Examensarbete

HÖGSKOLAN I SKÖVDE

WEBSOCKETS OCH SÄKERHET I STARTUPBOLAG

En studie i säkerhet kring WebSockets

WEBSOCKETS AND SECURITY IN STARTUP COMPANIES

A study in security around WebSockets

Examensarbete inom huvudområdet informationsteknologi
Grundnivå 30 högskolepoäng
Vårterminen 2017

Robert Roos

Handledare: Kristens Gudfinnsson
Examinator: Jeremy Rose

Sammanfattning

WebSockets är en ny teknik för att möjliggöra snabb kommunikation på internet mellan två eller fler användare.

Målet med denna studie var att undersöka de säkerhetsrelaterade problem introduktionen av WebSockets kunde medföra för startupbolag, samt specifikt hur XSS-attacker från ett serversideperspektiv skulle kunna avvärjas. Detta i syfte att ge startupbolag ett underlag att arbeta proaktivt med säkerheten, samt att initialt inte behöva köpa in externa säkerhetstjänster.

En kvalitativ undersökning har genomförts med metoden litteraturstudie. Tidigare forskning i ämnet har granskats, såväl inom WebSockets, som påverkan dataintrång och specifikt XSS-attacker kan medföra för en organisation. Denna metastudie har haft som syfte att binda samman tidigare forskning för att besvara studiens frågeställning, en efterfrågad metod inom informatik som lider brist på metastudier för tillfället och där tvärvetenskaplig kunskap inte binds samman i den takt som är önskvärt.

Undersökningen resulterade i att lyfta fram de viktigaste hoten att skydda sig mot, bland annat kontrollen av från vilken källa en klient de facto försöker ansluta från till en WebSockets server. Men även olika typer av XSS-attacker, där specifikt callback-modifikation identifierades som en sårbarhet med stora konsekvenser.

Avslutningsvis kunde utifrån genomförd litteraturstudie en slutsats presenteras innehållande specificerade rekommendationer vid implementering av WebSockets.

Nyckelord: WebSockets, DDoS, Security, XSS, Startups

English summary

WebSockets is a new communications protocol for the web, enabling fast communication between two or more clients.

The overall goal with this study was to investigate the security related problems the introduction of WebSockets could have in start-up companies. Also, how XSS-attacks from a server-side perspective could be averted. This to give the foundation for how start-up companies should work proactively with the security, also not have to turn to external security services.

A qualitative study has been performed with the method literature study. Earlier research in the field has been reviewed and analysed. This for both WebSockets and the impact hacking and specifically XSS-attack could have on an organization. This ’metastudy”s main purpose has been to connect earlier research to answer the problem statements. This has according to research been asked for a lot in the informatics field, where there is a lack of these kinds of ’metastudies’.

The study resulted in acknowledging the most important threats to protect against, among others the importance of inspecting what source a client is connecting from to a WebSockets server. But also, several XSS-attacks where specifically callback modification was identified as a vulnerability with big consequences.

In conclusion based on the literature study, recommendations for the proactive security work could be presented.

Keywords: WebSockets, DDoS, Security, XSS, Startups

Innehållsförteckning

Förord

Tack till mina föräldrar Olle och Carina för allt ovillkorligt stöd genom alla år, utan er hade jag inte vart här idag. Jag vill även rikta ett tack mot Kristens Gudfinnsson för kontinuerlig feedback, pedagogisk vägledning och hjälp under arbetets gång.

Definitionslista

Webbläsare: Med detta begrepp åsyftas det program en användare interagerar med webben genom. Vanliga exempel på webbläsare är Internet Explorer, Firefox, Google Chrome, Opera samt Safari.

Popup: Ett fönster som öppnas i webbläsaren utöver den sida du besöker, vanligen använt vid att visa reklam.

Förfrågningar/requests: Med detta avses när en fysisk person genom en webbläsare begär data från en server.

Polling: Refererar till förfrågan men i avseendet att ställa en förfrågan upprepade gånger fram till att efterfrågad/nya data faktiskt finns att tillgå. Innebär alltså att flertalet förfrågningar kan resultera i att ingen data returneras till klienten.

Klientsidan/klientperspektiv: Med detta menas en fysisk persons perspektiv i sin interaktion med en webbtjänst över Internet, genom en webbläsare.

Serversidan/serverperspektiv: I motsats till klientsidan så åsyftas här de datorer som tar emot förfrågningar från klientsidan och sedan behandlar dessa för att ge ett svar i form av att data sänds tillbaka, alternativt tas emot.

Exekvering: I relation till denna studie så åsyftas vid bruket av begreppet exekvering när kod körs. Detta kan ske såväl på klientsidan som serversidan.

Hackare: En användare vars intention är att utnyttja en sårbarhet i en organisations serverarkitektur/webbapplikation för att endera sänka systemet. Alternativt få tillgång till och förvärva känslig information.

Attacker/hackerattacker: I denna studie så åsyftas med begreppet attacker att en hackare exploaterar en sårbarhet i serverarkitektur/webbapplikation.

XSS: Cross-Site-Scripting, kod som endera körs eller injiceras i en webbapplikation för att förvärva information.

DoS/DDoS: Denial-of-Service är en attackmetod som går ut på att sända stora mängder requests mot en server i syfte att sänka den/att den inte kan behandla riktiga requests. DDoS utgår från samma princip men med flertalet klienter som utför attacken, vanligen från ett botnätverk bestående av kapade datorer.

Session hijacking: En typ av XSS-attack där målet är att förvärva en annan användares sessionscookie (identifierare för att användaren är inloggad). Väl förvärvad kan den utsattes session kapas och information kan förvärvas.

PHP: Serversidespråk för webben, körs vanligtvis på exempelvis en Apacheserver där skripten tolkas i realtid istället för kompileras.

Python: Objektorienterat skriptspråk som ofta används för serverlogik och GUI-lösa applikationer.

JavaScript: Skriptspråk som tolkas direkt i en användares webbläsare i motsats till exempelvis PHP som tolkas på serversidan.

DOM: Document Object Model, ett gränssnitt som ger programmeringsspråk möjligheten att läsa, uppdatera ett dokuments (ex. webbsida) innehåll och struktur.

Node.js: Baserat på Google Chromes V8 skriptmotor, möjliggör att köra JavaScript såväl direkt i webbläsaren (klientsidan) som på serversidan.

Realtidskommunikation: Här menat som möjligheten för en server att sända data till en klient direkt nya data finns tillgänglig.

TCP: Transmission Control Protocol är ett dataöverföringsprotokoll som används för att sända data mellan två datorer.

TLS: Samma som TCP fast krypterat.

WS/WSS: WS/WSS är WebSockets protokoll, där WS körs över TCP protokollet och WSS över det krypterade TLS protokollet.

Man-in-the-middle: En attackmetod, där attackeraren manipulerar uppkopplingen mellan två parter för att förvärva/manipulera den data som utbyts.

1. Introduktion

I en tid där alltmera information utbyts mellan användare och tekniska enheter samt mellan enhet till enhet ”internet of things” (IoT) eller M2M, ”machine-to-machine”. Krävs även snabba uppkopplingar med låg fördröjning och snabb leverans av data (Xia *et al.*, 2012). Ett exempel på denna explosionsartade växt av internet är att mellan åren 2000 och 2016, har användningen av Internet ökat med cirka 900% (MiniWatts Marketing Group, 2006). Detta sätter även säkerheten kring kommunikation över Internet i ett stort fokus och är fundamentalt för såväl organisationer som privatpersoner. Etablerade tekniker på marknaden idag så som HTTP och Ajax har många år av utveckling och forskning i ryggen gällande säkerhet, men hur ser det ut för nyare tekniker så som WebSockets?

Under tidigt 90-tal och ända fram till början av 2000 dominerade vad som benämns som ”stateless web”. Detta bestod av synkrona anrop och svar mellan användare och server, vilket innebar att en användare efterfrågade något genom exempelvis en hyperlänk, förfrågan sändes vidare till servern, behandlades och sändes sedan tillbaka till användaren som presenterades med en ny sida, detta över HTTP-protokollet (Hypertext Transfer Protocol).

Under mitten av 2000 började allt fler webbläsare få stöd för en teknik som kallas för Ajax (Asynchronous JavaScript + XML). Med Ajax introduceras ett tredje lager mellan klient och server (XMLHttpRequest objektet), vilket innebär att när en användare exempelvis efterfrågar information, så är det Ajax-lagret som sköter kommunikationen mellan klient och server asynkront, istället för som med HTTP-protokollet direktkommunikation mellan klient och server (Puranik, Feiock and Hill, 2013).

Ett nästa steg i detta är en teknik som efter 2010 vunnit mera mark när det kommer till att så snabbt som möjligt kunna leverera data mellan exempelvis två klienter med en server emellan, eller direkt mellan klient till server, WebSockets (Puranik, Feiock and Hill, 2013). Skillnaden mot tidigare tekniker här, blir en direktuppkoppling klienterna emellan utan mellanlager.

Detta leder oss in på hur pass säker denna teknik är att använda sig utav idag. Flertalet tidigare studier, bland annat den av Pimentel & Nickerson (2012) undersöker WebSockets ur aspekten prestanda. Där man med prestanda åsyftar bandbreddsåtgång vid bruket av WebSockets jämte Ajax (Pimentel and Nickerson, 2012).

Resultaten av deras studie pekar på tydliga förbättringar i prestanda sett ur bandbreddsperspektivet, men även ur fördröjningsperspektivet. Alltså hur lång tid det tar för ett meddelande att nå mellan två punkter. Man lyfter siffror som pekar på att fördröjningen mellan leverans av meddelanden när dessa sänds över HTTP-protokollet är mellan 3-4.5 gånger så lång, som vid bruket av WebSockets (Pimentel and Nickerson, 2012).

Vad man däremot inte lyfter och som är en mycket relevant fråga i sammanhanget, sett till hur WebSockets arbetar nära med bland annat JavaScript, är hur säker denna kommunikation egentligen är och vilka hot den står inför. Mycket forskning har gjorts inom ämnet när det kommer till överföring av data över HTTP-protokollet (Loreto *et al*, 2011). Motsvarande forskning när det gäller kommunikationssäkerheten med hjälp av WebSockets är däremot bristfällig vid dags dato, om man där exkluderar den forskning som gjorts kring TCP-protokollet i sig.

En annan begränsning med tidigare forskning är att den i huvudsak är inriktad mot större organisationer, med större resurser för säkerheten. Det finns idag system för att skydda mot exempelvis ”flooding” av servrar (se kapitel 2.4), men dessa är kostsamma och i många fall inte ett alternativ för en mindre organisation.

1.1. Problemområde

Med de fördelar som lyfts i inledningen WebSockets ger i form av direktkommunikation och konstant uppkoppling mellan klient och server med mindre bandbredsåtgång, är det rimligt att antaga att många organisationer kommer att påbörja en migration från tidigare tekniker såsom Ajax- mot WebSockets. I och med detta så bör även en grundlig analys kring vilka hot mot säkerheten som finns genomföras och ställas i kontrast till de hot som finns mot tidigare tekniker. Troligt är att tidigare tekniker i större grad har utförligare underlag för hur en säker applikation bör konstrueras, liknande underlag bör även sättas upp för WebSockets för att undvika olaga dataintrång.

I och med hur JavaScript används i webbläsaren för att möjliggöra såväl kommunikation med Ajax som WebSockets så får säkerhetsaspekten två perspektiv. JavaScript är ett programmeringsspråk som exekveras direkt i klientens webbläsare, detta innebär att när kod hämtas från webbservern så körs denna först när den laddats in i klientens webbläsare. Således är även koden i sig publik och kan enkelt läsas av användaren, vilket innebär att densamma även kan manipuleras (Ritchie, 2007).

Den stora möjlighet som finns att genomföra attacker mot och genom webbaserade tjänster styrks även genom att enkäter gjorda inom ämnet påvisar att den överlägset vanligaste formen av hackerattacker idag riktar sig mot webbtjänster (Erlingsson, Livshits and Xie, 2007). XSS, eller Cross-Site-Scripting utmärker sig som en av de vanligaste typerna av attackmetoder där JavaScript körs i en klients webbläsare, något som görs såväl med webbplatser som använder sig av Ajax, som WebSockets. Detta ger tyngd till påståendet att säkerheten kring webbtjänster idag är bland det viktigaste en organisation kan satsa på, då så många idag har en stor del av sin verksamhet webbaserad. Eller med system som står i direktkoppling ut mot det öppna intranätet.

Ett klassiskt exempel på en attack är så kallade XSS-attacker, där hackaren i exempelvis ett kommentarsfält på en blogg eller liknande matar in skadlig kod. Koden sparas ned av webbservern i databasen och när denna data sedan hämtas igen och presenteras på en webbsida kan den även exekveras direkt. Som tidigare nämnt så exekveras JavaScript på klientsidan, vilket möjliggör denna typ av attacker (Erlingsson, Livshits and Xie, 2007). En annan vanlig form av XSS-attack är att genomföra en så kallad sessionskapning, vilket kortfattat innebär att en attackerare tar över en användares inloggningssession och på så vis får tillgång till all dennes data (Erkkilä, 2012).

Som tidigare nämnt under introduktionen till denna studie, så är en av fördelarna med att lyfta anrop och arbete som väger tungt på bandbredd och servar- till klientsidan istället då JavaScript exekveras först där. Detta är dock till bekostnad på säkerheten, då man lämnar det slutna system man som organisation själv har full kontroll över- ut till okända klienter (Rodriguez and Posegga, 2015).

Ett annat exempel i kontexten är hur hackare kunnat utnyttja det faktum att WebSockets baseras på en konstant uppkoppling mellan klient och server. Genom att simulera att tusentals unika klienter kopplar upp sig mot en server genom en XSS-baserad attack, har man därigenom kunnat genomföra DDoS (Distributed denial of

service) attacker mot serverägare. Då verktyg vid dessa attacker inte fanns tillgängliga för att avgöra om en klient var äkta eller syntetisk, överbelastades systemen snabbt och fallerade (Rodriguez and Posegga, 2015). Ett katastrofalt scenario för många organisationer, såväl sett ur ekonomisk synvinkel om webbtjänsten ligger till grund för inkomster. Som rent förtroendemässiga, där användare som inte kommer åt en tjänst de endera investerat resurser i, eller behöver för exempelvis sitt arbete. Inte minst är detta ett problem för mindre bolag eller startups, där omkringliggande struktur för backupsystem kanske saknas- något som större organisationer i regel besitter, men i utbyte med större ekonomiska summor.

I en studie genomförd av Rodriguez & Posegga (2015) exemplifieras och mäts den påverkan denna typ av hackning medför, där tydliga resultat presenteras gällande överbelastning av server med små medel. Den slutsats de lyfter fram i sin forskning relaterar i stort till klientsidan av problemet. Detta då under perioden då detta testades, innehöll webbläsarna Google Chrome samt Safari en sårbarhet som möjliggjorde denna typ av ”flooding” av anrop och uppkopplingar mot målservern (Rodriguez and Posegga, 2015). Vad som däremot inte diskuteras och lyfts upp är den fortsatta problematiken från serversidans perspektiv. Liknande attacker är fortsatt fullt genomförbara med andra applikationer. Författarna vänder sig i huvudsak mot att minska problembilden genom att se till att de mest använda webbläsarna inte möjliggör denna typ av angrepp. Men lämnar ingen lösning för hur syntetiska klientanrop från serversidan kan kunna avvärjas eller identifieras, vilket är centralt i problematiken då så mycket ansvar förflyttats från server till klient.

Det blir därmed viktigt för organisationer som hanterar såväl affärsdata som kunddata i sina IT-system att ha ett proaktivt skydd mot kända typer av attacker som kan leda till dataintrång, eller att deras webbaserade tjänster tas offline. I sken av att fler privatpersoner än någonsin tidigare delar personliga uppgifter med organisationer, såsom personnummer, bankuppgifter, med mera som kan användas vid exempelvis identitetskapning, kan mycket stora skada ske vid dataintrång (Fisher, 2013). Intrången kan ge stora efterdyningar ekonomiskt för dessa organisationer, direkt genom exempelvis skadeståndskrav, eller indirekt genom dålig publicitet (Fisher, 2013). Avslutningsvis finns det idag på marknaden tredjepartstjänster som erbjuder skydd mot exempelvis XSS-attacker, men ofta till hög kostnad och dessutom skapar ett ”single point of failure” problem. I och med kostnaderna är detta även något som kan vara svårt för nystartade bolag att införskaffa (Cloudflare, 2017).

Denna studie avser att ge startupbolag intresserade av att använda sig av WebSockets rekommendationer kring vilka säkerhetsaspekter de bör vara medvetna om samt hot de bör skydda sig mot.

Baserat på det som lyfts fram ovan har studiens frågeställning fastställts.

1.1.1. Frågeställning

Hur kan XSS-attacker genomförda genom tekniken WebSockets stävjas/avvärjas från ett serversideperspektiv?

För att besvara denna frågeställning, har den delats in i 2 delfrågor:

1. Hur kan hackare från ett serversideperspektiv när WebSockets används, avvärjas från uppkoppling?

2. Vilka skydd ur ett serversideperspektiv kan användas för att minimera risken för XSS-flooding-attacker när WebSockets används?

1.2. Avgränsningar

I och med ämnets bredd har ett flertal avgränsningar gjorts för denna studie, för att hålla den inom fokuserade och rimliga ramar. Vid genomgång av HTTP-protokollet och säkerhetsaspekterna där, något som granskats utförligt i tidigare forskning. Kommer denna rapport inte fokusera på fördjupande fakta i ämnet.

Denna studie kommer även avgränsa sig från att granska medelstora och stora företag, då tonvikten i studien ligger vid att ge rekommendationer och lösningar för nystartade (så kallade startupbolag) samt mindre företag/microbolag med större möjligheter till direktkontroll över sin IT-miljö. Enligt den svenska definitionen av detta begreppet innebär detta mellan 1–10 anställda samt 10–49 anställda.

Studien avgränsar sig även från andra typer av attacker mot serverarkitekturen än XSS-attacker och där XSS-attacker inriktade mot WebSockets.

Studien avgränsar sig även från andra programmeringsspråk och ramverk för arbete med WebSockets än JavaScript, Node.js, HTML5 och PHP.

Slutligen så avgränsas studien från att granska säkerheten ur ett klientperspektiv, det vill säga ur webbläsarens perspektiv.

2. Bakgrund

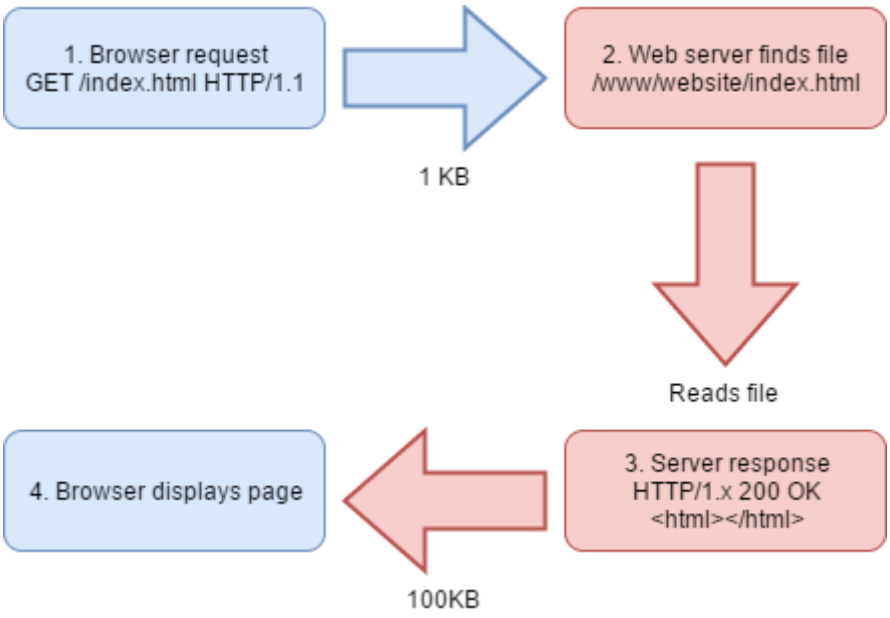
Detta kapitel tar upp den teoretiska bakgrunden för ämnet i studien, för att ge läsaren förståelse och en uppfattning kring ämnet. I detta kapitel beskrivs såväl HTTP-protokollet, som Ajax och slutligen WebSockets. För alla tre tekniker presenteras även hur XSS-attacker är kopplade till dessa, samt en avslutande sektion med utförligare information kring XSS-attacker. Avslutningsvis ges en kortare sammanfattning.

2.1. HTTP-protokollet

Som framgick av inledningen har HTTP-protokollet länge vart och är en central del när det kommer till kommunikation klienter emellan vid bruk av webbtjänster. I grunden har HTTP-protokollet aldrig haft som roll att möjliggöra det vi benämner som realtidskommunikation, där det till grunden är ”stateless”.

Ett exempel på vad som menas med begreppet stateless i detta sammanhanget, är att användaren matar in en sökfras i ett formulär, klickar på sök, användarens data sänds till serven som behandlar denna och sedan sänder tillbaka resultatet till användaren där tidigare sökningar nu raderats från det temporära minnet på klientens sida (Zhaoyun, Xiaobo and Li, 2010). Detta innebär att skulle samma sökning behöva genomföras igen exempelvis, så måste samma anrop ske igen mellan klient och server, där en ny sida/svar måste genereras.

HTTP har däremot simulerat realtidskommunikation genom en teknik som kallas HTTP polling, vilket går ur på att klienten konstant skickar en förfrågan till en server och efterfrågar ny information (Pimentel and Nickerson, 2012). Givetvis skapar detta problem sett till prestanda (mätt i bandbredd och resursåtgång från server), då så många förfrågningar från klient sker med jämna intervall, även då nya data inte finns att hämta.



Figur 1. Visar hur ett traditionellt HTTP-anrop tas emot och levereras

Ett annat problem är också att om en fixerad polling-rate sätts, exempelvis till varannan sekund- så vid förändringar i hur snabbt ny data finns tillgänglig skapas en latency där man kan säga att den data klienten får inte längre är i realtid (Pimentel and Nickerson, 2012). Denna typ av efterfrågan av ny information har vart och är vanligt förekommande i bland annat chattapplikationer och liknande meddelandeapplikationer.

Ett tredje vanligt problem med HTTP polling är att det leder till en omladdning av klientens webbsida. Detta innebär att även om klienten efterfrågar en liten mängd data, exempelvis ställer frågan ”har jag några nya notifikationer?”. Så måste en helt ny-exekverad version av webbsidan sändas i retur till klienten och inte endast den del av webbsidan som presenterar nya notifikationer, något som påverkar bandbredd och prestanda negativt (Furukawa, 2011).

En variant av HTTP polling är HTTP long polling. Som togs upp rörande HTTP polling, är en av de största nackdelarna där, den stora mängd onödiga anrop som skickas från server till klient i väntan på att nya data finns att tillgå. Detta söker HTTP long polling (Furukawa, 2011) att råda bot på genom att istället för att ta emot en request och genast ge ett svar, vänta med att ge klienten ett svar fram till att ny data finns att tillgå. Ett sätt att arbeta på som känns igen i många av de applikationer som idag använder sig av push notifikationer och liknande. Detta minskar antalet anrop avsevärt och den data som måste sändas mellan klient och server om och om igen (Pimentel and Nickerson, 2012).

När det kommer till säkerhetsaspekten kring HTTP är den annorlunda i sin natur att granska, då HTTP är ett protokoll för vilket bland annat Ajax använder sig av för att fungera. Vanliga påpekanden kring HTTP är bland annat hur den data som sänds mellan klienter, eller server och klient inte är krypterat, däremot finns HTTPS som är ett tillägg till protokollet som möjliggör kryptering. Andra vanliga säkerhetsproblem berör bland annat SQL-injections, vilket i grund och botten är snarare ett säkerhetsproblem vid serversidekoden (ex. PHP, C# el. liknande) framför ett säkerhetshål i protokollet (Rouse, 2010). För att genomföra en SQL injection rent genom HTTP-protokollet, så måste den skadliga SQL-koden köras/anropas direkt genom ett adressfält i en webbläsare exempelvis. Målet är här att få åtkomst till en databas för att förvärva information, eller på annat sätt manipulera den data som finns lagrad där (Rouse, 2010).

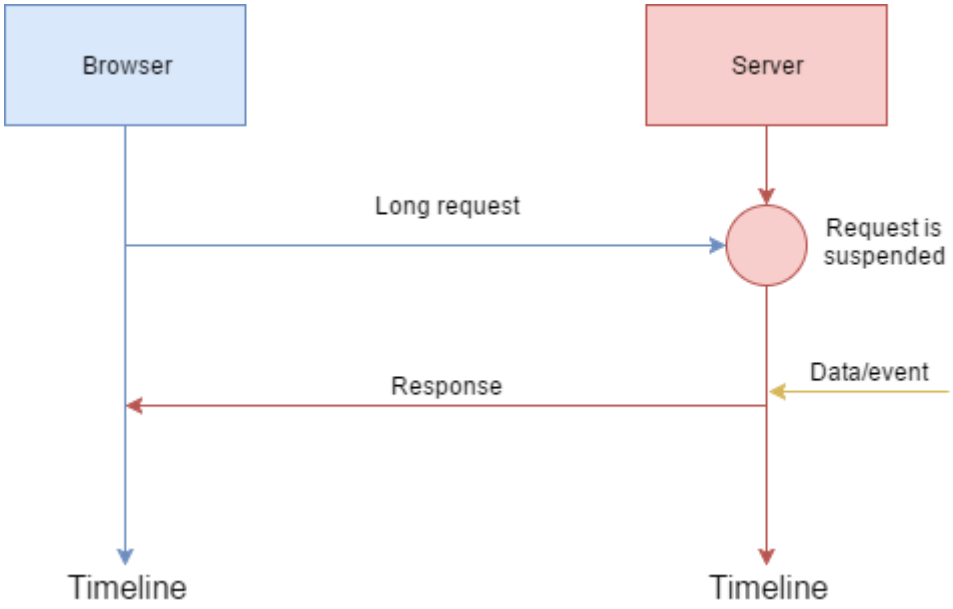
Ett exempel på hur en sådan attack kan se ut är som följer:

```
2
3
4   SELECT * FROM Users WHERE UserId = 105 OR 1=1;
5
```

Figur 2. Visar SQL-kod för att genomföra en SQL-injection attack och presentera alla användare I en databas.

Ovanstående figur skulle exempelvis kunna användas för att hämta ut alla användare som finns registrerade i den databas man avser att attackera. I och med att frågan som ställs mot databasen blir att endera hämta ut användaren med Id 105, eller alla användare om 1=1, vilket alltid är sant. Man kan dock som inledningsvis poängterades

debattera huruvida detta är kopplat till säkerheten hos HTTP-protokollet eller SQL i sig.



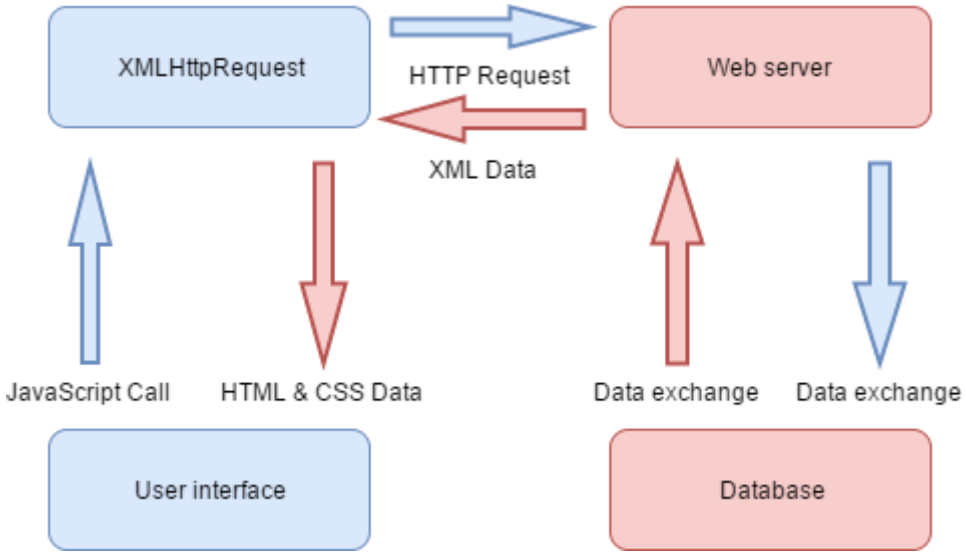
Figur 3. Visar hur ett asynkront anrop mot server hanteras

När det kommer till XSS-attacker så skiljer sig målet inte (detta tas upp under sektionen för XSS nedan) i större grad än när Ajax och WebSockets kommer in i bilden. De senare teknikerna agerar snarare katalysator. Detta innebär att så länge en webbplats tillåter inmatning av data och en webbläsare exekvering av JavaScript ökar risken för att XSS-attacker kan genomföras i och med att skadlig kod kan exempelvis lagras i en databas. När en besökare sedan laddar in en webbsida som hämtar information från denna databas, följer det skadliga skriptet med och körs i klientens webbläsare (Erkkilä, 2012).

2.2. Ajax

Ajax (Asynchronous JavaScript and XML) kan enklast förklaras som ett tredje lager mellan klient och server (XMLHttpRequest objektet), där målet är att nå en situation mera lik realtidskommunikation än det HTTP polling kan ge. Men är i jämförelse med WebSockets fortsatt ett ramverk som baseras på endera tidsbestämda anrop, som går genom mellanlagret, eller aktiva anrop som passerar därigenom (Puranik, Feiock and Hill, 2013).

Vad detta innebär är att när användaren efterfrågar exempelvis svar på en sökning, så skickas denna data genom XMLHttpRequest objektet, som sedan sänder klientens data vidare till en server. Servern ger ett svar som sänds tillbaka till XMLHttpRequest objektet och presenteras sedan för klienten utan att dennes webbsida behöver laddas om helt, som vid synkrona anrop. Istället uppdateras befintlig sida med ny information, vilket ger användaren en mera applikationslik känsla, men spar också bandbredd och serverresurser i och med att hela sidan inte behöver exekveras och laddas om- endast relevant data (Zhaoyun, Xiaobo and Li, 2010).



Figur 4. Visar hur ett Ajax-anrop går till, där användare efterfrågar nya data och får denna levererad

Det Ajax möjliggör i motsats till HTTP polling, är att sända och mottaga endast den information som krävs. I praktiken innebär detta att en användare kan i exempelvis ett webbgränssnitt med en tabell över städerna i södra Sverige, genom ett klick kan uppdatera denna tabell att istället presentera städer i norra Sverige, där denna data asynkront hämtas från en server/databas och sedan presenteras utan att användarens hela webbsida laddas som (Puranik, Feiock and Hill, 2013).

Flödet för en sådan request kan beskådas i figur 3 ovan. Här ser vi hur användaren genom sin webbläsare initierar ett JavaScript anrop, vilket kan startas genom exempelvis ett klick på en knapp på webbsidan. Anropet triggar XMLHttpRequest objektet, som utgör det ”mellanlager” tidigare nämnt. XMLHttpRequest sänder i sin tur en vanlig HTTP request till webbservern, som där behandlar ärendet, hämtar efterfrågad information från en databas och returnerar sedan svaret i XML-format tillbaka till XMLHttpRequest objektet. Slutligen så sänds denna nya data till användarens interface, i detta fallet en webbläsare exempelvis, för att presenteras till synes utan att en omladdning av webbsidan gjorts (Puranik, Feiock and Hill, 2013).

Analyserar vi säkerheten hos Ajax så bör vi först ha i åtanke att i och med att Ajax använder sig av HTTP-protokollet. Följer de säkerhetsproblem med som finns där, vilket också givetvis då går åt andra hållet när det gäller ökad säkerhet i det fall man kör HTTPS. Med Ajax tillkommer utöver detta givetvis nya problem, i mångt och mycket beroende på att ett nytt språk som exekveras i webbläsaren introduceras; JavaScript (Erkkilä, 2012).

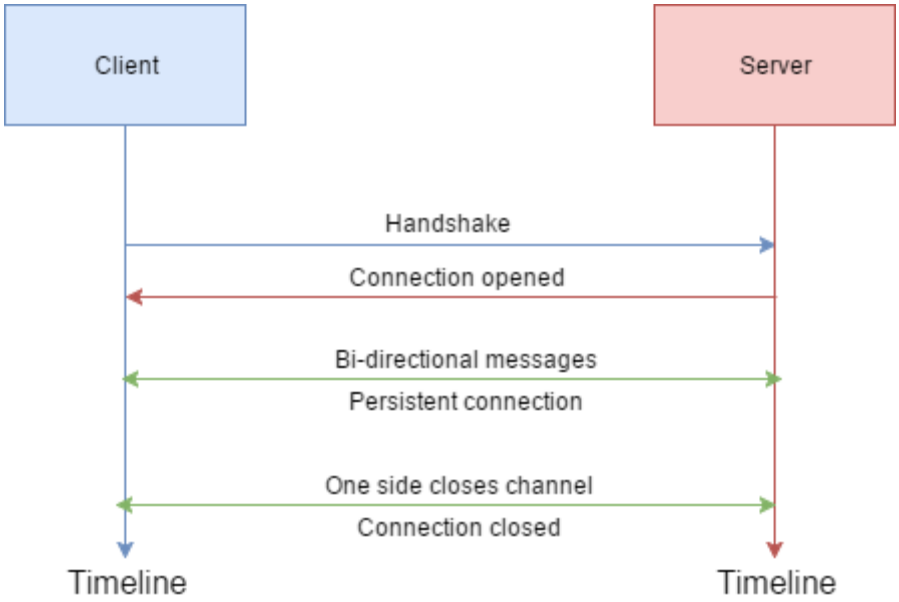
I och med att JavaScript körs direkt i webbläsaren så öppnas möjligheterna för att genomföra XSS-attacker. Exempel på detta presenteras under punkt 2.5.

2.3. WebSockets

Som tidigare påpekat har försöken till att få HTTP-protokollet att agera realtidskommunikatör inneburit en hel del problem i form av bandbreddsåtgång, fördröjningar av leverans (delay) och serverresurser som slösats i onödan. Med WebSockets avser man att lösa detta genom att introducera en fast uppkoppling mellan

klient/er och server/ar där meddelanden kan sändas asynkront mellan de båda (Erkkilä, 2012).

WebSockets opererar genom en enkel socket över webben som bygger på en TCP uppkoppling mellan klient och server, där arkitekturen har två delar. En typ av handskakning mellan klient och server där en uppkoppling sker. Detta genom att klienten skickar ett meddelande till servern med en förfrågan om att få koppla upp sig, där servern sedan sänder ett svar tillbaka (positivt/negativt). Om servern tillåter att en klient kopplar upp sig mot den, så ändras även protokollet med vilket kommunikationen sker, från HTTP/HTTPS till TCP/TLS (Pimentel and Nickerson, 2012). Samt att dataöverföringen mellan parterna i sig som är konstant fram till en part väljer att avbryta uppkopplingen (ex. genom att stänga sitt webbläsarfönster) (Pimentel and Nickerson, 2012). Detta gör också att kopplingen mellan de två (eller flera) blir i motsats till det vanliga HTTP-protokollet, ”stateful” istället för ”stateless” (Pimentel and Nickerson, 2012).



Figur 5. Visar hur en uppkoppling baserat på WebSockets fungerar

Erkkilä (2012) påvisar att i de fall där lite nya data tillkommer och med längre intervaller, så kan bruket av WebSockets spara uppemot 500 gånger så mycket bandbredd ställt emot klassisk HTTP polling och uppemot tre gånger mindre fördröjning (latency).

Satt i relation till ovanstående exempel för HTTP-protokollet när man vill få push notiser till sin mobil, så istället för att aktivt fråga servern med jämna mellanrum om det finns något nytt att hämta, kan man med WebSockets ha en konstant uppkoppling där data sänds till klienten endast när det faktiskt finns nya data att sända. Andra exempel på där WebSockets är en mycket duglig arkitektur att använda sig av är vid sportuppdateringar, uppdateringar från börsen etc. tjänster där realtidsuppdateringar är av högsta vikt för användarna (Kulshrestha, 2013).

WebSockets har en låg inlärningströskel i och med att det initialt var tänkt att tillhöra det som kallas HTML5 (men senare lyftes ur det sammanhanget). Detta innebär att allt man behöver för att få det att fungera är en modern webbläsare med stöd för HTML5

och JavaScript. Med modern webbläsare så avses Google Chrome 13 eller senare, Safari på MacOS, Firefox 4 eller senare, Opera 10.5 eller senare eller Internet Explorer 7 eller senare. Detta innebär också att de flesta bärbara enheter under operativsystemen Android och iOS kan köra applikationer som använder sig av WebSockets, förutsatt att klientens webbläsare är uppdaterad (Pimentel and Nickerson, 2012).

Skillnaden mellan exempelvis traditionell HTTP long polling och WebSockets är dock att en separat server krävs för att kunna hantera och bibehålla dessa uppkopplingar, detta påverkar dock inte den vanliga infrastrukturen där användare kopplas upp mot port 80. Port 80 är vanligtvis den port som används för en klient att ansluta mot en webbserver. En WebSockets server kan med fördel köras under samma port som en traditionell server, där sedan en inkommande request levereras till korrekt punkt i arkitekturen (Pimentel and Nickerson, 2012).

Det finns dock problem med just dessa fasta uppkopplingar ur ett säkerhetsperspektiv. Hackare har kunnat utnyttja det faktum att WebSockets baseras på en konstant uppkoppling mellan klient och server. Genom att simulera att tusentals unika klienter kopplar upp sig mot en server, har man därigenom kunnat genomföra DDoS (Distributed denial of service) attacker mot serverägare. Då verktyg vid dessa attacker inte fanns tillgängliga för att avgöra om en klient var äkta eller syntetisk, överbelastades systemen snabbt och fallerade (Rodriguez and Posegga, 2015).

Precis som med HTTP protokollet så finns samma problematik med SQL-injections och man-in-the-middle attacker (Heroku, 2017). Värt att nämna är också att WebSockets likt HTTP kan köras icke-krypterat, men har som HTTP har HTTPS, ett krypterat protokoll WSS, istället för WS (Heroku, 2017).

Detsamma gäller även när det kommer till XSS attacker och andra sårbarheter som kan utnyttjas vid bruket av JavaScript.

2.4. XSS-attacker

Gemensamt för alla tre tekniker ovan är att de alla är öppna för något som kallas XSS eller ”Cross-Site-Scripting”. Som lyftes under sektionen för HTTP så är grunden för att genomföra en XSS-attack möjligheten till att sända information till en webbplats (exempelvis ett kommentarsfält), samt att en klients webbläsare kan tolka JavaScript, vilket alla moderna webbläsare vid denna studies publikationsdatum kan (Erkkilä, 2012).

XSS-attacker är en typ av injektionsattacker, där script sänds in i en vanligtvis säker webbsida, för att sedan köras när användaren laddar in sidan i sin webbläsare. Eftersom att webbläsaren tror att skriptet som körs kommer från en pålitlig källa så ges det åtkomst till alla cookies, sessioner, eller annan känslig information som finns inläst i webbläsaren kopplat till siten som utsätts för en XSS-attack. Eftersom att JavaScript direkt kan påverka en webbsidas DOM-element, så kan hela sektioner eller webbsidan göras om direkt genom scriptet (OWASP, 2017a).

```
2
3
4   alert("This is a popup message!");
5
```

Figur 6. Visar upp JavaScript för att generera en popup på en webbsida.

Om vi antar att en användare postar denna kod i exempelvis ett kommentarsfält på en blogg, där koden inte avbryts från att köras. Så kommer nu en popup att visas varje gång sidan laddas in, för alla användare som besöker sidan med koden på. Ett relevantare exempel är XSS ”Cross-Site-Scripting” vilket i korthet går ut på att exempelvis en attackerare kapar en inloggning genom att förvärva sig den sessionscookie en annan användare har. På så vis kan hackaren överta denne användares inloggning och få tillgång till personens data (Hoffman and Labs, 2006).

Viktigt att framhålla här är att XSS är något som kan utföras utan att Ajax är involverat, däremot så med Ajax involverat agerar den en katalysator för attacken. Mestadels på grund utav hur data kan sändas och tas emot utan att en sida behöver laddas om för användaren. En användare kan ha råkat ut för en så kallad session hijacking genom XSS, där förfrågningar och data sedan sänds fram och tillbaka utan att användaren är medveten om att detta sker (Hoffman and Labs, 2006).

```
43  var sockets = []
44  while(true){
45      sockets.push(new WebSocket("ws://server:port");
46  }
```

Figur 7. Exempel på kod för att begära nya uppkopplingar mot en server i hög hastighet fram till att servern slutar svara.

WebSockets är lika utsatt för XSS-attacker som Ajax då JavaScript används även här. På samma sätt som tidigare så kan kod injiceras på en webbplats som i sin tur drar nytta av att webbplatsen använder sig av WebSockets.

Av figur 7 kan vi se att en loop i JavaScript körs vars enda uppgift är att skapa en ny uppkoppling mot servern i all oändlighet, en form av XSS-flooding-attack. Resultatet här skulle bli att webbservern till slut inte kan hantera alla nya uppkopplingar och därmed avböjer alla försök till en uppkoppling. Det finns dock skydd mot detta inbyggda i alla moderna webbläsare, Google Chrome exempelvis tillåter endast 255 simultana uppkopplingar från samma webbläsare, utslaget över alla öppna flikar (Chromium, 2017). Därav att det krävs ett större nätverk av användare för att göra denna attack gångbar. Här blir XSS återigen ett alternativ, exempelvis om denna skadliga kod placeras på en mycket välbesökt webbsida och därmed körs och hundra eller tusentals unika besökare.

Det finns dock sätt att arbeta proaktivt på för att skydda sig mot XSS-attacker. Det mest fundamentala är att alltid granska det användare sänder in i formulär på den webbplats man vill skydda. Inte tillåta att så kallade element-taggar i HTML som bäddar in JavaScript kod och alltid se till att avbryta kodstycken i element och text som potentiellt kan innehålla skadlig kod (OWASP, 2017a).

2.5. Sammanfattning

Som ovanstående presenterar så finns det idag i stort dessa tre tekniker som används frekvent när det kommer till kommunikation över webben. Som framgår är HTTP protokollet det som ligger till grund för den moderna webben, där vi genom att mata in webbadresser kan slå upp och navigera webbsidor. En förfrågan sänds från en klient

till en server om att hämta en webbsida, där webbsidan sedan sänds från servern och renderas i klientens webbläsare.

Nästa steg i detta var Ajax som introducerade ett tredje lager, ett mellanlager mellan klient och server som sköter kommunikationen. På så vis behöver inte användaren längre ladda om en hel webbsida när bara en specifik bit av en sida behöver uppdateras med nya data, eller sända data.

Slutligen WebSockets där ett helt nytt protokoll introducerats, WS/WSS. Vilket möjliggör fast uppkoppling mellan klient och server och därmed även ”statefulness” vilket innebär att servern minns vem klienten är under sin uppkopplingstid.

XSS-attacker är som framhållet i bakgrunden en form av injektionsattacker, där skadlig kod skjuts in i en webbsida. Detta kan ske exempelvis genom att JavaScript kod sänds in i ett kommentarsfält och att skydd mot att koden körs saknas. I och med att koden sänts in på webbsidan så tolkar i sin tur webbläsaren koden som säker och kör den, vilket leder till att full tillgång ges till de cookies och sessioner som finns i användarens webbläsare kopplat till webbplatsen. Detta kan i sin tur leda till exempelvis en så kalla sessionskapning, där scriptet tar över användarens inloggning på webbplatsen och hackaren kan i sin tur få tillgång till användarens personliga data. Som lyfts i sektionen om XSS så kan även XSS användas som ett verktyg för att rikta en DoS/DDoS attack genom anrop mot servern från ett script i webbläsaren, en typ av XSS-flooding-attack.

Som framhållet ovan så eftersom att tekniker efter HTTP bygger vidare på och lägger till ny funktionalitet så kan vi även se att sårbarheter ärvs. Med högre abstraktionsnivå ökar även mängden infallsvinklar för attacker. XSS-attacker bör framhållas här som bland de farligaste typerna av attacker, då de kan åstadkomma mycket stor skada med liten insats. Från det relativt harmlösa exemplet ovan där JavaScript-kod sänds in i ett formulär som sedan postas på en webbsida och presenterar popups, till man-in-the-middle attacker, session hijacking och mycket mera. Som nämnt tidigare eftersom dessa tekniker bygger vidare på varandra, är JavaScript grunden till problematiken, där Ajax kan agera katalysator och WebSockets i sin tur ytterligare bensin på elden. Exempelvis genom att Ajax sänder och tar emot data utan att användaren ser en omladdning av webbsidan denne befinner sig på.

Även om nya säkerhetsproblem uppstått så skall detta sättas i jämförelse med de fördelar som WebSockets ger. Som presenterats tidigare så var möjligheten för tvåvägskommunikation begränsad och egentligen endast möjlig genom att använda HTTP-protokollet på ett sätt det i grunden inte var avsett för, exempelvis long polling.

Andra fördelar med WebSockets som redan lyfts är att det kräver mindre bandbredd än tidigare tekniker, i och med att data bara sänds när det behövs. Latensen minskar även utefter samma logik. Slutligen så är tekniken som initialt utvecklades som en del under HTML5-standarden, kompatibel med de flesta moderna webbläsare, inklusive de som körs i mobila enheter.

I relation till allt detta bör återigen organisationsperspektivet understrykas och vikten för startupbolag att såväl vara medvetna om de säkerhetshot som finns, samt vilket proaktivt arbete som kan genomföras för att minska risken för att utsättas för hoten. Som framhålls av Fisher (2013) delar vi idag med oss av mera information idag online än någonsin tidigare. Som framhålls är det av yttersta vikt att göra det man kan som organisation för att skydda sina besökare och kunders data, såväl för att undvika de

direkta kostnader det kan innebära om dessa läcker. Som indirekta genom dåligt rykte och minskad tilltro till företaget (Fisher, 2013).

Studien framhåller även att det är billigare i det långa loppet att direkt från start av ett bolag, vilket denna studie fokuserar på, att investera i och granska den säkerhet man har. Framför att först i efterhand inse att man har bristande säkerhet i sina system och först då börjar göra något åt detta (Fisher, 2013).

3. Metod

Följande kapitel behandlar tillvägagångssättet för denna studies genomförande, för att producera det resultat som presenteras. Samt för att besvara given frågeställning. En genomgång av applicerbara metoder görs, där valda metoder för denna studie sedan presenteras. Planerat genomförande redovisas och slutligen presenteras validitet, de etiska aspekterna och reliabiliteten.

3.1. Tillgängliga metoder

För akademisk forskning krävs att de resultat man når fram till presenteras tillsammans med hur man de facto nått fram till dessa. Utan denna helhet blir det såväl svårt att avgöra en studies riktighet, som vid behov kunna återskapa resultaten då processen för hur studien genomförts saknas (Oates, 2005). Man använder dessa metoder som verktyg för att få fram resultat i den undersökning som genomförs. Men också för att säkerställa kvalité genom användandet av en vetenskapligt beprövad metod (Berndtsson *et al.*, 2008).

Det finns två olika paradigm att förhålla sig till här i bruket av dessa verktyg, kvalitativa och kvantitativa. Där det kvalitativa i en grov generalisering kan ses som exempelvis intervjuer med ett fåtal utvalda personer, med öppna frågor som oftast ger ett bredare spektrum i svaren än enkla ja och nej. Det kvantitativa paradigmet kan i motsats till exemplet ovan istället ses som något som är kvantifierbart. I relation till exemplet skulle detta kunna vara en stor enkätundersökning istället med starka frågor och direkta svar såsom ja eller nej (Oates, 2005). Anledningen till skillnaden i ansatserna härstammar i grunden från två olika skolor av vetenskap, där kvalitativa ansatsen har sin grund i social vetenskap. Respektive kvantitativa ansatsen sin grund i den traditionella vetenskapen: matematik, kemi och fysik med mera. Man kan avslutningsvis kortfattat säga att den kvalitativa ansatsen fokuserar på att studera mönster, sammanhang och ställa frågor som ”hur” och ”varför”. Så fokuserar istället den kvantitativa på att besvara frågor som kan kvantifieras, ”antal förekomster”, ”upprepningar” och så vidare. Detta resulterar också i att resultaten blir olika i sin natur, där kvantitativa ofta kan vara mycket specifika, medan kvalitativa i större mån ger möjlighet till att skapa modeller och teorier kring studerat ämne (Berndtsson *et al.*, 2008).

Detta innebär dock inte att bruket av det ena tillvägagångssättet automatiskt exkluderar bruket av det andra. I studier delas ofta det man vill undersöka upp i olika mål, där dessa olika mål med fördel kan ha olika metoder kopplade till sig för att anskaffa data (Berndtsson *et al.*, 2008).

3.1.1. Litteraturstudie

Litteraturstudie går ut på ett systematiskt granskande av ett problem, genom en analys av tidigare publicerade källor- utifrån en specifik frågeställning och syfte (Berndtsso*n et al.*, 2008).

Litteraturstudie skall dock inte förväxlas med att granska tidigare arbete för att utifrån detta ge någon typ av recension eller för att bekanta sig med ämnet. Målet här är att utifrån sökord identifiera och granska tidigare litteratur i ämnet, exempelvis vilket är relevant för denna studie. Ställa den empiri som förvärvas i kontrast med tidigare genomförda likartade studier. Men också för att bygga en kunskapsbas kring tidigare forskning och granska tidigare teorier, modeller och förslag på lösningar på

problemområdet, för att här identifiera ett mönster som kan appliceras som grund inför den kvantitativa delen av denna studie (Berndtsson *et al.*, 2008).

När det kommer till litteraturstudier inom informatik eller datavetenskap så är arbetet ofta så att den implementation, ramverk eller förslag på lösningar man ger. Ställs i kontrast med tidigare forskning och liknande lösningar där (Berndtsson *et al.*, 2008).

Det är också av stor vikt att man är uppmärksam på de potentiella konsekvenserna ens valda strategi för insamling av data kan ha. Systematisk analys och granskning av varje enskild källa man använder sig av är mycket viktigt, samt att använda sig av källor som är relevanta. Såväl kopplat till ämnet som undersöks, som kopplat till exempelvis publikationsdatum och tillförlitlighet (Berndtsso*n et al.*, 2008).

En annan svårighet vid arbete med en litteraturstudie är att veta när man har tillräcklig mängd data insamlad. Det är givetvis omöjligt att egentligen veta detta, men man bör alltid se till att ha en mycket god täckning för ämnet man studerar, samt för de resultat man presenterar. Samt att ge läsaren en god förståelse kring varför de specifika källor man använder sig utav har valts (Berndtsso*n et al.*, 2008).

Avslutningsvis är det viktigt att vara självreflekterande och uppmärksam på sin egen kunskap och subjektivitet inom ämnet. Att se till att man hela tiden försöker hålla sig opartisk och objektiv såväl i val av litteratur som de delar man väljer att lyfta till resultat och analys (Berndtsso*n et al.*, 2008).

Litteraturstudier är för min studie ett bra val och effektiv insamlingsmetod, detta trots att spjutspetsartiklar fokuserade på WebSockets finns i mindre antal. Men studiens bredd som innefattar även informationssäkerhet och specifika attacktekniker gör att grunden för tidigare forskning blir mycket bred.

3.1.2. Intervjuer

Det finns en rad olika sätt som intervjuer kan genomföras på och med olika mål. I och med detta så finns det en rad olika aspekter som måste tas i beaktning när valet av typen av intervju (Berndtsso*n et al.*, 2008).

Ett sätt att genomföra intervjuer på är genom så kallade öppna intervjuer, där forskaren har ingen eller liten kontroll över det som lyfts under intervjun. Så även om målet med intervjun är klar för intervjuaren, så är ämnen som lyfts inte planerade i förväg. Intervjuaren försöker istället att styra konversationen mot det område som avses att granskas. Detta är ett effektivt sätt att få fram områden intervjuaren själv inte kunnat/tänkt sig i förväg, men också för att undvika att hamna i en intervju där frågor som bara kan ge ja eller nej svar ställs (Berndtsso*n et al.*, 2008).

Motsatsen till detta är mera stängda intervjuer där en rad frågor redan i förväg definierats av intervjuaren, som här tar kommando tydligare till vilken väg intervjun skall gå.

Viktigt är också att välja en relevant respondent, exempelvis experter inom valt område. Planera en god flödesstruktur för intervjun. Använda sig av inspelningsutrustning eller inte- exempelvis kan det under öppna intervjuer vara svårare att hinna med att anteckna och inspelningar kan då vara till hjälp (Berndtsso*n et al.*, 2008).

Intervjuer skulle för min studie kunna fylla en viss roll i relation till informationssäkerheten. I övrigt en tidskrävande metod som skulle sett till den tekniska biten i denna studie ge lite avkastning på mycket tid investerat.

3.1.3. Enkäter

Enkätundersökningar är starkt förknippade med bruket av enkätsvar och statistiska tekniker för att analysera svaren. Ofta använt för att undersöka ett relativt välkänt fenomen, inom vilket det finns en större grupp respondenter med kunskap inom området (Berndtsso*n et al.*, 2008).

För min studie är en enkätundersökning mindre användbar, såväl utefter resonemanget som förs för intervjuer. Men också ur den aspekt att studien följer det kvalitativa paradigmet och inte kvantitativa.

3.1.4. Implementation

Enligt Berndtsson *et al.* (2008) så består många projekt inom datavetenskap och informatik av att utveckla nya lösningar, vilka kan vara nya mjukvaruarkitekturer, metoder, algoritmer och liknande. Dessa med målet att lösa ett befintligt problem på ett nytt sätt, med målet att det nya sättet skall ha fördelar över det gamla. Kopplingen mellan den kvalitativa delen av denna studie, med denna del blir här tydligare och styrker även de val av metod och strategier som gjorts. Teorin läggs härmed under den kvalitativa delen, där denna del sedan kan ta över för att testa det i skarp miljö.

Detta är även det övergripande målet med implementationsstrategin. Att med denna kunna påvisa att lösningen uppför sig på ett specifikt sätt eller har specifika egenskaper i relation till problemet den ska lösa. Utöver detta behöver implementationen i regel ofta jämföras med existerande lösningar innan slutsatser kan dras (Berndtsson *et al.*, 2008).

För denna studie skulle implementation kunna vara ett alternativ för att styrka de förslag som läggs fram, samt jämföra med litteraturstudien. Det ökar dock avsevärt studiens omfattning till den grad att det inte tidsmässigt skulle vara genomförbart.

3.1.5. Experiment

Ett experiment har som mål att undersöka ett fåtal variabler och på sättet dessa påverkas genom experimentella förutsättningar (Berndtsso*n et al.*, 2008). Typiskt sett så formas ett antal hypoteser vilka sedan avses att verifieras eller avkastas utefter experimentet genomförts. Inom datavetenskap och informatik som implementeras ofta en modell av ett system, där simulationer sedan körs för att se hur denna modell påverkas av olika variabler.

För denna studie skulle experiment kunna vara aktuellt att använda utefter de resultat som litteraturstudien ger. Dock utefter samma resonemang som under ovanstående punkt är projektet mycket tidskrävande och resultaten kan vara svåra att förhålla till fältet informatik.

3.2. Val av metod

Syftet med denna studie är att undersöka ta reda på de kritiska säkerhetshot som en övergång från HTTP-polling och/eller Ajax till WebSockets ger. För att utifrån givet resultat ge förslag för startupbolag hur implementationen av WebSockets kan göras för att minimera risken för XSS-attacker. Utifrån detta syfte så skulle det passa bra att genomföra en kvalitativ studie, då studien behöver grunda sig på tidigare forskning inom ämnet, standarden för WebSockets samt att insamling av data kring eventuella tidigare lösningar måste ske. Beaktning måste dock tas till studiens omfång, där en studie med denna bredd är tidsmässigt svår att genomföra. Därav kommer den

kvantitativa metoden ej att användas och istället ligga som rekommendation i avslutande diskussion för framtida forskning. Som tidigare nämnt så hade exempelvis en implementation eller experiment vart passande för studien, men rekommenderas istället för vidare studier.

Valet av en kvalitativ metod är alltså lämpligt i och med att det inom detta specifika område, då det är vid dags dato bara ett par år gammalt. Saknar en bredare grund av tidigare forskning, men det som finns ger en mycket god grund för att undersöka case och där det behövs komplettera upp med icke-akademiska artiklar. Med detta avses facklitteratur och artiklar från väletablerade organisationer inom informationssäkerhet och kopplade till utvecklingen av WebSockets-standarden, exempelvis Mozilla (Mozilla Developer Network, 2017b). Det skall också framhållas att även om spjutspetskompetensen inom WebSockets ännu är av mindre skala, så baseras WebSockets på tidigare tekniker med en bred akademisk grund. Utöver detta så skall även attackformen XSS granskas, som inte heller är i jämförelse med WebSockets alls lika nytt, där samma gäller att den akademiska grunden är mycket bred inom området och det finns mycket information att tillgå.

3.2.1. Litteraturstudie

Den kvalitativa metod som valts är av typen litteraturstudie, vilket innebär en systematisk granskning av artikel/artiklar av vetenskaplig typ relaterade till problemet med ett specifikt syfte i åtanke (Berndtsso*n et al.*, 2008).

Litteraturstudien kommer ligga till grund för att hjälpa mig besvara följande delfrågor i min frågeställning:

1. Hur kan hackare från ett serversideperspektiv när WebSockets används, avvärjas från uppkoppling?

2. Vilka skydd ur ett serversideperspektiv kan användas för att minimera risken för XSS-flooding-attacker när WebSockets används?

3.2.2. Strategi för litteratursökning

Urvalet av artiklar kommer ske utefter hur relevanta de är för granskat ämne, men även aktualitet. Det vill säga, är det som lyfts fram i artikeln vid dags dato fortsatt relevant. Eller har senare studier tillkommit som endera bygger vidare på given artikel, alternativt förkastar tidigare resultat. Just hur man avgör hur relevant en artikel faktiskt är, framhålls av bland annat Berndtsson *et al.* (2008) som något inte helt trivialt. Det finns dock ett flertal tekniker för att säkerställa att man faktiskt arbetar med relevanta källor och har övervägt alla relevanta källor.

Artiklarna kommer granskas utefter ett flertal kriterier. Metoden för artikeln har vart central att granska, bland annat för att kunna se i vilken skala undersökningen gjorts på, samt hur generell eller specifik de svar/modeller artikelförfattarna levererat faktiskt är. Vidare har även den ”impact” artikeln haft inom den akademiska världen gett en indikator om dess trovärdighet och relevans för denna studie. Detta i enighet med vad Berndtsson *et al.* (2008) rekommenderar. Vidare bör det framhållas att när icke-akademiska/peer-reviewed källor använts har trovärdigheten baserats på organisationens roll/kunskap inom ämnet samt hur etablerade de är/motsvarar akademiska källors resultat.

3.2.3. Analysmetod

Vidare kommer en analys av de resultat som ges av utvalda artiklar att genomföras. Dessa kommer ligga som grund för de slutsatser som dras i denna studie.

Det är slutligen viktigt att framhålla att en litteraturstudie kan ha många olika bakomliggande strategier för hur det är tänkt att man skall förvärva data (Oates, 2005). Denna litteraturstudie utgår ifrån att granska tidigare forskning kopplat till ämnet, för att genom detta förvärva relevant data för vidare analys.

En annan anledning till valet av metod grundar sig i det bland annat Webster och Watson (2002) framhåller rörande att det idag finns väldigt mycket forskning och artiklar inom informatik, men att det råder en brist på forskning som går tillbaka till tidigare publikationer, binder dessa samman för att täcka upp nya områden samt lyfter fram områden som kräver vidare forskning. Detta överensstämmer väldigt väl med det syfte och frågeställning jag har för denna studie- då den återknyter till tidigare forskning, använder denna för att applicera på ett nytt område (inriktning startupbolag) samt avslutningsvis har som avsikt att lyfta fram inom vilka områden kopplat till frågeställningen vidare forskning krävs.

En annan anledning som lyfts är hur IT är ett tvärvetenskapligt område, som drar kunskap från många olika områden, såsom datavetenskap och organisationsteori. I och med detta och i kombination med bristen på litteraturstudier som konkret binder dessa samman skapas en lucka inom fältet som i ökande grad har ett behov av att täppas till (Webster and Watson, 2002). Författarna lyfter också hur detta kan gå till och ger förslag på två tillvägagångssätt, det första där en bred teoretisk grund redan existerar- från vilket en litteraturstudie kan ske med tillägg i form av egna tester och modeller för att bygga vidare på denna kunskapsbas. Det andra förslaget som ges är mera relevant för denna studie, då författarna framhåller att vid en mindre teoretisk grund bör arbetet fokusera på detta nya och växande område och arbete för att exponera ämnet mera och mera grundläggande teoretisk grund presenteras för vidare studier i framtiden (Webster och Watson, 2002). Ett vanligt begrepp för detta är en metaanalys, där originalobservationer vanligtvis inte är en del av studien utan en sammanställning av resultat från redan publicerade artiklar.

Denna studie grundar sig i detta och i samverkan med det som lyfts fram av Berndtsso*n (*2008) har litteraturstudien genomförts.

3.3. Validitet

En viktig del i en vetenskaplig studie är att kunna styrka och fastställa att den information som förvärvats och framställs utifrån den datainsamling som gjorts är tillräckligt trovärdig.

I fallet kring litteraturstudien så bör begreppet behandlas utifrån forskaren och dennes fördomar, hur saker och ting uppfattas. En annan viktig punkt är fullständighet, vilket avser att under en litteraturstudie kunna avgöra när man har samlat in tillräckligt med material (Berndtsson *et al.*, 2008). Detta är givetvis omöjligt att egentligen veta, men genom att påvisa en tydlig strategi, ha en röd tråd genom sin studie och välja ut relevant innehåll ökar sannolikheten för att läsaren också litar på att tillräckligt med material har samlats in och analyserats.

Utöver detta är det även viktigt att uppmärksamma sin egen förståelse kring ämnet och det som studeras. Att arbeta med att analysera i ett självreflekterande sätt sitt eget

beteende under undersökningen för att minimera möjliga hot mot validiteten (Berndtsso*n et al.*, 2008).

Avslutningsvis är det viktigt att tydligt framhålla inte bara hur studien är tänkt att genomföras, genom given metodbeskrivning. Men också den strategi som valts och hur den har tillämpats under undersökningen.

3.4. Reliabilitet

Enligt Berndtsson *et al.*, (2008) beskrivs reliabilitet som hur robust och noggrann metoden är för de valda mätningarna. Det är därför av stor vikt att ha fokuserade frågor och frågor som har en spets tillräcklig för att kunna återupprepas i senare studier och tester. Det är också viktigt att frågorna som ställs är relevanta och ett faktiskt problem. Utöver detta att bland annat i den litteraturgranskning som sker, använda sig av bra källor och gärna flertalet inom samma område som undersöks för att bygga upp ett starkare stöd för de resultat och antaganden som presenteras/görs.

3.5. Etiska aspekter

För metoden som används för denna studie krävs inga typer av användaruppgifter eller annan data som kan kopplas till fysisk person. För den kvantitativa delen av studien så används endast slumpade tecken i randomiserad ordning, för att kunna mäta svarstider och överföringshastigheter. I och med detta så finns det ingen risk för att den data som används på något sätt går att koppla till fysisk person, eller väcka anstöt mot person eller folkgrupp.

För den kvalitativa delen av denna studie har artikelgranskning genomförts, där i huvudsak modeller och teorier för hur säkerheten vid bruk av WebSockets skulle kunna se ut. Även här har ingen data använts som kan kopplas till fysisk person. Teorier och modeller från tidigare forskning har tydligt refererats till och presenterats vid applicerbara punkter ligga till grund för den kvalitativa delens experiment.

För exempel där kod presenteras som potentiellt skulle kunna användas för skadliga attacker, så görs detta endast i kodstycken där kompletterande bitar saknas för att ge ett fullt fungerande attackverktyg. I relation till detta så presenteras även hur ett attack-exempel kan motverkas. Det är nödvändigt att uppvisa dessa sårbarheter både för läsarens förståelse men också för att belysa problemet och hur det kan åtgärdas proaktivt innan en WebSockets-baserad tjänst går live.

3.6. Genomförande

Under detta kapitel redovisas hur studien genomförts, avsikten med kapitlet är att visa att utvalda metoder faktiskt har applicerats. Samt redovisa hur planeringen följts och i viss mån ändrats om vid behov.

3.6.1. Litteraturstudie

Studien genomfördes genom att inledningsvis granska tidigare litteratur och dokument i ämnet. Här för att få en bild av och bakgrund till problemområdet, samt ge god grund för påståenden och resultat som lyfts fram. Frågeställningen arbetades fram genom att löpande granska de delar i tidigare forskning som ur den vinkling denna studie har (nystartade företag) ger bristande lösningar- för att där genom de resultat som i denna studie uppnåtts kunna täcka dessa.

För att få fram tillräckligt med empiri och kunna besvara de delfrågor som ligger till grund för frågeställningen har flertalet vetenskapliga artiklar behandlandes samma område granskats och ställts mot varandra. Detta för att ge en hög reliabilitet samt påvisa att granskat område och resultat har tydlig förankring i tidigare arbeten.

Litteraturstudien har i sin tur ofta presenterat resultat baserade på tester gjorda i Node.js, varav detta adopterades även för denna studie. Där resultat och lösningar som redovisats gjort så i Node.js/JavaScript. Detta påverkar dock inte resultaten i övrigt mera än att syntaxmässiga skillnader kan finnas/finns vid efterföljande tester baserat på denna studie.

Utifrån den litteraturgranskning som skett har empiri för såväl identifikation av problem förvärvats, som lösningar på dessa kunnat arbetas fram. En återkommande modell för hur arbetet sett ut presenteras under planerat genomförande.

3.6.2. Sökning och sökstrategi

Rörande urvalsprocess och anskaffande av litteratur så har följande databaser och sökmotorer använts: World Cat, IEEE, Springer och Google Scholar, där förstnämnda är av typen databas och sistnämnda av typen sökmotor över flertalet databaser. Verktyget Mendeley har även använts för samordning av artiklar. Tillgång till databaserna har givits genom Högskolan i Skövde. Sökord som använts vid letandet av artiklar inkluderar, men är inte begränsade till: *WebSockets, WebSockets security, HTTP, Ajax, DDoS, WebSockets Origin, XSS*. Även detta i enighet med de rekommendationer Berndtsson *et al.* (2008) ger.

Vidare bör framhållas att artiklar som granskats även valts utefter såväl teoretisk grund, med detta avses där artikelförfattare framhåller den teoretiska bas för vilket deras verktyg är tänkt att användas. Men även aktuella händelser i form av fallstudier, där artikelförfattare tillämpat given teori i verkligheten, för att därigenom genomföra en analys av de resultat man förvärvat sig. Detta ger en bredare grund för studien och en tydlighet mellan teori och de facto resultat i verkliga världen.

I och med hur WebSockets är en ny teknik så har brister inom den akademiska litteraturen vägts upp med trovärdiga källor från stora organisationer direkt involverade i standardens utveckling. Detta också för att komplettera upp den bristande forskning som finns inom studiens nisch. För att hitta information hos dessa organisationer, såsom exempelvis Mozilla Foundation ansvariga för bland annat webbläsaren Firefox (Mozilla Developer Network, 2017b), har interna sökfunktioner kompletterat med sökningar genom Google gjorts.

Detta även i enighet med det som lyfts fram av Webster och Watson (2002) där tidigare forskning från olika vetenskaper, tillsammans med nya källor vävs samman för att täcka upp ett område inom rådande forskning som i dagsläget saknas.

Med återkoppling till inledande del av detta delkapitel så bör även lyftas hur den litteratur som sökts fram sedan har valts ut. Genom sökning i ovan presenterade databaser så har utifrån söktermerna resultaten sett ut som följer:

Sökterm	Peer-reviewed artiklar (st)
WebSockets	43
WebSockets security	17
HTTP	1039995
Ajax	4189
DDoS	1423
Websockets Origin	15
XSS	358

Tabell 1. Uppvisar sökord med tillhörande antal sökträffar.

Här framgår tydligt den diskrepans som finns mellan äldre tekniker såsom HTTP och Ajax, med ett överväldigande antal artiklar innehållande söktermen. I jämförelse med WebSockets, där termen utan tillhörande bi-termer ex. ”security” endast genererar 43 resultat.

I och med den enorma mängd artiklar exempelvis HTTP resulterade i, så baserades urvalet här helt enkelt på vilken artikel som kunde ge en god bakgrund för denna studies bakgrundskapitel, samt baserat på peer-reviews och citeringar ansågs vara av mycket god kvalité inom den akademiska världen.

Pimentel och Nickerson (2012) samt Furukawa (2011) blev här basen för bakgrunden kopplat till HTTP, HTTP-polling samt HTTP long polling. Utvalda enligt ovanstående resonemang, men även då deras studier hade en inriktning mot säkerhet och vidare i studien satte tidigare protokoll i kontrast mot WebSockets. Fler källor har även använts i koppling till förklaring av HTTP-protokollet, baserade på samma urvalsprincip- men skall ses som stödjande källor framför primära som dessa två ovanstående.

Samma procedur genomfördes sedan för även Ajax, WebSockets samt mera specifika attacktyper utifrån problembeskrivning och avgränsning: DDoS, XSS med flera.

När det kommer till urvalet av de artiklar som fanns tillgängliga matchande söktermen ”WebSockets” så är antalet träffar där missvisande. Förvisso matchar de sökningen, men innehåller i många fall bara referenser i studien sedan till tekniken utan att vidare gå in på den. I grund och botten identifierades 10 stycken gångbara artiklar inom ämnet, där ett par snabbt kunde sorteras bort då de talade mera om vad WebSockets kommer kunna erbjuda (artiklar publicerade innan standarden sats), ett par andra fokuserade på prestanda och resultaten där av att gå från exempelvis Ajax till WebSockets. Dessa artiklar var förvisso av intresse och har inkluderats i studien, bland annat inom bakgrunden. Eftersom denna studie fokuserar på säkerhet så hamnade totalen på fem artiklar, där Erkkilä (2012) i sin studie fokuserar rakt igenom på säkerhetsaspekterna kring WebSockets. Denna artikel blev därmed den grund för vilket kompletterande källor sedan användes.

Nackdelen med studien är också i och med hur snabbt teknik utvecklas idag, att den vid denna studies genomförande redan är fem år gammal. Generellt så för protokoll och övergripande teknik har lite förändras, men exempelvis fler sårbarheter har framkommit allteftersom tekniken börjat tillämpas i större utsträckning. Här har whitepapers och stöd-källor i form av analyser från ansedda storföretag såsom Mozilla (2017) samt organisationer inom säkerhet såsom OWASP (2017). Dessa har kunnat ge en nyare bild och komplettera upp där Erkkilä (2012) haft endera bristande insikt eller att nya hot tillkommit med tiden.

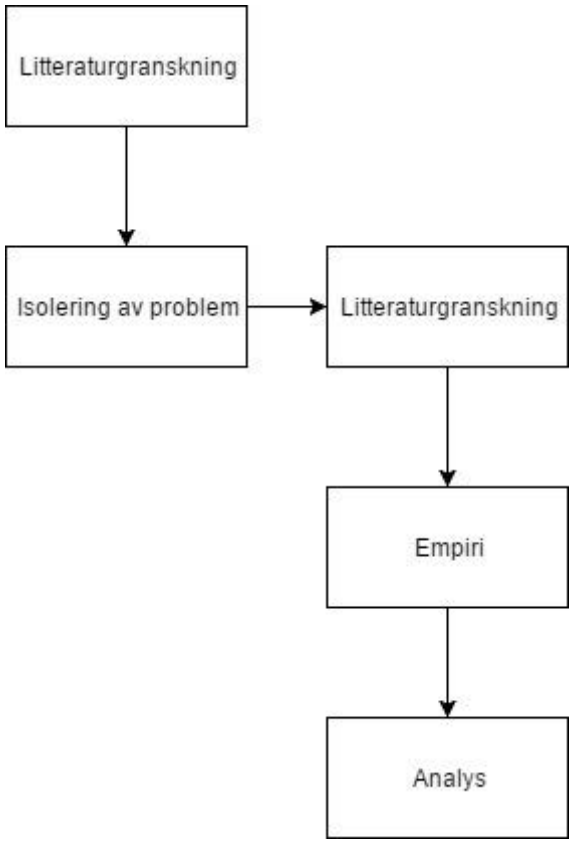
Större detalj i vilka kompletterande källor som valts ut, än de som är givna ovan till de grundläggande källorna kommer jag här inte att gå in på. De har valts utefter samma premisser och utifrån de resultat som sökningarna landat i.

Som lyfts fram i metodkapitlet är ett problem att veta när man egentligen har tillräckligt med data för att sammanställa det man funnit, analysera och dra slutsatser utifrån. Detta gjordes efter att rigorösa sökningar genomförts och litteratur kopplat till studiens ämne nått den grad att frågeställningen tydligt kunde besvaras och resultaten tydligt styrkas av de källor som använts.

3.6.3. Analys av litteratur

Efter genomförd analys av de resultat som litteraturstudien gett kunde sedan en slutsats formuleras med tydliga rekommendationer utifrån frågeställningen.

Avslutningsvis genomfördes en diskussion som reflekterade över studien och vald metod, samt presenterade förslag på fortsatt forskning inom ämnet.



Figur 8. Visar arbetsprocessen från bakgrund, till isolering av problem och förvärv av empiri för analys och slutsatsdragning.

Som modellen ovan påvisar leder litteraturgranskningen för de isolerade problemen (utgår från satt frågeställning) fram till en analys. Utefter den analys som sker kan sedan en slutsats presenteras där rekommendationer avses att redovisas för startupbolag vars mål är att adoptera tekniken WebSockets i sin verksamhet.

4. Resultat av litteraturstudie

Under detta kapitel redovisas de data som samlats in under genomförd studie, uppdelade utefter de delfrågor som framställts utefter frågeställningen. Syftet med detta kapitel är att på ett tydligt strukturerat vis lyfta fram den empiri som förvärvats utifrån definierad metod, för att sedan i följande kapitel möjliggöra följande analys och slutsats av studien.

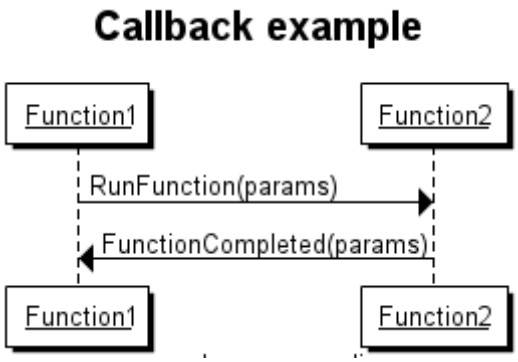
4.1. XSS-attacker mot WebSockets servrar

Under följande delkapitel lyfts resultaten från litteraturstudien fram kopplat till frågeställningens inriktning mot XSS-attacker, för att i följande delkapitel ytterligare behandlas utefter de specifika delfrågor som arbetats fram.

4.1.1. Specifika XSS sårbarheter hos WebSockets

Erkkilä (2012) framhåller att WebSockets likt Ajax är sårbart för XSS-attacker och kan i och med hur data sänds mellan klient och server utan att en webbsida till synes från klientens håll uppdateras, agera katalysator i en XSS-attack. Kapitel 2.4 ger bakgrund till detta.

En ny specifik XSS-attack som möjliggörs vid bruket av WebSockets, är möjligheten att utnyttja ett säkerhetshål i applikationen för att manipulera det som kallas för ”callbacks”. Callbacks i JavaScript är en funktion som vid slutförandet av en tidigare anropas och körs (Erkkilä, 2012).



Figur 9. Exempel på callback där Function1 anropar Function2 som körs. Efter Function2 är klar sänds en callback tillbaka till Function1 (Mozilla Developer Network, 2017a).

Detta sätt att genomföra en XSS-attack på möjliggör sniffning av trafiken (granska den data som sänds mellan klient och server), manipulera denna data eller implementera en man-in-the-middle attack mot WebSockets uppkopplingen (Erkkilä, 2012).

4.1.2. Skydd mot XSS-attacker

De steg man kan ta för att skydda sig mot dessa typer av attacker grundar sig i ett proaktivt skydd implementerat på klientsidan, såväl som på serversidan för att undvika att skadlig kod letar sig in i systemen (Kuosmanen, 2016). Nyckelordet här är inmatningsvalidering, där kontroller sker för att redan innan koden når in i ens system, desarmera attacken. HTML entity encoding går ut på att ersätta alla ASCII tecken med

deras HTML Entity motsvarigheter. Exempelvis ersätta ”>” med ”<” vilket får webbläsaren att tolka ”<” som en del av det HTML-element tecknet befinner sig, framför att se det som början på en ny HTML-tag (OWASP, 2017a). Denna inmatningsvalidering kan ske direkt på klientsidan där skadliga tecken efter att de matats in i en databas exempelvis och sedan hämtas ut, oskadliggörs. Inmatningsvalideringen kan även ske på serversidan, där en kontroll utförs av det innehåll som matats in och där skadliga tecken sedan ersätts eller tas bort innan data sparas ned i databasen (OWASP, 2017a).

Vidare bör samma typ av desarmering av potentiellt skadliga tecken på platser inom en webbsida där dessa kan köras ske. Utöver detta finns idag bland annat parametern ”HTTPOnly” vilket kan läggas till i en sessionscookie. Vad denna gör är att säkerställa att denna cookie inte kan nås genom script, detta medför därmed att XSS-attacker med sessionskapningar som mål avväpnas (Microsoft Developer Network, 2017). Denna metod för att skydda användare från session hijacking har enligt bland annat Atwood (2008) underskattats och är i realiteten ett av de mest effektiva skydden mot just detta.

OWASP (2017) lyfter även en alternativ skyddsmetod för att minska risken för XSS-attacker, genom en parameter som anges på webbplatsen ”Content-Security-Policy”. Denna kan se ut som följer:

”Content-Security-Policy: default-src: 'self'; script-src: 'self' secure.mydomain.se”

Syftet med att ange denna policy är att informera webbläsaren om att endast skript som hämtas från den definierade domänen skall tillåtas att köras. I och med detta kan domänen skripten finns på, levereras från filer utan skrivrättigheter eller möjlighet att manipulera (OWASP, 2017b).

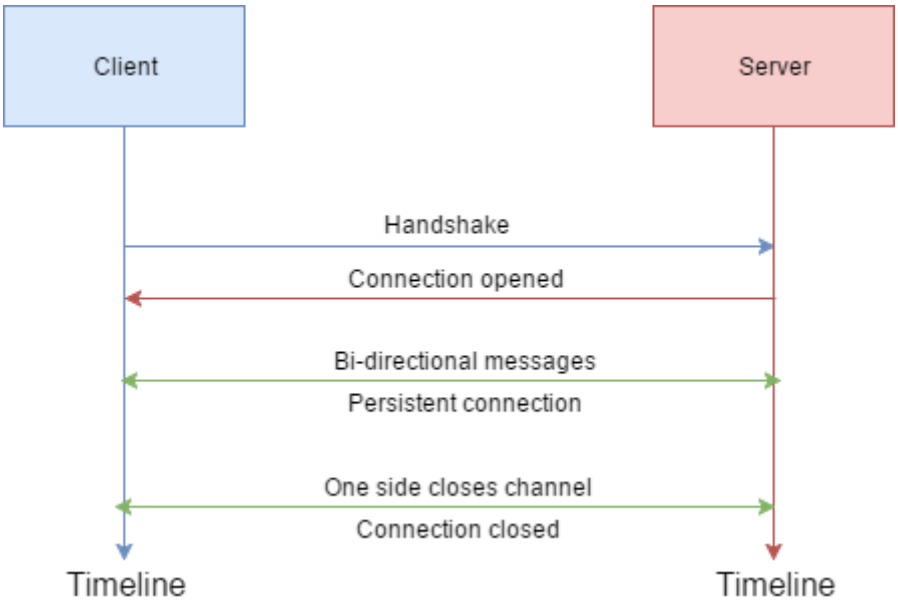
4.2. Avvärjning av uppkoppling

Under följande delkapitel lyfts resultaten från litteraturstudien fram kopplat till följande delfråga framställd från frågeställningen:

”Hur kan hackare från ett serversideperspektiv när WebSockets används, avvärjas från uppkoppling?”

4.2.1. Uppkoppling och verifiering av klient

Uppkoppling mot en server som kör WebSockets sker alltid i två steg, där en klient över HTTP-protokollet inleder en handskakning med servern och ber om att få koppla upp sig. Servern kan här ge ett positivt svar där uppkoppling sker och en TCP/TLS uppkoppling skapas mellan klient och server. Alternativt kan servern avvisa förfrågan från klienten varav en uppkoppling över TCP/TLS ej inleds (Erkkilä, 2012).



Figur 10. Visar hur uppkopplingsfasen ser ut

Uppkopplingen sker utan vidare insats från klientens sida än att besöka den IP- eller webbadress där WebSockets servern befinner sig. Detta sker vanligtvis genom en webbläsare så som Internet Explorer, Google Chrome, Safari eller Opera (Erkkilä, 2012).

Alla ovanstående webbläsare har givna parametrar för hur många simultana uppkopplingar med WebSockets som kan ske samtidigt (utslaget över alla flikar öppnade), där exempelvis Google Chrome accepterar uppemot 255 stycken (Chromium, 2017). Detta innebär i praktiken att exempelvis 20 flikar kan öppnas i Google Chrome, där alla sedan får en uppkoppling mot given WebSockets server med ett max tak på 255 simultana uppkopplingar.

Medföljande i varje förfrågan av en klient mot en server om uppkoppling är ett flertal parametrar/headers för att identifiera användaren och knyta denne till den eventuella fasta uppkoppling som initieras. Dessa headers är kortare textsträngar som innehåller information om exempelvis men inte exklusivt; klientens webbläsare, cookie-referenser och uppkopplingsstatus (Fette *et al.*, 2011).

```
GET /chat HTTP/1.1
 Host: server.example.com
 Upgrade: websocket
 Connection: Upgrade
 Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
 Origin: http://example.com
 Sec-WebSocket-Protocol: chat, superchat
 Sec-WebSocket-Version: 13
```

Figur 11. Exempel på hur de headers som sänds från klient till server vid förfrågan om uppkoppling ser ut (Fette *et al.*, 2011).

Origin (källa från vart uppkopplingen sker ifrån) är en av dessa och indikerar för servern från vilken IP- eller webbadress klienten uppger sig ansluta sig från, denna header sätts automatiskt av webbläsaren (Erkkilä, 2012). Enligt Fette *et al.* (2011) är Origin ett skydd för att förhindra ”cross-origin” bruk (tillgång till servern från andra källor än vart servern huserar). WebSockets skiljer sig från tidigare tekniker på så vis att här kan en klient ansluta sig från ett IP till ett annat (över domäner) fritt, något som exempelvis inte är möjligt med Ajax (Karlström, 2015). Vad detta innebär rent konkret är exempelvis att en webbplats finns på adressen *”www.exempel.se”* som i sin tur ansluter till en WebSockets server vid *”ws.exempel.se”*. En person med avsikt att tillgå information från denna server skulle i sin tur kunna sätta upp en webbplats på *”www.exempel.com”* och ansluta mot *”ws.exempel.se”* (Karlström, 2015).

Rent konkret innebär detta att du har din egen webbplats och WebSockets server, som en tredje part kan koppla upp sig mot och hämta/sända data från/till och därmed dra resurser från din server och bandbredd. Genom att WebSockets servrar kan anropas från domäner utanför sin egen öppnar det även upp för session hijacking (se kapitel 2.4) (Schneider, 2013).

Origin lyfts fram som central i arbetet för att förhindra att klienter ansluter från källor serverägaren ej önskar. Vidare konstateras dock av Erkkilä (2012) att en enkel vitlistning av källor som får ansluta till servern är otillräckligt då manipulation av origin är möjlig och enligt Schneider (2013) enkelt att genomföra genom exempelvis ett script i Python eller PHP som körs utanför en webbläsare och dess inbyggda skydd mot detta.

```
var WebSocketServer = require('websocket').server;
var http = require('http');

var server = http.createServer(function(request, response) {
    console.log((new Date()) + ' Received request for ' + request.url);
response.writeHead(404);
    response.end();
});
server.listen(8080, function()
{
    console.log((new Date()) + ' Server is listening on port 8080');
});

wsServer = new WebSocketServer({
    httpServer: server,
    autoAcceptConnections: false
});
```

Figur 12. Exempel i JavaScript för hur en server sätts upp i Node.js och JavaScript (Node Packaged Modules, 2017).

I figur 12 ser vi hur en WebSockets server sätts upp i Node.js med modulen Socket.IO (Socket.io, 2017), där språket JavaScript används såväl på klient- som serversidan. Variabeln *”autoAcceptConnections”* är av stor vikt att den anges och vid publicering av

webbplats är satt som *”false”*. Detta är ett första skydd mot att förhindra att klienter ansluter från vilket domän som helst.

Detta förhindrar dock inte anslutning från okända källor i högre grad än att se till att de protokollregler som satts upp för servern följs korrekt. För detta behövs en separat funktion som granskar från vilket domän en klient försöker ansluta (Node Packaged Modules, 2017).

```
function originIsAllowed(origin) {
  // put logic here to detect whether the specified origin is allowed.
  return true;
}

wsServer.on('request', function(request) {
    if (!originIsAllowed(request.origin)) {
      // Make sure we only accept requests from an allowed origin
      request.reject();
      console.log((new Date()) + ' Connection from origin ' + request.origin + ' rejected.');
      return;
    }
}
```

Figur 13. Visar exempel på hur en funktion kan läggas till för att granska att en klient ansluter från en godkänd källa (Node Packaged Modules, 2017).

I figur 13 presenteras hur en sådan funktion kan läggas till i kodblocket för initiering av en WebSockets server. Genom att definiera denna kontrollmetod och koppla den till händelsen *”on request”* vilket innebär när servern anropas, kan klientens origin header jämföras mot en eller flertalet godkända källor definierade av serverägaren och på så vis förhindra att klienter från andra håll kan ansluta till servern. Källorna anges som vanliga URL:er alternativt IP-adresser (Node Packaged Modules, 2017).

Origin headers skall i sken av ovanstående fortsatt inte ses som ett sätt att säkra applikationen på, från anslutning från okända källor. Utan främst som ett sätt att skilja på requests från olika platser och källor (Heroku, 2017). Detta då så kallad spoofing (engelskt uttryck) eller på svenska manipulation/förfalskning av denna kan ske på flertalet sätt och därmed kringgå verifikationen i exemplet ovan.

Som tidigare nämnt så sätts origin värdet automatiskt i den webbläsare man använder (Fette *et al.*, 2011), men detta kan enkelt kringgås genom att anropa en WebSockets server genom exempelvis ett script i PHP eller Python, där attackeraren själv är i kontroll över vilka värden som sänds till servern (Erkkilä, 2012).

Att definiera vilka källor som skall ges tillgång till WebSockets servern skall därmed ses som ett steg i att sinka en attack och skydda från exempelvis tillgång till servern av misstag (Erkkilä, 2012). Det har dock även en skyddande effekt mot session hijacking i och med att även om en attackerare har en webbplats vars syfte är att kapa besökande klienters session, så har dessa personer i sin tur inte gjort något för att dölja från vilken källa de försöker koppla upp sig mot WebSockets servern. (The Open Web Application Security Project, 2017).

Ett annat sätt att hantera felaktiga origins på, i det fall origins helt saknas vid ett anrop kan med fördel headern ”referer” granskas istället. Referer innehåller data från vilken

sida användaren navigerat från för att initiera en uppkoppling mot WebSockets servern (Fette *et al.*, 2011). I det fall denna är från en domän godkänt av servern kan en anslutning öppnas, i annat fall avböjer server uppkopplingen (Kuosmanen, 2016).

WebSockets protokollet i sig saknar lösning för autentisering av användare på serversidan. Istället är rekommendationen här att för att autentisera användare inloggade på en webbplats, att använda denna autentisering istället. Detta kan ske genom exempelvis kontroll av sessionscookies (Fette *et al.*, 2011).

4.3. XSS-flooding-attacker

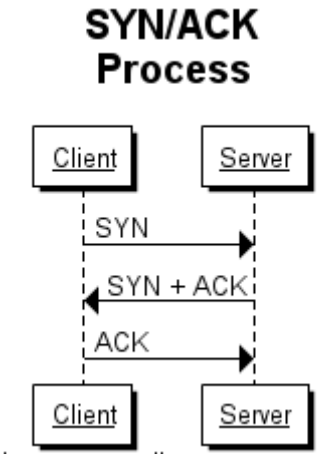
Under följande delkapitel lyfts resultaten från litteraturstudien fram kopplat till följande delfråga framställd från frågeställningen:

”Vilka skydd ur ett serversideperspektiv kan användas för att minimera risken för XSS-flooding-attacker när WebSockets används?”

4.3.1. XSS-flooding och protokollexploits

DoS-attacker är bland de vanilgast förekommande typerna av attacker mot servrar och webbplatser, mycket beroende på att de kräver liten insats ur teknisk synvinkel (implementation) för att genomföras men med förödande resultat (Whitman & Mattord, 2011). Målet med en DoS-attack är att överväldiga målet med enorma mängder requests, med målet att ta tjänsten offline. En DDoS-attack skiljer sig på så sätt mot en DoS-attack att istället för att en klient genomför denna attack, genomför ett nätverk av klienter denna attack för att sänka ett mål (Whitman & Mattord, 2011).

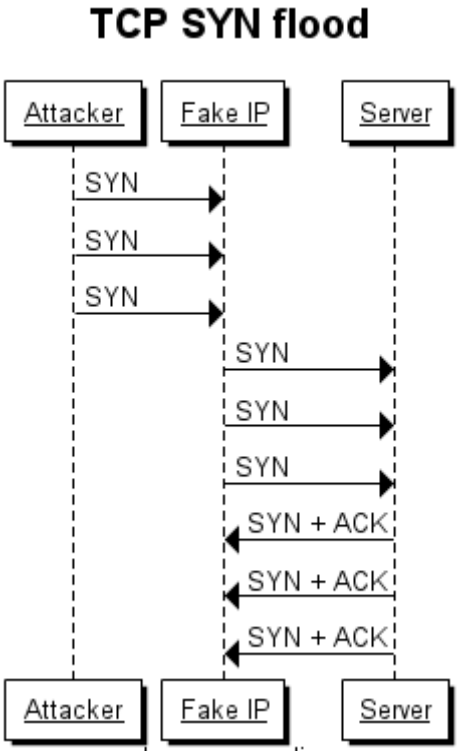
Det finns olika typer av DoS-attacker, bland de vanligaste utnyttjar olika typer av sårbarheter i det protokoll en server arbetar med/mot. I fallet med WebSockets är det TCP-protokollet och en DoS-attack som kallas TCP SYN Flood Attack (Jain and Singh, 2012). Sårbarheten som utnyttjas här är kopplat till hur en uppkoppling mellan klient och server görs vid uppgraderingen från en HTTP-uppkoppling, till TCP, specifikt själva handskakningstillfället (Jain and Singh, 2012).



Figur 14. Visar hur en handskakning mellan server och klient skall se ut (Jain and Singh, 2012).

Figur 14 uppvisar hur en handskakning mellan klient och server skall se ut mellan en klient och server. Denna handskakning ligger till grund för en uppkoppling mot TCP-

protokollet och sker alltid genom att när en klient vill koppla upp sig mot en server sänder den en SYN-request till servern, server svarar med en SYN/ACK för att synkronisera med klienten, där klienten avslutningsvis besvarar detta genom att sända en ACK (Jain and Singh, 2012).



Figur 15. TCP SYN flood genomförs där servern inväntar svar från ett ogiltligt IP (Jain and Singh, 2012).

När en DoS-attack genomförs utnyttjar man sättet denna handskakning sker på, genom att klienten sänder en SYN-request till servern, men med en ogiltlig IP-adress. I och med att minne allokeras på serversidan när en SYN + ACK sänds tillbaka till klienten, som i och med en felaktig IP-adress aldrig levereras, utnyttjas löpande mera minne fram till att inget mera finns att tillgå och servern avvisar fler uppkopplingsförsök, figur 16 illustrerar detta (Jain and Singh, 2012).

Ett liknande sätt att genomföra denna typ av attack på är att tillåta att en uppkoppling mellan klient och server faktiskt sker, men i sådan takt att det överbelastar servern. För att detta ska vara möjligt dock så krävs ett större nätverk av användare som genomför attacken, en såkallad DDoS-attack (Jain and Singh, 2012).

Genom att exempelvis genom en XSS-attack injicera en loop i JavaScript vars enda uppgift är att skapa en ny uppkoppling mot servern i all oändlighet kan detta genomföras. Det finns dock skydd mot detta inbyggda i alla moderna webbläsare, Google Chrome exempelvis tillåter endast 255 simultana uppkopplingar från samma webbläsare, utslaget över alla öppna flikar (Chromium, 2017). Därav att det krävs ett större nätverk av användare för att göra denna attack gångbar. Attackmöjligheterna kan även stävjas genom att införa en kontroll av origin-headern (lyft i föregående fråga).

Ett annat sätt att stävja denna typ av överbelastningsattack på är att konfigurera värdet för timeout på en uppkoppling. WebSockets pingar konstant en klient för att se om denne fortsatt är uppkopplad och aktiv. Detta kallas för ”ping pong” och för att styra detta kan man använda sig av ett ramverk som kan sända och ta emot dessa paket. WebSocket-Node är ett sådant ramverk där genom parametern ”setKeepAlive” sätts ett värde i millisekunder för hur ofta denna kontroll skall ske, genom att sätta den lägre bryts uppkopplingen med en inaktiv användare snabbare (McKelvy, 2014).

För andra typer av DDoS-attacker ligger skyddet i högre grad hos hårdvara än mjukvara, exempelvis riktade attacker mot en specifik IP-adress, något som dock faller utanför omfånget av denna studie (Jain and Singh, 2012). För detta finns även tjänster som tillhandahåller DDoS skydd, exempelvis CloudFlare som erbjuder skydd mot en fast månadskostnad (Cloudflare, 2017).

5. Analys

Under detta kapitel analyseras resultaten av litteraturstudien i kombination med teoretiska bakgrunden, för att besvara de delfrågor som arbetats fram utefter satt frågeställning. Samt avslutningsvis frågeställningen i helhet.

5.1. Avvärjande av hackare från serversidan

Resultatet av den studie som genomförts presenterar att det i dagsläget finns ett flertal olika aspekter att ha i beaktande när det kommer till att skydda sig från att hackare försöker ta sig in och/eller påverka en tjänst funktionalitet. Som inledningsvis lyfts av Erkkilä (2012) grundar sig mycket kring de säkerhetsproblem som finns vid bruket av WebSockets, hur protokollet fungerar. Speciellt i momentet där en uppkoppling från klient mot server sker över HTTP, för att sedan uppgraderas till en TCP uppkoppling.

Den initiala identifikationen och uppgraderingen av uppkopplingen är basal i sitt utförande, där en så kallad ”handshake” genomförs mellan klient och server, enkelt förklarat parameterutbyten mellan klient och server. Här lyfter Erkkilä (2012) speciellt en parameter som sänds med i det paket av ”headers” som utbyts mellan klient och server, *”origin”*. Denna Origin-parameter innehåller information som berättar för servern från vilket domän/IP-adress en klient försöker ansluta (härefter benämnt källa). Exempel på hur denna data ser ut presenteras i figur 10.

En av de kombinerade för- och nackdelarna med WebSockets som lyfts i anslutning till detta, är möjligheten att koppla upp sig mot en server från källor som ligger utanför den som WebSockets servern själv huserar. Detta är en unik funktionalitet i jämförelse med tidigare tekniker så som Ajax, men också en av de mest sårbara aspekterna av WebSockets om det inte hanteras korrekt (Karlström, 2015). Det skydd som naturligt finns mot XSS i och med hur exempelvis Ajax fungerar, med same-origin-policyn (att bara kunna posta inom samma källa) lyfts här bort helt.

Origin kommer här in i bilden som ett verktyg för att kontrollera från vilken källa en klient försöker ansluta. Schneider (2013) bland annat lyfter dock fram sårbarheten i detta och presenterar hur denna header kan manipuleras för att utge sig komma från en källa den i själva verket inte kommer från. Vad detta innebär rent konkret är att en WebSockets server kan ha applicerat skydd genom exempelvis en form av vitlista för vilka källor som får ansluta till servern. Men genom att manipulera origin headern kan klienter från helt andra källor fortsatt koppla upp sig mot WebSockets servern och därmed ges tillgång till att såväl sända och ta emot data.

```
GET /private/information HTTP/1.1
Host: www.exempel.se
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:23.0) Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: se-sv
Accept-Encoding: gzip, deflate
DNT: 1
Sec-WebSocket-Version: 13
Origin: https://www.exempel.se
Sec-WebSocket-Key: x7nPlaiHMGDBuJeD6l7y/Q==
Cookie: JSESSIONID=1A9431CF043F851E0356F5837845B2EC
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

Figur 16. Exempel på de headers som utbyts mellan klient och server

I följande exempel baserat på förvärvad empiri i föregående kapitel, sänds cookie headern för användarens session med i handskakningen med WebSockets servern. Vi kan också se att origin här är samma som för den WebSockets server vi avser att ansluta till. Förutsatt att ett skydd mot felaktiga origins inte är implementerade på serversidan kan detta lätt utnyttjas för att överta sessionen och därmed få tillgång till klientens alla data och information (Schneider, 2013).

```
GET /private/information HTTP/1.1
Host: www.exempel.se
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:23.0) Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: se-sv
Accept-Encoding: gzip, deflate
DNT: 1
Sec-WebSocket-Version: 13
Origin: https://www.exempel.com/fake-site/
Sec-WebSocket-Key: x6nPlaiHgGDBuJeD6l7y/Q==
Cookie: JSESSIONID=1A9431CF043F851E0356F5837845B2EC
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

Figur 17. Uppkoppling mot WebSockets servern från en skadlig webbplats

Figur 17 visar hur samma WebSockets server vid domänen ”exempel.se” kopplas upp emot, där samma sessions cookie header sänds med i handskakningen. Med skillnaden att detta sker från en annan domän *”exempel.com/fake-site/”*. I och med detta kan nu

andra script under denna domän ges åtkomst till användarens information vid *”exempel.se”* och användarens session är övertagen av denna skadliga webbplats. Detta är återigen som resultaten från litteraturstudien framhåller, bland annat från definitionen av WebSockets protokollet inte en miss som gjorts när standarden spikats. Snarare en funktionalitet som i vissa specifika tillämpningar kan vara önskvärd, men med bieffekten att utan att ha koll på problemet och motmedel implementerat, ett öppet hål in i en webbapplikation (Fette *et al.*, 2011).

Vidare lyfts även att trots att manipulation av Origin är ett problem, så har åtgärder tagits för att försöka stärka upp skyddet mot att detta utnyttjas genom att göra manipulationen tidskrävande samt att egna verktyg måste implementeras för att kunna utnyttja sårbarheten. Ett exempel på detta är att alla moderna webbläsare vid denna studies publikationsdatum, inklusive de för mobila enheter (iOS och Android) omöjliggör manipulering av Origin headern (The Open Web Application Security Project, 2017).

Andra förslag som litteraturgranskningen lett fram till är bland annat att på serversidan implementera en kontroll av vilken källa en klient ansluter. Att skapa en så kallad vitlistning, vilket innebär att en eller flera fördefinierade källor tillåts att ansluta till WebSockets servern, resterande avvisas.

```
function originIsAllowed(origin) {
  if(origin == "https://exempel.se")
  {
    return true;
  }
  else
  {
    return false;
  }
}

wsServer.on('request', function(request) {
    if (!originIsAllowed(request.origin)) {
      // Make sure we only accept requests from an allowed origin
      request.reject();
      console.log((new Date()) + ' Connection from origin ' + request.origin + ' rejected.');
      return;
    }
}
```

Figur 18. Exempel på hur en specifik origin kan anges för att kontrollera om klientens origin är densamma.

I figur 18 presenteras hur detta görs mot en enda specifik källa, här *”https://exempel.se”*. Där om klienten ansluter från denna källa servern accepterar uppkopplingen, annars avböjer den med ett meddelande sänt till klientens webbläsarkonsol. Denna typ av kontroll kan även användas i de fall det behövs som en svartlistning av domäner istället för som exemplet ovan uppvisar, vitlistning. Tillvägagångssättet är i princip detsamma, men där domäner som ej skall ha tillgång till WebSockets servern listas och kontrollerar, istället för vice versa.

Som framhålls inledningsvis i denna analys så förklarar andra källor, såväl med akademisk bakgrund så som Erkkilä (2012) som icke-akademisk/icke peer-reviewed (Schneider, 2013) att denna typ av kontroll bara ger ett visst skydd. Med detta menar man att ett skydd sätts upp mot de användare som ansluter till en WebSockets server genom en webbläsare, vilket är det traditionella sättet att ansluta på. I och med att alla moderna webbläsare vid denna studies publiceringsdatum omöjliggör manipulation av origin, skapas på så vis ett visst skydd.

Vi skulle kunna ponera exempelvis att en klient försöker ansluta i icke-skadligt syfte, exempelvis för att kunna deltaga i en chatt men från en annan källa än där WebSockets servern huserar. Nackdelen med detta blir dock att man även på grund av detta skulle kunna igla på WebSockets servern och den får hantera trafik för en tredje part. Här kan ett skydd som kontroll av origin vara ett bra skydd, då ett antagande kan göras att när en uppkoppling nekas kommer klienten att avbryta sina försök att koppla upp sig från en annan källa. Detta är en personlig åsikt och analys baserat på de resultat som presenterats av litteraturstudien.

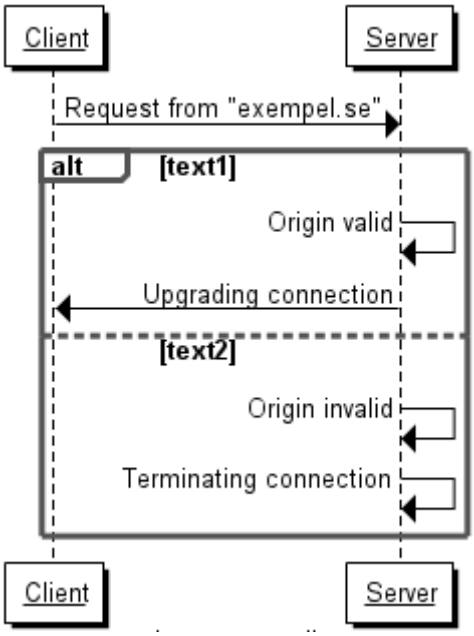
Att sätta upp en vitlistning för vilka källor som skall ges tillgång till att koppla upp sig mot en WebSockets server kan dock kringgås relativt enkelt. Detta genom att implementera ett script som inte körs i en webbläsare och därmed kan sända egendefinierade headers till servern.

```php
$headers = array(
  'host'                  => "http://exempel.se" . ":" . "8080",
  'user-agent'            => 'define here what browser to spoof',
  'connection'            => 'Upgrade',
  'upgrade'               => 'websocket',
  'origin'               =>  'define here the origin to tell the server you are from',
  'sec-websocket-key'     => 'key generated in a separate funcion',
  'sec-websocket-version' => '13',
);
```

Figur 19. Figuren visar ett kodblock från ett PHP-script som kan manipulera källan en klient ansluter från.

I figur 19 presenteras ett exempel i PHP där en array skapas och alla värden som normalt sänds med i en header till en WebSockets server manuellt definieras, detta baserat på resultaten från litteraturstudien. Detta innebär at såväl vilken webbläsare man använder, som vilken källa man ansluter från definieras av attackeraren.

Resultaten från litteraturstudien påvisar också att i det fall om origin eller referer header inte sänds med i en handskakningsförfrågan från klient till server, kan detta enklast hanteras genom att implementera en kontroll likt den för vitlistan. I denna kontroll försöker man läsa av origin och referer, om detta misslyckas så avvisar man uppkopplingen och klienten uppgraderas inte från en HTTP till en TCP uppkoppling. Flödesdiagrammet i figur 20 visar hur flödet för denna kontroll kan se ut.



Figur 20. Validering av att origin är korrekt, vid korrekt origin uppgraderas uppkopplingen till TCP/TLS, annars avbryts uppkopplingen med klienten.

Avslutningsvis rörande Origin så pekar även resultatet på att vid bruk av ramverk/moduler utvecklade för att arbeta med WebSockets. Exempelvis Socket.IO som erbjuder fördefinierade parametrar att modifiera för att tillåta full tillgång till en WebSockets server från alla källor, inga förutom den källa servern utgår ifrån, eller enligt en vitlista. Rekommendationen här från såväl organisationer som Mozilla Developer Network (2017b), Heroku (2017) som akademiska källor som Kuosmanen (2016) framhåller vikten av att alltid sätta dessa parametrar till att blockera alla externa källor, eller till en vitlista om inte speciella skäl finns och man är medveten om problemen som kan uppstå.

Slutligen så framhåller resultatet att det inte finns ett inbyggt stöd för autentisering av användare i WebSockets protokollet. Därmed behöver detta ske på annat vis och kan med fördel sedan kombineras med kontrollen av origin för att bara ge inloggade användare tillgång till anslutning (Fette *et al.*, 2011).

5.2. XSS och XSS-flooding-attacker

Som resultatet av litteraturstudien framhåller så är de nyckelbegrepp som använts här XSS samt DoS. Vid första anblick av resultatet kan korrelationen mellan DoS och XSS-flood-attacker kännas förvirrande (personlig åsikt). Anledningen är i grund och botten kopplad till begreppet DoS/DDoS bredd och de många infallsvinklar som finns för att genomföra en sådan attack. Utefter de avgränsningar som gjorts för denna studie och satt frågeställning har en specifik typ av DoS-attack utförbar genom XSS behandlats, då denna som resultaten påvisar snabbt kan få en WebSockets server att sänka sig själv genom överbelastning av förfrågningar.

5.2.1. XSS-attacker

Utöver detta så har även en granskning av litteraturen kring XSS kopplat till WebSockets genomförts, där en specifik typ av XSS-attack genom så kallade callbacks

i JavaScript lyfts fram som ett extra stort hot vid bruket av WebSockets (Erkkilä, 2012). Denna typ av attack möjliggör sniffning av trafik mellan klient och server, manipulation av data och man-in-the-middle attacker. Alla potentiellt mycket skadliga då känslig information som utbyts mellan klienter och server kan kapas och genom att utöver detta genomföra en man-in-the-middle attack, få total kontroll över klienternas interaktion med servern. I och med att en server kan hantera en stor mängd klienter simultant, blir detta extra allvarligt och med små medel lönsamt för en hacker att förvärva sig stora mängder data.

De resultat som lyfts fram från artikelgranskningen gällande skydd mot denna typ av XSS-attacker samt XSS-attacker på ett mera generellt plan nämner uteslutande att arbetet bör ske proaktivt. Har en hackare väl tagit sig in i ett system genom en sårbarhet kan skadan redan vara skedd, även om sårbarheten efter detta täpps till (Kuosmanen, 2016).

Detta proaktiva säkerhetsarbete kring att säkra upp sin webbplats mot XSS-attacker visar resultatet i störst utsträckning vara kopplat till inmatningsvalidering. Med detta avses att endera på klient eller serversidan alternativt båda granska den data som matas in av användare och vid behov radera eller genom något som kallas HTML entity encoding konvertera om potentiellt skadliga tecken till säkra (OWASP, 2017a).

Utifrån det som refereras till som skadliga tecken av bland annat OWASP (2017a) har följande tabell sammanställts med potentiellt skadliga tecken och deras säkra motsvarighet ur HTML entity encoding.

Potentiellt skadlig	Konverteras till ->	HTML entity encoding
&	-->	&
<	-->	<
>	-->	>
”	-->	"
’	-->	'
/	-->	&#x2F

Tabell 2. Exempel på potentiellt skadliga tecken och deras säkra motsvarighet med HTML entity encoding, baserat på OWASP rekommendationer (OWASP, 2017a).

Två andra skydd mot XSS-attacker upptäcktes också i och med litteraturstudien, där den ena grundar sig i att skydda att användares cookies kan läsas av. Detta förhindrar exempelvis session hijacking. HTTPOnly parametern styr detta och bör läggas in i alla cookies som exempelvis har med en användares session att göra, eller som på annat sätt kan leda fram till att känslig information kan förvärvas genom en XSS-attack (Microsoft Developer Network, 2017).

Utifrån litteraturstudien har följande exempel tagits fram på hur en cookie kan se ut där denna parameter är implementerad:

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
Vary: Accept-Encoding
Server: Microsoft-IIS/7.5
Set-Cookie: ASP.NET_SessionId=rr1337hxr; path=/; HttpOnly
X-AspNet-Version: 2.0.50727
Set-Cookie: user=t=bf038dudkamkdfi48; path=/; HttpOnly
X-Powered-By: ASP.NET
Date: Tue, 15 May 2017 09:00:00 GMT
Content-Length: 2838
```

Figur 21. Exempel på hur en sessions cookie har fått HttpOnly parametern tillagd för att skydda mot session hijacking.

Som resultatet av litteraturstudien framhåller så har denna tillagda parameter i cookies visat sig vara ett väldigt effektivt skydd mot session hijacking och bör som standard implementeras för varje cookie som innehåller känslig information (Atwood, 2008).

Avslutningsvis presenteras ett alternativt tillvägagångssätt för att stävja möjligheterna för att genomföra XSS-attacker, genom att bara tillåta att skript från en godkänd extern källa tillåts köras. Exempelvis skulle detta kunna vara att vi har en webbplats på ”www.exempel.se” och tillåter att skript bara får köras som hämtats från ”skript.exempel.se”. Detta regleras genom att implementera följande i webbsidans header:

”Content-Security-Policy: default-src: 'self'; script-src: 'self' secure.mydomain.se”

5.2.2. XSS-flooding-attacker

Den typ av DoS-attack som granskats i denna studie är av typen TCP SYN Flood Attack och är direkt kopplad till WebSockets i och med att WebSockets protokollet till skillnad mot HTTP kommunicerar över TCP protokollet.

Resultatet av litteraturstudien påvisar att sårbarheten som attackers här är kopplat till hur själva uppgraderingen mellan HTTP protokollet till TCP protokollet vid handskakningen mellan klient och server fungerar. Genom att vid uppkopplingsförfrågan från klienten, sända med ett falskt IP-nummer (origin) så stannar uppkopplingen upp då servern efter att ha tagit emot förfrågan om uppkoppling, sänder tillbaka en bekräftelse som klienten i sin tur skall besvara (Jain and Singh, 2012).

I och med att en felaktig källa sänds in, återkommer aldrig ett svar från klienten. För varje gång en sådan förfrågan sker allokeras minne vid servern för att förbereda för uppkopplingen. Detta sker fram till att minnet tar slut och därmed börjar servern att avvisa alla förfrågningar om uppkoppling, även från icke-manipulerade klienter (vanliga användare) (Jain and Singh, 2012).

Detta är en typ av flooding attack, där servern överbelastas med så många förfrågningar att den går offline. En liknande typ av attack är när en klient ansluter från en äkta källa (dvs. ett svar från servern kan nå fram till klienten), men i sådan takt att det överbelastar servern.

Utifrån de resultat som litteraturstudien givit så förefaller det att för att detta skall gå att genomföras måste en initial XSS-attack genomföras, därav termen XSS-flooding-attack. Anledningen är främst att klienten måste ansluta från en godkänd källa (vilket med injicerad JavaScript kod blir fallet då källan är den egna webbplatsen), samt att scriptet körs av en större mängd användare samtidigt. Detta också på grund av de restriktioner i webbläsare som finns och sats till att endast tillåta ett visst antal uppkopplingar mot WebSockets servrar, för Google Chrome är detta exempelvis 255 stycken (Chromium, 2017).

Stegen som behöver tas blir således:

- Skadligt skript injiceras på en eller flera webbsidor på en webbplats som har en WebSockets server
- Skriptet körs igång automatiskt och loopar mot servern, där målet är att skapa en stadig ström av nya uppkopplingar.
- Skriptet körs av många simultana användare samtidigt vilket ökar trycket på servern fram till att nya uppkopplingar inte kan tas emot

```
43  var sockets = []
44  while(true){
45      sockets.push(new WebSocket("ws://server:port");
46  }
```

Figur 22. Exempel på kod för att begära nya uppkopplingar mot en server i hög hastighet fram till att servern slutar svara.

Baserat på de resultat som förvärvats från litteraturstudien har ett exempel på hur ett script som åstadkommer detta sats samman och presenteras i figur 22.

Av litteraturstudien har även skydd mot detta för att stävja attackerna identifierats. Grundskyddet består i som lyft ovan att alla moderna webbläsare idag har en gräns för antalet simultana uppkopplingar som kan göras. Utöver detta så faller även denna typ av attack inom ramen för att ett proaktivt skydd mot skript injektioner genom inmatningsvalidering går en lång väg mot att undvika XSS-flooding-attacker då förutsättningen är att klienter kör det skadliga skriptet.

Ett annat sätt som lyfts är att begränsa timeouten vid servern för hur länge en klient får vara inaktiv innan anslutningen bryts. Detta kan vanligtvis regleras genom ramverk som används vi bruket av WebSockets, exempelvis Socket.IO eller WebSocket-Node (McKelvy, 2014).

5.3. Sammanfattning

Följande delkapitel sammanfattar det som lyfts under analysen genom att sammanställa de viktigaste punkterna, som även kommer ligga som grund inför studiens slutsats.

- Om WebSockets används tillsammans med ett ramverk är sannolikheten stor att parametrar finns för att blockera tillgång till servern från externa källor. Detta bör i princip alltid göras om inte situationen kräver annat.

- Kontroll av origin headern bör alltid kontrolleras på serversidan, kommer anropet från en trovärdig källa?
- Implementation av en så kallad vitlista där källor som skall ges tillgång definieras, dessa kommer att jämföras mot den origin header klienten sänder vid uppkopplingsförfrågan.
- Saknas en origin och/eller referal header bör en anslutning avböjas.
- Oavsett om kontroller av origin finns implementerade kan man aldrig med säkerhet avgöra en klients källa. Autentisering av en klient kan dock med fördel ske genom session cookies för att endast tillåta att inloggade användare kan ansluta till WebSockets servern.
- Inmatningsvalidering är centralt för att på ett tidigt stadium avvärja script injections
- Potentiellt skadliga tecken som sänds in genom exempelvis kommentarsformulär bör hanteras på såväl klient- som serversidan. Där dessa tecken genom HTML entity encoding oskadliggörs.
- Ett utmärkt steg i att skydda användare mot session hijacking är att använda sig av HttpOnly parametern i cookies.
- En Content-Security-Policy kan implementeras på webbplatsen, för att endast tillåta att skript från en tillförlitlig källa kan läsas in och köras.
- WebSockets kan i och med att protokollet baseras på TCP utsättas på TCP SYN Flood attacker. En typ av DoS attack som genom avbrutna anslutningsförsök överbelastar WebSockets servern.
- WebSockets kan även utsättas för XSS-flooding-attacker, där skadlig kod injiceras på en webbplats. Genom att sedan köra detta skriptet bestående av en loop som skapar nya uppkopplingar kan servern överbelastas i och med att effekten blir en DDoS-attack.
- Dessa typer av flooding-attacker är svåra att skydda sig mot. Men genom att ha ett bra grundskydd mot XSS-attacker kan XSS-flooding-attacker avvärjas.
- TCP SYN attacker kan avvärjas genom att sätta en lägre tröskel för när en klient anses ha gått inaktiv och därmed kopplas ned.

6. Slutsats

Detta kapitel redovisar studiens slutsatser som framkommit från studiens resultat och analys. Utifrån genomförd studie så kan frågeställningen given i inledningskapitlet besvaras.

6.1. Modell

Denna studie har avsett att besvara följande frågeställning:

Hur kan XSS-attacker genomförda genom tekniken WebSockets stävjas/avvärjas från ett serversideperspektiv?

Utifrån de resultat som ovan presenterats med tillhörande analys har en modell om 3 hot identifierats, med tillhörande 3 föreslagna lösningar.

6.1.1. Identifierade hot

Handshake	**Script-injection**	**XSS-flooding**
Origin header	Injicering av skadlig kod	Överbelastning av server
”Spoofing” av headers	Session hijacking	
Ingen inbyggd autentisering	DOM-kontroll	

Tabell 3. Uppvisar identifierade hot, samt olika typer av attacksätt.

6.1.2. Avvärjning

Handshake	**Script-injection**	**XSS-flooding**
Whitelist/blacklist	Hantering av skadliga tecken	Tidig timeout av användare
Kontroll av origin/referer på serversidan	Kontroll på serversidan	Extern autentisering
Extern autentisering	HttpOnly Cookies	Inlärning av beteende
Avvisning av uppkopplingsförsök		Skydd mot script-injection

Tabell 4. Uppvisar identifierade hot, samt olika sätt att förhindra att dessa utsätts för en attack.

6.2. Proaktivt skydd

Studien har påvisat att genom att arbeta proaktivt med säkerheten kan man säkra upp sina WebSockets-baserade system utan att behöva förlita sig på tredjepartstjänster. I och med startupbolag och mindre organisationers ekonomiska förmåga tydliggör studien att skydd mot XSS-attacker vilket kan medföra stora konsekvenser, på ett fördelaktigt sätt går att skydda sig mot genom att från start konstruera sina system på ett säkert sätt. Studiens avgränsning mot XSS-attacker genomförda mot WebSockets-baserade tjänster presenterar tre huvudsakliga attackvägar med tillhörande attackmetoder. Studien presenterar också för vart och ett av dessa förslag på hur det kan motverkas, enligt den modell som presenteras såväl under kapitel 5, som i den modell som skapats under kapitel 6.1. En sammanställning av analysen lyfter också i större detalj fram stegen som bör tas, detta under kapitel 5.3.

6.3. Bidrag

Studien har avsett att utifrån ett tvärvetenskapligt perspektiv binda samma tidigare forskning inom såväl informatik som datavetenskap. Genom den form av meta-analys som genomförts av tidigare litteratur har en bakgrund getts kring hur dataintrång negativt kan påverka en organisation och lyft vikten av att arbeta proaktivt med att säkra sina data. Studien har i relation till detta granskat den nya tekniken WebSockets och de säkerhetshot som finns avgränsat till XSS-attacker, för att utifrån detta arbeta fram en modell för de huvudsakliga hot som finns, samt hur dessa bör bemötas. Syftet har här vart att med den kombinerade kunskapen mellan informatik och datavetenskap, ge en modell för startupbolag att arbeta utefter vid implementationen av WebSockets i sina tjänster, för att arbeta proaktivt istället för reaktivt. Samt ge ett skydd utan att behöva förlita sig på tredjepartstjänster.

Genom den modell som presenterats kan organisationer och framtida forskning grunda sina system/vidare forskning på den grund som här lagts, för att komplettera upp med de andra aspekter som antagningsvis bör finnas vid försök till hackning av systemen.

7. Diskussion

Detta kapitel behandlar personlig diskussion kring genomförd studie.

7.1. Metodval

Valet av metod för genomförd studie var inte given från start, utan har i samarbete med handledare och examinator diskuterats fram för att bäst passa studien. I och med studiens inriktning mot mindre och medelstora företag samt deras möjligheter att adoptera studiens resultat i sin egen verksamhet. Krävdes en metod som inkorporerade tidigare forskning i ämnet för att ge en solid bakgrund och empiri för föreslagen lösning. Valet föll på att göra en kvalitativ undersökning med litteraturstudie som metod.

Metodvalet synkroniserade även väl sett till att studien genomförts inom området informatik, där andra metoder såsom exempelvis experiment valts bort då detta skulle till onödig grad komplicerat studien, samt möjligen gett resultat mera i linje med en datavetenskaplig studie framför en inom informatik.

Det negativa med mitt metodval är att som inledningen framhåller finns det idag relativt begränsat med forskning kring just tekniken WebSockets. Detta innebär att akademiska källor matchande den inriktning denna studie har är en handfull, även om kompletterande källor för tekniker, språk och metoder funnits att tillgå. Det blir också mera fokus på att använda sig av tidigare forskningsresultat, analysera och tillämpa i min slutsats, än att själv ha möjligheten till att testa allting fullt ut i exempelvis ett experiment. Men detta får ses i relation till ovanstående förklaring där koppling måste finnas tydligt mot ämnet informatik, samt studiens omfång som är mera specificerad kring ett väldigt specifikt problem, än utmålande och djupt granskande i flertalet olika infallsvinklar. Det bör också ses utifrån att det trots allt är en metod som efterfrågats mera inom IT, denna typ av metaanalys. Då det finns mycket spridd forskning, men få granskande studier som binder dessa samman.

Utifrån detta resonemang så skulle man även kunna argumentera för att denna studie ytterligare kunde avgränsats, exempelvis genom att lyfta ut XSS-attackerna och endast fokusera på validering av användare som kopplar upp sig mot en WebSockets server. Detta skulle även gett mera utrymme för exempelvis implementation och testning. Men detta får då ses i sken av ovanstående resonemang, att studien genom detta fyller en viktigare roll än ännu en studie som fyller en väldigt specifik lucka utan att knyta samman tidigare forskning.

Beträffande arbetet med valda metoder så bör det återigen lyftas i relation till anskaffandet av akademisk litteratur, sökning etc. att här finns det idag ett begränsat utbud i och med att området är så pass nytt och standarden antagen bara för ett par år sedan. Detta kan möjligtvis påverka resultatet och det bör också framhållas att då standarden står i utveckling och förändring kan framtida resultat vara avvikande denna studies. Det är därför i vidare forskning av stor vikt att uppmärksamma de datum källor hämtats i denna studie, samt denna studie i sig.

7.2. Resultat och slutsats

Jag känner mig nöjd med studiens resultat då viktiga aspekter av säkerhetsarbetet kring implementationen av WebSockets har tydliggjorts och behandlats. Samt att konkreta exempel på aktuella sårbarheter har lyfts och sedan under analysfasen fått förslag på hur de skulle kunna hanteras baserat på litteraturstudiens resultat. En brist

jag ser dock är att studien av tidsskäl fått en relativt snäv avgränsning kring de olika typer av XSS-attacker som kan genomföras, tillsammans med potentiellt andra typer av attacker som kan genomföras mot WebSockets servrar. Valet av just XSS dock gjordes utifrån dess popularitet och med relativt låg risknivå stora konsekvenser ett hack baserat på XSS kan ge.

En annan brist är att implementation / experiment skulle kunna vart en metod relevant för studien, även om grundsyftet har vart själva metastudien av tidigare litteratur. Jag tror dock inte att detta påverkat resultatet negativt och att resultaten baserat på litteraturstudien är av god kvalité. Men exempelvis implementation hade kanske kunnat ge mera grund för vidare forskning baserat på denna studie.

Värt att ha i åtanke kring resultatet är också att i och med att detta är ett relativt nytt ämne så har den akademiska basen av forskning vart sparsam. Ofta med referenser mellan/till varandra. Dock har dessa artiklar alla vart av god kvalité, hög andel citeringar och med källor direkt kopplade till exempelvis WebSockets protokollutvecklare.

7.3. Etiska och samhälleliga aspekter

För de etiska aspekterna av genomförd studie hänvisas till metoden och delkapitel 3.5.

Sett till samhälleliga aspekter så har studiens avsikt vart att ge underlag för startupbolag hur deras säkerhetsarbete kan se ut om de väljer att använda sig av WebSockets. I och med att vi idag som privatpersoner delar med oss av mera information än någonsin tidigare med olika typer av organisationer. Är även behovet av god informationssäkerhet av stor vikt. Den samhälleliga nyttan här som studien hoppas bidraga med, är att se till att startupbolag förstår att det är viktigt att skydda sina användares uppgifter. Såväl ur juridisk som ekonomisk och etisk mening, där problem så som sessionskapning snabbt kan leda vidare till stora problem för den utsatte individen.

Vidare har studien lagt grunden för startupbolag att kunna ha ett proaktivt arbetssätt kring säkerheten när det kommer till WebSockets på ett ekonomiskt vis. Externa säkerhetstjänster har av resultaten bevisats initialt inte är nödvändigt. Så länge man har koll på de säkerhetshot som finns och hur dessa kan motverkas av bra implementation.

7.4. Vetenskapliga aspekter och forskningsbidrag

Utifrån vetenskapliga aspekter så anser jag att denna studie kan medföra värde genom att knyta samman tidigare forskning inom såväl den mera tekniskt inriktade scenen, som organisationsteoretiska och informationssäkerhet. Studien har haft som syfte att genomföras som en metaanalys av befintlig forskning, för att med tidigare forskning täcka upp ett kunskapsluckor, istället för att bli ytterligare ett väldigt specifikt men nytt inslag. Studien står även på en god grund då litteraturstudien grundats i väl ansedda akademiska artiklar och sedan vid behov under analysfasen kompletterats upp med källor från väl ansedda organisationer med kunskap matchande studiens frågeställning.

Studiens främsta bidrag till forskningen utgår ifrån att ha fört samman såväl tekniska som samhällsvetenskapliga aspekter inom samma studie, för att skapa en mera samlad bild kring ett specifikt problem. Studien har även visat att det med proaktivt säkerhetsarbete går att förhindra mycket av de problem som annars lätt uppstår och

sedan i efterkonstruktioner måste säkras upp. Att det går att med rätt säkerhetstänk från start se till att bygga säkra applikationer med WebSockets utan att det är ekonomiskt krävande.

Studien bidrar även med att öka förståelsen kring de säkerhetsproblem som adopteringen av nya tekniker kan föra med sig och att analys av de tidigare protokoll och tekniker denna bygger på krävs för att förstå de säkerhetsproblem som kan uppstå.

Avslutningsvis belyser studien WebSockets och de fördelar protokollet har och hoppas att agera uppmanande inför framtida forskning för ytterligare utveckling av såväl realtidskommunikationen. Som säkrare protokoll och kommunikationssätt.

7.5. Framtida forskning

Som framhållet tidigare under denna diskussion har studien vart snävt avgränsad mot en väldigt specifik attacktyp, samt i koppling till mindre organisationer och företag utefter svenska mått mätt. Här finns det stor potential till att i framtida forskning bredda studien för att inkorporera ytterligare attack-tekniker såväl genomförda genom XSS som helt andra typer av attacker och hur dessa kan förebyggas från ett serverperspektiv.

För framtida forskning skulle även experiment kring det förslag på lösningar som lyfts fram i denna studie vara av intresse att utförligare testa och mäta. Samt ställas i kontrast till externa system som idag finns på marknaden specialiserade på exempelvis att stävja DOS och DDoS-attacker. Som denna studie framhåller så läggs grunden för att sätta upp ett bra skydd för små- och medelstora företag där man arbetar proaktivt med säkerheten istället för reaktivt. Där syftet är att säkra sitt system till låg kostnad, i kontrast till att reaktivt köpa in en extern lösning där det redan kan vara för sent. En väg att gå framåt här är att bygga vidare på den rekommendationsspecifikation som lagts fram för att ytterligare säkra upp företagens IT-system när de väljer att ta steget från tidigare tekniker, över till WebSockets.

Det skulle också vara intresse att på en mera teknisk nivå ingående analysera problemet med Origin headers och preventiva metoder där för att skydda sig mot session hijacking och liknande.

Referenser

Atwood, J. (2008) *Protecting Your Cookies: HttpOnly*. Available at: https://blog.codinghorror.com/protecting-your-cookies-httponly/ (Accessed: 15 May 2017).

Berndtsson, M., Hansson, J., Olsson, B. and Lundell, B. (2008) *Thesis Projects: A Guide for Students in Computer Science and Information Systems*, *Springer*. doi: 10.1007/978-1-84800-009-4.

Chromium (2017) *Chromium specification*. Available at: https://cs.chromium.org/chromium/src/net/socket/client_socket_pool_manager.cc?q=WEBSOCKET_SOCKET_POOL&sq=package:chromium&type=cs&l=29 (Accessed: 15 April 2017).

Cloudflare (2017) *Cloudflare*. Available at: https://www.cloudflare.com/ (Accessed: 20 April 2017).

Erkkilä, J.-P. (2012) 'WebSocket Security Analysis'.

Erlingsson, U., Livshits, V. and Xie, Y. (2007) 'End-to-End Web Application Security.', *HotOS*, pp. 2–7. Available at: http://www.usenix.org/event/hotos07/tech/full_papers/erlingsson/erlingsson_html/.

Fette, I., Google Inc, Melnikov, A. and Isode Ltd (2011) *The WebSocket Protocol*. Available at: https://tools.ietf.org/pdf/rfc6455.pdf (Accessed: 20 February 2017).

Fisher, J. A. (2013) 'Secure My Data or Pay the Price : Consumer Remedy for the Negligent Enablement of Data Breach', 4(1).

Furukawa, Y. (2011) 'Web-Based Control Application Using Websocket', *Proceedings of ICALEPCS2011*, pp. 673–675. Available at: http://epaper.kek.jp/icalepcs2011/papers/wemau010.pdf.

Heroku (2017) *WebSockets Security*. Available at: https://devcenter.heroku.com/articles/websocket-security.

Hoffman, B. B. and Labs, S. P. I. (2006) 'Ajax Security Dangers', *SPI Dynamics*.

Jain, A. and Singh, A. K. (2012) 'DISTRIBUTED DENIAL OF SERVICE (DDOS) ATTACKS - CLASSIFICATION AND IMPLICATIONS', 3(1), pp. 136–140.

Karlström, J. (2015) 'The WebSocket Protocol and Security : Best Practices and Worst Weaknesses', *University of Oulu*.

Kulshrestha, A. (2013) 'An Empirical study of HTML5 Websockets and their Cross Browser behavior for Mixed Content and Untrusted Certificates', *International Journal of Computer Applications*, 82(6), pp. 13–18. doi: 10.5120/14119-2221.

Kuosmanen, H. (2016) 'Security Testing of WebSockets', (May).

Loreto, S., Saint-Andre, P., Salsano, S. and Wilkins, G. (2011) 'RFC 6202 - Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP', *Internet Engineering Task Force*, pp. 1–19. doi:

10.1017/CBO9781107415324.004.

McKelvy, B. (2014) *WebSocketServer*. Available at: https://github.com/theturtle32/WebSocket-Node/blob/master/docs/WebSocketServer.md (Accessed: 20 April 2017).

Microsoft Developer Network (2017) *Mitigating Cross-site Scripting With HTTP-only Cookies*. Available at: https://msdn.microsoft.com/en-us/library/ms533046.aspx (Accessed: 12 May 2017).

MiniWatts Marketing Group (2006) *World Internet Usage Statistics and Population Stats*. Available at: http://www.internetworldstats.com/stats.htm (Accessed: 23 February 2017).

Mozilla Developer Network (2017a) *Declaring And Using Callbacks*. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes/Using_js-ctypes/Declaring_and_Using_Callbacks (Accessed: 10 May 2017).

Mozilla Developer Network (2017b) *WebSockets*. Available at: https://developer.mozilla.org/sv-SE/docs/Web/API/WebSockets_API (Accessed: 1 May 2017).

Node Packaged Modules (2017) *WebSocket*. Available at: https://www.npmjs.com/package/websocket (Accessed: 15 April 2017).

Oates, B. J. (2005) *Researching Information Systems and Computing*. 1st edn.

OWASP (2017a) *Cross-site Scripting*. Available at: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) (Accessed: 11 May 2017).

OWASP (2017b) *Testing WebSockets*. Available at: https://www.owasp.org/index.php/Testing_WebSockets_(OTG-CLIENT-010) (Accessed: 5 May 2017).

Pimentel, V. and Nickerson, B. G. (2012) 'Communicating and displaying real-time data with WebSocket', *IEEE Internet Computing*, 16(4), pp. 45–53. doi: 10.1109/MIC.2012.64.

Puranik, D. G., Feiock, D. C. and Hill, J. H. (2013) 'Real-time monitoring using AJAX and WebSockets', *Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems*, pp. 110–118. doi: 10.1109/ECBS.2013.10.

Ritchie, P. (2007) 'The security risks of AJAX/web 2.0 applications', *Network Security*, 2007(3), pp. 4–8. doi: 10.1016/S1353-4858(07)70025-9.

Rodriguez, J. D. P. and Posegga, J. (2015) 'Why Web Servers Should Fear Their Clients', in *International Conference on Security and Privacy in Communication Systems*, pp. 401–417. doi: 10.1007/978-3-319-28865-9_22.

Rouse, M. (2010) *SQL Injections*, *TechTarget*. Available at: http://searchsoftwarequality.techtarget.com/definition/SQL-injection (Accessed: 10 March 2017).

Schneider, C. (2013) *Cross-Site WebSocket Hijacking*. Available at:

https://www.christian-schneider.net/CrossSiteWebSocketHijacking.html (Accessed: 20 April 2017).

Socket.io (2017) *Socket.IO*. Available at: https://socket.io/ (Accessed: 15 May 2017).

The Open Web Application Security Project (2017) *CSRF*. Available at: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#Checking_the_Origin_Header (Accessed: 20 April 2017).

Webster, J. and Watson, R. T. (2002) 'Analyzing The Past To Prepare For The Future: Writing A Litterature Review', *MIS quarterly*, 26(2).

Whitman, M. E., & Mattord, H. J. (2011) *Principles Of Information Security*. Cengage Learning.

Xia, F., Yang, L. T., Wang, L. and Vinel, A. (2012) 'Internet of Things', *International Journal of Communication Systems*, 25(9), pp. 1101–1102. doi: 10.1002/dac.2417.

Zhaoyun, S., Xiaobo, Z. and Li, Z. (2010) 'The Web Asynchronous Communication Mechanism Research Based on Ajax', pp. 370–372.