OWASP top ten - What is the state of practice among start-ups?

Halldis Margrete Søhoel

Master of Science in Communication Technology
Submission date: January 2018
Supervisor: Colin Alexander Boyd, IIK
Co-supervisor: Martin Gilje Jaatun, SINTEF

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	OWASP top ten - What is the state of practice among start-ups?
Student:	Halldis Søhoel

Problem description:

The Open Web Application Security Project (OWASP) maintains a list of the top ten most critical security vulnerabilities on the web, along with tools and resources on how to test them and how to avoid them. For anyone interested in developing a secure application this would be a great place to start. However, studies have shown that developers often dont pay enough attention to security which in turn results in basic security issues being overlooked.

This master thesis will study how well start-ups are managing to create secure applications. Because the OWASP top ten constitute a consensus of the most common and most severe vulnerabilities on the web, and because the document will be well known to any threat agent, they represent a good measure on whether the most basic security needs are covered.

The master thesis will document the state of software security at a number of start-up companies, that offer web-based services as their primary business. Web applications from the five companies will be tested according to a test plan developed to cover the OWASP top ten. In addition, the companies will be interviewed in order to determine their level of knowledge on OWASP, and what security measures they have implemented.

Responsible professor:	Colin Boyd, IIK
Supervisor:	Martin Gilje Jaatun, SINTEF

Summary

New apps and web services are increasingly serving our everyday needs, and they are appearing at high speed. How secure are all these services?

This thesis has tested the security of five web services developed by startups. The startups were interviewed, and penetration testing based on OWASP top ten were conducted.

The results show that none of the companies were using a systematic approach when working with security. Two out of the five companies were familiar with OWASP and had done some arbitrary measures to prevent these. The three other companies seemed very little concerned about security and had the point of view that security was to be implemented once the service was functioning and steady.

Testing show that the three companies not concerned about security, all had serious security flaws. For the two companies actively working with security, although some security flaws were found, these were fewer and less serious. In general, the companies were more successful in avoiding implementation flaws, such as SQL injection and XSS, while architectural security holes were more common. SQL injection and XSS were also more widely known among the startups than the other OWASP top ten vulnerabilities.

Third party code played a huge part in securing all the applications. Still some of the companies failed by trusting this third-party code too much and by not considering security as an overall solution. Any company handling sensitive information about customers have the responsibility to make sure that the information is handled securely throughout the application and ensure full coverage.

Sammendrag

Nye applikasjoner og webtjenester betjener i økende grad våre daglige behov, og de oppstår i høy hastighet. Hvor sikre er alle disse tjenestene?

Denne oppgaven har testet sikkerheten til fem webtjenester utviklet av oppstartsbedrifter. Oppstartartsbedriftene ble intervjuet, og penetrasjonstesting basert på OWASP topp ti ble utført. Resultatene viser at ingen av selskapene brukte en systematisk tilnærming når de jobbet med sikkerhet. To av de fem selskapene var kjent med OWASP og hadde gjort noen vilkårlige tiltak for å forhindre disse. De tre andre selskapene virket svært lite bekymret for sikkerhet og hadde en holdning om at sikkerhet skulle implementeres når tjenesten fungerte og var stabil.

Testing viser at de tre selskapene som ikke var bekymret for sikkerhet, hadde alvorlige sikkerhetsfeil. For de to selskapene som aktivt jobbet med sikkerhet, ble det funnet noen sikkerhetsfeil, men disse var langt færre og mindre alvorlige. Generelt var selskapene mer vellykkede i å unngå implementeringsfeil, som SQL-injeksjon og XSS, mens arkitektur-sikkerhetshull var mer vanlige. SQL-injeksjon og XSS var også mer kjent blant oppstarts-bedriftene enn de andre OWASP topp ti sårbarhetene.

Tredjepartskode spilte en stor rolle i alle applikasjonene. Noen av selskapene stolte likevel for mye på tredjepartskoden, og mislyktes med å vurdere sikkerhet som en helhetlig løsning. Ethvert selskap som håndterer sensitiv informasjon om kunder har ansvar for å sikre at informasjonen håndteres på en sikker måte gjennom hele løsningen.

Preface

This Master's Thesis is the final submission to the Master of Science Degree in Communication Technology at Norwegian University of Science and Technology (NTNU) with specialization in Information Security. This concludes five years of study at NTNU and one semester abroad. Work done in this thesis was performed in the autumn of 2017 and continues the pre-project submitted in the autumn of 2016.

I would like to thank my responsible professor and supervisor, Colin Boyd and Martin Gilje Jaatun, for exceptional guidance and feedback throughout the whole process. Thank you for always being available and for all your time and efforts. I am very grateful.

At last I would like to thank the startups for their time and willingness to partake in this thesis, and wish them good luck for the future.

Halldis Søhoel
January 2017

Table of Contents

List of Tables

List of Figures

Abbreviations

API	=	Application Programming Interface
B2B	=	Business-to-Business
BSIMM	=	Building Security In Maturity Model
CIA	=	Confidentiality, Integrity and Availability
CLASP	=	Comprehensive Lightweight Application Security Process
CSRF	=	Cross-Site Request Forgery
CVE	=	Common Vulnerabilities and Exposures
CVSS	=	Common Vulnerability Scoring System
DOM	=	Document Object Model
DOS	=	Denial of Service
HPKP	=	HTTP Public Key Pinning Extension
HSTS	=	HTTP Strict Transport Security
ICT	=	Information and Communication Technology
openSAMM	=	Open Software Assurance Maturity Model
OWASP	=	Open Web Application Security Project
REST	=	Representational State Transfer
SDL	=	Security Development Lifecycle
SSDL	=	Software Security Development Lifecycle
SSG	=	Software Security Group
SSL	=	Secure Sockets Layer
SQL	=	Structured Query Language
TCP	=	Transmission Control Protocol
UDP	=	User Datagram Protocol
URI	=	Uniform Resource Identifier
URL	=	Uniform Resource Locator
WP	=	Wordpress
XSS	=	Cross-Site Scripting

Chapter 1

Introduction

“The Internet of half-baked Things is upon us” the security expert Marcus Ranum states on the podcast the Silver Bullet (Ranum, 2016). Today we are putting chips in everything. This result in the existence of major unused and poorly secured computing capacity connected to the internet. At the same time Apple and Google have found great success in open development environments opening the door to the software developing world to anyone willing to learn coding.

And people are willing; every month hundreds and thousands of new apps and web services appear online with varying quality. We are using these services to fill our needs in every aspect of everyday lives; sleeping, eating, travelling and shopping. Consumers rarely ask questions about the security of these systems.

Steve Bellovin states: “any program, no matter how innocuous it seems, can harbour security holes” (McGraw, 2006). A statement that was nicely illustrated on October 2016 when a DOS attack on the internet infrastructure provider, Dyn (York, 2016), resulted in downtime for Twitter, Netflix and several other high-profile web pages. It turned out that the attack was performed by a botnet of poorly secured IoT devices like, web cameras, video recorders and surveillance cameras.

As web usage increase and become part of our day to day life, compromising these systems become increasingly rewarding. Attackers no longer need physical access to the victims, they can attack more than one at the time and the risk of being caught and brought to justice is minimal. Today both digital forensic and jurisdiction issues face huge challenges when it comes to the prosecution of offenders on the web. The appearance of crime-as-a-service ensures that the means fall into the hands of those with an intent to use it (Bilge and Dumitras, 2012).

In any other science discipline, the first thing you learn are the consequences of making mistakes. They teach you that poorly designed buildings or bridges may cause great disasters. In computer science the focus is quite different. In fact, at many universities and university colleges, although security courses are offered, they are not a mandatory part of the ICT programs leaving many graduates without the most basic knowledge about security (Lysne et al., 2015). The attitude among developers is often that as long as it functions the intended way, understanding the system is unimportant. The problem arises when hackers start using the system in an unintended way.

1.1 Research questions

How secure are all these rapidly appearing services? This thesis will contribute to answering that question. In particular the thesis will investigate how well security is maintained in five start-up companies. Startup companies are an interesting group to explore because there are no formal requirements to who can start an IT-company. Anyone willing to learn how to code can start a business.

Another reason to think that security might be neglected in startups is that building software is expensive and startups do not have the resources to invest in anything that will not bring them closer to the next round of financing. McGraw and Ranum believe that the core of the problem is that the software industry is more rewarding to people that are a part of the problem, rather than those that are a part of the solution. If companies spend that extra time and resources on creating good secure software they will get delayed in the release cycle. Especially for startups this can mean life or death for the company.

On the other hand startup companies have two major advantages; their high motivation drive them to make better products and because they are small in size they can quickly adapt to secure development life cycles. In fact, smaller companies implement secure development life cycles at a higher rate than larger companies, likely because they have fewer decision levels and are less prone to bureaucracy (Geer, 2010).

Literature suggests that startups manage to make secure applications even when their knowledge level indicates differently (Bau et al., 2012). There can be two reasons for this; they are highly motivated and personally invested in the product, so they put in the extra effort to do it right. The other reason may be that high profile companies like Microsoft, Google and Facebook are making complete functionality that takes care of critical transactions and secure sample code, that help developers make more secure applications.

Different applications have different security needs based on the sensitivity of the information stored and of the transactions the application is used for. Some applications can accept a higher risk than others. To evaluate this we cannot only look at the technical aspects, but need to also consider the threats and the business values of the system. There needs to be compliance between the security needs and the security achieved.

Open Web Application Security Project (OWASP) top ten constitutes a consensus among security experts of the ten most critical web vulnerabilities. A test plan based on this list will provide technical insight on the security of the applications, which is the main focus of the thesis. The OWASP top ten provide a good basis to check whether the most basic security needs are achieved.

The thesis will answer the following research questions:

1. Do Start-ups cover the basic needs when it comes to software security?
    (a) Is security a concern?
    (b) Do they have knowledge about common tools?
    (c) Are their applications protected against OWASP top ten?

2. Are tools available that can help start-ups uncover vulnerabilities in a fast and easy way?

1.2 Thesis structure

To answer the research questions five web applications made by startup companies have been tested. The work done to answer these questions are divided into three phases; Interviews, Testing and Risk Assessment.

Interviews During the interviews each company is asked about the development process, the threats and values of the systems and OWASP top ten. The aim is to determine the level of knowledge, what activities are done to ensure security and to gain information that will later be used in the risk assessment process. The insight from the interviews are used to answer research question 1a and 1b. The threats and values identified will later be used to determine the risks associated with the vulnerabilities found during testing.

Testing To answer research question 1c penetration testing of the five applications will be performed. The test plan is based on OWASP top ten and closely follows the tests described in OWASP testing guide.

Risk Assessment When each OWASP top ten vulnerability have been tested for the applications they are evaluated in the light of the business context determined during the interviews. The technical flaws together with the threats and values are used to determine the severity of the vulnerabilities found.

The thesis start off with a literature study in Chapter 2 and basic theory in Chapter 3. The method is described in detail in Chapter 4 and the tools that are used during testing are presented in Chapter 5.

In Chapter 6 the companies are presented followed by the findings of the interviews. A summary of the company context can be found in table 6.1. In Chapter 7 the test plan and the results from testing are described. A summary of the results are found in table 7.1 to 7.6. In Chapter 8 the risks are evaluated and discussed. Finally, the research questions are revisited and answered in Chapter 8.3.

Chapter 2

Literature Review

This chapter is divided into two sections; background study and related work. The background study sets the context of this thesis and present the field. Here contributions from well known security experts and high profile companies will be presented and some history is provided. Then, in the second section, similar works to this thesis will be presented along with their results.

2.1 Background

The aim of this section is to set the context of web application security and present important contributors. We start off with the umbrella, computer security, and show how it has matured from being merely a infrastructural effort to include the equally important software security approach.

Section 2.1.1 presents the traditional approach to security and illuminate important voices on why this approach is not sufficient. Section 2.1.2 present the advanced field of software security that together with the traditional approach makes a complete solution. This is a field still under development and important contributions to developing a best practice are presented. Section 2.1.3 presents the narrow field of web application security, that makes use of both said approaches, and is the topic of this thesis.

2.1.1 Computer security

Computer security is the idea of protecting computer systems and the information stored on them. In this context the security requirements or goals are often defined by the CIA paradigm; Confidentiality, Integrity and Availability. It is common to expand on this to also include authenticity, non-repudiation and accountability (Lysne et al., 2015).

Confidentiality To ensure that information is only disclosed to those who have the right to access it.

Integrity The data is valid and accurate and has not been manipulated either by accident or deliberately by unauthorized people.

Availability Making sure the data or services are continuously available to legitimate users.

Authenticity Ensuring the origin of data and correct identification of the sender.

Non-repudiation An action is undeniable. It should be possible to prove that an action or transaction was performed by the specific individual.

Accountability A potential attack is traceable and those responsible can be held accountable.

As we shall see, attention has often been focused on securing the infrastructure and putting defences outside the system. In the recent years attention has been shifted onto also creating secure software. Today everything is connected to the Internet making resources and data accessible to a larger group of attackers and attacks have becommed more anonymous. To an attacker, digital stealing may not seem as serious as stealing something material, but the intrusion can often feel just as invasive to the victim and the loss can be just as significant. This leads to the indisputable importance of web application security.

The traditional approach to computer security

In the past, security has been looked at as an infrastructure matter. The security has been handled at the network layer with firewalls, anti virus programs, intrusion detection and other security functionality. The problem is that vulnerabilities in the software, make it possible to bypass these security technologies. Security has also been handled in a reactive way rather than proactive, with the penetrate-and-patch approach and creating new anti virus signatures as new attacks are discovered. However, this ensures that computer systems remain insecure.

The penetrate-and-patch method

The penetrate-and-patch approach consists of a tiger team of security experts trying to penetrate an already live system. The vulnerabilities that are disclosed during testing are fixed in patches in a post-facto way. Traditionally this testing was based on secret methods and the secret knowledge of the security experts. Today a lot of this knowledge and techniques has been captured and made available online in the form of tools, tutorials and resources, making them accessible for everyone to use. Security testing is no longer reserved for the experts.

This approach is problematic for a number of reasons. McGraw (1998) argues that the approach happens too late; once an attack is discovered it is already too late and it allows hackers to stay ahead of the game. Once a patch is introduced system administrators and consumers may stall on installing the latest updates, leaving the system exposed to any attentive hacker. Bilge and Dumitras (2012) found that exploits for 42 % of newly discovered vulnerabilities appeared on the Internet within 30 days of their disclosure. Also their study shows that anti virus programs continue to detect exploits for vulnerabilities years after patches are available. This indicates that there still exist unpatched hosts.

Another reason to scrap penetrate-and-patch is that patches often introduce new vulnerabilities. Because of the pressure to cook up a fast solution and because the patches often are not tested adequately 10 % of patches introduce new bugs (Bilge and Dumitras, 2012). Also fixing bugs in software later in the development life cycle according to IBM is a lot more expensive. Fixing it during testing is 15 times more expensive than during design and fixing it during maintenance is 100 times more expensive than during design (Edwards, 2017).

Full disclosure policy

The full disclosure policy is the much disputed practise where vulnerabilities are disclosed to the public even if patches are not yet available(Bilge and Dumitras, 2012). The purpose is to add transparency and prevent security-through-obscurity. Also it gives vendors incentives to fix the vulnerabilities faster and add pressure. A publicly disclosed vulnerability is a vulnerability that is assigned a CVE(Common Vulnerabilities and Exposures) identifier and is maintained in the public CVE database. This database maintains a list of all known vulnerabilities and information about release date, exploits and severity. The policy is disputed because it exposes the affected computer systems to malicious attacks before the vendors has had a chance to fix the flaw. In fact, the number of detected attacks on a vulnerability can in some cases increase as much as 100,000 times after disclosure and the number of malware variants can sometimes increase up to 85,000 times. Today there is a debate whether the trade-off between faster patching and more attacks adds more value to society.

Another essential fact adding transparency is that the previously secret methods of security experts today is captured in easy-to-use tools and comprehensive resources and tutorials. This brings security testing to the fingertips of all developers, testers, consumers and also malicious actors. One of those resources is the OWASP awareness documents, testing guides, tools and top ten lists. It is fair to say that if a web page is vulnerable to these exploits, it is fairly easy for anyone to exploit them.

“Software security is not security software”

Another approach used to secure a computer systems is to add a lot of security software and/or features to the system at the end and hope for the best. But as McGraw (2003) states. “You cannot bolt security onto an existing program”. These mechanisms, no matter how useful, will not fix implementation bugs or design flaws (Howard and Lipner, 2003). Security software are any program designed to enhance security like anti-viruses, anti-spyware, firewalls, intrusion detection etc. Vendors will add features to existing programs to protect them but what they might actually end up doing is increasing the attack surface. Microsoft reports that one of the lessons learned during the 2001 security push was to disable features and services by default to limit the attack surface and contain the effects of an attack (Howard and Lipner, 2003).

As a complete solution application security needs to be both built in from the start and protect the software once it goes live. The field of building secure software is quite new and the first book on the subject appeared in 2001(Viega and McGraw, 2001).

2.1.2 Software security

"Software security is the idea of engineering software so that it continues to function correctly under malicious attacks" McGraw (2004). The field emerged with initiatives like the 2002 Windows security push (Howard and Lipner, 2003) and the 2003 DIMACS software security workshop (McGraw, 2003) as the traditional reactive method had proven insufficient. It fortifies the traditional security approach by building software so that it can resist attacks in a proactive way. In order to achieve this we need to implement security activities into the regular software development life cycle to eliminate as many security flaws as possible. Obviously it is easier to protect software that is not faulty, and the cost of fixing bugs increase drastically the further down the development life cycle it is discovered. Security holes in software are common and the problem is growing (McGraw, 2003).

The trinity of trouble - Why is software security a bigger problem now than in the past?

To explain why we are seeing an increase in attacks on software McGraw (2003) use what he calls "The trinity of trouble".

The first element of the trinity is the growing internet connectivity. Today we are connecting more and more of our devices to the internet. As a result, attackers no longer need physical access to a system in order to exploit it. At the same time more and more of our daily business is made available online.

The second element is the phenomenon of extensible systems. Vendors updates their software through so called mobile code to evolve the system incrementally. This enables attackers to slip in malware as unwanted and even unnoticed extensions.

The last element is the growing size and complexity of systems. The flaw rate tends to increase as the square root of the code size, therefore it is simply impossible to avoid bugs in large systems. This element may be exaggerated or limited by using unsafe/safe programming languages.

Best Practice

First out to develop a best practise was Microsoft with their 2002 security push (part of Microsoft Trustworthy Computing Initiative). They took all 8,500 development staffers, from designers, coders and testers, out of their daily tasks to work solely on security. It focused on four areas; education, code review, threat modelling, and making design changes. The security push was the start of a cultural change within the organization; to never again emphasize feature over security.

For education they divided designers, coders and testers into groups with different tracks for each role. For the designers and architects the lesson were to reduce the attack surface, disable features by default and reduce privileges (least privilege). For coders the main lesson was to never trust input and to stay away from insecure function calls and techniques. The testers were taught to use intelligent fuzzing and how to make threat models to base their testing on.

The coders were then put to review all the code, both using tools and manually following the data flow through the application and ensure that it was handled in a secure way. The designers started doing threat modelling by decomposing the application, identifying threat categories and making attack threes. In the end they made changes to the design by reducing the attack surface. Other changes that were made to the development process was; making secure sample code as developers will reuse them as they are written, assign an owner to each source file that needs to sign it off only when it has been properly reviewed and making the current version under development the main target for changes as it is harder and more annoying to make changes to products already in use. They also believe that if a critical mass in the organization has knowledge and care about security they will affect the people around them. In the "Trustworthy Computing Initiative", Microsoft make the case for implementing so called security cycles into the regular development cycle, these are called Secure Development Life cycles (SDL).

Microsofts noteworthy efforts are still highly relevant today. The mindset of considering security throughout the whole development life cycle has evolved and been formalized. One that has always been an advocate for implementing security cycles throughout the entire development process is McGraw. In 2004 he published the security touchpoints in figure 2.1 which makes a good reference to how security cycles can be implemented at different stages during development.

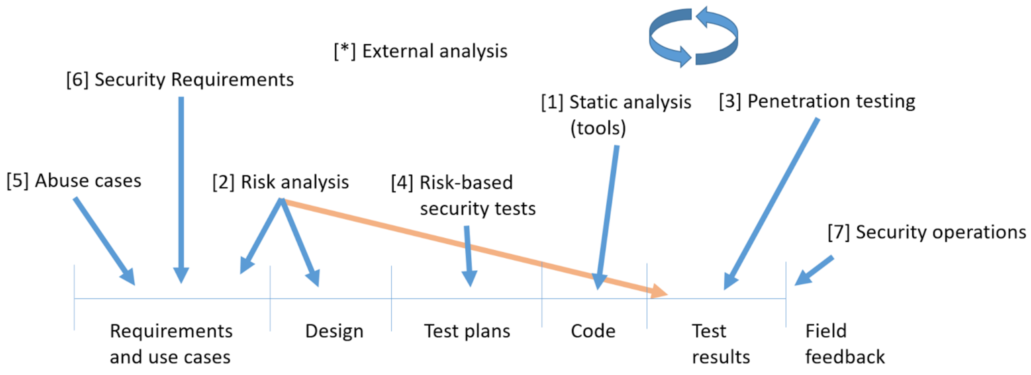


Figure 2.1: Software security touchpoints (McGraw, 2009)

Through observation and measurement we can move from opinion to fact. In order to move from assumptions to science we need measurement. That was the motivation to start the Building Security In Maturity Model (BSIMM) comparing the security initiative in 78 firms to determine the level of maturity within (McGraw, 2016). With BSIMM the companies can compare what they are currently doing to ensure secure software with the activities of other companies and they receive a score or a maturity level indicating if they are better or worse off than their counterparts.

In "Four Software Security Findings" McGraw (2016) present 4 facts learned after working with BSIMM. Firstly, BSIMM is applicable to any company in any industry and of any size. Furthermore, in order to successfully work with the BSIMM activities there is a need for a dedicated software security group(SSG). The average ratio of an SSG to development is 1.52 %. This fact is quite interesting cosidering startups where the number of developers in many cases are one-digit. Further the SSG should contain coders, architects and personnel with teaching skills to be effective. Another thing that has an impact is the experience and age of the security initiative. Firms with the highest maturity levels have older security initiatives. The last finding is that the top firms have large satellites. Its a myth, although common, that developers should just take care of security. As the security initiative becomes more sophisticated it is distributed and institutionalized in the organization. As start-ups are only getting into the game, it is a safe bet that the security initiatives may not be as sophisticated as more mature businesses.

2.1.3 Web application security

"Application security follows naturally from a network-centric approach to security, by embracing standard approaches such as penetrate-and-patch and input filtering and by providing value in a reactive way" (McGraw, 2004). From this follows techniques such as sandboxing code, detecting and defusing malicious code, obfuscating code, monitoring and using technology to enforce the use policy. This approach comes from the fact that the infrastructure people, that more often deal with security, are operators not builders. Therefore they move the security techniques into their line of work and apply security software. "The problem is that vulnerabilities in the software let malicious hackers skirt standard security technologies with impunit" writes McGraw (2004) "although there is some real value in stopping buffer overflow attacks by observing HTTP traffic as it arrives over port 80, a superior approach is to fix the broken code and avoid the buffer overflow completely". Therefore both the traditional approach to security and software security is a natural part of protecting applications (McGraw, 2004).

Open Web Application Security Project

A world without some minimal standards in terms of engineering and technology is a world in chaos

Keary (2014)

"Internet-enabled software applications [...] present the most common security risk today and are the target of choice for malicious hackers" (McGraw, 2003). From 2001 the Open Web Application Security Project (OWASP) has worked as a renowned and acknowledged actor to improve the software security in web applications. "The problem of insecure software is perhaps the most important technical challenge of our time." writes Keary (2014), global board member of OWASP, "we are trying to make the world a place where insecure software is the anomaly, not the norm."

Open Application Security Project (OWASP) is an open community of security experts working to make software security visible in web applications and help organizations and individuals make informed decisions. They work in a collaborative way providing open source tools and resources freely available online. The organization is neutral and independent, aiming to provide unbiased information. Through awareness documents like OWASP top ten, testing guide, code review guide and development guide they aim to provide the necessary information to build and maintain secure applications.

OWASP top ten

One of the awareness documents provided by the OWASP community is the ten most critical web application vulnerabilities. The latest version that received consensus by the start of this thesis is the OWASP top ten 2013 (Owasp.org, 2013). The list was developed through analysing 500,000 vulnerabilities found in thousands of applications. The vulnerabilities are rated based on number of occurrences, exploitability, detectability and impact. The 2013 document is listed in Table 2.1.

A1: Injection
A2: Broken Authentication
A3: Cross-Site Scripting (XSS)
A4: Insecure Direct Object References
A5: Security Misconfiguration
A6: Sensitive Data Exposure
A7: Missing Function Level Access Control
A8: Cross-Site Request Forgery (CSRF)
A9: Using Components with Known Vulnerabilities
A10: Unvalidated Redirects and Forwards

Table 2.1: OWASP top ten 2013

OWASP top ten 2017 was released in the spring 2017, but was later pulled back as the community disagreed on its validness. The final version was finally published on October 20, unfortunately too late to be part of this thesis. In the 2017 release two of the vulnerabilities from 2013, A8 and A10, were retired, and A7 and A4 were merged into one under "Broken Access Control". There are three new vulnerabilities added; XML External Entities (B4), Insecure Deserialization (B8) and Insufficient Logging & Monitoring (B10). OWASP top ten 2017 is listed in table 2.2.

There has been quite a lot happening on the application development front the last 4 years. Microservices, RESTful APIs and Single Page Applications have completely changed the architecture of web applications and come with their own set of security challenges. The fundamental technologies have changes and are now dominated by new web frameworks such as Angular and React (Owasp.org, 2013). Many of the vulnerabilities are rearranged. Luckily all the vulnerabilities from OWASP top ten 2013 are still relevant, only two were taken off the list due to low occurrences. However, they are still among eight vulnerabilities in the 2017 release under "Additional Risks to Consider". B4, B8 and

B10 from 2017 were not tested in this thesis.

B1; Injection
B2: Broken Authentication and Session Management
B3: Sensitive Data Exposure
B4: XML External Entities (XXE)
B5: Broken Access Control
B6: Security Misconfiguration
B7: Cross-Site Scripting (XSS)
B8: Insecure Deserialization
B9: Using Components with Known Vulnerabilities
B10: Insufficient Logging & Monitoring

Table 2.2: OWASP top ten 2017

2.2 Related work

This section presents works that are related to this thesis. In section 2.2.1 the results on a survey on Secure Development Lifecycles show that in 2010 not many companies were using secure development lifecycles but smaller companies were implementing them at a higher rate than the bigger companies. Section 2.2.2 present two studies on the state of practice in Norwegian organization. Section 2.2.3 shows an experiment where the security of applications made by freelance developers were compared to the security of applications made by start-ups. It showed that even though start-ups in general had less knowledge about security than the freelancers they still manages to make securer applications due to high level of motivation.

2.2.1 Are companies using SDL?

A secure development lifecycle (SDL) is a term used for addressing security throughout the software development process and implementing security activities through all phases of development. Another, and perhaps more precise, term used to describe this is Secure Software Development Life cycle (SSDL). Today there exist different methodologies to achieve this; Microsoft SDL, Microsoft SDL Agile, Open Software Assurance Maturity Model (openSAMM) and Comprehensive Lightweight Application Security Process (CLASP). Because there were so many different opinions and approaches, The Building Security In Maturity Model (BSIMM) was started to describe “existing software security initiatives”(BSIMM.com, 2017). In 2010 Errata conducted a survey to determine if companies had heard about the said methodologies and models and whether they had implemented any of them. The results were summarized by Geer (2010).

The survey showed that 81 % of the asked companies had heard about the methodologies, however only 30.4 % were using them. When asked about why they did not use them

23.9 % provided that they were too time consuming, 15.2 % that they required too many resources and 4.3 % that they were too expensive.

The survey also showed that smaller companies implemented secure software development life cycles at a higher rate than larger ones. This is natural as smaller companies are more flexible because they do not have as many decision levels and protocols to follow.

2.2.2 Software security in Norwegian organizations

Two studies have recently been conducted to determine the state of practice in Norwegian organizations with regard to security. Nicolaysen et al. (2010) studied the software security initiatives of six companies using agile software development methodologies and Jaatun et al. (2015) studied 20 public Norwegian organizations developing software.

Consistent with the Errata survey Nicolaysen et al. (2010) found that very few of the companies were utilizing methodologies for creating secure software. The developers had no formal training in developing secure software and very few were concerned about security. Functionality were often prioritized over security.

The study conducted by Jaatun et al. (2015) showed big variations in the number of security activities being performed. Out of the 112 activities the organization was asked about, one organization did only nine, while another did 87. On average the organizations did 44 out of the 112 activities. One area where the organizations did very well were activities connected to compliance and policy. This is probably because of strict regulations from authorities.

The study also revealed that few of the organizations were using a systematic approach to create secure software and the activities being done depended on initiatives from individuals. Also, developing secure software was not a priority, instead the organizations were relying on infrastructure to solve the security needs.

2.2.3 Software security in applications made by Silicon Valley start-ups

Bau et al. (2012) conducted an experiment where 19 silicon valley start-ups and 8 hired freelancers were asked to take a basic software security quiz and submit an applications. The security of the applications were then tested through static analysis and penetration testing.

The results from the quiz showed that the scores for the startups were very dispersed. Some of them scored close to 100 % while others knew less that 50 % of the questions. The startups also did slightly better on the quiz than the freelancers, but this was not statistically significant.

The testing showed that the startups made significantly more secure applications than the freelancers. Furthermore, the negative correlation between the quiz score and the number of vulnerabilities found were also bigger among the startups. This indicates that the startups did a better job at using their security knowledge in practise and implementing what they knew.

The tests also showed that in many cases the startups were successful in developing secure code even in areas where they had failed on the quiz. This shows that even in areas were they lacked knowledge they were able to figure it out and implement it in a correct and secure way.

In other words startups were more successful in using their knowledge in practise and when they did not have the knowledge, they in some cases still managed to implement secure code. Bau et al. (2012) believe that there are two factors contributing to these facts; the startups are more motivated and dedicated to make good solution, and often developers use framework or copy-paste secure code which may "save them". One example of this was one of the freelancers that answered on the quiz that a secure way to store a password was i plain-text, still in the application delivered he had hashed the password with a salt.

2.2.4 Connections to this thesis

The common opinion among experts is clear, but studies show that organizations often do not implement best practice. Previous studies have looked at the security in private established companies and public Norwegian organizations. This thesis will look at how the status is among startups in Norway. Startups are a heterogenic group in relation to knowledge and experience. Anyone can register a business and put a web page online. However, they are small, motivated and have few hierarchical levels which may play out to be an advantage when it comes to making secure applications. Also there are large amount of tutorials, sample code and libraries made by large companies like Google, Facebook and Stripe, that can help even inexperienced developers make secure applications.

Another thing that separates this thesis from previous work is that the security will be tested on real life running applications. The testing is similar to the experiment conducted by Bau et al. (2012), but unlike this experiment the testing is performed on live applications and the test plan is based on OWASP top ten. Also the experiment was performed in 2012 and a lot as happened in the field of web application security since then.

Chapter 3

Basic Theory

3.1 Definition of a vulnerability

Common Vulnerabilities and Exposures (CVE) defines a vulnerability as a weakness in the software or hardware code that may affect the confidentiality, integrity or availability of a system when exploited. Examples are DoS, getting unauthorized access etc. This differs from an exposure that is a configuration or software issue that may be used as a stepping stone into the system. An exposure therefore does not allow direct compromise of a system but can be an important component of an attack, for example a state that allow information gathering activities or hide malicious activity. OWASP define the term “vulnerability” more loosely to include any weakness that can cause harm to the stakeholders and the OWASP top ten include both direct vulnerabilities and indirect exposures. The life cycle of a vulnerability is seen in figure 3.1:

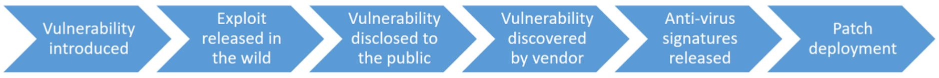

Vulnerability introduced
Exploit released in the wild
Vulnerability disclosed to the public
Vulnerability discovered by vendor
Anti-virus signatures released
Patch deployment

Figure 3.1: Life cycle of a vulnerability (Bilge and Dumitras, 2012)

Implementation bugs vs. Design flaws

Software vulnerabilities can range from local implementation errors to higher level design flaws. The difference is the amount of code one has to consider in order to disclose it. The simplest errors, like unsafe system calls, live on an isolated line of code and can be detected with a lexical analysis. Further vulnerabilities that depend on the behaviour of several functions are mid-range. At the highest level we have logical flaws in the design. These mistakes require high expertise and knowledge to detect, and we must consider how multiple components work together. The design flaws occur in the design phase while implementation bugs occur at the coding phase. McGraw compares building secure software to building a house. The kind of brick we use is important but its even more important that the house is designed to have four walls and a roof. In the past software security has paid much more attention to bricks than to walls. OWASP top ten can occur both at the design phase and the implementation phase.

3.2 The OWASP top ten explained

This section will look a little closer at OWASP top ten and explain each one. The OWASP top ten list explained here is that of 2013 which has been the basis for the vulnerability testing.

3.2.1 A1: Injection

The most widely injection attack is the SQL injection. The injection attack happens when untrusted user data is sent to an interpreter as part of a query or a command without properly separating user data and code. The danger is that the user input entered is executed as code and in that way the attacker is able to perform actions or access data that should be restricted. Other forms of injection attacks are buffer overflows, command injection, LDAP, XML and many more.

3.2.2 A2: Broken Authentication and Session Management

Broken authentication and session management vulnerabilities are those connected to "assume other users identities" (Owasp.org, 2013). These vulnerabilities are often architectural flaws and occur when credentials, tokens, hashes or sessions are not successfully secured. Here the developer must consider brute forcing, log out functionality, token duration, account provisioning etc.

3.2.3 A3: Cross-Site Scripting (XSS)

XSS is an attack where the attacker is able to execute scripts in the victims browser. This attack can be used to steel session variables, redirect the victim or change the look of the web site. This vulnerability occurs when the web server fails to validate user input before sending it to the browser.

3.2.4 A4: Insecure Direct Object References

Insecure Direct Object References occur when objects like files, directories or database entries can be accessed without authorization by manipulating references. The vulnerability is present when access control is not applied to sensitive objects. One example of an attack is if the attacker is able to access some object by changing parameter values in the URIs.

3.2.5 A5: Security Misconfiguration

The applications, framework, server and database need to be up to date and correctly configured. If not attackers can take advantage of known loop holes and attacks.

3.2.6 A6: Sensitive Data Exposure

Sensitive Data Exposure occur when sensitive data such as; personal data, private data, financial data, heath information, academic records, are not properly protected when stored or in transit or access control is insufficient. Information leakage about the application, that an attacker can use to further exploit it should also considered.

3.2.7 A7: Missing Function Level Access Control

This vulnerability is present when access control to functionality is not done on the server for each request. Even when functionality is not visible for certain users on the user interface, an attacker can send the HTTP request for that functionality directly to the server. If the server does not verify the user identity at this stage, the attacker can perform unauthorized functions.

3.2.8 A8: Cross-Site Request Forgery (CSRF)

This vulnerability occur when it is possible to trick an authenticated user to sending a forged request to the web site and the request is accepted by the application. The browser will automatically add the session variables of the user to the request, therefore additional protection is necessary to prevent this vulnerability.

3.2.9 A9: Using Components with Known Vulnerabilities

If an application utilizes libraries, frameworks or other components that contains known vulnerabilities, an attacker can use this information to perform target attacks. Known vulnerabilities are stored in a public record and are easy to find.

3.2.10 A10: Unvalidated Redirects and Forwards

When untrusted user data is used as the destination address in a forward or redirect without being further evaluated. If this vulnerability is present attackers can trick users to click on links that appears to be for the legitimate site, but instead they are redirected to a phishing or malware site.

3.3 Startup

In the pre-project (Søhoel, 2016) the term start-up is defined as:

Start-up is a loosely defined term but in general it refers to a newly established company in its initial stage of operations. It often has a small number of employees and it has a scalable business model, meaning that they have the intention of growing (Robehmed, 2013). A startup is an entrepreneurial venture with a lot of risk related to it and might initially be funded by its founders (Investorpedia.com, 2007). In this project I will define the term as a newly launched business (0-4 years) that may or may not generate income. Also it is still in the developing phase of its product and organizational structure, with mainly its founders as its employees.

The same definition will be used in this thesis.

3.4 HTTP requests and responses in attacks

Hypertext Transport Protocol (HTTP) is the primary communication protocol used on the web at the application layer. The protocols works with the client sending a http request to the server and the server responding to the request with a http response. The protocol is stateless, which means that the server does not store any information between two requests. Therefore each request/response pair can be looked at as stand alone operations. Manipulating HTTP request lays the foundation for many attacks. By using tools like Burp Suite it is possible to interact with any web page without going through the browser. By doing this it is possible to obtain additional information about how the application is configured, session is handled and gain direct access to functions and data not accessible through the browser. The following subsection will explain the basic structure of http request/response pairs. The focus will be on the components relevant to the penetration testing performed in this thesis.

3.4.1 HTTP method

A HTTP method is added in each HTTP request to indicate what type of action the client wishes to perform on the specified resource. The possible methods are:

GET retrieves the data on the specified Uniform Resource Identifier (URI).

HEAD identical to the GET method but without the message body.

POST can be used to submit a resource on the specified request-URI and might change the state of the server.

PUT can be used to update an existing resource or create a new one if it does not already exist.

DELETE delete the resource on the specified URI

CONNECT open a tunnel with the requested destination.

OPTIONS returns the allowed HTTP methods for the provided resource.

TRACE is used for diagnostic purposes and mirrors back to the client what was received by the server.

PATCH change the requested resource according to a “patch document”.

According to OWASP testing guide some of these methods pose security risks to the web site when enabled. The methods that should not be used are:

1. PUT
2. DELETE
3. CONNECT
4. TRACE

However, when the Application Programming Interface (API) is RESTful methods like PUT and DELETE are commonly used.

3.4.2 HTTP security headers

In the HTTP response there exist security headers that when set can help secure the application. They restrict how the browser interact with the server and prevent common vulnerabilities. The headers recommended by OWASP Secure header project are:

1. HTTP Strict Transport Security (HSTS)
2. Public Key Pinning Extension for HTTP (HPKP)
3. X-Frame-Options
4. X-XSS-Protection
5. X-Content-Type-Options
6. Content-Security-Policy
7. X-Permitted-Cross-Domain-Policies
8. Referrer-Policy
9. Expect-CT

Among these HSTS is especially important because it ensures that data is always sent over an encrypted channel. The Content-Security-Policy protects against a range of common vulnerabilities and is very useful. Although there is a trade off between usability and security, the developers should consider each one and set those that are relevant and necessary for their application.

Chapter 4

Method

To answer the research questions a handful of techniques have been selected. This chapter will discuss what those techniques are and why they have been chosen. The thesis started with a literature study. Then all the companies selected went through a semi-structured interview to determine the business context. Each application was tested based on a standardized test plan, using both automated penetration testing and manual penetration testing. At the end, each vulnerability was rated based on the OWASP risk rating methodology. Based on the experience acquired during testing some tools and resources will be recommended based on criteria given in section 4.5.

4.1 Literature study

The literature study started of with some material recommended by the supervisors. A search on Google Scholar for material relevant to the topic was also done. Then the references of the sources found was followed and a search for additional material by relevant authors. The background study is a historical review showing how traditional computer security has evolved into the approaches seen today and how these are used in web application security. The related work section present finding of work that are relevant for this thesis.

4.2 Interview

To answer research question 1 a semi-structured interview was performed. The goal was to have a somewhat free conversation where the subjects talked about their application and their development process. A list of topics necessary to cover was planned beforehand. The companies would talk about each topic and when necessary follow-up questions were asked. The goal was to determine if and how they work with security, establish the business context, listen to their thought around the risks connected to the application and get some idea of how familiar they were with the OWASP top ten.

4.3 Testing

To establish whether any of the applications are exposed to the OWASP top ten, vulnerability testing was performed. There are two possible techniques to test for security holes; black-box testing and white-box testing.

Black-box testing or penetration testing is based on penetrating the running application from a user perspective. This means that the tester has no special privileges or prior knowledge of the application other than that available for everyone. This is the approach most similar to a real attack scenario where an adversary is trying to compromise the application. The tester adapts the mindset of an attacker and illustrate real security holes instead of hypothetical ones. The disadvantage of this approach is that the discovery of some vulnerability using this method establishes with certainty that the vulnerability exists, however if a vulnerability is not found this is no proof that none are there. Also, the success of the testing relies upon the skills of the tester and it is assumed that the skills of the attacker does not exceed those of the tester.

White-box testing or static analysis is based on reviewing the source code looking for security flaws. This approach is quite time consuming and require high expertise as some logical flaws may require detailed knowledge of the workings of the entire system.

Additionally the testing can either be done manually or automatically by using automated tools.

Austin and Williams (2011) found that in order to fully disclose the state of security in an application, multiple techniques should be used. They found that the different techniques discover very different types of vulnerabilities. Automated penetration testing was found to be the most effective way of finding vulnerabilities. However, the tools sometimes produce false positives and each vulnerability needs to be verified manually. Manual penetration testing was the best approach for finding architectural flaws. Automatic static analysis was better at finding implementation flaws. To get the full picture of the security it is recommended to use all three techniques.

The technique used in this thesis is a combination of automated penetration testing and systematic manual penetration testing. Testing followed a test plan developed to cover OWASP top ten and multiple tests were done for each. The advantage of following a test plan is that by the end of testing the tester is confident that all vulnerabilities are covered. For some of the vulnerabilities there exist useful tools to get a quick assessment. The vulnerabilities found must then be verified manually to determine whether. For other OWASP top ten vulnerabilities, for example “Broken Authentication and Session Management” that are of an architectural nature, manual testing was mostly used.

A combination of these two techniques was found to be the most time efficient way to cover the focus area of this thesis. The tools used were Burp Suite, Whatweb, Netcat, Dirbuster, Nikto, openSSL and Nmap. All tools were available on Kali Linux.

4.4 Risk rating

The most important thing is to find out what is the most important thing

Shunryu Suzuki

When the testing was finished and the list of vulnerabilities was completed, each of the vulnerabilities were evaluated based on the OWASP risk rating methodology (Owasp.org, 2017c). The OWASP risk rating methodology is a method for evaluating the risk connected to a vulnerability based on severity. The severity is measured by the likelihood and impact of the vulnerabilities according to the parameters in figure 4.1. The vulnerabilities are rated as either; high, medium or low.

Because the OWASP risk rating methodology is a tool for evaluation it is necessary to consider the correctness of the rating. The value of the method is not the exact value assigned, but the assurance that all relevant aspects are reflected in the evaluation. Because of this the evaluation done only partly follows the OWASP risk rating methodology. The likelihood and the technical impact and business impact follow the methodology accurately. When the impact is evaluated the methodology states that only the business impact should be considered. In this thesis both the technical and business impact are considered, and the severity is evaluated as the average of the likelihood and impact. This is because it in my view gives a more correct picture of the severity. The reason for this may be that the business impact was not accurately assessed as this normally is a longer process. Still I wanted both the technical and the business context to be reflected in the final evaluation to capture all the known information. I believe that this is appropriate because after all the goal of the process is to obtain a true evaluation of the risk. After acquiring the results following this method, the risk connected to some of the vulnerabilities are further adjusted. The final list of vulnerabilities and risks forms the basis for discussing how well security is maintained in the startups.

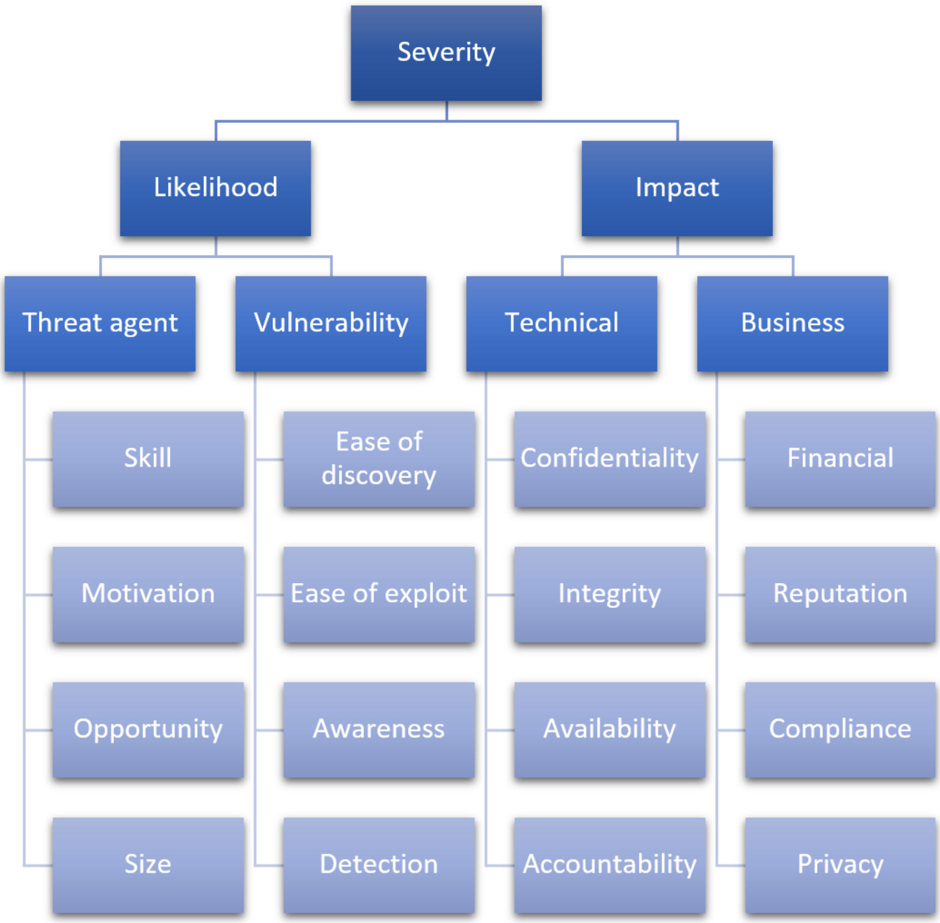


Figure 4.1: OWASP Risk Rating Methodology

4.5 Evaluation/recommendation of relevant tools and resources

Based on the experience gained during testing a set of tools and resources will be recommended. In the pre-project (Søhoel, 2016) the following criteria were defined for this:

- *The tools should be free or not too costly*; start-ups often do not have a lot of resources to pay for expensive tools.
- *The tools should be quick and easy to install and use*; for non-experts whose main priority is developing a good application, it should be easy and intuitive to get started with security improving tools.
- *There should be a limited amount of tools to cover all of OWASP top ten*; the collected set of tools should cover all of OWASP top ten but the number should be limited to make it both easier and faster to get acquainted with all.
- *One should not need to be an expert in order to benefit from the tools*; the recipient is a non-expert.
- *The tools should give a limited number of false positives*; If the tools gives too many false positives, verifying all positives get more time consuming than manual testing.
- *The tools should be well documented*; This is important for accelerating the learning process and making the tools useful in less time.

Chapter 5

Resources and tools

During testing there were some tools and resources that proved to be very useful. This chapter gives a introduction to each of these and can be used as a encyclopedia for terms that are unknown to the reader.

5.1 Resources

The resources used in this thesis are the OWASP testing guide, OWASP cheat sheet and the Common Vulnerabilities and Exposures (CVE) database.

5.1.1 OWASP Testing Guide

OWASP Testing Guide (Meucci and Muller, 2014) gathers a comprehensive selection of security tests for web applications. For each test there is a tutorial on how to do black-, grey- and white-box testing and tools are recommended. This is a great place to start for anyone who wants to learn how to perform security tests and very little knowledge is required. It includes 83 defined tests; some of these relevant to the OWASP top ten were selected for the testing perfomed in this thesis.

5.1.2 OWASP Cheat Sheets

The OWASP cheat sheet (Manico et al., 2017) is a series of documents providing information on how to successfully prevent certain vulnerabilities. These are easy to understand and break down step by step what needs to be done. This can be used by developers both as a learning document and a checklist.

5.1.3 CVE

CVE is a database where all publicly known vulnerabilities are registered. Online it is possible to search for vulnerabilities connected to vendors and product and one can review statistics (CVEdetails.com, 2017). Each vulnerability is registered with an identifier and a Common Vulnerability Scoring System (CVSS) severity score.

5.2 Tools

During testing seven tools were used; Burp Suite, Whatweb, Netcat, Dirbuster, Nikto, openSSL and Nmap. They are all available on Kali Linux. This section will give a short description of all of them.

5.2.1 Kali Linux

Kali Linux (Kali.org, 2017b) is a penetration testing Operating system that can be run on a virtual machine, a dvd or usb drive. It contains a number of useful tools that can be used for penetration testing and digital forensics. It is a Linux distribution that is open source and free to use. With Kali Linux there is only one thing to download and set up and a number of well documented tools are ready to use. All tools used in this thesis is available at Kali Linux.

5.2.2 Burp Suite

Burp suite (Portswigger.net, 2017) is a tool that can be used both for manual penetration testing and automated penetration testing. It has multiple useful functionalities that are seemingly integrated. Burp suite is set up as a HTTP proxy server and all traffic is cached by this proxy. All HTTP packets can be inspected, repeated and manipulated before being sent. The functionality used in this thesis were; target, proxy and repeater.

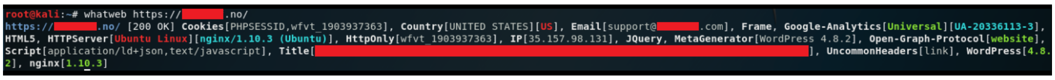
The target functionality can be used to spider the application and determine the attack surface.

The proxy functionality is by far the most useful function. It can be used to review packet history and all HTTP requests and responses can be inspected. This reveal a lot of information about both the flow of the web-site, the configuration of the server and how session and authentication is managed. It is possible to use the proxy with interception mode where all traffic can be reviewed and changed before sending, or in regular mode were the user browses the application in the browser and can look at the HTTP history later. Furthermore, interesting HTTP requests can be sent to the repeater for further use.

The repeater can be used to replay packages and manipulate them if required. When the response is returned it is showed to the user.

5.2.3 Whatweb

Whatweb (Morningstarsecurity.com, 2017) is a tool that can be used to find; IP address, platforms, libraries, plugins, country, cookies, headers and more. It can be used both in passive mode and active mode. In this thesis only the passive mode was used to get a quick overview of frameworks and get the IP address. Typing in the command “whatweb” and the URL of the website in a terminal will return this information. This can be seen in figure 5.1.



```
root@kali:~# whatweb https://[redacted].no/
https://[redacted].no/ [200 OK] Cookies[PHPSESSID,wfvt_1903937363], Country[UNITED STATES][US], Email[support@[redacted].com], Frame, Google-Analytics[Universal][UA-20336113-3],
HTML5, HTTPServer[Ubuntu Linux][nginx/1.10.3 (Ubuntu)], HttpOnly[wfvt_1903937363], IP[35.157.98.131], JQuery, MetaGenerator[WordPress 4.8.2], Open-Graph-Protocol[website],
Script[application/ld+json,text/javascript], Title[[redacted]], UncommonHeaders[link], WordPress[4.8.
2], nginx[1.10.3]
```

Figure 5.1: Whatweb

5.2.4 Netcat

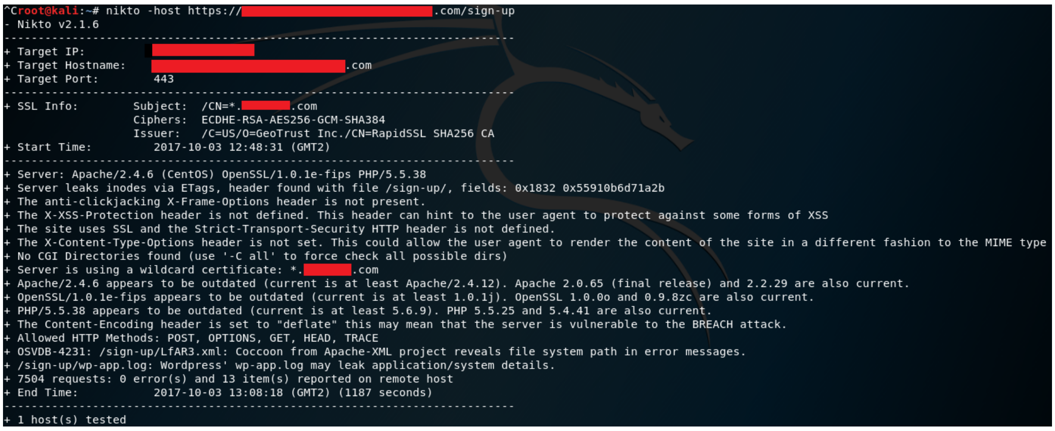
Netcat (Nmap.org, 2017a) is a networking tool normally used to test TCP or UDP connections. In this thesis it was used to establish a TCP connection to the host and inspect the headers of a HTTP response. Burp Suite could be used for this same purpose and was normally used instead.

5.2.5 Dirbuster

Dirbuster (Kali.org, 2017a) can be used to brute force file names and directories in order to spider an application. The user can choose among some standard word-list or create a unique one. Dirbuster sends a request to the website with all the entries in the word-list as the URI and keeps track of which ones returned a success response.

5.2.6 Nikto

Nikto (Sectools.org, 2017) is a web server scanner. The user simply enters the URL of the site, and Nikto will return a list of potential security flaws found. The result from one of the websites can be found in figure 5.2



```
^Croot@kali:~# nikto -host https://[redacted].com/sign-up
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          [redacted]
+ Target Hostname:    [redacted].com
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /CN=*.[redacted].com
                   Ciphers:  ECDHE-RSA-AES256-GCM-SHA384
                   Issuer:   /C=US/O=GeoTrust Inc./CN=RapidSSL SHA256 CA
+ Start Time:         2017-10-03 12:48:31 (GMT2)
---------------------------------------------------------------------------
+ Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips PHP/5.5.38
+ Server leaks inodes via ETags, header found with file /sign-up/, fields: 0x1832 0x55910b6d71a2b
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The site uses SSL and the Strict-Transport-Security HTTP header is not defined.
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Server is using a wildcard certificate: *.[redacted].com
+ Apache/2.4.6 appears to be outdated (current is at least Apache/2.4.12). Apache 2.0.65 (final release) and 2.2.29 are also current.
+ OpenSSL/1.0.1e-fips appears to be outdated (current is at least 1.0.1j). OpenSSL 1.0.0o and 0.9.8zc are also current.
+ PHP/5.5.38 appears to be outdated (current is at least 5.6.9). PHP 5.5.25 and 5.4.41 are also current.
+ The Content-Encoding header is set to "deflate" this may mean that the server is vulnerable to the BREACH attack.
+ Allowed HTTP Methods: POST, OPTIONS, GET, HEAD, TRACE
+ OSVDB-4231: /sign-up/LfAR3.xml: Coccoon from Apache-XML project reveals file system path in error messages.
+ /sign-up/wp-app.log: Wordpress' wp-app.log may leak application/system details.
+ 7504 requests: 0 error(s) and 13 item(s) reported on remote host
+ End Time:           2017-10-03 13:08:18 (GMT2) (1187 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Figure 5.2: Nikto

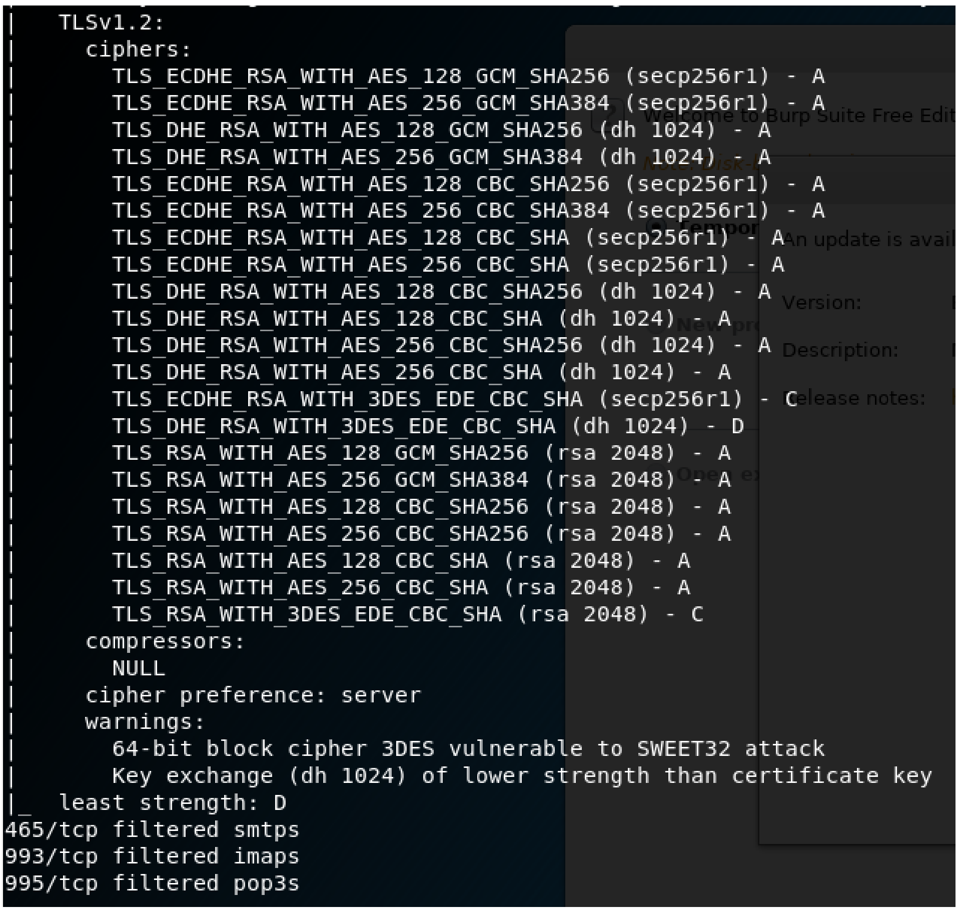
5.2.7 OpenSSL

OpenSSL (Openssl.org, 2017) is not really a hacking tool but an open source implementation of the Secure Sockets Layer (SSL) protocol. It can be used to manually check for

SSL vulnerabilities. In this thesis it was used to check for secure renegotiation, public key size, SSL version and more.

5.2.8 Nmap

Nmap (Nmap.org, 2017b) is a security scanner that was used to scan open ports and establish enabled crypto systems on each port. Nmap also rates each crypto system with a grade from F to A and the weakest enabled crypto system is highlighted. This is important because even if the default crypto system is secure on the port, it is possible to force communication to be done with a weaker crypto system. Therefore the weakest enabled crypto system should also be secure. The results of Nmap for one of the applications can be seen in figure 5.3.



```
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (secp256r1) - A
|       TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (dh 1024) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - A
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - A
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256r1) - C
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - D
|       TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C
|     compressors:
|       NULL
|     cipher preference: server
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       Key exchange (dh 1024) of lower strength than certificate key
|_  least strength: D
465/tcp filtered smtps
993/tcp filtered imaps
995/tcp filtered pop3s
```

Figure 5.3: Nmap

Chapter 6

Company context

This chapter provides some background information on the companies. Section 6.1 present the five applications and the start-ups behind them. In sections 6.2 the insight gained from the interviews are presented.

6.1 The five participating start-ups

To take part in this study five very diverse start-up companies have been recruited. Some store very sensitive and critical information about their users, while others are less critical in the sense that compromising the site will not have major consequences for the users. Some of the start-ups have a lot of knowledge and experience about web development and security, while others are more inexperienced and “learn-by-doing”. One of the companies does not develop the application themselves but has outsourced all coding to a firm abroad. The founders all have backgrounds in business and management.

6.1.1 Company A

The first Company is a B2B business delivering a time management and project management systems to companies. The site stores sensitive information about their customers such as customer relationships, invoices and personal information about the employees such as national identities, bank accounts and e-mails. The founders all have backgrounds in business and management and have outsourced development to a company abroad.

6.1.2 Company B

Company B is a crowd-sourcing website for educational materials. The users are generating content in the form of Q/A, quizzes and discussions. What each user is allowed to do depends on the level of privileges. A user acquire higher privileges by contributing to the site and getting up-votes from other users. It is also possible to visit the site as a guest user. The company consists of 4-5 students studying computer science and related fields.

6.1.3 Company C

Company C is an open source project for online booking of laundry facilities. The users can create laundries and invite tenants to book machines. The start-up consist of 4 software development engineers with experience with web development and security. The application is a hobby project beside full time jobs.

6.1.4 Company D

Company D provide a communication solution for medical personnel, patients and dependents. It stores highly sensitive and personal information about patients connected to a medical institution. The content can be of the form of pictures, videos, texts or timetable of patient, routines and even when the patients last went to the toilet. One of the things that make this site so critical is that many of the patients are not legally competent and cannot consent to their information being made public. It is therefore important to protect this group from being exploited. The application was developed as an alternative to publishing and sending this information through Facebook.

6.1.5 Company E

Company E are providing tutoring services for children in primary school. The tutor has to upload a certificate of good conduct from the police and their diploma on the web site. The students enter what grades they have in the subject they want tutoring in and what grade they want to achieve. Also the students enter payment information. The founders have degrees in computer science and related studies.

6.2 Interviews

During the interviews the companies were asked about what was the most important business assets in the application and together we did some risk analysis. I wanted to know what was critical to protect on the page, who the threat agents were and what the worst-case scenario is if the application were to be attacked. They were also asked about how they worked with security. In the end we went through the OWASP top ten to determine their knowledge. We also discussed which one of OWASP top ten were the most relevant for protecting their specific assets.

6.2.1 Company A

During the interview we agreed that their most critical assets are the sensitive information about people and businesses, and leakage of this information would lead to end of business for the start-up. The most critical vulnerabilities are those connected to authentication and access control. They said that until now the priority has been on functionality and creating a working application, but now that they have real customers they want to focus on making the application secure. From the OWASP top ten only injection attack was known to them. They have no idea whether their application is secure or not and have so far not been doing any security activities. It is clear that the founders are eager to deliver the best possible product to their customers and that they now realize that security is a part of that.

6.2.2 Company B

There is no sensitive information on the page that is not already open for everyone to see. Therefore the greatest risks on this page are destruction and spamming. CSRF, XSS and privilege escalation are therefore the most severe vulnerability. Unsafe redirections are also relevant for this page because the site can be used as an enabler for phishing attacks. Apart from using secure libraries they haven't done any security activities and the coders are in charge of handling security. They had heard about OWASP from taking a security course at university and had knowledge about some of vulnerabilities like injection and XSS, but they did not have detailed knowledge about OWASP top ten. They suspect that there might be a lot of vulnerabilities on the site and was really eager to have it tested. Also, they offered to help with the testing and clearly wanted to learn more about security.

6.2.3 Company C

The application is developed by 4 professional web application developers next to their full time jobs. They seem to have a lot of knowledge and experience on both web development and security. They store as little information about the users as possible to make the application simple to use. The most sensitive info on the site are the e-mail addresses of users. We agree that the biggest threats to the system are leakage of e-mail addresses and passwords, spamming, changing and deleting of bookings and general destruction. To ensure security they make sure to use secure frameworks and use the guides of renowned developers on how to use them correctly. For example once a week they use Snyk to scan through all dependencies in their project to look for libraries with known vulnerabilities. They have been testing for access control and enumeration of e-mails. They were familiar with OWASP top ten and strives to keep themselves up to date on them.

6.2.4 Company D

The patients are in many cases not able to consent to how their information is managed and it is therfore super critical to protect it. The motivation for the app was to prevent distribuation of this information on Facebook. The information stored is not really valuable in itself but because of how sensitive it is to the individuals it belongs to it is so important to protect. In some cases information from medical records may appear on the page. Because of the sensitivity the business has a lot of focus on security. They have security reviews every six months, make sure to have all frameworks up to date and keep themselves updated on known vulnerabilities. Also they get a lot of advice from the Norwegian Data Protection Authority (Datatilsynet) to make sure they comply with Norwegian law. When asked about the OWASP top ten they gave me a list stating how they protect themselves against each one. They are currently working on implementing BankID which require the highest security level in Norway (Difi.no, 2017).

6.2.5 Company E

After a few months of being unable to get back in contact with this company, I was finally able to schedule an interview. They did not show up for that interview and when trying to schedule a new one later on I did not hear from them again. Therefore I in this case was unable to do an interview with this company. I still wanted to test the application as I was curious to find out whether failure to participate in the interview, meant secuirty has low priotiy.

	Outsourcing	Information sensitivity	Knowledge about OWASP	Web Development Experience
Company A	Yes	Medium/High	Low	Low
Company B	No	Low	Low	Low
Company C	No	Low	Medium	High
Company D	No	Very High	High	Medium
Company E	No	High	-	-

Table 6.1: Summary of companies

Chapter 7

Testing

Chapter 7 goes through the test plan and the results from testing. Chapter 7.1 explains the order and technical details of the test plan. Chapter 7.2 presents detailed results for each test. The results are summarized in table 7.1.

7.1 Test plan

The order of which each vulnerability is tested is carefully chosen and follows the order in which they are listed below. They are grouped together with related vulnerabilities where the tools and/or testing technique is similar. They appear in this order because some information from one vulnerability is later used in testing another. Some vulnerabilities are tested together because they both need the same base work. This is the case when two vulnerabilities both need a HTTP request/response, the HTTP request/responses of all critical transactions or all entries where it is possible to enter user input. All in all this order was found to be the most time efficient.

Information Gathering The first phase of any good penetration test is information gathering. This phase consists of gathering as much information about how the site works and maybe find hints that can be used later on in the testing. The goal is to determine how the application is configured and what components are used (A5 and A9). As part of this, figuring out how encryption work(A6) is also essential. Tools like Whatweb, Nikto, Nmap and openSSL are used to get a quick overview. Then the findings are verified by inspecting HTTP responses from the server in Burp Suite and replay messages to verify assumptions. The source code, cookies and error messages are also inspected to look for any relevant information.

Transactions & Authentication When knowledge about the web server and application is acquired, we move on to understanding how transaction, authentication and access control work and test for CSRF (A8), broken authentication and session management (A2). The test consist of navigating on the page and getting an overview of critical transactions while saving all HTTP request/response in Burp Suite. The goal is to determine how account provision, authentication, session management and access control work, and if it is possible to bypass it. The HTTP packets are used to determine if CSRF is possible.

Access Control Next phase is testing for unvalidated redirects and forwards (A10), missing function level access control (A7) and insecure direct object references (A4). Now that Burp Suite has cached all possible transactions on the page we go through all of them looking for interesting file references, function calls and redirects and try to exploit them.

Injection All fields that accept input from a user both in the browser and http packets are tested for SQL injection (A1) and XSS (A3).

7.1.1 A5 - Security Misconfiguration

To test for Misconfiguration seven tests are done from the OWASP testing guide; OTG-INFO-002, OTG-INFO-008, OTG-INFO-009, OTG-CONFIG-002, OTG-CONFIG-006, OTG-ERR-001 and OTG-ERR-002. The goal of these tests is to get as much information about the server and application as possible that later can be used in A9. We also look at how the HTTP responses are configured, look for default files and directories and try to provoke error messages.

First step is to determine the type of web server. Whatweb is used to find the ip-adress of the target. Then Burp Suite or Netcat is used to inspect HTTP response to find the type of server and framework in the HTTP headers. Further I go to web page and use the command “document.cookie” in the console to see if the cookies used reveal something about the application. It is also possible to look at file extensions and source code to find more information about the framework and versions. In the source code I look for comments, application specific paths and script variables.

Next we are trying to determine the file structure by using either Dirbuster or Burp Suite spidering and checking the robots.txt file.

Further I scan the application using Nikto to get a quick overview of security headers, HTTP methods and other security configurations. All the results I get form this test I need to verify manually to check whether the security issues reported actually constitute a real vulnerability.

Next I want to verify that the HTTP methods found with Nikto is correct and for what URI they are enabled. I use Burp Suite to send an OPTION request to different pages and look at the allowed responses. I then try to generate an error message on the page to gain further information about the page.

7.1.2 A9 - Using Components with Known Vulnerabilities

Testing for this vulnerability simply involves googling the components discovered under A5 and checking the CVE database to see if there exist vulnerabilities. First I run Nmap on the application to enumerate if there are more services running on different ports and what they are.

7.1.3 A6 - Sensitive Data Exposure

To test for Sensitive Data Exposure I do the three tests OTG-CRYPST-001, OTG-CRYPST-003 and OTG-CONFIG-007. I use openSSL to check how well SSL/TLS has been configured and Nmap to inspect the enabled crypto schemes. I also check that all sensitive information is sent over an encrypted channel and whether the strict-transport-security header is set on the HTTP requests sending this information. Also relevant for this vulnerability is leakage in source code, error messages and exceptions.

7.1.4 A8 - Cross-Site Request Forgery (CSRF)

CSRF can be a bit tricky to test for.

In the OWASP Testing guide the test described for Cross-Site-Request-Foregry(OTG-SESS-005) is simply to try and perform the attack to see if it works. This to me seemed like a lot of trouble when all I am really interested in is whether it is possible or not. Also, testing through performing the attack means testing all factors at once, and if the attack is unsuccessful I still would not have established if it is possible or not. Instead I followed the OWASP CSRF Prevention Cheat Sheet to see if the best practise here are followed.

According to OWASP the first ting you need to do to prevent CSRF is to check the origin and/or reference fields and compare them to the host fields in the HTTP request. If neither one of the headers are present the server should block the request. OWASP also emphasize that in addition to this measure a token should be sent together with each request to authenticate the user. The best practise here is to use a synchronized token that changes for each request or to use a Anti-CSRF token that is stored as a hidden field in the input form(as recommended by both OWASP(Owasp.org, 2017b) and acunetix(Acunetix.com, 2017)).

According to RFC6750 (Michael B. Jones, 2012) using a bearer token is also secure against CSRF as long as it is not stored in a cookie. However, according to OWASP the session token will be submitted automatically by the browser no matter if the user has sent the request voluntary or was tricked(Owasp.org, 2017a). Therefore the site is vulnerable if session management is‘relying only on information which is known by the browser [...] “Known by the browse” refers to information such as cookies, or http-based authentication information (such as Basic Authentication; and not form-based authentication), which are stored by the browser and subsequently present at each request directed towards an application area requesting that authentication.” (Owasp.org, 2017d). Therefore, when authorization seems to rely solely on tokens that are only present in the authorization header and not in forms in the HTTP request entity-body, I will assume that it might be vulnerable to CSRF. However this is ambiguous as the specifications imply the opposite. Also, there might be framework specific defences not visible to me while testing. As of 2017 CSRF is no longer on the OWASP top ten list, mainly because a lot of frameworks include CSRF defences.

7.1.5 A2 - Broken authentication and session management

Testing for Broken authentication and session management consist of 12 tests; OTG-IDENT-001, OTG-IDENT-003, OTG-IDENT-004, OTG-AUTHN-001, OTG-AUTHN-002, OTG-AUTHN-003, OTG-AUTHN-006, OTG-AUTHN-007, OTG-AUTHN-009, OTG-SESS-003, OTG-SESS-004, OTG-SESS-006. Here account provisioning, usernames, password policy, lockout mechanism, log-out functionality, browser cache, session variables and password recovery is investigates. In the start tests concerning cookies were also performed but it became clear that these issues were not relevant anymore due to tokens replacing cookies in session management.

7.1.6 A10 - Invalidated Redirects and Forwards

Testing for this is based on OTG-CLIENT-004 from the OWASP testing guide. I start by catching all HTTP request/responses while I go through the web page doing all sorts of actions and transactions. Then I look for requests that have the parameter “redirectUrl” in the URI and try to manipulate it to see if I get redirected to the manipulated page. If I am redirected the A10 vulnerability exist. Otherwise I get an error message.

7.1.7 A7 - Missing Function Level Access Control

Testing for this vulnerability is based on the three tests OTG-AUTHZ-001, OTG-AUTHN-004 and OTG-AUTHZ-002 from the OWASP testing guide. Here I try to manipulate parameters in URIs, forced browsing and doing high privilege transactions with low privilege credentials. If I can access any functions, files or pages I were not supposed to the A7 vulnerabilities exist.

7.1.8 A4 - Insecure Direct Object References

Here the tester tries to load objects or pages by changing parameters in URIs or get restricted files returned through manipulation location of files retrieved.

7.1.9 A1 - Injection

For this vulnerability I mainly test for SQL injection. I go through all input files and other places where user input is accepted; like URIs, cookies and tokens, and try to provoke a database error message. I add semicolon or quotes and inspect the HTTP responses from the server. If I get a database error the quote or semicolon is not filtered properly and the field is vulnerable to sql injection. Another family of severe injection attacks is the buffer overflow but because of the time constrains of the thesis there is not enough time to test for this.

7.1.10 A2 - Cross site-scripting

There is a lot of XSS protection in the browser. We first need to disable all of these. I then use Google Gruyere to verify with a XSS that all the different protections are in fact disabled. Then I try to insert JavaScript in all input fields to see if I can get it to run. I enter the attack “<script>alert(XSS) </script >” to see if I get an alert box. I then go to inspector to verify if I need to adjust the code in any other way. Here it is also interesting to see if any of the characters are filtered. If they are this might mean that site uses only blacklisting to protect against XSS. This is not a good idea since there are a lot of filter evasion techniques to get around this filtering. If the script does not run, even after getting around the filtering, the site separates code and user input in a safe way.

7.2 Results

This section present the results form testing. Section 7.2.1 gives a summary of the findings for all the companies while 7.2.2 to 7.2.6 present detailed results for each company and explanation of the finding. The findings for each company are summarized in tabels 7.2 to 7.6.

7.2.1 Result summary

Table 7.1 show a summary of the results. “P” here stands for “Passed” and “F” stands for “Failed”. Detailed description of each company follows under chapter 7.2.2 through 7.2.6. In general, vulnerabilities of an architectural nature such as A2 and A6 were common, while for vulnerabilities connected to implementation such as A1, A3 and A10 almost none were found.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Company A	P	F	P	P	F	F	P	F	F	P
Company B	F	F	P	F	F	F	F	F	P	P
Company C	P	F	P	P	P	P	P	F	P	P
Company D	P	F	P	F	P	F	F	P	F	P
Company E	P	F	P	P	P	F	P	P	P	F

Table 7.1: Summary of results

7.2.2 Findings: Company A

This subsection presents the detailed results from testing application B. A summary of the results can be found in table 7.3. There were found a lot of vulnerabilities for this application. Their biggest issues were connected to authentication and session management and configuration.

A5, A6 and A9 - Configuration, Ciphers and Components

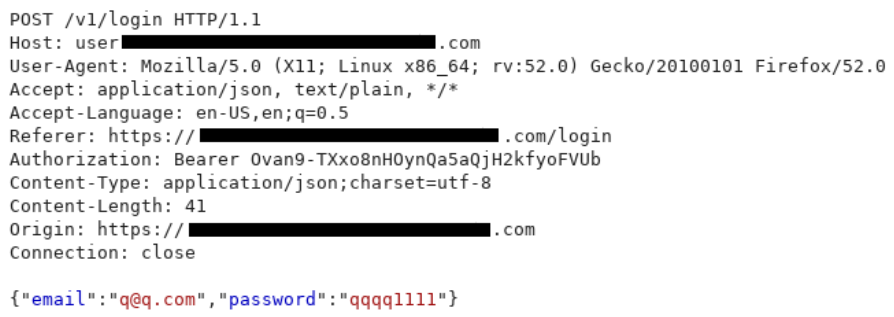
In the HTTP responses from the web server the Server field is not hidden and lets us know the type and version of the server, programming language and openSSL. All the versions where outdated. The PHP version was last supported two years ago and it is stated on the website that not even with the appearance of security issues will the release be patched. For the openSSL version there where registered several vulnerabilities in the CVE database among them some with CVSS score 10.

Vulnerability	Result	Evaluation
A1	None found	Passed
A2	Weak account provisioning, weak password policy, no lockout mechanism, no session timeout	Failed
A3	Use filtering	Passed
A4	None Found	Passed
A5	Framework outdated, Security headers not set, insecure HTTP methods enabled	Failed
A6	HTTP Strict Transport Security not set, weak crypto scheme, Leakage in source code	Failed
A7	None found	Passed
A8	Does not follow OWASP recommended best practice	Failed
A9	Outdated components that are no longer supported	Failed
A10	None found	Passed

Table 7.2: Results company A

Another pretty severe security issue is that none of the security headers are set such as; X-XFRAME-Options, X-XSS-Protection, X-content-Type-Option, Content-security-Policy and Referrer-Policy. This means that the site is susceptible to cross site request forgery, XSS, clickjacking etc. Also the HTTP methods TRACE, PUT and DELETE are enabled. This can be seen in figure 7.1 and 7.2

There are a lot of comments in the source code that reveal information about the application that there is no reason for clients to know.



```
POST /v1/login HTTP/1.1
Host: user[redacted].com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Referer: https://[redacted].com/login
Authorization: Bearer Ovan9-TXxo8nHOynQa5aQjH2kfyoFVUb
Content-Type: application/json;charset=utf-8
Content-Length: 41
Origin: https://[redacted].com
Connection: close

{"email":"q@q.com","password":"qqqq1111"}
```

Figure 7.1: HTTP request sending username and password

SSL/TLS ciphers and transport layer protection The application uses HTTPs but the HTTP-strict-Transport security header is not set when sending credentials which means the website is vulnerable to downgrade attacks. If a user is forced to use HTTP when logging in he/she will send both username and password in cleartext and the server will respond by sending the authentication token also in clear text. I use openSSL and find that they use a secure version of SSL, use secure renegotiation, the certificate is fresh and valid

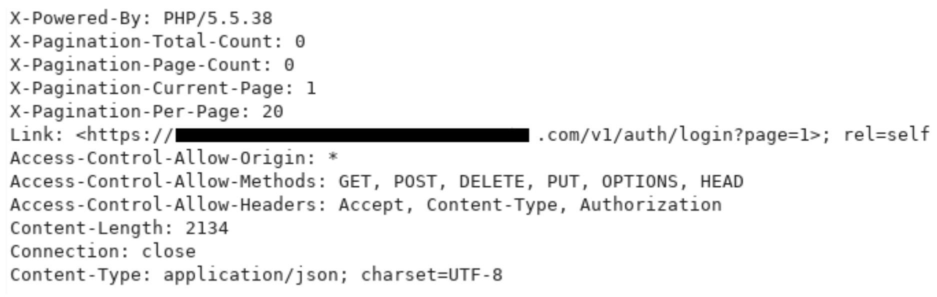

```
X-Powered-By: PHP/5.5.38
X-Pagination-Total-Count: 0
X-Pagination-Page-Count: 0
X-Pagination-Current-Page: 1
X-Pagination-Per-Page: 20
Link: <https://[redacted].com/v1/auth/login?page=1>; rel=self
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, DELETE, PUT, OPTIONS, HEAD
Access-Control-Allow-Headers: Accept, Content-Type, Authorization
Content-Length: 2134
Connection: close
Content-Type: application/json; charset=UTF-8
```

Figure 7.2: HTTP response of the request in figure 7.1. Security headers not set

and the server public key is long enough. However, the weakest available crypto system is a Diffie-Hellman 1024, which is too low. Again this makes the application sensitive to downgrade attack even though the default settings are secure.

Information leakage in source code In the source code there is a lot of comments that reveal a lot about directories, functions and parameters. Especially critical is that the comments reveals how admins and managers are verified and what functions/elements they have access to. This information could be very helpful for an attacker planning an attack. There is no reason to have these comments and best practise is to remove all comments.

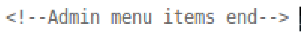

```
<!--Admin menu items end-->
```

Figure 7.3: Admin menu

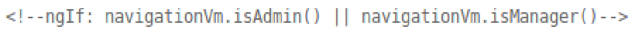

```
<!--ngIf: navigationVm.isAdmin() || navigationVm.isManager()-->
```

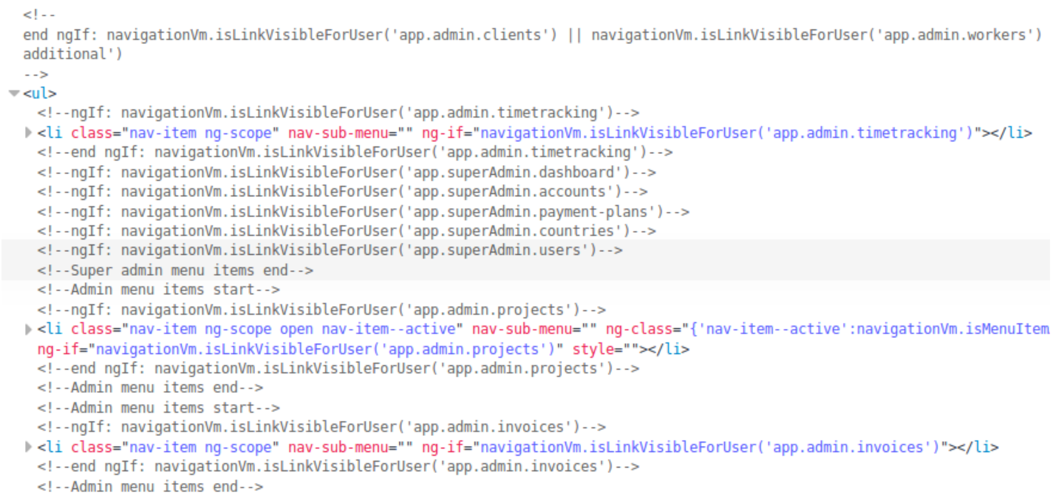
Figure 7.4: Information leakage in the source code

A8 - Cross Site Request Forgery

I successfully loaded the admin page and created new admin users without including the Origin or Reference field. I conclude that the Origin and Referrer headers are not checked. The second check is to verify whether authentication relies only on the information known by the browser. Each request is authenticated through a bearer token in an authorization header. There is no synchronized token or aditional CSRF token. Because of these findings I conclude that the web server does not implement the OWASP recommended settings to prevent CSRF and that the requirements that need to be present to perform the CSRF are all there.

A2 - Broken Authentication and Session Management

This is what we agreed upon during the interview was one of them most critical ones because if unauthorized users are able to log in as other users or admin users, then all the



```
<!--
end ngIf: navigationVm.isLinkVisibleForUser('app.admin.clients') || navigationVm.isLinkVisibleForUser('app.admin.workers')
additional')
-->
▼<ul>
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.admin.timetracking')-->
 ▶<li class="nav-item ng-scope" nav-sub-menu="" ng-if="navigationVm.isLinkVisibleForUser('app.admin.timetracking')"></li>
  <!--end ngIf: navigationVm.isLinkVisibleForUser('app.admin.timetracking')-->
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.superAdmin.dashboard')-->
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.superAdmin.accounts')-->
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.superAdmin.payment-plans')-->
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.superAdmin.countries')-->
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.superAdmin.users')-->
  <!--Super admin menu items end-->
  <!--Admin menu items start-->
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.admin.projects')-->
 ▶<li class="nav-item ng-scope open nav-item--active" nav-sub-menu="" ng-class="{'nav-item--active':navigationVm.isMenuItem
  ng-if="navigationVm.isLinkVisibleForUser('app.admin.projects')" style=""></li>
  <!--end ngIf: navigationVm.isLinkVisibleForUser('app.admin.projects')-->
  <!--Admin menu items end-->
  <!--Admin menu items start-->
  <!--ngIf: navigationVm.isLinkVisibleForUser('app.admin.invoices')-->
 ▶<li class="nav-item ng-scope" nav-sub-menu="" ng-if="navigationVm.isLinkVisibleForUser('app.admin.invoices')"></li>
  <!--end ngIf: navigationVm.isLinkVisibleForUser('app.admin.invoices')-->
  <!--Admin menu items end-->
```

Figure 7.5: Information leakage in the source code

information stored for that company is compromised. All in all i executed 14 tests for this vulnerability. Below I will summarize the most interesting results.

Weak provisioning process On the front page of the company web page a 14-days free trial is promoted where you register your e-mail and get a verification link sent to your e-mail. This is a good precaution because you need some form of verification that the e-mail address exists and that it belongs to the user. However, if you go straight to the log-in page you get the option to register and on this page there is no verification process except the email address has to be on the form char@char.com. So I register the user q@q.com and company name q. Also it turns out that the verification link sent on email actually is only a link to this page and does not verify your e-mail at all. Now administrator q of company q can create users for all its employees from the admin dashboard and set their roles and password. The employees e-mail is even now not verified. Furthermore, Administrator Q can make another administrator Y without any second authentication, Y in-turn can change the status of Q to inactive and lock Q out of the platform. Therefore if anyone gets access to an administrators account accidentally or by targeting the page, the threat agent can take over the platform and lock out everyone else. A user cannot create users with higher privileges than oneself, but in the source code I find comments stating that if isAdmin() is true then there should be some functionality here.

Account enumeration When you try to log in with wrong the username and password you get the message “The e-mail or password is incorrect” which is good because it does not reveal whether it is the username or password that is wrong. Also I could not find any difference in the server response either with the number of/type of packets or how much time it took to process the request. However, if you go to the “forgotten password”-page you can write in any e-mail address and either get a response that “e-mail to re-store password” is sent or you get a “e-mail not in use” error-message. That way we can check or bruteforce username. Because I know that there exist a administrator dashboard for the start-up founders and because I know that the username has to be on the form char@char.com I check the e-mail address of the guys in the start-up and get the success message. I assume that I now know the username for the admin users. Also I find on the start-up web-page a “satisfied users”-section. On this page i find the name of the company and the CEO’s of customers, from that point it is not hard to find the e-mail addresses(and most likely usernames) of companies using the platform.

Weak lockout mechanism There is no lockout mechanism when entering wrong a password hence bruteforcing password is possible. I have the username of both system admins and company admins and I can bruteforce their passwords.

Weak password policy All signs are allowed in the password which is good because it does not limit the search area for bruteforcing. The password must contain at least eighth characters, one letter and one number. Depending on the strength of the password it may take less than a minute or more than a year to bruteforce the password. In other words it is up to each user to make a password that is strong enough.

Cookies and session management The application does not use cookies for session management but an authentication token is sent to the client right after authorization and is included in every HTTP request when navigating the page. The authorization bearer is regenerated for each log-in and remains the same throughout the session. There is no session timeout, which basically means that if the customer log-in on a borrowed computer and forgets to log-out the owner of that computer can use that session forever. Even if you close the browser without logging out the session is still active. Considering that simply closing the browser is a quite common way of ending a session we can assume that there are a lot of inactive but valid sessions out there. Only by pressing the log-out button is the authorization bearer changed to inactive.

Log-out functionality and cache The log out button works as expected. The session is closed and the authentication bearer is no longer active. When loading pages in some HTTP packets the no-cache flag is set while in others packets it is not set. After log out i tried to reload pages containing sensitive information but did not succeed. Then I went to about:cache in the browser to see if I could find something sensitive left. I found some remains from www.example.com/invoices page. I could see that the page had been accessed but the content was only encoded gibberish and I was not able to recover it. Otherwise all traces of the page was cleaned out from cache.

A4, A7 and A10 - Access Control and Redirects

I use wfuzz to bruteforce the page for directories. The page use soft 404 which is not a security problem but can still be considered a misconfiguration. Other than that I was not able to manipulate URLs, perform DOM XSS or buy-pass authentication with these techniques.

A3 and A1 - Injection attacks

The app has a lot of filtering on all input fields so you can only write characters or only numbers except in the password field. Using some filter evasion techniques I tried different attacks but were not able to perform either XSS or SQL injection. Even If I was not able to bypass the filtering I hope that this is not the only mechanism protecting the site from XSS as it is not best practice.

This subsection presents the detailed results from testing application B. A summary of the results can be found in table 7.3. The security was rather low on this site, and basic vulnerabilities like SQL injection, Insecure Direct Object Refrences and Missing Function Level Access Control were found.

7.2.3 Findings: Company B

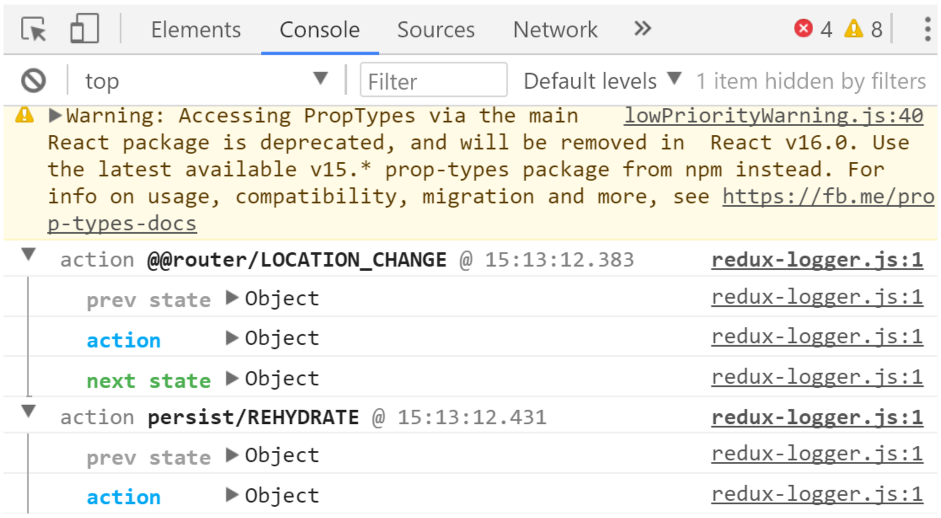
Vulnerability	Result	Evaluation
A1	Can return token of all users	Failed
A2	Token does not expire, log-out functionality does not work	Failed
A3	None found	Passed
A4	Can change device_id to get another users token returned	Failed
A5	Security headers not set, Debugging information printed in console	Failed
A6	Possible to send token over http	Failed
A7	Can perform high privilege action without privileges	Failed
A8	Does not follow OWASP recommended best practice	Failed
A9	No known vulenaribilies	Passed
A10	None found	Passed

Table 7.3: Results company B

A5, A6 and A9 - Configuration, Ciphers and Components

I find that web server is Cowboy and the application uses Express with Node.js. The web site uses Google analytics cookies with 2 years, 24 hours and 1 minute expiration, but these are not used to manage the session. In the source code I find that the application uses react. Also there are some comments in the code that are clearly from a template meant as a tutorial for the developer, however they do not reveal anything too sensitive. In console the application prints a lot of information that looks it like is used for debugging. From Nikto it looks as though some security headers are missing such as; X-XSS-Protection, anti-clickjacking and X-Content-Type-Options. From inspecting the HTTP packets in Burp Suite I find that none of the recommended security headers are set. The application uses HTTP for the start page, but switches to HTTPS after logging in as a guest, this is when a token is generated. However, the Strict-Transport-Security header is not set. In burp suite it looks like the packets are sent over HTTPS, however my browser states that the connection is not secure. It is therefore possible that the token in some cases are sent in clear text, at the very least it might be possible to force the browser to send the token over HTTP. In burp suite I could do privileged actions over HTTP sending my privileged token in the clear. Otherwise the SSL is configured correctly with weakest crypto system graded as C, the rest of the crypto systems had grade A.

I cannot find any known vulnerabilities for the components I discovered. However because the X-powered-by header is not disabled targeted attacks against the applications is a danger. Best practice is to disable this header.



```
▶Warning: Accessing PropTypes via the main    lowPriorityWarning.js:40
React package is deprecated, and will be removed in  React v16.0. Use
the latest available v15.* prop-types package from npm instead. For
info on usage, compatibility, migration and more, see https://fb.me/pro
p-types-docs
▼ action @@router/LOCATION_CHANGE @ 15:13:12.383        redux-logger.js:1
     prev state ▶Object                                 redux-logger.js:1
     action     ▶Object                                 redux-logger.js:1
     next state ▶Object                                 redux-logger.js:1
▼ action persist/REHYDRATE @ 15:13:12.431               redux-logger.js:1
     prev state ▶Object                                 redux-logger.js:1
     action     ▶Object                                 redux-logger.js:1
```

Figure 7.6: Debugging info in console

A8 - Cross-Site Request Forgery (CSRF)

The site uses a token that is changed for each time the user reaches a higher level. However, it is possible to use previous tokens also for privileged actions. Therefore the token acts as a static token and not a synchronized one. The origin and referrer headers are not checked and no special CSRF tokens are used. I base my assumption that CSRF is possible on this site on the material provided by OWASP.

A2 - - Authentication and Session Management

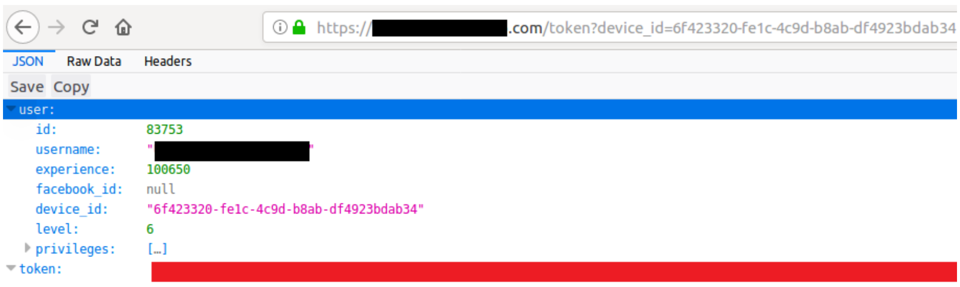
The role definitions are interesting. There are 7 levels of user privileges with increasing functionality for each level. Anyone can gain the highest privileges by contributing to the site. The lowest privilege can only view while the highest can delete materials and comments. You also gain points if other users credit your contribution. The idea is that untrusted users can contribute on some levels, while trusted users that have proven to contribute materials of high quality over a long period of time get an editor role. A user can register with Facebook or sign in as a guest user. No one can provision user accounts or delete them.

The log in either goes through Facebook or you get a guest user with no password. There is no sensitive information that is not already available for anyone through the guest user. Old tokens received at lower levels can be reused to perform actions, therefore the privileges are connected to the identity and not the token. The token is refreshed when the browser is closed and by clicking on the log-out button. However, the token does not expire and can still be used, hence the refreshing of the token has no purpose. Days after a token was generated I could still perform actions with it. Tokens in general should not live more than an hour. The log-out functionality does not work as the tokens are still active and it is possible to log back in as the same user without authentication. It is not possible to change to another user, even by deleting all cookies.

There is a limitation as to how many contributions one can make in one day. By deleting cookies it is still possible to get a new guest account. However, this restricts spamming and users are not able to pass through multiple levels in a short time period through spamming.

A4, A7 and A10 - Access Control and Redirects

There are many actions on the page that requires different levels of privileges. I tracked them using burp suite and then tried to resend them with the token belonging to a user with level 0. On all but one I got the response that I did not have the privileges to do that action. For the last one I was able to change a question. There were no unsafe redirects. When trying to use forced browsing without being logged in one is redirected to the log-in page. There are some insecure direct object references. The most severe is on the URI "/token?/device_id=..." where the token for a user is returned. If an attacker is able to guess or brute force the device_id it can steal the token and act as that user. Even though the device_id is a large random looking number it is much smaller than the token. On this page I also get a database error when entering the wrong device_id. I is later used when testing for SQL injection.



```
https://[redacted].com/token?device_id=6f423320-fe1c-4c9d-b8ab-df4923bdab34
JSON   Raw Data   Headers
Save  Copy
▼ user:
    id:            83753
    username:      "[redacted]"
    experience:    100650
    facebook_id:   null
    device_id:     "6f423320-fe1c-4c9d-b8ab-df4923bdab34"
    level:         6
  ▶ privileges:    […]
▼ token:           [redacted]
```

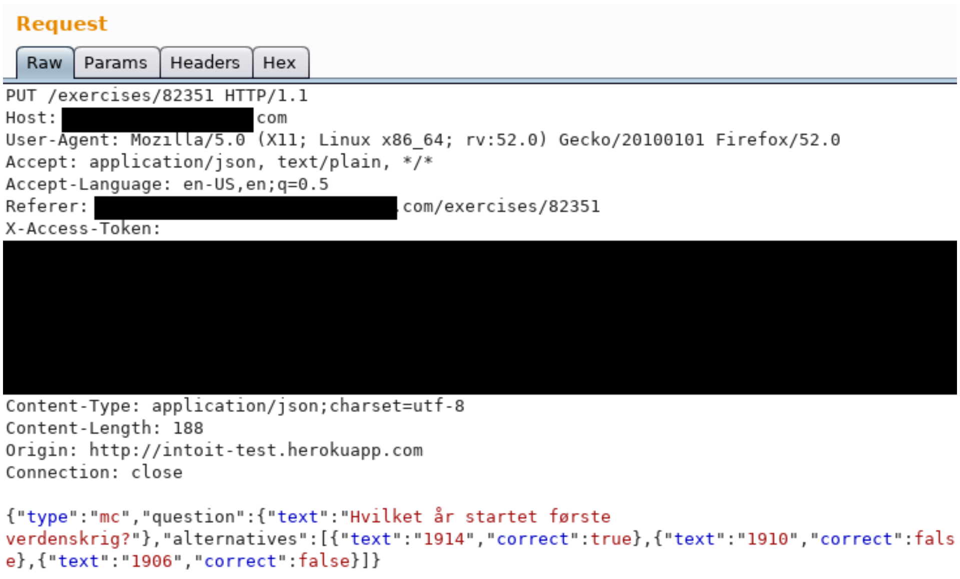
Figure 7.7: Insecure Direct Object References

A1 - Injection

I go back to the URI on "/token?/device_id=...". At the bottom there is an entry called query that writes out what the query should be as shown in figure 7.7. So i write "device_id=1' or '1' = '1" and a list of all users with their respective device_id is returned. I can then use the device_id to steal the token of any user and start doing mischief. By adding "AND level=7" I can filter on the users with the highest privileges.

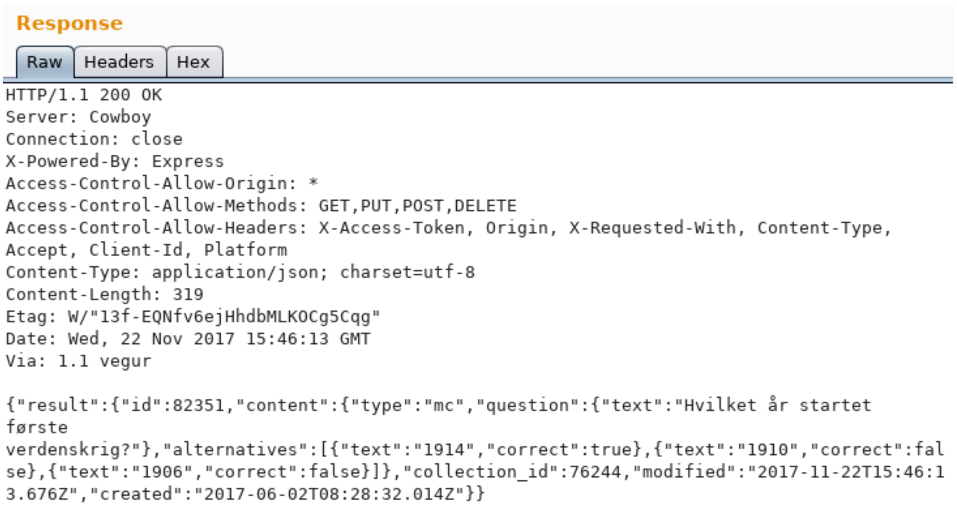
A3 - Cross-Site Scripting

The site does not use filtering but the user input is properly separated from code. I was not successful in performing any XSS attack.



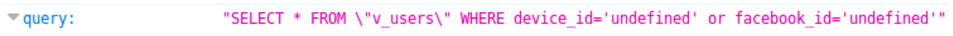
```
PUT /exercises/82351 HTTP/1.1
Host: [redacted]com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Referer: [redacted]com/exercises/82351
X-Access-Token:
[redacted]
Content-Type: application/json;charset=utf-8
Content-Length: 188
Origin: http://intoit-test.herokuapp.com
Connection: close

{"type":"mc","question":{"text":"Hvilket år startet første
verdenskrig?"},"alternatives":[{"text":"1914","correct":true},{"text":"1910","correct":fals
e},{"text":"1906","correct":false}]}
```

Figure 7.8: HTTP request for changing a question. The user privileges are too low to perform this action.



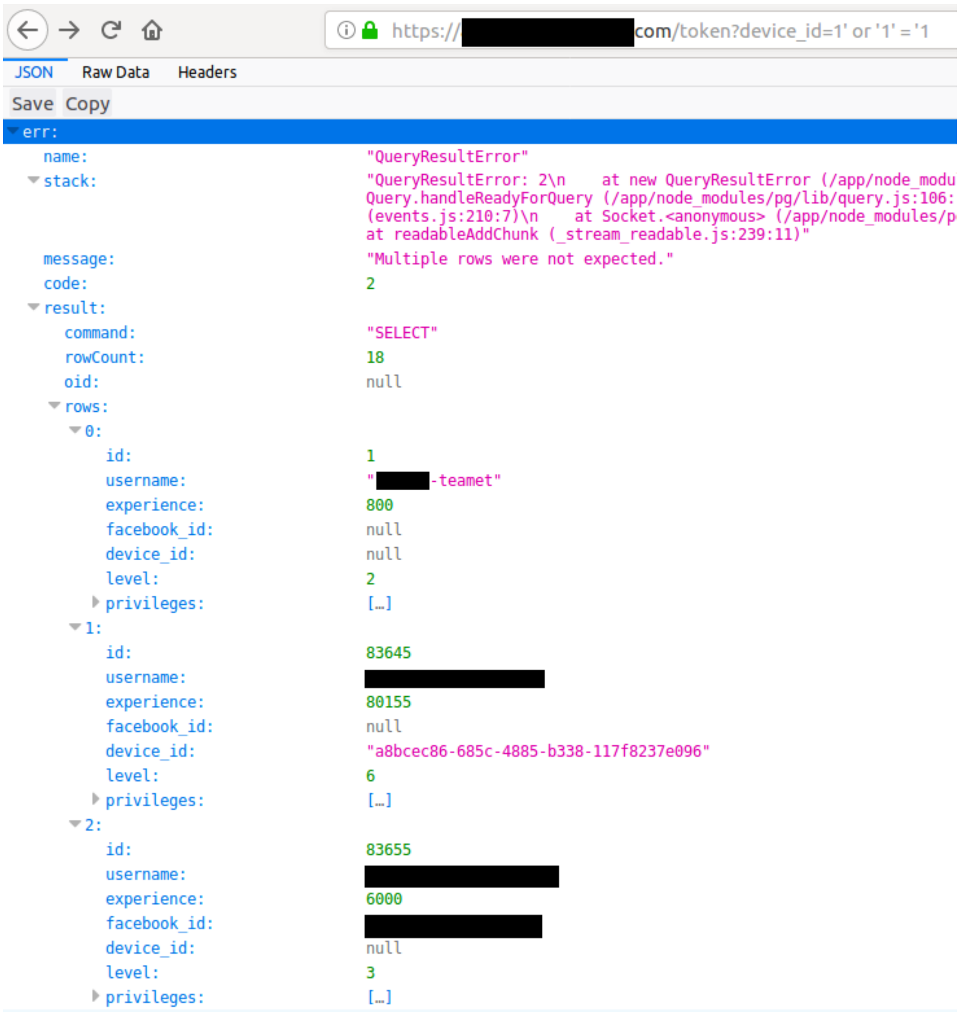
```
HTTP/1.1 200 OK
Server: Cowboy
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,PUT,POST,DELETE
Access-Control-Allow-Headers: X-Access-Token, Origin, X-Requested-With, Content-Type,
Accept, Client-Id, Platform
Content-Type: application/json; charset=utf-8
Content-Length: 319
Etag: W/"13f-EQNfv6ejHhdbMLKOCg5Cqg"
Date: Wed, 22 Nov 2017 15:46:13 GMT
Via: 1.1 vegur

{"result":{"id":82351,"content":{"type":"mc","question":{"text":"Hvilket år startet
første
verdenskrig?"},"alternatives":[{"text":"1914","correct":true},{"text":"1910","correct":fal
se},{"text":"1906","correct":false}]},"collection_id":76244,"modified":"2017-11-22T15:46:1
3.676Z","created":"2017-06-02T08:28:32.014Z"}}
```

Figure 7.9: HTTP response of the request in figure 7.8



```
▾query:          "SELECT * FROM \"v_users\" WHERE device_id='undefined' or facebook_id='undefined'"
```

Figure 7.10: Query entry



```
https://[redacted]com/token?device_id=1' or '1' = '1
JSON   Raw Data   Headers
Save  Copy
err:
  name:                         "QueryResultError"
  stack:                        "QueryResultError: 2\n    at new QueryResultError (/app/node_modu
                                Query.handleReadyForQuery (/app/node_modules/pg/lib/query.js:106:
                                (events.js:210:7)\n    at Socket.<anonymous> (/app/node_modules/p
                                at readableAddChunk (_stream_readable.js:239:11)"
  message:                      "Multiple rows were not expected."
  code:                         2
  result:
    command:                    "SELECT"
    rowCount:                   18
    oid:                        null
    rows:
      0:
        id:                     1
        username:               "[redacted]-teamet"
        experience:             800
        facebook_id:            null
        device_id:              null
        level:                  2
        privileges:             […]
      1:
        id:                     83645
        username:               [redacted]
        experience:             80155
        facebook_id:            null
        device_id:              "a8bcec86-685c-4885-b338-117f8237e096"
        level:                  6
        privileges:             […]
      2:
        id:                     83655
        username:               [redacted]
        experience:             6000
        facebook_id:            [redacted]
        device_id:              null
        level:                  3
        privileges:             […]
```

Figure 7.11: All user names are returned with device id

7.2.4 Findings: Company C

This subsection presents the detailed results from testing application C. A summary of the results can be found in table 7.4. The security on this site was rather high, and the vulnerabilities found seemed as though they might be intentional to improve user friendliness. My impression is that security was considered at all levels of the application.

A5, A6 and A9 - Configuration, Ciphers and Components

This web site is very clean and well organized. And the craftsmanship is to admire. The HTTP responses does not leak anything except the server and version of the server and all

Vulnerability	Result	Evaluation
A1	None found	Passed
A2	Token does not expire, log-out functionality does not work, token stored in cache	Failed
A3	None found	Passed
A4	None found	Passed
A5	Well configures	Passed
A6	Weakest crypto scheme has grade C	Passed
A7	None found	Passed
A8	Does not follow OWASP recommended best practice	Failed
A9	Automatically updates dependencies	Passed
A10	None found	Passed

Table 7.4: Results company C

the security headers are set properly. The server version is recent and a mainline version. The source code is very clean with no comments and no links to paths except the ones already known to the user. The code reveals that the application uses react and redux. The robots.txt file lists one entry, the sitemap.txt. The sitemap.txt lists all the URIs already visible to an unauthenticated user. An authentication token is checked for all GET requests and transactions and if it is missing the user is redirected to the log-in page. When writing document.cookie in console, nothing is returns. Nothing, no warnings or debugging info, is printed to console. The application uses PUT and DELETE HTTP methods, which in general should never be used, but because the API is RESTful this is not a misconfiguration.

I did not find any weaknesses for the server and I also know from the interviews that the application uses “Snyk” (Snyk.io, 2017) that automatically test the application for new vulnerabilities and updates dependencies.

The cryptosystems in use were all strong (A) except one that had grade (C). Strict-transport-security header is set in all responses.

A8 - Cross-Site Request Forgery (CSRF)

The referrer and origin headers were not checked and neither synchronization token or anti-CSRF token were present. The application uses a bearer token in the authorization header which is not the OWASP best practice. The token has a very long expiration date. After one week of testing it had still not expired and hence it is possible that it will never expire. If this is the case, the token can be used forever if it is leaked once. I could delete laundries and machines with a week long token even when the user was not logged in even if I removed both the Referrer and Origin headers in the HTTP response. I cannot rule out CSRF on this application.

A2 - - Authentication and Session Management

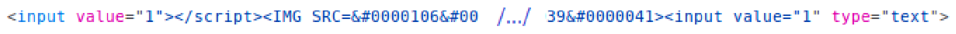
To authenticate on the site the user has to verify with either Facebook or Google. The user is authenticated using a authorization bearer (OAuth.2) for each response. This is always sent with the strict-transport-policy header set. However the token never expires, neither when i push the log-out button nor on timeout. I was able to delete machines and laundries with a week old token without being logged in. Also the cache-control header is not set when HTTP requests are sent to socket.application.com where the token is sent in the URI. This means that I could find valid authentication tokens in the cache of the browser and use these to forge transactions. Password policy is six digits, which is appropriate for this type of application but there is no lockout mechanism, hence brute force is possible. It is also possible to verify usernames by trying to reset the password for a given e-mail. You either get prompted that an e-mail is sent or that the username is not in use.

A4, A7 and A10 - Access Control and Redirects

There were no unsafe redirects. When trying to load pages or do transactions I were not authorized to, I got “404 nor found” or “403 forbidden”.

A1 and A3 - Injection and XSS

I did not get any database errors when playing around with user inputs. Likewise the site does filter any special characters from input field, but the input is correctly separated from the rest of the code.



```
<input value="1"></script><IMG SRC=&#0000106&#00  /.../  39&#0000041><input value="1" type="text">
```

Figure 7.12: User input is correctly separated from code

7.2.5 Findings: Company D

This subsection presents the detailed results from testing application D. A summary of the results can be found in table 7.5. The security was relatively high on this site, although some architectural flaws were found.

A5, A6 and A9 - Configuration, Ciphers and Components

The server is well configured using Nginx and Angular and I can not tell the version form any of my tests. The API is restful and Node Security Project is used to check and update all dependencies on every pull request. Therefore all versions are up to date. Security headers are set, like strict transport security an cache control. Neither cookies, source code nor error messages reveal anything.

Only secure crypto systems are used and the least strength had grade A. Further more the server discards all packets sent over HTTP.

There are some information leakage about other users, which is possible to get hold of if you have a valid token. See Insecure Direct Object References (A4) for more information.

Vulnerability	Result	Evaluation
A1	None found	Passed
A2	Token still active for one hour after log-out	Failed
A3	None found	Passed
A4	Get information about other users returned by manipulating parameters in URI	Failed
A5	All areas are well configured	Passed
A6	Information leakage about other users	Failed
A7	Can comment as another user	Failed
A8	Framework specific CSRF protection	Passed
A9	Use Node Security Project to keep all components up to date	Passed
A10	Description of findings	Passed

Table 7.5: Results company D

A8 - Cross-Site Request Forgery (CSRF)

Like the other companies they use a token but they do not check the Referrer or Origin headers. I find some calls to a script called CSRF.js and some Angular specific protection against CSRF appears to be in use. I assume that the site is not vulnerable to CSRF.

A2 - Authentication and Session Management

There are four types of users; superuser/admin, staff, employee and service receiver that all have access to different functions. All users can also be active or inactive. There are no user name enumeration even when recovering a password. Lock-out mechanism is strong with one minute delay per four wrong password tries. This is the only application that uses the recommended expiration time on the token of one hour. However, even after pressing the log-out button the token is valid for another hour.

A4, A7 and A10 - Access Control and redirects

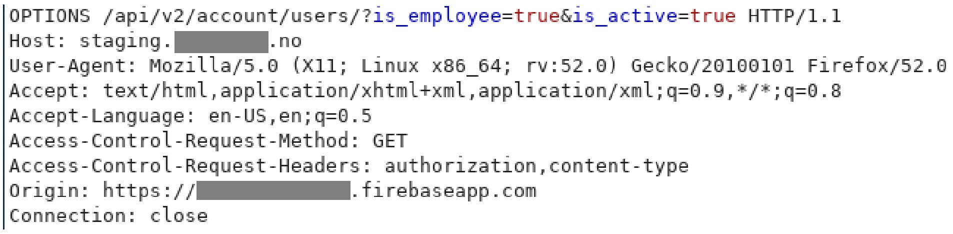
I found some Insecure Direct Object References where I could return information about different user groups by manipulating parameters in the URI. As seen in figure 7.13, by changing the parameters of is_employee and is_active I get information return about users as seen in figure 7.14 and figure 7.15.

When making a comment it is possible to change the "created_by" field and make comments as seen in figure 7.16. Even though I am logged in as "Lise" I can make comments as "Robert" and "Gamlefar".

No unsafe redirects or forwards were found.

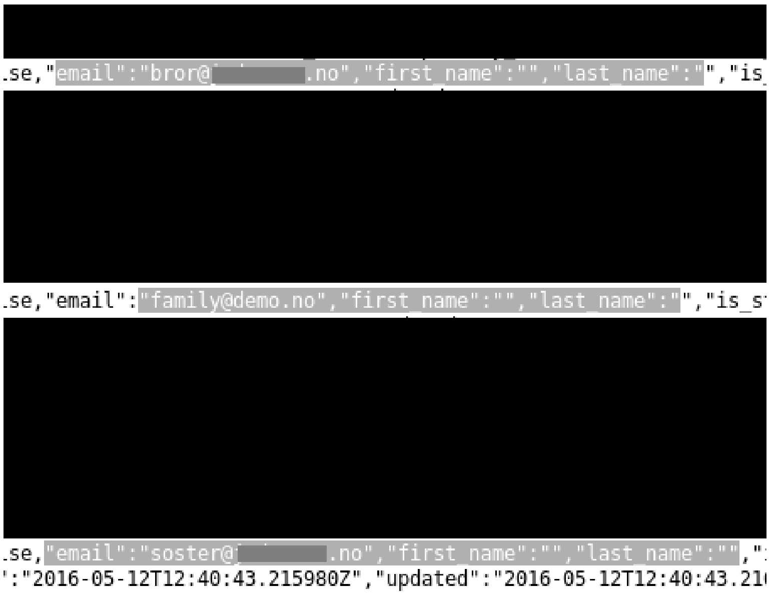
A1 and A3 - Injection and XSS

The application is well protected against SQL injection and XSS.



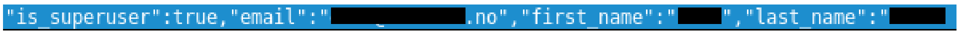
```
OPTIONS /api/v2/account/users/?is_employee=true&is_active=true HTTP/1.1
Host: staging.[redacted].no
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization,content-type
Origin: https://[redacted].firebaseapp.com
Connection: close
```

Figure 7.13: Object call



```
[redacted]
.se,"email":"bror@[redacted].no","first_name":"","last_name":"","is_
[redacted]
.se,"email":"family@demo.no","first_name":"","last_name":"","is_s
[redacted]
.se,"email":"soster@[redacted].no","first_name":"","last_name":"",":
':"2016-05-12T12:40:43.215980Z","updated":"2016-05-12T12:40:43.21
```

Figure 7.14: Information leakage about other users



```
"is_superuser":true,"email":"[redacted].no","first_name":"[redacted]","last_name":"[redacted]
```

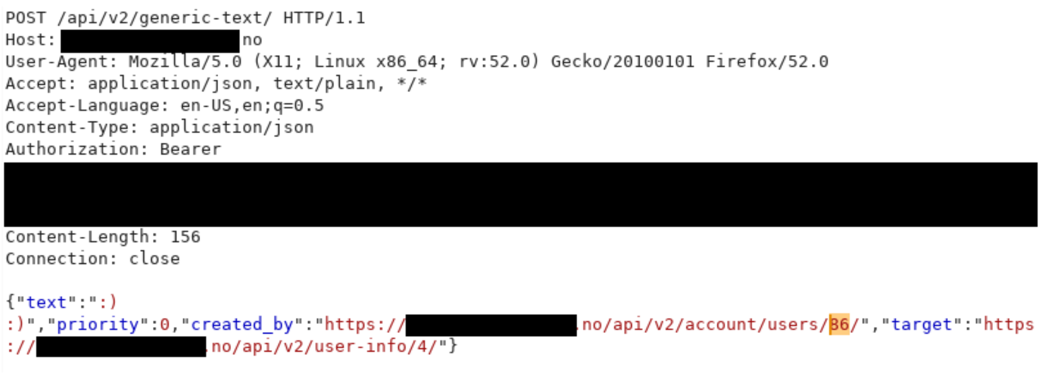
Figure 7.15: Information leakage of superuser

7.2.6 Findings: Company E

This subsection presents the detailed results from testing application E. A summary of the results can be found in table 7.6. Not many vulnerabilities were found for this application, and it relies heavily on third party code. My impression is that more in depth testing, would disclose more security flaws.

A5, A6 and A9 - Configuration, Ciphers and Components

From the HTTP response from the web server I can only determine that the server is LiteSpeed but not what kind of version it is. I can also find that WordPress (WP) is used



```
POST /api/v2/generic-text/ HTTP/1.1
Host: [redacted]no
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/json
Authorization: Bearer
[redacted]
Content-Length: 156
Connection: close

{"text":":)
:)","priority":0,"created_by":"https://[redacted].no/api/v2/account/users/86/","target":"https
://[redacted].no/api/v2/user-info/4/"}
```

Figure 7.16: Comment as another user

Vulnerability	Result	Evaluation
A1	None found	Passed
A2	Weak and inconsistent password policy, setting a password on the website does not work	Failed
A3	None found	Passed
A4	None found	Passed
A5	Not well configured when the target site is the host (as opposed to Google/Firebase,) insecure HTTP methods enabled on some URIs	Passed
A6	Insecure crypto schemes enabled on some ports	Failed
A7	None found	Passed
A8	Synchronized token	Passed
A9	No known vulnerabilities	Passed
A10	Could perform phishing attack	Failed

Table 7.6: Results company E

by looking at the URIs sent and paths in the source code. Other than that I find Jquery, bootstrap, stripe and some other libraries. The cookies are called AJS and I find that these cookies are used on other pages but not where they come from. On the first log-in page none of the security headers are set. This is also the only place where the URL of the website is set as the host in the HTTP request. As soon as I log-in, the URL of the site is only set as either Origin or Referrer but the host is some third party site and here all HTTP headers are set appropriately. This tells me that the system is well configured besed on secure third parties.

In the robots.txt file I find two entries, one allowed which shows an empty page, and one page called WP-admin which is disallowed. This last entry return the HTTP response "forbidden" and redirects to log-in. When spidering the site with Dirbuster I find a lot of pages in use and it seems to iterate endlessly and Dirbuster is not able to finish the whole search. Dirbuster finds a lot of empty WP directories and files.

The allowed HTTP methods are GET HEAD POST OPTIONS, but I do find some URL where also DELETE and PUT are enabled.

In the CVE database I cannot find any registered vulnerabilities for LiteSpeed Web Server. A number of vulnerabilities are registered for WordPress, the latest from 2017.

The web server only use cipher schemes with very high security port 433(security grade A). There are other ports open for IMAP, POP3 and SMPT with very low crypt systems enabled (security grade F).

The certificate for the start page shows a warning for mixed content but this is before log-in and before any credentials are present.

In console there is printed some debugging information and there is a content-security-policy warning.

A2 - Authentication and Session Management

The application uses Firebase provided by Google. Firebase is an application development platform developed on top of Google cloud platform. It provides a lot of ready to use functionalists and guides and samples of how to use it. All the authentication is handled by Google which makes it more secure. It is not possible to create a user without verification either through Google or Facebook. Log-out functionality and session timeout work as expected and the application is protected against brute force by blocking the device for 3 seconds by every 5th failed log-in attempt which will slow down an attack. Even when the session terminated, some of the cookies are still set, one of these store the name and email adress of the user and with this cookie set I can access the page HTTPs://application.no/user/user-id with a picture and some information about the user. I discovered this page by chance and were not able to invoke it again by putting other user-ids or even through the same link in the same browser.

The password policy is weak and inconsistent. 6 characters if you set the password through the application directly and 6 characters with at least one number if you reset the password through Google. Also with a failed log-in the application prompts you whether it is the password or the username that was incorrect. I can verify that the email of the founders are in use and also a “test@test.no”. Caching works as it should; no user content is cached. However, I can show most pages of the application without logging in, but without any user content. For instant it will state “welcome undefined” if I try to go straight to the dashboard.

Except for the authentication implemented though Firebase there are some really strange authentication mechanisms on the page that does not work properly at all. For instance, after first log-in (though Google or Facebook) I get a prompt to set a password but when I push the “save”-button nothing happens. The only way to set a password for the first time is to log out and use the “reset password” function provided by Google. Now it is possible to change my password on the dashboard, however if I enter a password that is not consistent with the password policy I am prompted to re-authenticate in order to reset my password. If I enter a password that is not consistent with the password policy nothing happens. Likewise if i enter the wrong credentials or simply close the re-authentication window, nothing happens. When I log on to the page using Google or Facebook I get an alert telling me that the email is badly formatted because the email field is empty and then I get logged in.

My impression is that this is an application that is saved by third party code and is secure because it relies on Google development tools. Clicking around on the Firebase platform I find that it is really user friendly and comprehensive and provides guides, samples and libraries in a really straightforward way.

A10 - invalidated redirects

Because the site uses Firebase and most critical actions go through Google it is hard to find an entry point for an attack, but one off the things the application has to handle are redirects. Whenever the host is Google or other third party products the redirects are secure, but when the application handles the redirection itself it will accept any URL provided by the client. In the attack the link below redirected the browser to Facebook.

```
HTTPs://application.firebaseapp.com/__/auth/handler?apiKey=
AIzaSyAUCJL0LMU-iDI3TC2DodURB_A-uptzmPE&appName=%5BDEFAULT%5D&
authType=signInViaRedirect&providerId=Google.com&scopes=profile&
redirectUrl=HTTPs%3A%2F%2FFacebook.com%2F%23%2F&v=3.9.0
```

this attack is dangerous because it can be used in phising attacks in order to fool the user into thinking that the link leads to the legitimate site when in fact the user is redirected to some other malicious site.

A1 - Injection

After clicking around on the page and entering ';' in all input fields and in the URL i go through all the HTTP request. I got no error messages. In the collection of HTTP responses most requests are sent to third parties like Google and Stripe. I doubt I would be able to do a SQL injection on Google or Stripe, apart from that fact, I am not authorized to try. The only requests to the target web site are GET requests to JavaScript files. It seems like the application never retrieves anything from their own database and most of the application is driven by JavaScript.

A3 - XSS

The application does not use filtering but entering JavaScript in the input field I can see in the HTML that my input is properly separated from the code.

A8 - CSRF

The only transaction on the page relevant for this attack is the changing password request, which goes directly to Google. In the changing password request there is a unique token that is changed for each request. However other requests to the site do not check the origin and referrer headers, but they do not seem to do any critical transactions. All critical transactions go through other sites and are secure.

A4 and A7

I did not find any vulnerabilities here. When trying to access pages without authenticating I am redirected to the log-in page. In some cases the site is loaded but without any information or “undefined” which is strange but not really a security problem.

Chapter 8

Analysis

In this chapter I will analyze the results. First section 8.1, evaluates each vulnerability found with OWASP risk rating methodology (Owasp.org, 2017c). Section 8.2 analyze the overall results.

8.1 Risk rating

The results create a picture of the state of security among the start-ups. As seen in table 7.1, some of the companies have many vulnerabilities while others have very few. Some vulnerabilities seem more critical than others. At first sight, using SQL injection to steal tokens for all users could appear to be a serious flaw. So does doing high privilege actions with low privileges. But, as mentioned before, there is a huge difference in the type and the sensitivity of the information present on each site. Therefore a vulnerability found in company A, D and E are a lot more severe than the same vulnerability found in company B or C.

Another thing to be taken into consideration is who the threat agents are. Needless to say it is far more likely that an exploit will take place if anyone at the internet has the opportunity do do so, as opposed to only authenticated users. Taken into account the technical skills of the threat agent might change the picture significantly. This is why for example, the scenario of a nurse exploiting the information leakage vulnerability in application D, where you need to set up a proxy to capture the HTTP packets and inspect them, is unlikely.

Other factors affecting the likelihood of an exploit is the trade-off between the reward and how easy the exploit is to achieve. If the reward is high an attacker might invest a lot in gaining it. If the exploit is fairly easy, however, someone might do it just for fun.

8.1.1 Evaluation of likelihood and impact

Below the OWASP risk rating methodology is used to evaluate the likelihood and the impact of each vulnerability. Table 8.1 and 8.2 rates the likelihood for each. table 8.3

and 8.4 rates the impact. The number before the dot in the ID stands for the company where A=1, B=2, C=3, D=4 and 5=E, and the number after the dot stands for the OWASP top ten vulnerability. The impact is rated based on the technical impact which is loss of confidentiality, integrity, availability and accountability, and business impact which is financial damage, reputations damage, non-compliance and privacy violations.

ID	Company	Description	Threat agent	Vulnerability	Likelihood
1.2	A	Weak account provisioning, weak password policy, no lockout mechanism, no session time out	9	8	8.5
1.5	A	Outdated framework, Missing security headers, insecure HTTP methods	9	8	8.5
1.6	A	Stict Transport Security not set, weak crypto systems, information leakage in source code	8	8	8
1.8	A	CSRF	8	6	7
1.9	A	Outdated framework	8	8	8
2.1	B	SQL injection used to return all tokens	7	8	7.5
2.2	B	No session timeout, no log-out	7	5	6
2.4	B	Change device_id to get other users info	7	7	7
2.5	B	Security headers not set, debugging info in console	7	7	7
2.6	B	Possible to send token over http	7	7	7

Table 8.1: Likelihood of each vulnerability

ID	Company	Description	Threat agent	Vulnerability	Likelihood
2.7	B	Can perform high privilege action with low privileges	7	6	🟥 6.5
2.8	B	CSRF	7	6	🟥 6.5
3.2	C	Token doesn't expire, token stored in cache, log-out functionality doesn't work	6	6	🟥 6
3.8	C	CSRF	7	6	🟥 6.5
4.2	D	Token still active after log-out	7	5	🟥 6
4.4/7	D	Information leakage about other users in HTTP response	5	5	🟨 5
4.8	D	Can comment as another user	5	5	🟨 5
5.2	E	Weak and inconsistent password policy, Setting password doesn't work	6	6	🟥 6
5.6	E	Insecure crypto schemes enabled on some ports	7	6	🟥 6.5
5.10	E	Phishing attack	9	7	🟥 8

Table 8.2: Likelihood of each vulnerability

ID	Company	Description	Technical	Business	Impact
1.2	A	Weak account provisioning, weak password policy, no lockout mechanism, no session time out	6	6	6
1.5	A	Outdated framework, Missing security headers, insecure HTTP methods	6	3	4.5
1.6	A	Stict Transport Security not set, weak crypto systems, information leakage in source code	5	3	4
1.8	A	CSRF	4	4	4
1.9	A	Outdated framework	7	2	5
2.1	B	SQL injection used to return all tokens	7	2	5
2.2	B	No session timeout, no log-out	7	2	5
2.4	B	Change device_id to get other users info	7	2	5
2.5	B	Security headers not set, debugging info in console	5	2	4
2.6	B	Possible to send token over http	5	2	4
2.7	B	Can perform high privilege action with low privileges	3	1	2
2.8	B	CSRF	4	2	3

Table 8.3: Impact of each vulnerability

ID	Company	Description	Technical	Business	Impact
3.2	C	Token doesn't expire, token stored in cache, log-out functionality doesn't work	4	2	3
3.8	C	CSRF	4	2	3
4.2	D	Token still active after log-out	6	5	5.5
4.4/7	D	Information leakage about other users in HTTP response	5	3	4
4.8	D	Can comment as another user	3	2	2.5
5.2	E	Weak and inconsistent password policy, Setting password doesn't work	5	4	4.5
5.6	E	Insecure crypto schemes enabled on some ports	4	2	3
5.10	E	phishing attack	5	2	** 3.5

Table 8.4: Impact of each vulnerability

ID	Company	Description	Likelihood	Impact	Severity
1.2	A	Weak account provisioning, weak password policy, no lockout mechanism, no session time out	8.5	6	7.3
1.5	A	Outdated framework, Missing security headers, insecure HTTP methods	8.5	4.5	6.5
1.6	A	Stict Transport Security not set, weak crypto systems, information leakage in source code	8	4.5	6.3
1.8	A	CSRF	7	4	5.5
1.9	A	Outdated framework	8	5	6.5
2.1	B	SQL injection used to return all tokens	7.5	5	6.3
2.2	B	No session timeout, no log-out	6	5	5.5
2.4	B	Change device_id to get other users info	7.5	5	6.3
2.5	B	Security headers not set, debugging info in console	7	4	5.5
2.6	B	Possible to send token over http	7	4	5.5
2.7	B	Can perform high privilege action with low privileges	6.5	2	4.3
2.8	B	CSRF	6.5	3	4.8

Table 8.5: Severity of each vulnerability

ID	Company	Description	Likelihood	Impact	Severity
3.2	C	Token doesn't expire, token stored in cache, log-out functionality doesn't work	6	3	4.5
3.8	C	CSRF	6.5	3	4.8
4.2	D	Token still active after log-out	6	5.5	5.8
4.4/7	D	Information leakage about other users in HTTP response	5	4	4.5
4.8	D	Can comment as another user	5	2.5	3.8
5.2	E	Weak and inconsistent password policy, Setting password doesn't work	6	4.5	5.3
5.6	E	Insecure crypto schemes enabled on some ports	6.5	3	4.8
5.10	E	Phishing attack	8	3.5	5.8

Table 8.6: Severity of each vulnerability

8.1.2 Evaluation of severity

In table 8.5 - 8.6 the severity for each vulnerability is given as the average between the Likelihood and the Impact. It is slightly different from the OWASP risk rating methodology which emphasizes business impact over technical impact. I have chosen to evaluate the severity in this way because I believe that it makes a more correct picture of the severity of each vulnerabilities. If the severity is 6 and above it is given the color red which indicates high risk, if it is 3 and above (but below 6) it is given the color yellow which indicates medium risk. The Severity table according to OWASP risk rating methodology can be found in the appendix.

In table 8.6 there are two vulnerabilities that almost qualify as high risk, 4.2 and 5.10.

Vulnerability 4.2 is the vulnerability that token is still active after pressing the log-out button. This should definitely be fixed as a user expect the session to be closed when logging out. When pressing the log-out button the user should be confident that no one can steal the session. However, the token is only active for one hour, giving an attacker a very short time window to perform an attack. In addition, because the attacker would need to get hold of the token, the only way to exploit this vulnerability would be to set up a proxy on the computer used by the victim before the session to sniff the traffic, then the attacker would need to get access to that same computer after the end of the session and compromise confidential information within one hour. This means that an attacker would need very specific access and use social engineering to perform the attack. Because this attack would be difficult to perform, I believe that a medium severity level is appropriate.

Vulnerability 5.10 is a phishing attack. The user thinks he/she will visit the target web site, but is instead redirected to a malicious site. The likelihood of this vulnerability is very high because it is a fairly easy attack to do technically and a very popular one. The business impact for this attack, however, is very low as it does not really affect the site in any way. Instead the tricked user is left with the consequences. Because of this I think that the severity should be considered high.

This leaves us with severity table 8.7.

8.2 Analyzing the results

On all applications there were significant security holes that needs to be fixed. Those companies with the lowest knowledge about OWASP also were the ones with the most critical security holes. Those who considered the security from the start had significantly better security.

Company A Company A had not focused on security and had prioritized functionality. Still they admitted that security breaches could lead to the end of their business and that they had multiple competitors that would be interested in taking over their market shares. Testing showed that their security architecture was faulty with multiple severe holes. Targeting a specific customer and gaining access to their accounts and even steal admin credentials did not seem at all too difficult.

Company B Even though application B does not contain any confidential information they were concerned about someone destroying material and spamming. The test-

ID	Company	Description	Severity
1.2	A	Weak account provisioning, weak password policy, no lockout mechanism, no session time out	High
1.5	A	Outdated framework, Missing security headers, insecure HTTP methods	High
1.6	A	Stict Transport Security not set, weak crypto systems, information leakage in source code	High
1.8	A	CSRF	Medium
1.9	A	Outdated framework	High
2.1	B	SQL injection used to return all tokens	High
2.2	B	No session timeout, no log-out	Medium
2.4	B	Change device_id to get other users info	High
2.5	B	Security headers not set, debugging info in console	Medium
2.6	B	Possible to send token over http	Medium
2.7	B	Can perform high privilege action with low privileges	Medium
2.8	B	CSRF	Medium
3.2	C	Token doesn't expire, token stored in cache, log-out functionality doesn't work	Medium
3.8	C	CSRF	Medium
4.2	D	Token still active after log-out	Medium
4.4/7	D	Information leakage about other users in HTTP response	Medium
4.8	D	Can comment as another user	Medium
5.2	E	Weak and inconsistent password policy, Setting password doesn't work	Medium
5.6	E	Insecure crypto schemes enabled on some ports	Medium
5.10	E	Phishing attack	High

Table 8.7: Final severity table

ing shows that even though there is not much to gain from this, it is so easy to do that someone might still just try. The developers were students with limited knowledge about OWASP and with only basic experience with web development. Their approach is "learning by doing". This is a good example that there are a lot of pitfalls when it comes to developing applications and that security needs special attention.

Company C Although application C does not contain any critical information, it is still one of the more secure. The application is very professionally made and very neat. The company has taken many measures to ensure security. They have been a little careless with the use of tokens which could in turn lead to someone deleting reservations or booking entire laundries. However, to steal a token the attacker still need access to the victims computer and a compromise would most likely be limited to only one individual. If an attacker were to get hold of a token once, they could be able use it forever.

Company D Application D contains very sensitive information and hence the security needs to be thereafter. The developers were very aware of OWASP and had taken measures to protect themselves right from the start. In fact they were very well protected in most ways although they had slipped up some places. The vulnerabilities here were caused by not considering the misuse cases of someone manipulating the HTTP request. The exploits for these vulnerabilities were limited in reach by the facts that only authenticated users could do them and because an attacker would have to be at a specific place at a specific time. Also, the account provisioning is strong. Nevertheless these are vulnerabilities of significant risk and need to be fixed.

Company E Application E in contrast to application C seemed very amateurish. There are multiple functions that do not work according to their purpose and the password policy is inconsistent. Simple functions like setting the password does not work at all. In spite of this the application is really secure. Looking at the HTTP packet capture almost none of the requests had the site as host. In their place were Google, Facebook, Stripe and other third parties. When inspecting the responses they were particularly well configured. So was their Firebase server.

When targeting the search to include only requests with the target application as host, I found an unsafe redirect that I could use to perform a phishing attack. In addition, though all critical transactions went through third party code, the less critical parts of the site was not configured to be very secure. I believe this illustrates that using secure third party code and sample code is a very good help for inexperienced developers wanting to make bullet proof applications.

The result of this thesis is consistent with the view that, regardless of the security needs of an application, developers who address security from the start make significantly more secure applications. This is illustrated by the findings in company A, B, C and D. A and B had very different security needs, but none of them had considered security in the development process. The two applications therefore showed a lot of security holes. In contrast we have application C and D. C had very low security needs and D had very high. They both were very up to date on OWASP and had implemented multiple measures to make secure applications. The applications showed few security holes and the vulnerabilities found were in general limited in reach.

The results also provide additional weight to the findings of Bau et al. (2012) that argue that start-ups make more secure applications because they are more motivated. Application A, which was the only application that was outsourced, was also the application with the most severe security holes, despite having high security needs.

Findings in application E also demonstrates that secure third party code and samples from large third party suppliers make a huge difference when smaller and slightly inexperienced companies develop applications. Whenever third party code was being used the security was exceptional. Third party material is a great asset to the overall security on the web.

8.3 Research questions revisited

This section will revisit the research questions and answer them based on the findings of this thesis. The research questions are:

1. Do Start-ups cover the basic needs when it comes to software security?
   (a) Is security a concern?
   (b) Do they have knowledge about common resources and tools?
   (c) Are their applications protected against OWASP top ten?
2. Are tools available that can help start-ups uncover vulnerabilities in a fast and easy way?

Is security a concern? How much the start-ups were concerned about security varied. Company C and D had clear goals for how to secure their application and could name a number of measures taken to maintain security, Company A and B agreed that security was important but did not appear to realize to what extent before after the interviews. Nor did Company A and B know if their applications were secure or not and they did not do anything to accomplish a secure application. After being in touch with many companies in regards to this thesis there is a clear division in attitudes towards security. Some companies were very eager to participate in the thesis, always replied promptly to e-mails and always provided me with additional information or documents needed. The four companies mentioned were all very grateful to have their application tested and asked for advice on how to move forward and improve their security. On the other hand, the major part of companies I talked to said that they did not have time, did not want to commit or were in general very hard to get in touch with. I was also in contact with several other companies who said that they wanted to participate but then did not have time to meet me or answer questions or I was not able to get in contact with them. The time required from the companies were ten minutes answering a questionnaire and 30 minutes for the interview. Possibly these companies were going through a hectic phase. Obviously it takes a lot of time and effort to start a business. However, because security requires special attention, if the attitude is "we don't have time" and "It will have to wait until later", likely the security will not be well preserved. As seen in the literature review, the reason many companies fail on security is because it is considered too expensive and not prioritized. Company E did not answer my e-mails for months and I had to send numerous reminders. In the end when I decided to exclude them, they finally responded. I decided to test them because I was curious to see if their attitude was reflected in some way in the security of the application. Because they did not show up for the interview we had scheduled, I am unable to conclude on how they approach security. Based on what I have seen during testing I am inclined to conclude they rely on third party code to secure the application and that they themselves do not know how to develop secure code. This is based on the observation that security was very high on all requests made to third parties and extremely low on request made to the target application.

Do they have knowledge about common resources and tools? None of the companies had done any prior testing of the security and hence did not use any testing tools for either static analysis or penetration testing. Company C and D both used dependency checkers and framework specific defenses to secure their applications. They also expressed adequate knowledge about OWASP top ten and company D had even done a risk assessment. Company D were also in contact with several institutions giving advise and contributing their security. Company A and B had rudimentary knowledge about OWASP. All used third party code to handle specifically sensitive transactions like log-in and credit card information. Company E used Firebase that really helped them securing their application. This is one helpful tool in achieving secure applications.

Are their applications protected against OWASP top ten? All the application contained at least one vulnerability, although their severity differed. Overall I am inclined to conclude that both application C and D passed the OWASP top ten penetration test as they had no high risk vulnerabilities. That said, both have vulnerabilities that need to be addressed. Specifically the information leakages and the missing function level access control in application D are not insignificant. However, in all applications there will always be vulnerabilities, and what is important to consider is the risk connected to them and whether the specific security needs of that application is covered. In my opinion security in application C and D is maintained.

Application A and B were not well protected against OWASP top ten with several high severity vulnerabilities. That said, I still feel safe using application B as it does not contain any security holes that would affect me as a user, as I do not need to provide any sensitive information about myself. If someone decided to take down application B, they probably would succeed. Although all the critical transactions on application E were handled in a secure way, they still had one high severity vulnerability present on the page.

Do Start-ups cover the basic needs when it comes to software security? Considering whether the basic security needs are covered comes down to whether they are protected against OWASP top ten or not. My impression is that Application C and D definitely cover the basic needs required. There is compliance between the security needs of the application and the actual security obtained. Application A and B did not provide even the most basic security needs. They are not well protected against basic attacks such as brute forcing and SQL injection which would be the first any person with little security knowledge would try. Application E is well protected against most vulnerabilities and all critical transactions and sensitive information are handled in a secure way. Still I would not feel safe uploading my private documents on this site, as I am not confident that they would be handled in a secure way throughout the application. They rely too much on third party code and the performance of the application does not demonstrate any knowledge or effort to secure the information. They do not convince me that the information is handled in a secure way, but simply assume that it is because they are using third party code. I also believe a more thorough penetration test not limited to OWASP top ten or the time constraints, might reveal more security flaws.

Are tools available that can help start-ups uncover vulnerabilities in a fast and easy way? There definitely are a lot of tools that start-ups can use to help them improve security that are both free and easy to use. Snyk and Node Security Project are two examples of dependency checkers that help maintain up to date components in an automatic way. Other dependency checkers exist for frameworks not covered by these two. Furthermore, I was generally impressed by Firebase, both on the security level maintained and how user friendly it was. Third party code that will handle financial transactions, log-in and storage of sensitive information are priceless for inexperienced developers wanting to develop secure applications.

As discussed in chapter 5.2, Kali Linux comes with a wide range of security testing tools. There is only one thing to download and set up, and it is free. It is very widely used, and there are numerous tutorials online. In Kali Linux the specific tools found useful were, Burp Suite, openSSL, Whatweb, Nikto and Nmap. How they were used is explained in chapter 7.1. Static analysis tools were not used in this thesis, but are helpful for developers to quickly review code. I believe that at least the SQL injection vulnerability could have been prevented if a static analysis tool had been used.

Chapter 9

Conclusion

Don't stop at 10. There are hundreds of issues that could affect the overall security of a web application

OWASP top ten 2017

This thesis performed penetration testing on applications made by five startup companies. The test plan was limited to OWASP top ten and a numbers of tests from the OWASP Testing Guide were selected for each vulnerability mentioned. Although they all contained significant vulnerabilities, considering these together with the business context revealed that 2 out of 5 had maintained adequate security. Application C and D had no high risk vulnerabilities. Also in 3 out of 5 the user interests were adequately maintained, even though application B had some serious security flaws the vulnerabilities would not lead to loss for any of the users. For company E there was only one high risk vulnerability found where the application could be used as a stepping stone for launching a phishing attack. Although not posing a threat to the business goals of the company, this security flaw could have great consequences for innocent users.

None of the companies had performed any prior security tests and they did not follow formal methods for implementing security. However, company D had done risk analysis and prepared a document discussing how to prevent OWASP top ten.

All of the applications were using third party code to handle critical and sensitive transactions. This helps maintaining security of the most sensitive parts of the applications. Also the companies were using other types of tools that increased security. Examples are, dependency checkers, Firebase and framework specific defences. Although third party code and security tools will help startups avoid common mistakes, they still have a responsibility to consider the whole system and ensure that security is maintained at all stages of the application. They need to ensure full coverage.

Looking at OWASP top ten only scratches the surface of the most basic security needs, and to really be confident in the result more comprehensive testing has to be performed. Also the disadvantage with penetration testing is that even if a vulnerability is not disclosed, this is no guarantee that it is not there. Other testing techniques and activities should be applied to increase security.

None of the startups were using a systematic approach to ensure security and the measures being done seemed somewhat arbitrary. This is not hard to imagine, as startups need to rely on their own knowledge and because they do not have dedicated security staff.

There is always a trade off between usability and security, and between cost and risk. One of the founders of company D said: “If you are afraid to drown, don’t swim”. An application will never be completely free of security flaws. It is also obvious that when a company is new it cannot be expected to have the experience and routines of an established company. A developer just learning how to code, cannot be expected to have the knowledge of a security expert. What is most crucial is that the startups need to be aware that security must be handled and what the consequences are if they are not. After only 30 minutes of talking to them about potential vulnerabilities and risks, they seemed to have whole new perspective. There are a number of helpful resources and tools online free of charge. It will, however, take time, effort and experience to succeed.

Bibliography

Acunetix.com, 2017. CSRF attacks, XSRF or Sea-Surf. Available online at: https://www.acunetix.com/websitesecurity/csrf-attacks/, Last accessed: 2017-11-14.

Austin, A., Williams, L., 2011. One technique is not enough: A comparison of vulnerability discovery techniques. In: Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM 2011, Banff, AB, Canada, September 22-23, 2011. IEEE Computer Society, pp. 97–106.
URL https://doi.org/10.1109/ESEM.2011.18

Bau, J., Wang, F., Bursztein, E., Mutchler, P., Mitchell, J. C., 2012. Vulnerability factors in new web applications: Audit tools, developer selection & languages. Tech. rep., Citeseer.

Bilge, L., Dumitras, T., 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In: Yu, T., Danezis, G., Gligor, V. D. (Eds.), the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012. ACM, pp. 833–844.
URL http://doi.acm.org/10.1145/2382196.2382284

BSIMM.com, 2017. About the BSIMM. Available online at: https://www.bsimm.com/about.html, last accessed: 2017-12-25.

CVEdetails.com, 2017. CVE details. Available online at https://www.cvedetails.com/ last accessed 2018-01-08.

Difi.no, 2017. Ulike sikkerhetsniv. Available online at: "http://eid.difi.no/nb/sikkerhet-og-informasjonskapsler/ulike-sikkerhetsniva", Last accessed: 2017-10-24.

Edwards, D., 2017. Devops: Shift left with continuous testing by using automation and virtualization? Available online at: https://www.ibm.com/cloud/garage/experience/deliver/dibbe_edwards_devops_shift_left/, Last accessed: 2018-01-08.

Geer, D., 2010. Are companies actually using secure development life cycles? IEEE Computer 43 (6), 12–16.
URL https://doi.org/10.1109/MC.2010.159

Howard, M., Lipner, S., 2003. Inside the Windows security push. IEEE Security & Privacy 1 (1), 57–61.
URL https://doi.org/10.1109/MSECP.2003.1176996

Investorpedia.com, 2007. Startup. Available online at: http://www.investopedia.com/terms/s/startup.asp, Last accessed: 2016-10-20.

Jaatun, M. G., Cruzes, D. S., Bernsmed, K., Tøndel, I. A., Røstad, L., 2015. Software security maturity in public organisations. In: Lopez, J., Mitchell, C. J. (Eds.), Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings. Vol. 9290 of Lecture Notes in Computer Science. Springer, pp. 120–138.
URL https://doi.org/10.1007/978-3-319-23318-5_7

Kali.org, 2017a. Dirbuster. Available online at https://tools.kali.org/web-applications/dirbuster last accessed 2018-01-08.

Kali.org, 2017b. Kali linux. Available online at https://www.kali.org/ last accessed 2018-01-08.

Keary, E., 2014. OWASP Testing guide - foreword. OWASP Foundation.

Lysne, O., Beitland, K., Hagen, J., Holmgren, A., Lunde, E., Gjøsteen, K., Manne, F., Jarbekk, E., Nystrøm, S., 2015. Digital sårbarhet - sikkert samfunn. Tech. rep., Norges offentlige utredninger, available online at: https://www.regjeringen.no/contentassets/fe88e9ea8a354bd1b63bc0022469f644/no/pdfs/nou201520150013000dddpdfs.pdf, last accessed: 2018-01-10.

Manico, J., Righetto, D., Krawczyk, P., Dhiraj, M., Kulkarn, S., Gigler, T., Coates, M., Williams, J., Wichers, D., Wall, K., Walton, J., Sheridan, E., Kenan, K., Rook, D., Donovan, F., Kang, A., Ferguson, D., Shah, S., Siles, R., Watson, C., Matatall, N., 2017. OWASP cheat sheets. Available online at https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series last accessed 2018-01-08.

McGraw, G., 05 1998. Testing for security during development: Why we should scrap penetrate-and-patch. Aerospace and Electronic Systems Magazine, IEEE 13, 13 – 15.

McGraw, G., 2003. From the ground up: The DIMACS software security workshop. IEEE Security & Privacy 1 (2), 59–66.
URL https://doi.org/10.1109/MSECP.2003.1193213

McGraw, G., 2004. Software security. IEEE Security & Privacy 2 (2), 80–83.
URL https://doi.org/10.1109/MSECP.2004.1281254

McGraw, G., 2006. Software security: Building security in. In: 17th International Symposium on Software Reliability Engineering (ISSRE 2006), 7-10 November 2006, Raleigh, North Carolina, USA. IEEE Computer Society, p. 6.
URL https://doi.org/10.1109/ISSRE.2006.43

McGraw, G., 2009. Software security touchpoint: Architectural risk analysis. Available online at: https://www.cigital.com/presentations/ARA10.pdf, last accessed: 2018-01-7.

McGraw, G., 2016. Four software security findings. IEEE Computer 49 (1), 84–87.
URL https://doi.org/10.1109/MC.2016.30

Meucci, M., Muller, A., 2014. OWASP Testing guide. OWASP Foundation.

Michael B. Jones, D. H., 2012. Oauth 2.0 bearer token usage. Available online at: https://tools.ietf.org/html/rfc6750, Last accessed: 2017-11-14.

Morningstarsecurity.com, 2017. Whatweb. Available online at https://www.morningstarsecurity.com/research/whatweb last accessed 2018-01-08.

Nicolaysen, T., Sasson, R., Line, M. B., Jaatun, M. G., 2010. Agile software development: The straight and narrow path to secure software? IJSSE 1 (3), 71–85.
URL https://doi.org/10.4018/jsse.2010070105

Nmap.org, 2017a. Ncat. Available online at https://nmap.org/ncat/ last accessed 2018-01-08.

Nmap.org, 2017b. Nmap. Available online at https://nmap.org/ last accessed 2018-01-08.

Openssl.org, 2017. openSSL. Available online at https://www.openssl.org/ last accessed 2018-01-08.

Owasp.org, 2013. Top 10 2013. Available online at: https://www.owasp.org/index.php/Top_10_2013-Top_10, Last accessed: 2016-10-13.

Owasp.org, 2017a. Cross-Site Request Forgery (CSRF). Available online at: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF), Last accessed: 2017-12-16.

Owasp.org, 2017b. Cross-Site Request Forgery (CSRF) prevention cheat sheet.

Owasp.org, 2017c. OWASP Risk Rating Methodology. Available online at: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology#Step_1:_Identifying_a_Risk, Last accessed: 2016-10-13.

Owasp.org, 2017d. Testing for CSRF (otg-sess-005). Available online at: https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005), Last accessed: 2017-12-16.

Portswigger.net, 2017. Download Burp Suite community edition. Available online at https://portswigger.net/burp/communitydownload last accessed 2018-01-08.

Ranum, M., Sept 2016. Silver bullet talks with Gary McGraw. IEEE Security Privacy 14 (5), 7–10.

Robehmed, N., 2013. What is a startup? Available online at: http://www.forbes.com/sites/natalierobehmed/2013/12/16/what-is-a-startup/#4a7d139e4c63, Last accessed: 2016-10-20.

Sectools.org, 2017. Nikto. Available online at http://sectools.org/tool/nikto/ last accessed 2018-01-08.

Snyk.io, 2017. Product information. Available online at: https://snyk.io/features, Last accessed: 2017-11-14.

Søhoel, H., 2016. Pre-project: OWASP top ten - what is the state of practice among start-ups? Tech. rep., Norwegian University of Science and Technology.

Viega, J., McGraw, G., 2001. Building Secure Software: How to Avoid Security Problems the Right Way. Addison-Wesley.

York, K., 2016. Dyn statement on 10/21/2016 DDoS attack. Available online at: https://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/, last accessed: 2017-11-27.

Appendix A

Severity according to OWASP Risk Rating Methodology

According to the OWASP Risk rating methodology the business impact is to be emphasized over technical impact when evaluating the overall impact. Table A.1 - A.2 indicate the overall severity for each vulnerability when following the OWASP risk rating methodology to the point. All vulnerabilities for company B and C are medium because the business impact is low. Company A and E both have one critical vulnerability, while company D has one high vulnerability, one medium and one low.

ID	Company	Description	Likelihood	Impact	Severity
1.2	A	Weak account provisioning, weak password policy, no lockout mechanism, no session time out	High	🟥	critical
1.5	A	Outdated framework, Missing security headers, insecure HTTP methods	High	Medium	High
1.6	A	Stict Transport Security not set, weak crypto systems, information leakage in source code	High	Low	Medium
1.8	A	CSRF	High	Medium	High
1.9	A	Outdated framework	High	Low	Medium
2.1	B	SQL injection used to return all tokens	High	Low	Medium
2.2	B	No session timeout, no log-out	High	Low	Medium
2.4	B	Change device_id to get other users info	High	Low	Medium
2.5	B	Security headers not set, debugging info in console	High	Low	Medium
2.6	B	Possible to send token over http	High	Low	Medium
2.7	B	Can perform high privilege action with low privileges	High	Low	Medium
2.8	B	CSRF	High	Low	Medium

Table A.1: Severity of each vulnerability

ID	Company	Description	Likelihood	Impact	Severity
3.2	C	Token doesn't expire, token stored in cache, log-out functionality doesn't work	High	Low	Medium
3.8	C	CSRF	High	Low	Medium
4.2	D	Token still active after log-out	High	Medium	High
4.4/7	D	Information leakage about other users in HTTP response	Medium	Medium	Medium
4.8	D	Can comment as another user	Medium	Low	Low
5.2	E	Weak and inconsistent password policy, Setting password doesn't work	High	Medium	High
5.6	E	Insecure crypto schemes enabled on some ports	High	Low	Medium
5.10	E	phishing attack	High	High	Critical

Table A.2: Severity of each vulnerability