Examensarbete

Investigating the current state of security for small sized web applications

av

Karl Johan Lundberg

LIU-IDA/LITH-EX-A--12/072--SE

2013-01-22
Examensarbete

Investigating the current state of security for small sized web applications

av

Karl Johan Lundberg

LIU-IDA/LITH-EX-A--12/072--SE

2013-01-22

Handledare: Dag Helstad, Anders Fröberg
Examinator: Erik Berglund
Abstract

It is not uncommon to read about hacker attacks in the newspaper today. The hackers are targeting governments and enterprises, and motives vary. It may be political or economic reasons, or just to gain reputation. News about smaller systems is, unsurprisingly, not as common. Does this mean that security is less relevant of smaller systems? This report investigates the threat model of smaller web applications, to answer that very question.

Different attacks are described in the detail needed for explaining their threat but the intention is not to teach the reader to write secure code. The report does, however, provide the reader with a rich source of references for that purpose. After describing some of the worst threats, the general cloud threat model is analyzed. This is followed by a practical analysis of a cloud system, and the report is closed with general strategies for countering threats.

The severe destruction that a successful attack may cause and the high prevalence of those attacks motivates some security practices to be performed whenever software is produced. Attacks against smaller companies are more common now than ever before.
# Table of Contents

Abstract .......................................................................................................................... 3

Table of Contents .......................................................................................................... 4

1. Introduction .................................................................................................................. 7
    1.1 Background .............................................................................................................. 7
    1.2 Objective ................................................................................................................ 7
    1.3 Scope ...................................................................................................................... 7
    1.4 Intended Audience ................................................................................................. 8
    1.5 Thesis Overview .................................................................................................... 8
    1.6 Business Context ................................................................................................. 9

2. Network Terminology ................................................................................................. 10
    2.1 SQL ...................................................................................................................... 10
    2.2 Cookies ................................................................................................................ 10
    2.3 Certificate ............................................................................................................ 11
    2.4 Web Proxy ........................................................................................................... 11
    2.5 HTTP ................................................................................................................... 11
        2.5.1 Referer Header ............................................................................................... 11
        2.5.2 Connection ................................................................................................... 12
        2.5.3 GET vs. POST ............................................................................................... 12
    2.6 SSL and HTTPS ................................................................................................... 12
    2.7 HTML .................................................................................................................. 13
    2.8 DOM .................................................................................................................... 14
    2.9 JavaScript ............................................................................................................ 14

3. Common Network Threats ......................................................................................... 15
    3.1 Command Executions ........................................................................................... 15
        3.1.1 SQL Injection ................................................................................................. 15
        3.1.2 XPath Injection ............................................................................................ 16
    3.2 Client-Side Attacks ............................................................................................... 16
        3.2.1 Cross-Site Scripting ....................................................................................... 16
            3.2.1.1 Reflexive vs. Persistent XSS ............................................................... 17
            3.2.1.2 DOM Based vs. Traditional XSS ...................................................... 17
            3.2.1.3 Sanitization .......................................................................................... 17
        3.2.2 Cross-Site Request Forgery ........................................................................... 18
            3.2.2.1 Login XSRF ....................................................................................... 18
            3.2.2.2 Defenses .............................................................................................. 19
        3.2.3 Differences between XSS and XSRF ............................................................. 20
    3.3 Session Attacks ..................................................................................................... 21
        3.3.1 Cookie Guessing Attacks ............................................................................. 21
        3.3.2 Cookie Eavesdropping ................................................................................. 21
1. Introduction

1.1 Background
Internet has grown rapidly over the few last decades, showing its huge potential to the public. When the use of the Internet exploded back in the 90’s, people’s intuition would be that their own data were stored locally on the disc and their programs run locally on their computer.

Today, cloud applications are giving us new opportunities of storage. Available services do not only provide its customers with physical space but also flexibility. As the Internet has grown it has become a natural part of people’s lives. People are nowadays used to bring connected devices everywhere, so cloud services have brought us the luxury of carrying our software with us everywhere.

Flexibility, however, comes at a price. As authorized users easily may connect to their cloud system, so does the malicious attacker. In newspapers are we used to read about hackers stealing accounts on social networks or shutting down big sites with DoS attacks.

But the victims we read about in the news tend to be huge companies so those articles often feels... remote. No one would attack a smaller project, no one would benefit from that. We do not need to spend time testing for security vulnerabilities. Or do we?

1.2 Objective
This report intends to answer the following questions:
- Which are the greatest threats to today’s web applications?
- What is the general threat scenario of smaller cloud applications?
- Is risk management necessary and feasible for smaller projects?

Hopefully, it will give the reader a picture of the current security state of smaller cloud systems. Its intention is to encourage secure development of new cloud applications, but also make the reader aware of the tradeoffs.

1.3 Scope
As hinted, this report will not cover security aspects of big, well-known cloud applications intentionally. Neither will it cover application-specific threats that are not characteristic for cloud applications. Those threats may be mentioned as examples of a more general phenomenon, though. The evaluation will be from a user’s perspective, who in the end is the potential victim to security vulnerabilities.
The report will focus on the more technical threats that are generally encountered in the cloud, such as SQL injections and XSS, to give the reader a picture of common traps developers tend to fall into. It will also mention organizational threats, which for example include malicious insiders, and highlight those as a concern. Businesses has to consider organizational threats whether they are building cloud applications or not, but their cloud systems must be implemented according to the given circumstances.

The intention of this report is not to teach the reader proper defenses against threats, correct strategic priorities or solid development. There are already a wide range of sources that explains those matters on the web (see references).

1.4 Intended Audience
The reader is assumed to be familiar with abbreviations like IP, SQL, HTTP, etc. and have some knowledge of how the Internet works. Prior knowledge about computer security, cloud mechanisms, or expertise in networking is not required.

1.5 Thesis Overview
An overview of the rest of the chapters in this thesis:

Chapter two consists of brief descriptions of some important components in network systems. This chapter is only supposed to support the following chapters and if the reader is already confident in his or her knowledge of networking, this chapter may be skipped.

Chapter three will go through a range of common attack patterns. The intention is to present the most fatal risks on the web, where fatality is determined by sites such as OWASP, McAfee and WASC. These attack patterns will be described in such a detail that would give the reader an idea of their impact and why they are common. This report will explain why security is a concern and what the traps look like. Those threats are just a selection of all risks that networking brings. They have been divided into groups, influenced by WASC’s Threat classification from 2004. While this is one of the oldest source used in this report, the author found that their classification explained the variety of threats clearly.

Chapter four consists of a general threat model of modern cloud systems in general. This will follow the recommendations of CSA. This chapter aims to put the threats given in chapter four in a cloud context. From this chapter, the reader will be able to prioritize risks for their own risk analysis regarding the current state of cloud security.

Chapter five consists of a practical example of an authentic cloud system that will be analyzed accordingly. This system is black box tested, then tested again with a static analysis tool. The results from the static analysis are being investigated and results will follow from all these tests.

Chapter six will briefly explain general strategies for secure developing. This chapter does not deal with any specific threat; its intention is to present ways in which threats in general may be dealt with.
1.6 Business Context

Symantec reveals in their trend report from 2011 that about 50 % of targeted attacks is directed towards companies with 2500 or less employees. About a third of those attacks hit small companies, with less than 250 employees. Targeted attacks increased 2011 and Symantec speculates that attacks on smaller organizations will increase further, as they may be used as a stepping stone to their bigger partners. Another trend is that those attacks that previously used to intrude via the CEO now have found other paths to the company as well, like the HR or Executive Assistants. In 2011, around 37 % of global businesses were adopting cloud solutions. Symantec then predicted that cloud usage would increase in the near future and that cloud technology may change the way business are being made. This could force new ways to protect the users and corporate systems [50].

A common error made among smaller companies is to believe that cybercriminals only targets big enterprises. In a survey from the National Cyber Security Alliance and Visa Inc., with around 1000 answers from smaller companies, 84 % answered that they are more prepared to an attack than big enterprises [34]. In a study made by Better Business Bureaus, 7.4 % of the asked US small business claimed that they had been victims from some kind of fraud [45].
2. Network Terminology

2.1 SQL

Structured Query Language (SQL) is a dedicated database programming language used to extract data in an easy manner. An example of SQL code may look like follows:

```
SELECT *
FROM Books
WHERE price < 100
ORDER BY title;
```

This code assumes that there is a table called ‘Books’ with the columns ‘price’ and ‘title’ in the current database. It will select all rows in Books that costs less than 100 and order them by title [49].

```
SELECT *
FROM Books
WHERE title LIKE '%s%'
```

Another book query. Instead of looking for cheaper books, this code searches for titles containing a ‘s’. The % character is called a wildcard and represents any non-empty string. The code above will return rows containing titles like “English Dictionary” or “Computer Security” but not “Advanced Math’s” or “Shantaram” [46].

2.2 Cookies

Cookies are small chunks of data that contains a user’s activity on a web page. This data is stored in the browser during the user session and sent to the website. Next time the user opens the page, it will retrieve the cookie from the browser. Cookies were first used for a reliable way to remember user states and activities in the past. One specific example is using cookies for authentication, which basically remembers whether a user is logged in or not. Although they cannot contain viruses or run malware on the user’s computer, they are often a big security concern.

Session cookies are deleted when the browser is closed, while persistent cookies last for months. The general properties of cookies are configured with flags, carried within. The secure flag confirms that traffic should be encrypted through TLS and the HttpOnly flag enforces pure HTTP (or HTTPS) traffic, preventing malicious JavaScript to run [49].
2.3 Certificate

A certificate is an electronic document that binds a public key to an individual. The certificate has to be signed by a certificate authority to be valid and contains a variety of data like validation time, validator data and signature, subject data and a signature algorithm. Certificates are managed and distributed by public key infrastructures (PKI) [49].

Assume that two individuals want to communicate for the first time. One of them wants to send a confidential message and keys are exchanged according to some method, say Diffie-Hellman. With certificates, entities may authenticate to each other, via a third party, in which both trusts. This trusted party is known as the Certificate Authority (CA) and is responsible for issuing certificates to users. Before creating a new certificate, the CA checks with a Registration Authority (RA) to verify the information sent from the certificate requester [1].

2.4 Web Proxy

A web proxy is an entity that acts as an intermediary between a client and a server. That is, the server treats the proxy as a client and the client treats the proxy as a server. There are several uses for proxies; filtering, directing, etc. A Forward Proxy is attached to the client and may be used as a firewall, in contrast, a Reversal Proxy is attached to the server and commonly used for DNS-calls, filtering and directing. There is also a third kind, the Open Proxy. An open proxy is not attached to a main computer or system, but acts as a server between anywhere on the net and anywhere on the net. It is often used as an anonymity server, which hides the users IP-address [49].

2.5 HTTP

HTTP is an application layer protocol for client-server structured networking. It is based on the idea that clients requests pages from servers, which respond appropriately. Currently, HTTP is running in version 1.1.

A HTTP request contains a request method, headers, and an optional message body. A mandatory header in all HTTP requests is the Host header, which specify target server domain name and port number.

2.5.1 Referer Header

The HTTP referer header contains the URL that initiated the request. Note the difference between the actual computer which sent the request and the referer header: Assume that a user is searching for ‘kittens’ at Google and clicks on the first link that pops up. The referer header will then contain the first page of Google’s search result over ‘kittens’. Also, note that the spelling “referer” is correct in this context.
2.5.2 Connection

Another important header is Connection. It may be used with the argument “Keep-Alive” to prevent the server from shutting down the connection. In the old days, loading a homepage could take a long time. Partly because of a slower connection, but also because all time wasted on reconnecting. If a homepage with ten pictures where to be loaded, it took eleven connections to load [40].

2.5.3 GET vs. POST

Typical request methods are the GET and POST methods. Although their function depends on the server-side and browser-side implementation, there are recommendations to consider.

The HTTP specification from 1999 states the following recommendation: “Authors of services which use the HTTP protocol SHOULD NOT use GET based forms for the submission of sensitive data, because this will cause this data to be encoded in the Request-URI. Many existing servers, proxies, and user agents will log the request URI in some place where it might be visible to third parties. Servers can use POST-based form submission instead” [16]

The difference between the two is important to point out because a browser should warn the user if he or she tries to reload the page. Consider a user who visits a web shop. The site may be requested with GET, unless the request coincides with some other action, such as user login. If the page fails to load, the user simply clicks reload and the page is sent again. The user decides to buy a new computer, which should be handled with a POST request. This time, if the page does not respond, the user should not be able to reload the form. That may cause one more purchase to happen and the client ends up with two computers in total.

HTTP does, however, not guarantee either that a GET request makes no changes on the server or that a POST request never reloads [24].

2.6 SSL and HTTPS

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are application layered security protocols on top of transport layer protocols such as TCP and UDP. SSL was released in 1996 as version 2.0 (1.0 was never publicly released) and was the predecessor to TLS. SSL/TLS is used in a variety of things, like mail services and VoIP.

They are used in order to create a secure connection in an insecure network, such as the Internet. SSL and TLS are often used together with another application layer protocol. For example, HTTPS enforces usage of SSL or TLS but is in other ways identical to HTTP. HTTPS is running on a different port (443) than HTTP (80). If communication is going on in the default port (in the case of HTTP; 80) the client may request that the server switches connection to SSL/TLS instead.

SSL/TLS provides a handshake protocol to ensure a secure connection. The client sends a hello message to the server, containing version, present time, random number, session id,
and cipher suites. The server responds in a similar manner, using asymmetric encryption to exchange keys and results in a symmetric encrypted connection between the two [49]. Usage of TLS will be assumed throughout the rest of the report, even though SSL still is in use.

2.7 HTML

Hyper Text Markup Language, more known as HTML, is a markup language for displaying pages on the web. It was developed at CERN in the early 90’s to make homepages visual and user friendly [49]. A HTML page consists of HTML elements, which contains HTML tags. Typically, those tags come in pairs and alter the text between them.

Example 1: \(<h1> Header </h1>\)

The above is an element that contains the tags \(<h1>\) and \(<h1>\). It will print “Header” as a header. A pattern should be recognized here. \(<h1>\) and \(<h1>\) are called the opening and the closing tag, any text outside this pair will be formatted as normal text unless the pair is encapsulated in some other pair of tags. There are so called empty elements as well, which are unpaired. An example of this is \(<br/>\), which is a line break. In some tags, additional information may be added. The string “Header” is called the content of the given element.

Example 2: \(<h1 style="background-color:#ff0000;"> Some text </h1>\)

The above element will draw “Some text” with altered background color. In this tag, the keyword style is an attribute of h1.

Example 3: \(<a href="http://www.liu.se/> This is my school </a>\)

Another example of an attribute and another type of tag. This element prints a link to the University of Linköping. In this context, a means “anchor” and href is “hypertext reference”.

Example 4: \(<a href="../page1.htm"> A link to page 1 </a>\)

Even relative links are allowed with the anchor tag. In this example, a link is created to page1.htm, which is located one level up. The observant reader may wonder if HTML restricts the use of some characters in clear text, like ‘<’. It does not. To print a ‘<’, a web developer uses HTML-entity encoding, in this case “&lt;”.

Example 5: \("\&lt;h1&gt; Not a header \&lt;/h1&gt;\"

The above HTML code will print “<h1> Not a header </h1>” to the page. Thus, it will not represent it like a header as in Example 1 [19].

2.8 DOM

A Document Object Model (DOM) is an interface which makes it possible for scripts and programs to dynamically access and alter documents. DOM is a W3C standard which is defined in three parts; Core DOM, XML DOM and HTML DOM.
DOM structures HTML code as nodes in a tree. In the HTML heading above, the reader were introduced to the concept of elements, attributes, etc. in HTML. All of those are nodes, as is text in elements, comments, and the whole document.

There are several methods and properties in HTML DOM. One method is `getElementById(id)`, which returns the element with the corresponding id. A property, `innerHTML`, returns the element’s text. The DOM is not being described in detail in this report, but it is essential that the reader has some idea of what it is capable of [46].

### 2.9 JavaScript

While HTML and CSS are languages for layout of the visual content of a website, JavaScript is an actual programming language which can change how the site behaves. In contrast to the common intuition, JavaScript is not related to Java. While Java is a heavy, complex language related to C++, JavaScript is a lightweight scripting language, mainly used for web pages.

As a scripting language, JavaScript needs an interpreter to run its code and every widely used and updated web browser contains a JavaScript interpreter. With this said, there are things that JavaScript cannot do. An example of this is that one cannot force running a script when the interpreter is turned off or not present [19].
3. Common Network Threats

There are several types of threats that are specifically effective against networks. McAfee is an antivirus developer which continually has released security reports during the past years. As this is written, the most recent report from McAfee is from the first quarter of 2012. Most prevalent was remote procedure calls, covering more than one quarter of all threats, tight followed by SQL-injections. The third quarter consisted in Browser or Cross-Site Scripting (XSS) threats [27]. Similar results are presented by the Open Web Application Security Project (OWASP) in their top ten threat list in 2010. This list highly emphasized attacks like data injections, XSS, Cross-Site Request Forgeries (XSRF), and tampering or bypassing of authorization [38].

3.1 Command Executions

Remote command executions, especially injection attacks, are among the most devastating risks out there [27][38]. This type of attack includes SQL injections, Buffer overflow exploits and XPath injections, to name a few [47]. An injection attack may consist in sending a query into an interpreter to perform some unauthorized task. This attack may cause unauthorized reading, writing or deleting of files.

3.1.1 SQL Injection

A SQL injection consists of a SQL command submitted by an attacker through an application that exposes a system database. A common way to perform those is to induce syntax errors in the first queries. Some servers respond to invalid queries in a helpful way, which often makes the attack easier. Returned errors are often indicating that a query passed single quote or some other illegal character. A lucky attacker may even get to see names of tables or columns. Other servers, however, do not give the user any clue of what the problem is. Those servers are harder, but not impossible, to attack with data injection. In that case, the attack is known as blind injection attack [38].

From now on, it will be assumed that the database is using MySQL. There are some differences between versions but this report will only cover some examples to give the reader an idea of the complexity in sanitizing SQL queries. For starters, review the following query:

\[\text{SELECT * FROM Users WHERE username = 'input1' AND password = 'input2'}\]

While seemingly legit, this code is easily exploited. The honest user Batman would type: \text{input1="Batman" and input2="123456"} (even though Batman generally is concerned about security, he has chosen probably the worst possible password of all time). The double quotes are implied from the language which created the form, say Java, and will be omitted in the SQL query.

A malicious user could hack Batman's account by typing \text{input1="Batman'--", input2="anytext". The single quote mark will cause SQL to break out of the string and}
interpret any further characters as SQL code. The double hyphen is a comment mark, every part of the query after this will be ignored. Thus, the input2 variable will never be read. The resulting code will look like follows:

```
SELECT * FROM Users WHERE username = ‘Batman’-- AND password = ‘input2’
```

Thus, granting the attacker access to Batman’s account.

Similar methods are:
- `input1="Batman’ #“ (since # and -- are equal)
- `input1="Batman’/" and input2="*/" (since */ this is a SQL comment */)
- `input1="Batman” and input2="anytext’ OR ‘1’=’1” (will run as password = ‘anytext’ OR ‘1’=’1’, which always returns true)

One way of sanitizing the above examples would be to escape all characters which let the input run SQL commands. Those would be: '-', '#', '/', and the single quote. But how would then Mr. O’Brien log in? This could be solved by converting all single quotes to double quotes before running the SQL syntax [15].

One way to prevent injection attacks is to deny untrusted dynamic queries to databases. OWASP recommends using a safe API that either provides a parameterized interface or avoids the interpreter entirely. If there is no such API, special characters should be carefully escaped [38]. In dynamic queries, sanitization tends to be blunter and less intuitive to implement for the developer. Most interfaces seem to be able to sanitize simple, static SQL code such as the one above. But dynamic queries using commands, such as LIKE, are generally required being manually escaped [7].

### 3.1.2 XPath Injection

XPath is a language that makes it easier to find data in a XML document. It is a W3C recommended language which searches for XML nodes [46]. XPath injections are similar to SQL injections and prevented in a similar manner [32]. One difference between the two is that XML does not restrict access to some parts of documents in the same manner as SQL databases may do. Neither is it possible to select the entire database in a general syntax in SQL as it is in XML documents [42]. Some XPath injections are almost identical to their SQL injection siblings, like the input2="anytext’ OR ‘1’=’1” example given above. Due to the similarities, XPath injections are only briefly mentioned in this report [18].

### 3.2 Client-Side Attacks

#### 3.2.1 Cross-Site Scripting

Cross-Site Scripting is commonly abbreviated as XSS, to resolve ambiguity with Cascading Style Sheets. While OWASP declared data injections to be the greatest risk in Web applications in 2010, they stated that XSS attacks clearly were the most prevalent [38]. In 2007, it was estimated that 80 % of all websites were vulnerable to XSS [21]. Though XSS recently has shrunk in prevalence [27], it is still one of the most common Internet threats out there. XSS is an attack which targets users on a webpage to spread malicious contents with
JavaScript. Some possible consequences of successful XSS-attacks is that the attacker hijacks an account, spreading worms, gains access to browser history and clipboard contents, controls the browser remotely or scans and exploits intranet appliances and applications [18][21].

3.2.1.1 Reflexive vs. Persistent XSS

Reflexive XSS attacks are echoed from the server as a response to the victims command. They may be sent via an email, containing a link with embedded JavaScript malware. The user will be requested to open the link and, upon doing so, the malware runs a script that gives the attacker control over the user session.

Another type of XSS attack is the persistent one, which manipulates the page itself. It may be a social media where users type a username that other users can see when they visit their page. If the developers have not sanitized the user name field, then the attacker may implement a malicious script that executes every time someone visits his page [18]. There are a third group as well, called the DOM based XSS attacks.

3.2.1.2 DOM Based vs. Traditional XSS

DOM based XSS was revealed by Amit Klein in 2005. Traditionally, a XSS attack would exploit vulnerabilities on the server while the newer DOM based version exploits the client side. If the reflexive attack example above is to be considered, the victim would get a link to a site which contains a server side exploit. A DOM based reflexive attack would do the same but exploiting the client instead. As the name suggest, this type of attack targets weaknesses in the client DOM [12].

3.2.1.3 Sanitization

The standard method of protecting a website from XSS is sanitization. A XSS attempt on a site could look something like “<script> [Malicious code] </script>". This possibility could easily be sanitized by just seek for every occurrence of “<script>". If the JavaScript tags were used in the target system, an attack may look like this:


In this case, removing script tags would obviously render the code useless. Note that, in this scenario, MaliciousString has at least two attack options. It may contain an apostrophe to break out from the string, but may also escape the script mode with a “</script>". This is something many automatic sanitizers fail to check. To protect an entire site one has to sanitize with respect to the context. The idea of context in this case is defined by where in the code untrusted data may appear.

HTML-entity encoding has been mentioned earlier. When a website is read by a HTML parser, any occurrence of HTML-entities will be printed on the viewed page. So if a HTML code contains the string “&lt;” it will be rewritten to ‘<‘, but never be part of a HTML tag. As a matter of fact, this encoding would make a sufficient sanitization method for all untrusted data in the body of HTML tags. However, problems will occur when a supposed attacker is typing a resource URI, such as the href or src attributes of a tag.

"<… onclick='showText(MaliciousString)’ … >"
The simplified code above gives another dimension to how complicated a sanitization process may be. `MaliciousString` is, again, in nested contexts. It may escape with a quote or double quote. But since `onclick` is going through JavaScript, all HTML entities will be decoded before it is run. Thus, a third attack vector is unfolded, and potentially restricted characters may be used. The larger the attack vector, the larger the threat. Some XSS attacks can have an arbitrary long attack vector, since they keep cycling between contexts.

Taking DOM based attacks into account; sanitization must be performed on both the client and the server. When all this is HTML and JavaScript related vulnerabilities, there are problems that need to be solved on other levels. A common error is that the sanitizer and the browser are using mismatched character sets.

Web developer frameworks tend to prevent XSS attacks in different ways. Some are implementing libraries needed with embedded XSS protection; others extend it in the application code (auto-sanitization). A study from 2011, made for the University of California, reviewed 14 popular web developer frameworks in a XSS sanitization perspective. While this kind of tools are known for being a far more secure option than implement protection as a developer, the study presented other results. It showed that only half of the tested frameworks used some kind of auto-sanitization and only three of those were context-sensitive.

While context-insensitive auto-sanitization may induce a false sense of security, it is considered more secure than no auto-sanitization. The latter alternative relies on developers being able to choose the correct sanitization library at every point and the auto-sanitization will be effective in those cases when the context is right [48].

3.2.2 Cross-Site Request Forgery

Cross-Site Request Forgery is another high rated threat on OWASP’s list [38]. While the abbreviation CSRF is commonly used, this report will use XSRF to denote its close relationship with XSS attacks. XSRF is an attack that tricks the user to load a page that contains a malicious HTTP request. This request will be operating under the same credentials as the user and acts on the victim’s behalf. Generally, XSRF attacks causes single state changes, for example changing the victims password, but sometimes it is used to access sensitive data. XSRF attacks may seem to be similar to XSS, but there are one significant difference: A XSS attack exploits the users trust to a web site, while a XSRF attack exploits a sites trust to the users browser [38][18].

```html
<img src="http://vulnerable.site/transfer?amount=1500&destinationAccount=123456789">
```

An example of XSRF attack may look like the code above. A user is logged in to their bank account and opens another web site, `some.malicious.site`. The browser finds an image on this other site and searches for its source. Instead of finding an image, however, the browser transfers 1500 of some predefined currency to the account 123456789. Note that the request in itself is perfectly adequate. If the user manually would send 1500 funds to 123456789, this very URL would be generated [18].
3.2.2.1 Login XSRF

Login XSRF is a reverted account hijacking. An attacker sends a request, containing their own username and password, to an honest user, which eventually logs in as the attacker. If the user is logged in to the attackers Gmail, their search history may be recorded on the malicious Gmail account while the browsing the web. Assume that the user would like to buy something via PayPal on a malicious website, and logs in to their PayPal account. The website silently logs in to its own PayPal account and records the user’s bank card number while it is entered [3].

3.2.2.3 Defenses

A common way to protect sites from XSRF is to induce randomness in all requests, or at least in all sessions. Typically, a bank web application would send a challenge to a user that wish to log in [18]. For instance, Swedbank sends an eight digit number, which the user responds to with an appropriate sequence of eight digits, generated from their security token. To transfer money while logged in, the user is required to repeat this procedure [43].

Often when a browser sends a HTTP request, or in other words: opens a web page, it attaches an attribute known as the HTTP referer header. This attribute tells the server which site created the request and may then distinguish same-site requests from cross-site requests. However browsers have, for some time now, contained bugs which have made it hard for them to endure referer spoofing.

There are two ways of using the referer header to counter XSRF. The lenient version would block any invalid referer header, but not lack of the same. This implementation is commonly used but easy circumvented; the attacker could make the browser suppress the referer header. Another variant is the strict referer validation. It differs from the lenient only by not accepting when the referer header is skipped. While this makes XSRF harder on the whole, it denies requests which legitimately suppress the referer header.

Unlike HTTPS, HTTP packets may be modified on the network and sometimes drop their referer header in proxies. Using strict referer validation on HTTP would cause too many false negatives. HTTPS, on the other hand, works very well with the strict method. Since pages a user logs in to in most cases uses HTTPS, this proves to be some protection against login XSRF. A main issue with strict referer validation is privacy.

In a whitepaper from 2008, Barth, Jackson and Mitchell suggest the Origin header defense. They introduce the origin header, which will be sent only in HTTP POST requests. The origin header contains a subset of the information given in the referer header, only enough to identify the initiator but not exact paths or queries. Recommended usage of POST and GET, as discussed above, is essential for origin headers [3].
3.2.3 Differences between XSS and XSRF

To clear up any confusion, differences between XSS and XSRF attacks will be sorted out here. This is necessary, due to the different approaches that these attacks are countered with. The following picture will explain attack paths those threats will take.

A traditional persistent XSS is, as expected, directed towards the server. It heavily relates to several attacks described in chapter 4.1, as it is a code injection. Typically, it will induce a script to the server which will be run at any time clients views a certain page. If an attacker would try to steal money from honest users, this script would transfer money from visitors to the attacker.

DOM based persistent XSS exploits vulnerabilities on the applications client side. This attack will force a client to behave in an unexpected way which would be met accordingly on the server. In the case of a greedy attacker as above, the client would send a request to the server, asking to transfer money to the attacker.
XSRF attacks may seem similar to DOM based XSS, but this time a conventional request is crafted in forehand and forwarded to the server via the browser. Thus, the client is not tampered with. It just forwards a request that seems legit.

Reflexive XSS attacks may be sent through any channel, such as a third party mail. This attack will redirect the user to a malicious server which runs a script, containing a server side exploit. A code injection is taken place on the server, which replies to the browser. The browser responds appropriately, since the message origins from a trusted server [36]. As discussed above, reflexive XSS attacks may also target client vulnerabilities, thus becoming a DOM based XSS [12].

3.3 Session Attacks

XSS and XSRF attacks were described above, and some of their counters. To guarantee a secure connection between a user and a server, however, the developer must be concerned about a larger attack area. This section will describe the process of session handling more in detail to give the reader an overview of different fundamental attack patterns.

To maintain a secure login process, the entire login process should be handled with a secure protocol, such as TLS. It is not enough to just hash or encrypt the password since it can be intercepted and retransmitted. Thus, an attacker may login without knowing the plaintext password. For best security, TLS may be used for the entire session. That would greatly reduce the risk that the session ID gets grabbed [35].

3.3.1 Cookie Guessing Attacks

Guessing attacks are feasible when the session ID is small or has low entropy. If the session ID is 32 bits or less, pure brute forcing may break it in a decently short period of time. About 128 bits are considered strong today.

As important is the generation algorithm. If it is incremental, it has zero entropy. An attacker would log in to their account twice and study the difference between the two session IDs that were received. If those equal to, say, 2045 and 2056, the attacker would simply set their cookie to 2049 and log in as somebody else. Even if the session is generated with a standard random algorithm, such as rand C, the entropy would be low. This could be hacked with a so called lattice reduction technique. If a developer do not own a hardware tool that generates random numbers, some frameworks like J2EE and .NET provides secure randomizers [39].

3.3.2 Cookie Eavesdropping

Many XSS attacks aim to steal a user’s cookie. One way to achieve this is to trick the user to click on a link that sends the JavaScript code “document.cookie” to the attacker. But no XSS is needed if plain HTTP is used, without TLS, then the cookie will be transmitted in clear text and may be intercepted by an eavesdropper [39].
3.3.3 DNS Poisoning

If a server (www.honest.site) is vulnerable to DNS poisoning, an attacker may set up a malicious server in the domain (not.honest.site) and trick the user to transmit via this site. With a packet sniffer, the attacker may then intercept the cookie in the same manner as an eavesdropper [39].

3.3.4 Session Fixation

There are techniques which allow an attacker to change a user’s session ID. If a user with an altered session ID logs in to their email account, the attacker could mirror their own session ID to the other user’s altered session ID and thus, operate the email account in the victim’s name [39].

3.3.5 Secure Cookie

As mentioned, a cookie that is sent over plain HTTP, is by default in clear text and thus trivial to intercept. However, if TLS is not implemented correctly, it may be tampered with in the same manner. For example, if the secure attribute of a session cookie is not set, the session ID will be transmitted in plain text to the server. This will make the user vulnerable to eavesdropping and session fixation as mentioned above. The browser still communicates as if HTTPS were used properly, so blocking port 80 (HTTP default port) with a firewall will not ensure a secure connection.

If the domain attribute does not require an exact hostname, the cookie is vulnerable to DNS poisoning. As in the example above, an attacker could trick the user to send their cookie to not.honest.site. If the server’s certificate is validated, the validation holds for the entire domain. Thus, not.honest.site will, be trusted by the client and receives all the sensitive data that is sent [39].

3.3.6 Alternatives to Cookies

There are three widely used ways to establish sessions between a client and a server. Thus far, only services using cookies have been considered in this report. Other methods are to hide the session ID in the URL and to keep it in a hidden form on the page. All the exploits mentioned in this section is applicable to alternatives to cookies as well [39].

3.3.7 Certificate Validation

A proper certificate validation process includes validation of the server’s certificate chain and verification of the server’s hostname. Some TLS libraries, such as OpenSSL, do not verify hostnames. A developer is recommended to use a higher level library, such as libCURL, to ensure a secure validation process. OpenSSL is implemented in libCURL and only validates the certificate chain. This includes ensuring that the server certificate chain leads to a trusted CA certificate, it checks all certificates within this chain and that none of those are expired [14].
3.4 User and Account Management

3.4.1 Password Model
Some web pages do not manage authentication properly. Passwords are an important concept related to this threat. The service should require strong passwords, determined by length and complexity. When users wish to change their password (which they should do, on a regular basis) they must be able to do so and be required to type their old password in the same process. A user should have a limited amount of attempts to login to their system. All passwords should be stored hashed on the server, or if the password is used on other services on the system, it should be encrypted [35].

Brute force attacks are a great concern to password sites. Several breaches has occurred and provided statistics of commonly used passwords:

- Rockyou.com leaked 32 million passwords in late 2009. The 5000 most common passwords were used by 20% of the users, i.e. 6.4 million [44].
- Around 6.5 million passwords were leaked for the social network LinkedIn in June, 2012 [17].
- 450000 passwords leaked due to a SQL injection during the summer of 2012 [31].

Mark Brunett, author of the book Perfect Password, claims that 98.8 per cent of people everywhere share the same 10000 passwords [26].

3.4.2 Password Recovery
Any system with login functions will sooner or later have to face the problem that occur when a user no longer remember their password, loses their security token or by some means are no longer able to log in to the service. The easiest way to handle those cases is to simply continue to deny access and ask the user to register a new account. If the server is aware of this, the account should be locked and thus maintaining the best possible security, given the situation. In many cases this method is not viable; some other approaches are described below.

3.4.2.1 Identification in Person
The best way to ensure a secure password recovery is to meet the user in person and identify him or her with a proper ID card or passport. Should it be a fraud, one may at least remember what the attacker looked like. This has the obvious downside that it is time consuming; it always requires human intervention and if the user works or lives far away from the IT support it may become infeasible due to long travels.

3.4.2.2 Faxed Documentation
Similar to above, but the ID or passport is electronically sent. This method comes with a similar set of advantages as well, and has the benefit that the user does not have to travel too far. On the downside, though, it is easier to exploit, since the identification token might be stolen or copied. Another disadvantage is that it does not cover how to send the new password to the user.
3.4.2.3 Simple Email Recovery

Sending a temporary password is a rather common way to deal with recovery. While it may not be as secure as many other methods, it is considered ‘secure enough’ for most applications. It is automated, easy to use, and since it uses a third party provider, attackers searching for vulnerabilities in the client-server structure will not be able to tamper with it. This does require users to use their mail accounts in a secure way and if the recovery mail is intercepted, the user account is compromised.

3.4.2.4 Encrypted Email Recovery

Encrypting recovery emails does provide another layer of security to the mail method, but it requires that users have public keys. This could be an optional feature for user that demands a higher level of security. Some extra considerations have to be taken regarding key expiration and revocation.

3.4.2.5 General

There are some important aspects of recovering passwords, regardless of which method is chosen, to consider. The user should be required to choose a new password, partly because it should be known by the user and by the user alone, and partly because the recovery may be compromised. This newly chosen password should obviously be typed in the same, or equivalently secure, way as if it was changed for other purposes.

Every recovery attempts should be logged and the procedure should only be allowed once within a given period of time, say three months [30].

3.5 Information Disclosure

3.5.1 Directory Indexing

Some pages rely on “security by obscurity” to when hiding secret files on a web page. When a user enters a homepage, they normally types a domain name, such as www.example.site. When the web server receives this call, it redirects the user to a default page. If there is no default page, a directory listing is returned. If a directory indexing attack is successful, backup, hidden and temporary files may be obtained by the attacker. The attacker may also obtain intelligence needed for a transition to another attack, such as naming conventions of variables and entities, user account enumeration, configurations, and script contents.

A directory indexing may occur when the web server is poorly configured, when individual components of the page needs separate configuration but are forgotten and when the URL to the secret pages are stored on a server from past. For example, Google search engine may keep records of links from past scans [47].
3.5.2 Information Leakage

Information leakage may intuitively lead the reader to think about leakage of confidential data, such as business secrets. This is correct, but leaks may contain information that does not seem sensitive and they can occur at places no one is assumed to search for them. Comments in HTML code are examples of both those cases. They are only visible to the users that manually check the site’s source code and it is not always clear whether it would be sensitive. The notion of blind SQL injections was mentioned earlier. If the application returns error messages from SQL queries, it will give an attacker clues of how the database works [47].

3.5.3 Path Traversal

Many web servers are equipped with an archive which users are allowed to browse. This archive may be public or restricted to a certain set of users. Path traversal attacks are attempts to breach out of this archive, to access forbidden areas of the server. One typical way to perform path traversal attacks is via the URL. If the server’s URL is www.honest.site, a path to the archive may look like this: www.honest.site/archive. Assume that someone types: www.honest.site/archive/../secretfile. If the site is vulnerable, it has just been exploited. Each “../” traverses the attacker up one level so if they succeeded here, that would mean that the attacker has reached a secret file one level above the archive file [47].

3.6 Logical Attacks

Code injections and traffic interception are central concerns for securing systems. But in some cases, applications greatest weaknesses are not in vulnerable components but in the logical structure. Logical attacks are an umbrella term which consists of several functional abuses of an application. Hackers performs DoS attacks to temporarily shut down sites, they registers thousands of email accounts for spamming purposes. Payment procedures may contain logical loopholes that, when abused, allows an attacker to buy goods without paying for them [47]. This section will only cover some common examples of this phenomenon.

3.6.1 DoS Attacks

Denial of Service (DoS) attacks are attempts to shut down a site. It will be divided into two parts, one being a logical abuse of a weakness in an application and the other is DDoS attacks, described below.

Giving a user too much privilege is costly. Untrusted data may abuse naïve generosity of rights that and cause the service to go down or worse. If a service lets the user freely choose how many of a certain object will be created the user could cause the server to run out of memory simply by choosing a very large number. If it lets users choose an arbitrary roof limit of a loop, they could cause the server could get stuck [37].
3.6.2 DDoS Attacks

The impact of a Distributed Denial of Service (DDoS) attack is the same as its above mentioned relative. DDoS attacks do not, however, rely on logical vulnerabilities on the targeted victim. In some sense, they are actually abusing vulnerabilities on intermediate attackers. They are performed through so called botnets, which consists of clients that are infected with a small script. This script is known as a bot and is controlled through the botnet. During an attack, the botnet spams messages to the victim and shuts it down. About 88 % of the DDoS attacks in 2011 consisted of HTTP requests.

Thousands of companies, organizations and governments have fallen victims to DDoS attacks. In many cases, the individual recovery and lost business from those attacks have cost millions of dollars. DDoS attacks typically lasts for about ten hours but in some cases they have persisted in months [2]. One well known recent attack was committed on Swedish government sites, believed to origin from the hacktivist group Anonymous [41].

3.6.3 Malicious Automation

The concept of automation is one of the most important reasons for industrial development in the last few centuries and now, in the age of computers, more important than ever. There are, however, tasks that should be denied automation for security reasons. Several web applications provide free trial service, which may be upgraded to a premium account if a customer so desires. Such privileges has been exploited with scripts, that instantly registers large numbers of accounts for spam, distributed DoS attacks or other mischiefs.

CAPTCHAs are a common counter to malicious automation. This term was introduced by von Ahn in 2000 and is defined as a challenge which is easily generated and solved by humans, but hard to solve with a computer. A CAPTCHA typically consist of a picture of a visually distorted alphanumeric word. An attacker may solve CAPTCHAs either by algorithms or labor. While the latter technically is not automation, it circumvents the point and security of the method and may be a concern.

The solving algorithms indicate an arms race that developers must be concerned about. Motoyama et al. argues in their report from 2010 that unlike most other security related arms races, this is a defender’s advantage situation. The best CAPTCHAs withstand most solving algorithms better than labor and if the developer periodically is changing CAPTCHA generator, an attacker must be flexible in their choice of algorithm. Regardless of which solving method is used, an attacker must be ready to pay a price for an efficient attack.

One of the main drawbacks of CAPTCHAs is usability. There are statistical claims that even though a CAPTCHA takes only a few seconds to solve manually, it may have a substantial negative impact on customer’s popularity [33].
4. The Cloud

The National Institute of Standards and Technology [20] defines cloud computing as:

“A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or cloud provider interaction”

Consider an email application, such as Gmail. One could argue that this service is not a cloud system, due to the fact that it is not scaling to the user’s demand. This report focuses on cloud systems, but the reader will find that the same security philosophy is applicable on most interactive web applications.

4.1 Cloud Classification

Given the definition above, the notion of a cloud may feel wide and hard to in depth analyze in reality. To be more specific, a widely used classification of cloud systems is briefly introduced below.

4.1.1 IaaS

Infrastructure as a Service (IaaS) is the subset of the cloud applications which provides network resources on demand. A customer of this service could host their own cloud application on the hired IaaS. That application would be a public cloud, since it is located at servers which are publically available to hire [4]. Apart from hardware, an IaaS provides the software necessary to run the hardware (drivers and BIOS) and virtualization software. An IaaS typically uses several computers, and interacts with even more users. The virtualization software provides the illusion of a 1-1 relation so that a user feels like being alone on one single computer. Important security aspects on this level are abuse of service, as will be discussed later.

4.1.2 PaaS

Platform as a Service (PaaS) lies upon the IaaS layer, so a customer of a PaaS is also, indirect, hiring (or providing) the underlying IaaS. PaaS provides what could be viewed as an operating system, on which users may upload compatible applications. A challenge for these services is to balance encryption and performance properly. PaaS generally makes software development faster, but less portable in contrast to IaaS structures.

4.1.3 SaaS

The actual applications for most end users are known as Software as a Service (SaaS). Examples of SaaS are Dropbox and Google Apps. Main threats to be concerned to at this level are password and account managements [4][22]. One may regard applications such as email and streaming as SaaS, even though they technically does not satisfy the definitions of cloud computing.
4.2 Security Advantages

While cloud based solutions may bring new threats, there are benefits as well. The physical servers in a cloud will be more maintainable than single servers out in smaller companies. If a company runs one server in the office, it will probably not be able to have full-time working employees to ensure security. But a cloud with a thousand servers may easily employ an entire staff for that one sake. This staff will then get more in-depth experience and be able to individually specialize in different types of issues. For the sake of maintenance, the cloud will probably be built uniformly and thus highly reduce the work time needed for upgrades and new software. Conveniently, this also applies for recoveries and backup. A large scale cloud will be able to load balance much better than the single office server, making it useful in moments of intense computing e.g. DDoS-attacks. The centralized computing will also be useful for lightweight clients, like embedded systems.

It may be tempting to criticize cloud networks for the security issue of storing data on a remote host. This is a correct intuition to have, but it should be compared with the risk of theft and computer crashes in the office [20].

4.3 Security Disadvantages

Cloud systems tend to be rather complex. They are built upon the same basis as a server which is dedicated to a certain system, but on top of that a cloud system also requires middleware to run concurrent software. This creates a large attack surface, in contrast to the office service. The concurrency itself may prove as a threat as well. It makes security dependent on strong logical controls since the client’s shares the physical storage.

A switch from a local intranet to the cloud means that information that previously passed inside the company now is released in the Internet. Even if this information is encrypted, this may prove to be a security concern. The company will also from now on be highly dependent of cooperation with the cloud provider. It will suffer some loss of control and be less able to determine how data is stored and managed. There may as well emerge juridical problems, for example if the region which holds the physical servers has different privacy laws [20].
4.4 Cloud Threats

Cloud Security Alliance presents their top list of threats in cloud systems in a report from March 2010. They will be reviewed below.

4.4.1 Abuse and Nefarious Use of Cloud Computing

Hackers abuse the service’s relative anonymity, to conduct attacks such as DDoS from the hired Platform/Infrastructure. While this threat has caused most damage to PaaS clouds, the attackers are now targeting IaaS systems in a greater extent than before. To mitigate this kind of behavior, strict registration validation should be utilized. It is also important to monitor credit card frauds, external blacklist from the own network and customer network traffic [11]. Examples of abusing cloud systems are the 80 individual malware incidents originated from Amazon S3 and EC2 between 2007 and 2009 [25].

4.4.2 Insecure Interfaces and APIs

Cloud services tend to use several different APIs to maintain their functionality. On ProgrammableWeb, there were, by 2009, 1300 different APIs and around 10-15 were created every week [9]. All these basic APIs provides functionality to the cloud system, but not always security. A system is not more secure than its weakest link so it is important to analyze security on every single API and take every result in consideration while analyzing the system as a whole. Important parts of any cloud system are proper authentication, authorization and encrypted transmissions [11].

4.4.3 Malicious Insiders

Malicious insiders are not an entirely new concern for companies which settles the cloud. But a bigger attack area is to be concerned about. A malicious insider could be an employee of the own company or an employee of the cloud provider. A cloud customer has generally no clue of who might be employed at that level or what they have access to in the cloud. Another kind of threat, though not intentionally malicious, in this branch is mistakes. Employment contracts and compliance reporting should deal with these threats to some extent. When employees are repositioned in the company their accesses and roles should be redefined accordingly and security breach notification processes must be determined [23] [11].

4.4.4 Shared Technology Issues

Vulnerabilities in low level software are quite common even in cloud systems, and may be severely devastating. This is exploited in Rutkowska’s rootkit Blue Pill and Kortchinsky’s Cloudburst, to name some [11]. While this threat lies on the IaaS layer, it concerns applications on top. Customers have to take this threat into account for their own security analysis and try to identify what type of virtualizations that are used [10].
A recurrent problem is that the core components in computers are not built for handling the multi-tenancy that a cloud system requires. A hypervisor is used for handling multiple users, which constitutes a new layer, with new unexplored possibilities or in other words; new threats. These hypervisors must be implemented with best practices in mind. The hardware should be frequently monitored for unexpected changes and when patches are necessary, they should be forced on to the cloud's PaaS and SaaS applications.

4.4.5 Data Loss or Leakage
Proper authorization, authentication and audit implementation are the key to protect data. Backup routines should be implemented and sensitive data transmission should be handled in secure connections, such as TLS. Unlinking of files and lost decryption codes are some events that compromise data [11].

Data loss or leakage is general threats which brings chapter four of this report back in scope. As previously demonstrated, threats like SQL injections may cause data to be lost or leaked. Data may be compromised by hijacked accounts, as will be discussed below.

4.4.6 Account or Service Hijacking
Hijacking techniques also relates to attack types mentioned in chapter four. Client side attacks such as XSS and XSRF may cause a session to obey an external attacker, performing arbitrary actions such as delete or alter data. Server side vulnerabilities may be exploited in similar manner, just as bad password or account management.

4.4.7 Unknown Risk Profile
As an unknown risk profile may provide some security by obscurity, it will deny external analysis and thus, trust. A security oriented company that develops SaaS will not hire an IaaS with unknown risk profile for their application if there are alternatives. The paranoid security expert will not use a web application that uses their own secure protocol instead of HTTPS. Even if those services are as secure as their developers claim (which is unlikely), there is no guarantee for that [10].

A risk profile contains information such as protocols or tools and their version numbers, security practices and intrusion attempts. A decent security analysis of a user’s application should include those in the math. The profile could also contain information of who else are using the infrastructure [11].
5. Practical Security Analysis

5.1 Method
This chapter consists of a practical security analysis, based on material given in previous chapters. The idea is to learn how to improve the security of a system in a narrow time span and no strict method will be used. Target of this analysis will be briefly explained and then will the test phase take place. A static analysis software, Brakeman, will be then used to give a quick review of the code.

The more general threats and vulnerabilities that are mentioned in this report will be investigated. If a threat is found, a solution should be suggested. Desirable solutions are simple, long-term and do not provide disadvantages to the usability of the system.

The manual part of this analysis was mostly covered by black box testing, in an attempt to actually hack the system or at least note some suspicious behavior. Some manual reading of code occurred, but most to get a general hang of the software’s structure.

Software used for this analysis, besides Brakeman, is Google Chrome (without add-ons) and Wireshark. Mozilla Firefox would also have been a reasonable choice of browser in combination with Firebug (similar to Chrome Developer Tools). Wireshark were used to confirm encryption made by TLS.

5.2 System Characterization
The system will be referred to as Target of Evaluation (ToE) and consists of a cloud application. The cloud is relying on several cloud services, but the main application is located on Heroku (Platform as a Service) and implemented with Ruby on Rails. Amazon S3 (Infrastructure as a Service) and Cloudant CouchDB (Software as a Service) is implemented in this app as add-ons, and so is the authentication system, Devise. Cloudant’s CouchDB is receiving data from another source, which is forwarded to the main Heroku application where it gets formatted. When the data is correctly formatted it is forwarded to an Amazon S3 database. All this data is to be considered secret and is transmitted over HTTPS.

There are two different accounts on the ToE. One is for common usage and another for administrative purposes. The administrator has rights to arbitrary edit, add and delete users, but are not allowed to use the system in any other regard. Users have opposite rights.
5.3 Test Results

Analysis of SQL injections are performed in all input components which relates to a database, such as the login page. This did not result in any success and no error messages. A few possible threats were found by Brakeman, which recommended some simple solutions, such as updating Rails and minor code changes.

A brief manual analysis showed no traces of XSS vulnerabilities. ToE contains several input boxes which texts are printed on the page. These prints escaped the following characters: &, <, > and ". No prints were found which occurred inside a HTML tag and contained untrusted data. As discussed in chapter three, this would imply that XSS are sanitized. Brakeman found a few XSS risks similar to the SQL injections.

A cookie is used to maintain a session between the browser and the server. It is secure and has a properly given domain. The session ID has a length of 128 bits and TLS 1.1 is used during the entire session, not just while logging in. All requests are handled with TLS and every HTTP request contains the referer header. Thus, XSRF attacks are not likely to succeed.

The only password restriction is that at least six characters must be used. While strong passwords may be enforced in agreements, there is no reason that it should be technically possible to break the agreements. While changing password, the user is required to type in their old password once and the new twice. No automatic password recovery mechanism exists on the page, but if a user contacts an administrator this can be solved anyway. Administrators have right to change user passwords.

No leaked information has been observed and as the HTML code is generated from Rails, it does not contain comments. Every attempt for path traversing and directory indexing has failed. The site has not allowed any usage of restricted URLs.

The site does not seem to contain any anti-automation features. This applies to user creation on the admin page (which could cause DoS) and login attempts (which could be serious in combination with weak passwords). For enhanced security, CAPTCHAs could be implemented in those areas.

Brakeman found around ten potential vulnerabilities. Those regarded XSS, SQL injection, Mass Assignment and DoS. About half of these were associated with a certain version of Rails and it was suggested that an upgrade of Rails would fix them. The others were single lines of code that seemingly just should be exchanged with another line of code.
5.4 Evaluation of Results

Except for the Brakeman results, not many threats were found. There were some doubts regarding weak passwords and poor anti-automation. Recommended fixes would be updating Rails, changing a few potentially vulnerable lines of code, more password restrictions (eight characters, mixed lower/upper case, numbers and specials) and CAPTCHAs for user login and user management. As CAPTCHAs were reviewed in chapter four, the developers should examine whether the advantages of CAPTCHA pays off for reduced usability.
6. General Strategies

There are several scenarios to consider in an attack on a cloud application. To name a few; it could be someone authorized to use the system that abuses their rights, someone unauthorized who performs a targeted attack against the system, or someone launching a weapon of mass destruction (such as a worm) which causes collateral damage to the investigated system. These are some general scenarios which should be confronted by general defense strategies, known as security principles.

6.1 Security Principles

Below follows eight security principles. While this is just a selection of a wide range of principles, those should be enough to cover most strategically considerations.

6.1.1 The Principle of Least Privilege
To minimize the threat of users abusing their rights, one could argue that every user should only be authorized as little resources as possible to perform their task.

6.1.2 Psychological Acceptability
A user may cause harm without intent to do so. Hence, it is important for a system to be easy to use but difficult to misuse.

6.1.3 Securing the Weakest Link
To prevent a targeted attack from the outside, it seems rational to assume that the attacker will be scrutinizing the system to find a weak link. Thus, to guarantee a secure system, every part of the system must satisfy a certain level of security.

6.1.4 Open Design
Using open designed software may raise skepticism to the unaware developer. The downside would be that a hacker will know how the system works. On the other hand, this kind of software has proven to be safer than any homemade system since it has been tested to cover a great deal of attacks.

6.1.5 Fail-safe defaults
Default set permissions should deny all access to the current object. This will constrain rights for newly created objects. If something fails, the problem would be that the system is too secure.
6.1.6 Economy of mechanism
A security mechanism should be as lightweight as possible. This would imply that the mechanism executes faster, but also make algorithms easier to understand and to correct if bugs are revealed.

6.1.7 Complete mediation
Every access should be checked for authority whenever it is requesting an object. This will ensure that every request’s source is identified. Some systems remember an entity’s identity between requests, for example via session ID. These designs should be investigated, and may prove to be safe.

6.1.8 Reluctance to trust
The environment, in which the system should operate, should be considered to be insecure. This includes input data from users and underlying systems. When analyzing web servers, even the own client system should be suspected, since it may have been compromised [28] [6] [8].

6.2 Practical Security
It is easy to think about security as something that slows the development process down, a necessary cost of software development. This may be true to some extent, but security should not be added on top of a development process, it should be integrated. A well-known phenomenon is that the longer it takes to detect an error, the more expensive it is to fix it.

In modern software development, functionality tends to be in focus. Security gets, in most cases, a second place in priority. It seems to be a skewed balance. Developers are in general more perceptive to the customer’s demands on functional features than the market value of selecting the correct security measures [13].

Most web software today is constructed with different frameworks. Those frameworks are greatly increasing effectiveness since the developers now do not have to write their own basic components, such as buttons and forms. Most frameworks are also providing a great deal of security. Input boxes are sanitized from injection attacks, third party payment applications manages their own security, and database components handles queries and catches SQL injection attempts. Frameworks are not flawless and even if they were, they would not be a complete security solution. But they are an example of speeding up development and increasing security at the same time [5].

6.3 Secure Development Lifecycle
Microsoft uses a development process known as Secure Development Lifecycle (SDL), which inculcates separate practices for secure software. SDL is a structured model for incorporating security in the software development process. It covers seven steps which the team follows throughout its lifecycle and clarifies how security may be implemented.
The general requirements which are defined in the beginning of any software production are complemented by security aspects. The system will then be designed according to the given requirements. Testing the software is also made with security in mind [29].
7. Conclusion

As straightened out above, there are real risks associated with cloud systems, even in a small size. To inculcate the importance of securing an application to network threats, consider an attacker which is specialized in SQL injections. If this attacker spends a few days to get access to a database, that data is compromised. If the attacker does not want to spend time on this, there are toolkits for free that will do the job much faster. Whether the data is secret or relied upon does not matter since successfully injected SQL code may pose a threat to both confidentiality and integrity. Effectively, a secure application needs to be secured to an entire attack surface, while a hack only requires one vulnerable hole. The given database may contain secret business related information that must not leak out; it may contain (Horror and revulsion!) clear text passwords which are reused on other sites. In both those given cases, alteration as well as unauthorized disclosure would likely harm business. The attacker is, as always, not known in advance, but may want to steal for profit, gain information to hack other sites or just damage business.

The network threats described in chapter three are generally either mitigated by proper usage of built-in sanitization or best practices as described in password theory. Some attacks, such as the client side XSS and XSRF, are more complex to sanitize than SQL injections. But an application that neutralizes SQL injections completely and most of the client side attacks are clearly more secure than an application without sanitization. In theory, this improvement should be easy to implement.

The cloud as a concept was introduced in chapter four. A brief review of the greatest vulnerabilities gave the reader an idea of what the main concerns would be in a cloud environment. This chapter may have raised questions less receptive to simple answers. But it is notable that all vulnerabilities are not concerning all applications on the developer’s level. If one were to create an email service, which would be a SaaS, no implementations would affect threats that depend on PaaS or IaaS vulnerabilities. A concerned developer with a low budget would probably still be able to spend some time reading about choices of platform clouds. Some of the mentioned vulnerabilities are directly related to network threats described in chapter three. Mitigations for those would be made anyway.

In chapter five, a practical analysis of an application was made. The analyst had no earlier experience with the application and performed a straightforward black box test, comparing it with threats given in chapter three. The static analysis tool, Brakeman, run for five minutes and its results were investigated. Most errors were solved with an upgrade of Rails.
An overview of different general strategic decisions was found in chapter six. Some basic security principles were stated. Those have not been referred to through the report, as they are more about rule of thumb than specific attack scenarios. But the reader may find that the philosophy they are representing follows in steps to mitigate a diversity of threats. It was suggested that security may be integrated in the program lifecycle, as an approach to streamline development.

A more strict development process, Microsoft’s SDL, were mentioned. While it may seem to be overkill for a small cloud application, several steps seems to be legit even without security awareness. Spending time setting up milestones, general requirements, and finding appropriate tools for testing and coding, are all actions that may streamline the development in the long run. Security just adds another dimension.
8. Discussion

As a result from my studies, I would suggest that any cloud system should integrate security in development. There are certainly tools which withstand most attacks out there. While these tools seem to be improving over time, I find that security issues are something that any software developer should be aware of.

My intention have been to highlight the greatest threats for cloud computing. I have given examples that may not strict count as cloud applications, for the sake of simplicity. I have not hesitated to bring the scope to general web security to indirectly explain problems that occur in the cloud. In a few years from now, say in 2015, the threat model will differ from what it looks like today and parts of this report will be expired by then. To make this report as up-to-date as possible, I have tried to use sources which are at most two years old. All technical descriptions in this report fulfill that requirement, while more abstract theory (such as password recovery management) relies on older sources.
9. Further Reading

I have tried to give the reader an idea of the current threat model of cloud systems. While there have been heavy focus on SQL injections, XSS and XSRF, those are just the top of the iceberg. If the reader is concerned to learn more about specific attacks, I would suggest taking a closer look on the given references. I have read numerous articles to get an understanding for a few attacks, rather than shallow knowledge of the majority of attacks. The articles that have proven to be of most use for my work are stated in the references. They deal with attack specific phenomena, such as sensitive sanitization or login XSRF. For a broader perspective of attacks, I recommend the following sources:

- Open Web Application Security Project (OWASP) - This is an excellent resource for anyone who wants to learn about and to deal with specific threats. Those are described far more detailed than in this report and covered with so called cheat sheets, containing several attack examples that a developer may use to test their own application. On top of that, they also publish a top ten report at approximately every three years with the most dangerous threats.
- Cloud Security Alliance (CSA) - In their reports, especially Security Guidance for Critical Areas of Focus in Cloud and Top Threats to Cloud Computing, they cover most of the security related material used in this report. I would argue that CSA and OWASP together cover almost anything that a cloud developer needs to know about security.
- McAfee and Symantec - Those two anti-virus companies are continuously keeping track of threats on clouds, but also in wider spectra. The concerned everyday web user will find McAfee to be a valuable resource for security news and Symantec covers threats in a more business related context.

The mentioned sites above will be found in the references chapter.
10. References

http://books.google.se/books?id=ERSfUmmthMYC&pg=PA11&redir_esc=y#v=onepage&q&f=true


Stanford University

Diversity Limited

http://software-security.sans.org/blog/2011/06/06/safer-software-through-secure-frameworks

U.S. Department of Homeland Security
https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/principles.html

http://www.cse.ohio-state.edu/rsrg/documents/2005/05BWS.pdf

Linköpings universitet

Websense
http://securitylabs.websense.com/content/Blogs/3402.aspx


https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finaldevanbu.pdf

[14] Diquet (2012) Everything you’ve always wanted to know about certificate validation with OpenSSL (But were afraid to ask). iSEC Partners

http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/


http://www.dailymail.co.uk/sciencetech/article-2157607/Letmein-123456-naughty-words-The-worlds-popular-passwords-revealed.html

http://www.veracode.com/security

[19] HTML.net (viewed 2012) HTML, CSS and JavaScript Tutorial
http://www.html.net

http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909494

https://www.whitehatsec.com/assets/WP5CSS0607.pdf

[22] Joe McKendrick (2012) SaaS, PaaS and IaaS: three cloud models; three very different risks
ZDNet
Bank Info Security  


[25] Landesman (2009) *Amazon Cloud has Rained Malware Before*  

http://www.independent.co.uk/life-style/gadgets-and-tech/features/how-safe-is-your-password-2316963.html


http://fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf

[31] Mills (2012). *Yahoo password breach shows we're all really lazy*  
http://news.cnet.com/8301-1009_3-57471265-83/yahoo-password-breach-shows-were-all-really-lazy/


http://static.usenix.org/event/sec10/tech/full_papers/Motoyama.pdf

[34] National Cyber Security Alliance, Visa Inc (2010) *Small Business Study*  
http://www.staysafeonline.org/stay-safe-online/resources/

[35] OWASP (2010). *Broken Authentication and Session Management*  
https://www.owasp.org/index.php/Broken_Authentication_and_Session_Management

https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
[37] OWASP (2011) *Denial of Service*

[38] OWASP (2010) *OWASP Top 10 - 2010*


[40] Rescorla (2001) *SSL and TLS: Designing and Building Secure Systems*

[41] Rolfer (2012) *Nätattackerna har börjat*
Dagens Nyheter
http://www.dn.se/nyheter/sverige/natattackerna-har-borjat


http://www.swedbank.se/privat/index.htm


[46] w3schools.com (viewed 2012) HTML DOM and SQL Tutorial
http://www.w3schools.com

http://www.net-security.org/dl/articles/WASC-TC-v1_0.pdf

http://en.wikipedia.org/

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopier för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Uphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Karl Johan Lundberg