pdf

[5] Web technology surveys
http://w3techs.com/technologies/overview/programming_language/all

[6] Mario Konecki *Secure web applications*
http://bib.irb.hr/datoteka/299783.Konecki_Hutinski_Orehovacki.pdf

[7] Kevin Heineman *Building Web Application Security into Your Development Process*
http://help.rmccurdy.com/scripts/docs/spidynamics/Webapp_Dev_Process.pdf (2005)

[8] *Advanced Network Design*
http://resources.infosecinstitute.com/book-advanced-network-design/

[9] OWASP Zed Attack Proxy Project
https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[10] Active Scan https://code.google.com/p/zaproxy/wiki/HelpStartConceptsAscan

[11] Bjoern Klinggaard http://dinbror.dk/bpopup

[12] HTML5 Introduction
http://www.w3schools.com/html/html5_intro.asp

[13] Regular Expressions
http://www.regular-expressions.info/

[14] PHP Data Objects
http://php.net/manual/en/book.pdo.php

[15] Http only https://www.owasp.org/index.php/HttpOnly

[16] Secure Web Applications http://www.security-assessment.com/files/documents/presentations/2007_DI_Exploiting%20Web%20Applications.pdf (2006)

[17] Symantec Network Intrusion Detection Signatures
http://www.symantec.com/connect/articles/network-intrusion-detection-signatures-part-one

Lisa 1. Ründepoliitika

Poliitika kategooria, selle alla kuuluvad testid ning nende mõju [17].

Policy	Test Name	Result	Code
Injection	CRLF Injection	Access to data	2a
	Cross Site Scripting	Access to data	2b
	Parameter Tampering	Access to data	2c
	Remote OS Command Injection	Access to data	2d
	Server Side Code Injection	Access to data	2e
	SQL Injection	Access to data	2g
Server Security	Path Traversal	Access to server files	3a