Middle-Earth Dating - Group 5

System Architecture and Security 2013

Casper Falck Hansen,
Pedram Hamishe Javan,
Jonatan Rueløkke and
Radu Alexandru Miu

Table of Contents:

1 Introduction

During the age of the the internet we are currently in, a huge security paradigm shift from the focus on physical perimeter defense, into the realm of software security. This means that the main defense used has been reactive and mostly behind the curve when it comes to the ingenuity and creativity of attackers.
This project is created from a project assignment at the university of ITU to learn and build systems that surpass this barrier of perimeter defense and reactive security, by following the doctrine of McGraw and following the international and danish rules for online security.

The project involves building a dating/social site with a number of prerequisites in relation to publicly available data, private data and some general functionality such as adding interests or friends. There are also a number of requirements from a security point of view, such as using a specific IP address to access the service as well as specifics relating to information available on the site and security functionality.
The goal has then been to build up a product (dating site) around the specifications given, making security choices and evaluating the choices. This does not mean the site has to be impenetrable, but any weakness has to be argued for.

This means the main focus is on McGraw's three pillars of software security, making a risk analysis and applying risk points to secure the project against the types of attacks deemed important.

The project has a standard system documentation setup with a focus on choices and analysis, more so than implementation. Appendix includes "team contract" and "week by week diary".

2 Project Principles

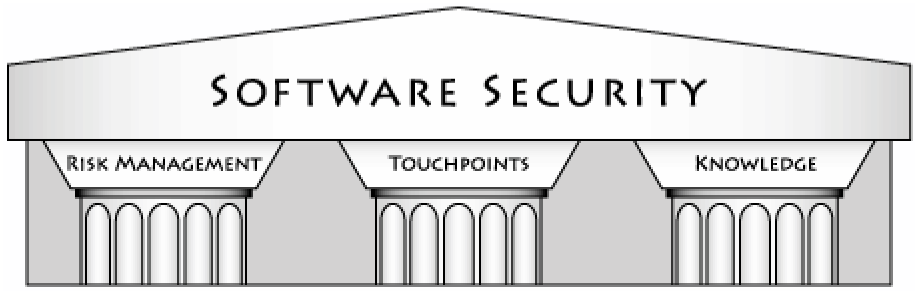
As this course is a security course, it is very important to outline the focus of the system in terms of security in terms of principles. The principles will be mentioned addressed as the principles of McGraw.

This project is first and foremost a student project. As such the concern from a security point of view is very different from that of an actual product. All the information our system contains is practically useless for anything else than claiming to have hacked and retrieved that information. This means that the main priority is not to create the most functional or aesthetically pleasing site, but to create a site that will simply not be hacked.

2.1 McGraw's Three Pillars

Software security is an increasing problem that has manifested and picked up attention over the past 5 to 15 years. Originally physical security of the actual computers were the main issues, but with the globalization of computers and the internet, the shift to software (non-physical)

security has occured. Note that this has not removed the need for physical security, but that as a different doctrine of security.
The main problem with methods of early software security that some even continue to rely on today was two main factors, perimeter defence and reactive defence.
Perimeter defence was adopted mainly from physical defence. The goal of perimeter defence is to create an outer layer of defence that should be impossible to get through. This, however, fails if the threat happens from inside the perimeter. Another problem is the lack of defence in depth, if you first breach the perimeter there is not further defence inside the perimeter.
Reactive defence has the problem of only counteracting known attack avenues. An example of this is anti-viruses. These have a list of attacks they look for, which means that any new attack would work until it is discovered and classified.
The solution McGraw introduced to software security is known as “The three pillars of software security”. These pillars are known as “Risk Management”, Touchpoints” and “Knowledge”.



McGraw’s three pillars claim to be the solution because it encourages the developers to be proactive to threats and run a lasting loop of fixing security issues. It claims that a system is never completely safe and as such it proposes solutions to iterative development of security.

2.1.1 The First Pillar: Risk Management:

Risk management is the first pillar and likely the most important pillar in terms of what to do from a non-implementation point of view. It is important to know that this pillar works on an architectural and design level as well as a lifetime cycle level.

Upon creating a product it is important to consider how the architecture protects or possibly opens avenues of attacks. Choosing an unsafe programming language such as C or C++ can be the right choice for a product, but it is important to look at the implementation level on how to limit the problems with the language. On the other hand it is also very important to track and keep a record of vulnerabilities and keep adding to it as new features are implemented or even removed. The RMF (Risk Management Framework) is proposed to mitigate iteration issues.

Risk management is, however, not a mathematical science at all times. At the best of times you can mainly only predict the severity of an issue. The probability of it happening can be very difficult to predict, but also be changing depending on the user base or feature implementation. As such risk management is also known as a mix between mathematics and fortune telling. On the other hand it also requires a lot of knowledge to even be able to just list issues covering a broad enough scope to properly protect the software.

The RMF framework proposed is a 5-stage framework that covers how risk analysis should be handled. The framework is a multi level loop in large products, but in general works in a standard iterative manner.

Stage 1: Understand the business context
The business context is generally very important, not always the software itself, but for the principles used when deciding on high level choices, such as choice of architecture or language. As an example, if you are asking a development team to choose a language they are less used to than another because it is safer, you are also risking product delays, which can be costly in terms of revenue or how the scheduling matches up with other parts of the company, like commercialization of a product and release of a product. This part also includes making sure that the development team is on the same page as the management in terms of what is more important, releasing on time or releasing a secure platform when it is ready.

Stage 2: Identify the business and technical risks
As stage 1 defined the business context correlating to the business goals, the next step is to evaluate all problems, both technical and business related that can hurt the goals defined. For example a website's uptime can be a main business goal, however, that could contradict with keeping services up to date, due to the downtime created by updating software.

It is also very important to identify technical risks in relation to business goals. A technical risk is significantly less severe if it does not affect any of the business goals. Say a service used internally in a company contains private information such as addresses and names of people in the company. This information could possibly considered a prime concern on social sites such as Facebook, but in an internal system that may not be a business goal at all, even if the information could be considered private and important to the employees.

Stage 3: Synthesize and rank the risks
After defining all of the technical risks, business risks and defined how they relate to the business goals, you now sit with an impossibly large list of problems. This list is practically unusable without the synthesization and ranking of these problems. This stage is to run through and identify severity and probability and create a list defining the most severe problems and a list defining the most probable problems. These lists essentially act as a priority list for the people working on development.

Some of the possible metrics considered in this ranking are risk likelihood, risk severity, risk impact and risks being discovered over time. The risk over time aspect is critical when adding features or when working in a changing environment, such as a growing user base.

Stage 4: Define risk mitigation strategy
At stage 3 you sit with a number of sorted lists outlining all the issues. No one is going to assign any resources to fixing these problems if you do not present a solution. As such stage 4 is used to define a solution strategy in a cost effective manner. This means instead of looking at each individual risk, you look at the bigger picture, group the risks into types of issues and find aggregate solutions.

It is still very important when defining the mitigation strategy to consider the business side of the system. Creating long term solutions and continue to consider newly occurring risks when implementing these mitigation strategies.

Stage 5: Carry out fixes and validate
This stage is pretty self explanatory. Stage 4 designed a strategy for fixing problems, this stage is essentially following that strategy to fix the said problems. The main thing to consider is validating the fixes, making sure to validate from several avenues and not only consider the simplest case.

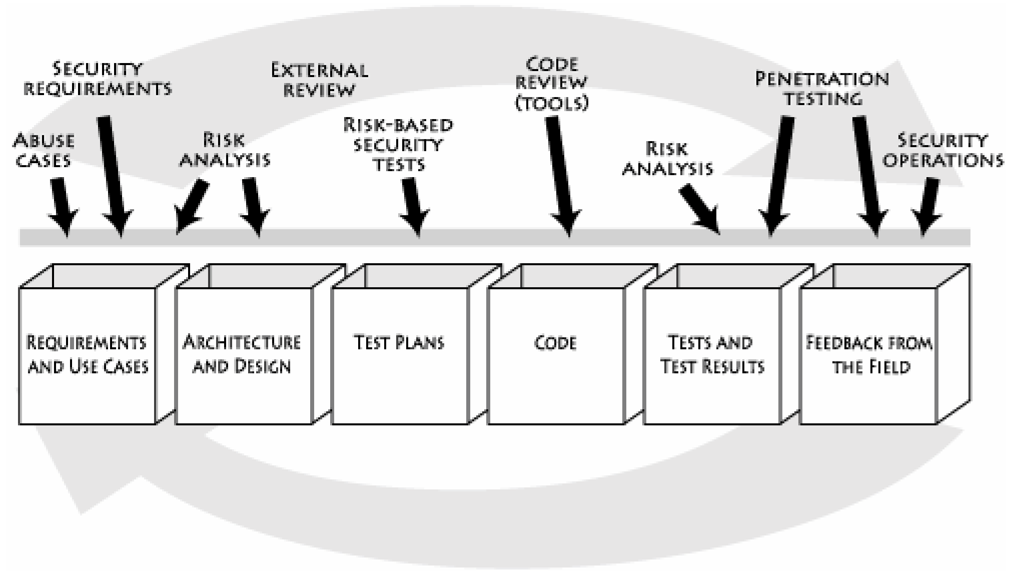
The loop:
To begin with, this was stated to be an iterative process. As such these five stages are run through from 1 through 5, then repeated. This iteration should happen fluently and things like updating threat lists should happen even outside of the loop. It is important to run this framework through continually to keep the system up to date.

In conclusion, a risk analysis will have both business risks as well as technical risks. A thin outline of what a risk analysis could contain could be: two kinds of lists, an outstanding risk list and an identified risk list. These types will exist for both business risks and technical risks. Then all of the risks will need to be evaluated against technical problems and business problems with a severity and priority ranking. This will lead into a list of mitigation strategies.

In terms of this report, it will mainly focus on technical risks and have a token business risk section to outline how it would appear in a real world scenario.

2.1.2 The Second Pillar: Touchpoints
Touchpoints deals mainly with changing the attitude away from being reactionary to being proactive about the security in the system. There are a lot of people who have misconceptions relating to security such as SSL being an all encompassing solution to security or that you only need to fix security issues as they appear, essentially clogging holes, instead of making sure that holes were never in the system to begin with.

Touchpoints work from a list of best practices shown in the image below.



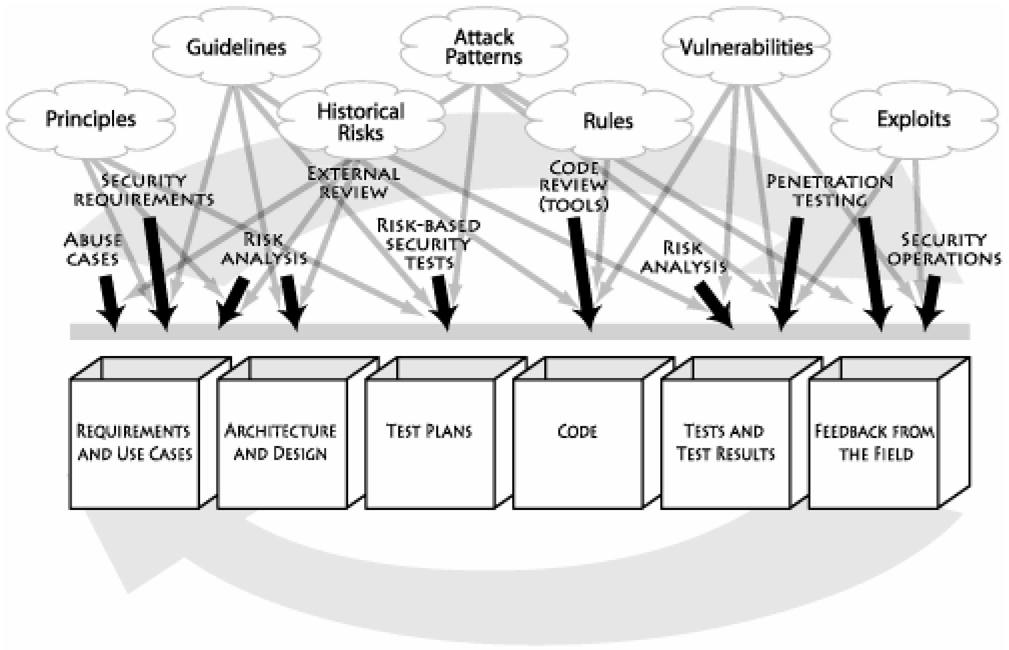
Firstly the boxes present a plan for where to look in the project. Firstly you look at requirements and use cases. This is where you consider abuse and set requirements. Then you look at the architecture and design, which is where the risk analysis mainly occurs. Then you test the risks found in the risk analysis, review the code, and get results from the test. This is then where the iteration occurs and a revised risk analysis is created. Then finally penetration tests and feedback is gathered.
Then as an overall principle, it is important to have external review all over the board, as outside perspective is often important to not get too attached to ones own system and general outside expertise. Finally iteration, running through the touchpoints again and again is very important to consider changes within the systems used, within the field of security as well as the life cycle of the system.

2.1.3 The Third Pillar: Knowledge

Knowledge is the third pillar and is important for the field as a whole. The pillar is essentially taking all of the knowledge gained in your system and sharing it, as well as learning from other peoples systems.
From an internal point of view, the knowledge pillar concerns re-evaluating previous principles, rules and guidelines depending on what historical risks or previously used attack patterns to secure your system against the most likely attacks depending on the type of attackers you have or the type of information you are trying to protect.

For example some sites may contain a lot of sensitive information and as such an attacker trying to get access to the data is a more critical avenue of attack compared to an attacker who is simply trying to deny the service from working. Both are issues, but keeping the sensitive information from getting into the wrong hands is more crucial in this case.

The third pillar uses the 7 catalogs presented in the image below. This image is a modified version of the image from the second pillar drawing arrows explaining how these 7 catalogs relate to the touchpoints.



One important aspect to knowledge is that it can and should be applied throughout the cycle of the system. Essentially knowledge involves keeping track of everything affecting the system. This is everything from malicious attacks both unsuccessful and successful and keeping track of normal use and abuse. This knowledge can be used to see where changes needs to be made and what kind of weighting should be applied in the re-evaluation of risk ranking and synthesizing.

The use of knowledge is this project mainly confirm with other groups. Groups who have their defense breached or groups who breach our defense are required to publicly announce this, allowing us and other groups to better secure their projects.

2.2 Security Principles

As for security principles, we are referring to the security principles defined by the Department of Homeland Security's design principles. While all of these principles are important, we have outlined and explained a few that we consider especially important to this project. These principles will be mentioned throughout the report when explaining design decisions.

The following 4 principles are the main principles of security this project will follow:
1: Reluctant to trust
2: Least privilege
3: Defence in depth
4: Failing securely
5: Securing the weakest link

6: Promoting privacy
7: Never assume your secrets are safe

Reluctant to trust:
Reluctant to trust is important due to a major part of attacks relate to trusting the user, especially user input. In general the system has to continually make sure that it is the right user that is logged in and that all input the user has is run through a security measure to avoid injection attacks.
Another reason trusting can cause issues relate to the privileges the user has. For example trusting an admin user can cause extra problems if that user uses an unsafe password or if in some other way gets compromised. This means vetting the actions or having a backup is important to encourage not trusting the user actually is doing valid actions with his privilege level.
Finally outside information should not be trusted at all. All information has to be white listed and restricted in size and content to avoid any kind of attacks. If anyone is to not be trusted, it is an outside source more than anything internally.

Least privilege:
Least privilege is important to make sure that every user is only able to access the few things that user needs. As such an admin user has access to more data, but due to our reluctant to trust measure, we will not even allow the admin to just freely change/remove data and or users.
Least privilege is generally just used to make sure that any user only has the possible actions that user needs to perform his function within the system. This limits the severity of any breach to that users specified area of expertise.

Defence in depth
As explained in the first section of McGraw's security pillars, one of the main issues is only having perimeter defence. Having several security checkpoints limits breaches to isolated sections of the system. While a breach in the wrong area can still be critical, this means that breaching any point of the system will not allow you access to the rest of the system.
As such means that we will verify input on both the web client end and the SQL end. Other measures will be taken such that accessing parts of the project will not give unrestricted access to the entire system.

Failing securely:
Lastly if our system has a failure, the log file accessed through the slice will allow us to find the issue - the users will not receive any error code or anything that might assist in breaching security. Conceptually we would have a back up SQL server in worst case scenarios.
Also catching errors and outputting controlled error messages instead of giving away entire stack traces or issues similar to that is also of paramount importance to not give away parts of the system to allow potential threats access to information that could help them breach the system.

Securing the weakest link:
The main point is identifying the weakest link. It is very important to be aware of your own system weak links and have a reason to leave that weak link intact. For example this specific project is created as a school project where denial of service attacks are illegal, as such having that as a weak link is complete fine as long as it is a conscious choice.
This is of course also important to always know the next step to fix in the iterative state of the risk management framework.

Promoting privacy:
The assignment in question relates to a dating site, which contains information about users. This means that privacy from a business goal standpoint has to be decently high. As such using the principles of least privilege continually promoting privacy in each part of the system is of importance to this project.
Besides that, internally as a group it is also important to promote privacy in terms of passwords and password handling. Also group members to keep from discussing the system with other project groups.

Never assume your secrets are safe:
Essentially we do not ever want to use anything relating to magic URL's or assuming that people will not guess some part of our system. Essentially we do not want our system security relying on something that can be exposed by word of mouth or guesses, even if we are promoting privacy.

3 Project Requirements

Now that the project principles has been defined based on McGraw's doctrine of system security, the actual user stories provided in the assignment will have to be made into a series of use cases and abuse cases. These use and abuse cases will be used to outline the system requirements together with the project principles.

3.1 Use cases

A person, lets call him Rafal, wants to create an account on this social network site to be able to socialize with his friends without entering the evil domain of facebook or google+. He of course wants his friends to be able to see his name and address so he adds that information to his profile when creating it.

Rafal has now created his profile but now he wants to be able to begin using this profile. Therefore he enters his profile by writing the username and the password he used when creating the profile. Rafal now wants to add some information about his hobbies so other people can see them and eventually choose to befriend him because of their shared interests of ponies or other relevant hobbies.

Rafal has moved to London over the summer and therefore wants to change his address on the site. At the same time he found out his surname Åsø was difficult for english people to pronounce so he chose to change that and now wants to change it on the site as well.

Rafal has been at a bar in the weekend where he met a girl named April who apparently also has a user on the site. Now he wants to find April using her name so he can see more about her hobbies.

After seeing that April is also a fan of ponies Rafal has decided to befriend her. He therefore wants to add her as a contact which he wants to label as a “friend” contact.

A month has passed since Rafal and April became friends on the social network. Since then Rafal has developed feelings for April but he is shy so instead of talking to her he wants to give her a virtual hug to show his affection.

Rafal has been told that April has a user on a competing dating site as well and wants to check whether that is true. Luckily he knows that Middle-Earth Dating and the competing site has a deal so the users can see which users are one the other site without creating a user. He therefore checks the list of users from the competing site.

Rafal has after several months been really happy with the site, however he does not like that he is seeing too much profanity on the site, therefore he wants to become an administrator on the site so he can remove users that are not living up to the terms of service.

Rafal is now an admin and has at the same time found out that April, who he is no longer friends with due to a conflict, has broken the ToS, he therefore sees no other way than to remove her user.

Rafal has been using all his living moments inside the social network and it has destroyed most of his social life in the real world. He has therefore decided that he wants to delete his account.

3.2 Abuse cases

A year has passed and Rafal is mad over all the problems he got in his real life because of the social network, therefore he has decided to try and destroy its very existence.

The person Rafal has the biggest grudge against is April and he therefore wants to start out by hacking her account to make her unpopular.

After hacking April’s account and changing her information Rafal is moving on to destroy the site. His first choice is to try and do a denial of service attack. Sadly for Rafal he does not know how to do a proper DDoS attack with several computers so he chooses to try and create a script that keeps on creating users instead so he makes the database overflow with users.

After doing the DoS attack Rafal has decided he wants to destroy the business by removing users. However he is not able to do this by using a normal profile so he needs to either get admin rights or hack an existing admin account.

After long and hard work he finds out about an admin user that has an account which is easy to hack. Using this account he begins selling information about different users to third party companies and people.
After earning quite a bit of cash selling information Rafal decides to make the final blow to the social network by deleting all accounts on the site, which causes the other users to get too frustrated and go back to facebook for some reason.

3.3 General requirements

The general product requirements are based upon the use cases as seen in the previous chapter and will have McGraw's build-in-security process as a reference to deal with these stories in a secure way from the beginning.

The assignment sets two specific requirements:

- The project has to be accessed through the slice server
- The landing page URL has to be your_ip_address/ssase13

The project besides that has a number of other requirements based on the use cases and the project principles. The following list outlines the set requirements for the project while the section below will qualify these decisions.

- A person has to be able to create a user with information
- A person has to be able to login to his/her user
- A user will be able to see and edit his information
- A user should be able to delete his/her profile
- A user should also be able to add other users as friends
- Friended users should be able to see each others information and give virtual hugs
- Unfriended users can only see each others name-information
- The webclient should "commercialize" itself with a section explaining the deadly sins it is secure against.
- Users can be tagged as admins by client administrators.
- Admins can remove other users
- The site has to show one other site users through REST
- The system has to send a user and interest list to one other site through REST

So there has to be functionality so that a person can add the following points of information to his user profile:

- Name
- Address

- Hobbies
- Friends

The person has to be able to log in to a private user that is only accessible by that person. As such the system needs to let a user create a profile on the web application and then edit these. In general the system promotes privacy and hides information not relevant to other users until they are friended promoting privacy.

When a user edits this information, the information will be checked if it is valid and then before the information reaches the database the data will be confirmed as being the same information from the correct user. Generally the system will be reluctant to trust the user and reconfirm the data a second time before making changes, due to the defense in depth principle.

Additionally as the information is private we have added the requirement of the users information to only be accessible by friends, which means that only users name and hobbies address will be public to find friends. These same information are also what will be available through our REST service.

The REST service requirement is to link up our database with one other group's database. This means we will be able to show some of the public variable from their database on our site, meanwhile they will show some of our public information from our database on their site.
From a security standpoint none of the imported data from their server will be added to our database. A separate sandboxed section will be added for their data, as well as having all their data sanitized. This corresponds with good practice of not trusting their data.
On the other hand, their access of our data will be entirely limited to GET statements and as such they will have no access to our database or system in any other manner than to receive the data intended.

Finally there is the requirement of having an admin user that can delete and create new users. This user will still not have access to already existing user's passwords. The goal is also to create the admin access such that the admin can only change tags in the database to "not visible" and then have the information deleted on the server end. This is due to the least privilege principle as well as reluctant to trust.

The addition of a "virtual hug" campaign has to be added. This system will essentially be created as a possible action between two friended accounts. One player can tag the other for a hug while the other person can then return the hug by clicking and removing the hug from his profile.

The frontpage also has to sport a commercial spiel telling their users exactly what the system is protected against. Due to the security issue of telling potential hackers parts of the system, which will simply allow them easier access to the site, this section will be written with statements of what the system is protected against without mentioning any system information or anything else that can give potential attackers information to help them breach the system.

This means the site will likely state:
The wizards working on middle-earth dating knows that your privacy is important, whether you are an evil warlord wanting your real identity to be kept a secret, or a dwarf afraid that people will know that you're interested in elves. Therefore this site uses magical state of the art security that is potentially unbreachable!

The site is protected against the following security threats:

- *SQL Injection*
- *Command Injection*
- *Man in the Middle Attacks*
- *XSS*
- *Magic URLs and Forged Cookies*
- *Information Leakage*
- *Weak Passwords*

3.4 Security requirements

Above is a list of the general requirements made from the use/abuse cases and the project principles. Now a section with different security requirements follows. First listing all the requirements and then going into more detail.

- Limit user slice access
- Maximum login attempts within a time period
- Maximum one user pr. email address
- Time limit on IP creating users
- Limit admin delete functionality to only "hide" information

The slice setup will only have very few users with limited access to each part of the slice. This is to avoid a breach in a single user to compromise the entire slice. Each user will be responsible for a part of the slice setup, such as separating the frontend and the backend systems in different users. This means there is no root user with global access.

By looking at the project reluctant to trust, it should be obvious that users are not reliable enough to always create safe, strong passwords. Therefore it should be advised that one IP can only make a limited amount of login attempts each hour or so, so that it is much harder to brute force access unto a user.

To make sure a malicious person cannot flood the database with users it should be possible to make the user have to confirm the creation of the user using an email and it should furthermore only be possible to create one user pr. email. As the system currently does not have an e-mail system it should just stop flooding of accounts from single IPs.

Due to the constant threat of being hacked, it has been decided that an admin will only be allowed to remove things from the web client - simply hiding the data. Then the data can be removed through the slice completely if required.

4 Risk Analysis

In terms of risk analysis, it will be split up into two segments, one containing all the technical risks and another containing all the business risks. The first section here will outline the business goals and from the goals the risks will be defined depending on how they affect the business goals.

4.1 Business Goals

In Middle-Earth dating between species is not something people look upon with open eyes. Other than a few elven/human relationships people mostly date their own race. Therefore it is necessary with a new dating site called Middle-Earth Dating.
This means if you are a hobbit wanting to find that one special orc that can make your life complete, or a dwarf just out for some fun with a nazghul, come to Middle-Earth Dating.

Since the internet and computers in general are a fairly new concept in Middle-Earth anything looks amazing in the eyes of the user, therefore the design of the homepage will be very low on visuals. However in a world where factions are in war all the time it is important that Sauron is not able to use the homepage to see where Frodo lives or that Gollum is not able to maliciously change Bilbo's information as revenge on that whole ordeal with the ring.

At the moment the website is on a testing level. However when it is released it will be kept up by commercialising in the way seen from for example facebook. We will also actively sell all user information to run the business.

So the following business goals are in place for Middle-Earth Dating.

-Have the first official release ready and online at 16.12.13
-Have an uptime of at least 95%
-Increase the amount of companies running commercials on the site within the first year by 30.
-Make the popularity of the site rise to at least one thousand residents of Middle-Earth by the first year.
-Make the online time for each user become an average of two hours each month within the first year.
-Increase the staff to ten people by the end of the year.
-Increase the knowledge amongst the average Middle-Earthian to include 5 percent of the population within one year.

4.2 Business risks

So with the business goals in place the next step is to find and analyze some business risks.

The business risks written below is just a small sample of what could be potential business risks for a company. The risks chosen are about the software not being ready in time. System crashes and downtime because of different events. Failure of critical operational features in the system and some aspects of failure in increasing the popularity of the product and failure in creating trust for users.

Business Issue	**Outcome**	**Chance of single occurrence**	**Severity (1-10)**	**Mitigation strategy**
Failure in increasing popularity	Loss of interest, and in worst case end of product	Possible (50%) It is possible that users does not get to learn about the site or simply does not like it when they do.	High Severity(10) With next to no users new users won't see any reason to enter the site, and without users, companies won't see a reason to advertise on the site which in the end will bring the end.	Make strong marketing campaign already before official release. Make sure that the users are satisfied with the product once it is released.
Failure in trust	Less personal information and if the trust is very low, loss of users.	Unlikely (30%) The common user does not think that much about his/her security on a webpage before things actually go awry. However things like media or bad experiences might increase trust issues.	High severity (7) Dating sites in general need a good trust level so users feel they can share intimate information with only the people they want to.	Add security measures, and limitations to information sharing and tell the user about those measures and limits.
The software fails to meet the acceptance criteria required for release	Middle-Earth Dating will be unable to be released to the market at the release date.	Likely (70%) It is not uncommon for software systems to be delayed, however this system is a smaller one and therefore the percentage is not in top.	Very high severity (10) For each day the system is not released no users will join and no companies will begin advertising and thereby earning the company money.	Work within a strict work schedule. Using tools like time plan, agile project management and iterative programming process.
System failures	Middle-Earth Dating will not	Very likely (90%) Even the strongest of	Varying severity (1-10)	Have backup servers take over when the

cause unplanned downtime	be able to fulfill its uptime requirement.	servers do not have 100% uptime so sometimes the server might crash	Depends alot on the downtime. If it is just a five minute server glitch that can easily be fixed by a restart it is not that important, however bigger system failures will possible make the system lose clients and revenue.	main server goes down. Have a log system that tells the developers in charge of uptime about server status regularly.
Security weaknesses cause system failure	Middle-Earth Dating will not be able to fulfill its uptime requirement.	Possible (50%) Malicious users can with enough intent break the system, however the chance of it happening to a critical degree is limited.	Varying severity (1-10) As with the previous business risk the severity depends a lot on the level of system failure happening.	Have backup servers take over when the main server fails. Have an error handling system logging what security weaknesses could have been exploited and fix them.
The software fails to perform critical operational functions properly.	People will be unable to create users or log into their users and might therefore stop using the site	Unlikely (20%) Not thinking about misuse the system should be robust enough with the few functions it has, to be able to run them properly.	Low severity (2) It is normal for users that a page sometimes does not work. The normal response is to try again later.	Log errors and other fails within the system and make sure the developers are aware a function did not work properly.

4.3 Technical risks

After the business risk analysis the next section will outline the different technical risks and specifically how these affect the business goals. In terms of personal risks any user getting their account compromised has very little personal information at risk. The biggest personal risk is address, but without age, gender, image or vacation status, this information does not really contribute a lot of risk. The worst case is getting an account deleted. In this case the user will need to have both his account compromised and his e-mail compromised for this to ever occur, unless the malicious act is done from an admin account.

In terms of admin accounts, these are pretty much the weakest link as described by McGraw. This is because they have a lot of privileges currently. To account for this the future solution will be to both limit privileges and create multiple admin rights that will be on different admin accounts. Another back-up solution will be to keep a secondary database that is incrementally updated every 24 or 48 hours, such that if an admin account is accessed and abused to change things in the database, the database can be reverted to the previous back-up one.

In terms of our back-end, having multiple low access users on the slice, means that breach seriousness will be severely limited, failing securely. This enforces the majority of McGraw's build-in-security principles with limiting and separating privileges and defence in depth.

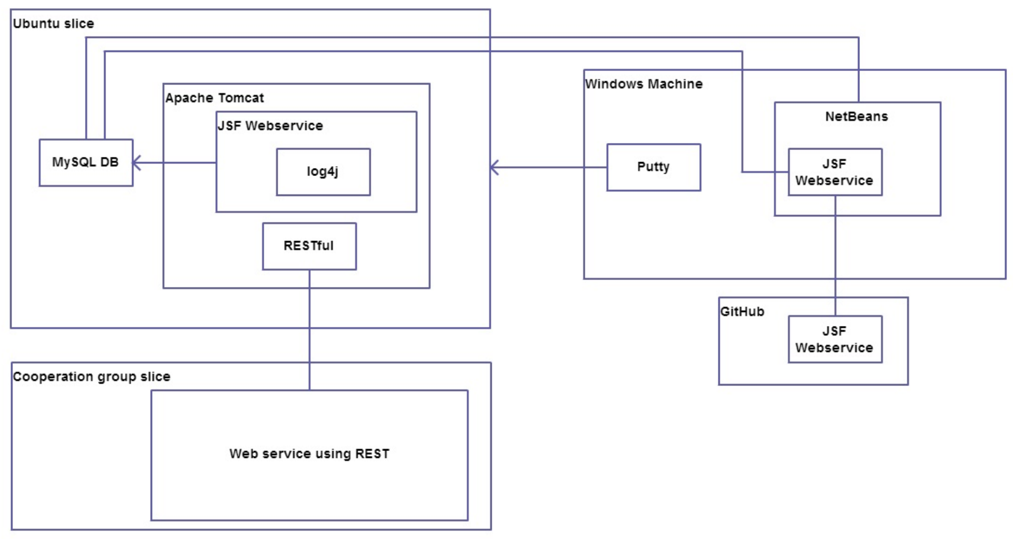
Security Issue	**Outcome**	**Chance of Single Occurrence**	**Severity (1 - 10)**	**Mitigation Strategy**
User account breach	User information is compromised and can be edited.	Likely (100%) Not accounting for bad passwords.	Private information compromised (2)	Force password strength - such as numbers, letters, signs, capitalized and such.
Admin account breach	Can access all users information as well as edit and delete information.	Unlikely (20%) Admins are trusted by developers and proper security measures are expected	All information compromised (7). Data is still not that severe, but global access.	Use a back-up server database to recover information. Force admin log-in from specific ip-address could be used to severely limit chance of occurrence.
Back-end slice breach	Can access one slice user. Slice does not have 1 global user so only 1 part will be compromised.	Very unlikely (5%). Firstly will require ip and port information, then username and password - which are a very limited number of access routes.	Severity (2-10) depends on what user is accessed. Wrong user can bring down system.	Creating several users, rotating passwords and possibly limiting number of people who know users and passwords.
Slice server crashing/resetting/is deleted	All information is lost from the server	Highly unlikely (0.1%) - Server is running officially by ITU.	High severity (10) - All server information is gone.	Keep a back-up of the database at minimum. However, keeping an entire slice back-up updating on a weekly or monthly basis would be appropriate.
Creating a user breaking basic TOS	A user can use the "hobby" field to create	Likely to happen	Varying severity (1-10).	Admins can be used to moderate

	bomb guides or use profanity for any reason.	(100%). Cannot really account for user diversity.	In general most likely not an issue - however creating a guide on bombs could be somewhat severe - Worst case would be another entity shutting down the site due to illegal content.	content! Content restriction or monitoring algorithms could be used to spot and deal with these transgressions.
User account breach from user end (key logged, giving away account information etc.)	User information compromised	Likely to happen (100%). With a dating site like this, with a just semi sized user base will have account breaches.	Low severity (0-2) Information has limited to no use, and if people care about their information they will not have their account breached. (Key loggers, friends stealing account etc.)	Encourage and inform users of password policies and account sharing policies.
Slice server admin goes malicious	Due to several low privilege admins only one part would be affected. This could be accessing/deleting database. It could be access to remove programs terminating the site in itself.	Very unlikely (5% or less) Current access is four people attempting to pass their exam.	High severity (10) Anything part of this could completely compromise the up-time of the system or even remove the databases	Have a system backup that updates every 24-48 hours to reduce the effect. Will still cause downtime, but reduce it significantly.

5 Software Documentation

This section will shortly outline each of the programs and their relation to each other. The picture in the start is an illustration showing how the different parts of the product is linked together. Each part will be explained further below. The “Windows Machine” is only in consideration while changes are being administered to the slice.

System Design



Terminal Emulator
The slice which has to run the webservice is running on ubuntu. So the first thing to choose is how to connect to said slice. Since all project members work on windows systems a terminal emulator was needed. The terminal emulator chosen was PuTTY which works as an SSH-Telnet- and Rlogin-protocol client. PuTTY enables the windows machine to emulate a linux terminal and thereby enables it to connect to the server. PuTTY has a long list of pros in regards to the project, one of them being added debugging information. At the same time it does not have any significant cons and therefore it is a good choice for a terminal emulator.

Coding Language and Environment
For the coding language Java was chosen. This is due to Java being easy to work with but at the same time it has the tools necessary for it to be used within for example the banking world and therefore it should be seen as a safe enough choice for a dating website. From the choice of working in Java, Netbeans has been chosen as the primary programming environment for the development of the web service. The two main java development environments (Eclipse and Netbeans) have very few distinct pros and cons compared to each other. However for the level of web service that has to be created NetBeans has a reputation of being a bit easier to work with for people who hasnt worked with Java before and it was therefore chosen.

Extra Programming Tools
To simplify the workload it was chosen to download a couple of extra programming tools instead of coding everything from scratch. The first tool downloaded is the JDK which is the largest software development kit and for example contains all the groundwork for creating a web service project. Furthermore the application framework JSF has been used for creating the frontend for the web service. This includes navigation and connection with databases.

DB Management System

A database is needed for storing the userdata on the server. To create this database a db management system is needed. MySQL was chosen for the project since that even though some people and companies have begun switching to a DB manager like MariaDB. MySQL is still widely popular and is designed with web and cloud services in mind. The queries were written from scratch instead of using an ORM tool.
A model showing the database design has been added in Appendix 9.3 on page 34.

Web Server Container
Another system that has been used is the Apache Tomcat. This is used as out web server container and makes an environment the code can run in. Tomcat was chosen since it is java-based, it has high availability and is frequently updated. Furthermore it is designed for web applications and is therefore an ideal choice for this project.

Logging Framework
When working on the web server it is smart to have a logging tool since without that there will be a very low amount of information available when an error occurs. For this apaches log4j has been chosen since it is a speed optimized logging tool that at the same time provides the information necessary.

REST Service
Implementing a REST service is extremely useful for several reasons. It exposes an API-like interface to the exterior world, therefore imposing strict conventions and clearly separating this communication module from the other areas. This makes it easier to deal with inbound and outbound data, which is beneficial for security related functionality, as well. In addition, it has become a best-practice for building AJAX based web applications, further separating the client from the server. We provide a this service for other team(s), enabling them to access our data in a secure, conventional way, but we did not yet take advantage of asynchronous technologies within our own site.

6 Testing

Given the nature of the project and it’s incremental addition of requirements over the entire development period, we have opted for a mile-stone based testing flow, rather than a test driven alternative. Overall, we focused on ensuring validity and consistency of the user-provided data, dealing in an efficient way with failure scenarios and mitigation of previously described technical risks, in descendant order of their assessed impact. Since our implementation is not complete at this moment, not all security considerations have been addressed. However, we focused on fail-proofing the sensible points of our system, particularly input fields and URLs, hence protecting against the most common and damaging attacks. We did so by both actively checking against possible exploits and by making specific architectural or technical decisions during implementation.

6.1 Database-related considerations

Maybe the most relevant example would be our way of protecting against SQL injection, an exploit against which we have devised three stages of defence:

- input validation and preparation - using regular expressions, white/black listing characters and similar techniques
- database interaction - use of JDBC and prepared statements for any form of database interrogation
- re-validating data upon retrieval from the database - important against stored XSS

As of yet, based upon our own tests, and lack of successful attacks from other teams, we managed to prevent any tampering with our DB attempted through malicious user-input. It is far from being a production-level implementation, though, and still needs to deal with special cases. For instance, checking retrieved data from the DB is only a consideration, for now, and there is no way of discouraging more sophisticated attacks, such as the ones using malicious graphic files for XSS. Moreover, while sufficient in the vast majority of cases, regular expressions and prepared statements could also be circumvented by an attacker with cultivated skills. As a general rule of thumb for our project, any data going to and coming from the database is never trusted and sanitized as carefully as possible, before being included in any query. Also, the way of building these queries is also an important consideration. Despite validation and sanitization methods applied to a given input, there always exists a probability of malformed data squeezing through and, if the query is built using string concatenation, rather than the prepared statement placeholders, the application becomes vulnerable. As warned by virtually any community member, JDBC is just as secure as any other connection pool, but can be rendered ineffective by bad practices as exemplified. Therefore, all queries we constructed, strictly use placeholders and never use concatenation.

6.2 Stored XSS

Because this is a form of attack that can be performed in a large variety of ways, we only focused on the most basic in our implementation. More subtle methods of achieving such an exploitation, are increasingly harder to protect against, requiring large amount of time and resources.
As of yet, we consider our application safe against JavaScript being injected in the database, thanks to the previously described validation steps. We were also very careful with how to present form validation feedback to the user, and never mirror one's input in our error messages. We are aware that regular expressions are not universally sufficient for enforcing accepted input structure and content. For instance, a regular expression for validating our user's hobby, might allow **%**, but check against **<**, **>** and their url-encoded representations **%3C**, **%3E** respectively. Because the latter are patterns in themselves, the check needs to be recursive, to ensure validity of its assessment. Otherwise, an input such as **%%3C3C** would still result in the intended **<,** once url-decoded.
As example of advanced security features relevant to our application, enforcing image validity

stands out as a must-have security measure. While this kind of XSS has a relatively low occurrence probability, it can potentially have serious consequences for the entire application and it’s users. To defend against it, we would have to white-list file types and implement a way of sanitizing the file on the server in some sort of sandboxed environment.

6.3 DOS

While fully protecting against a DDOS attack is not possible, there are several forms of the attack that can be prevented. As mentioned under our security requirements, we protected our application against scripts attempting to flood the database with users, by increasing the waiting time between registration attempts from the same IP. Of course, if under a distributed attack, this will not prevent the application from becoming unresponsive, but it effectively eliminates more permanent outcomes of such an event.

6.4 Brute Force

Brute force attacks, similarly to DOS, can not be completely prevented, but can be rendered inefficient by enforcing several best practices. First, since the main target for these attacks are the user-name and password pairs, it is a minimal requirement to impose a certain number of characters (6, in our case), and expect upper and lower case letters, and numbers for passwords. In this way, the probability of randomly matching a combination of length 6 or greater, to a user’s password, decreases dramatically. Needless to say, doing so does not provide absolute security, as the user may still choose a weak password, despite being compliant with our standards. In such eventuality, dictionary attacks can be quite prolific, if not for additional preemptive techniques. We chose to monitor the time interval between requests from the same client and incrementally stall the reply after each one. As such, checking large numbers of combinations becomes unviable simply because it would be extremely time-consuming.
Secondly, the response to failed authentication attempts has to be ambiguous and given in constant time for any length of input. For instance, an attacker may want to simply confirm existence of a certain username, or email address in the database. In this scenario, responding with an ambiguous message such as “invalid username or password” provides no reliable information, but merely a “Schrodinger box”. Moreover, it is important to give this reply in (close to) constant time, as to not provide any indication of what the length of the desired combination might be. Since we use regular expressions and built-in encryption, the time difference between processing strings of similar size is minimal and it would be virtually impossible to distinguish from normal variations in network delays. Of course, we could also allocate a constant time to this step (ex: 20ms), but we decided it would be somewhat of an overkill.
Another effective measure against brute-force offensives, comes in the form of Captcha, which generally discourages such intentions. We were planning on adding this feature to our login form, as other groups did, but it was not a top-most priority.

6.5 Routing and access points

We chose to use JSF and define routing rules in the projects web.xml file, where none of the paths defined expect any GET parameters. Hence, malformed URLs will be redirected towards our 404 page and/or striped of undesired parameters. Moreover, they are used strictly for navigating between different pages, but not for transmitting any data. All data to and from the server should eventually be handled solely by our REST service. In doing so, we greatly reduced the number of entry points to the server, leaving only well established protocols exposed to the outside world. This part represents the bottleneck in our applications security, and although still under development, we were particularly attentive with handling input throughout this section.

6.6 Testing on other slices

While trying to find vulnerabilities in other projects, Group 1 gained our focus, but we eventually applied the same treatment to most of the other. Our most advanced test was a PHP script designed to do one of two things: either try to register a large number of new users, or check if the interval between login attempts is constant. As of yet, Group 1 (Brocial Network) proved to be quite resilient and our script is served a 403 response header, despite our efforts to forge the request headers and cookies.
In addition, we also attempted basic SQL injection on several other projects, hoping to expose a weak point, but it seems everyone makes good use of prepared statements. Also, we tried out url/html encoded input and even tried to build a magic URL. None of these trials resulted in any exciting response, so we had to give up, due to limited time availability.

7 Discussion

Overall, the project presented a few challenges of both technical and theoretical nature. While the expected end-product should have been sufficiently complex to demonstrate various security related techniques, due to several set-backs we encountered during development, our project is not mature yet. Despite our efforts, the technology of choice, or rather our limited knowledge of it, proved to raise significant issues and we were forced to look for alternatives on a couple of occasions. During the first period, we looked into JSF and JSP, different IDEs and even built a couple of try-out projects in Grails, which had the advantage of automatizing large parts of development jobs, such as logging, DB connection, creating classes, etc. At the same time, we were drafting plans for the intended outcome (approximating requirements for following milestones) and tried to assign tasks in an efficient manner. Unfortunately, the instability we experienced during this first development period was something we could not fully recover from, hence, the incomplete status of our product.

Despite the numerous caveats in the existing implementation, we did spend a great part of our group time debating on how to best address the many security aspects relevant for this assignment and researching implementation possibilities in Java.

We are very well aware of missing functionality, including some of which we mentioned earlier in this report. For instance, it is possible, at this stage, to spam our database with auto-generated users, since the system is not aware of request initiator’s IP, despite the fact that we foresaw this weakness. Apart from previously reported as missing or partially functional features, it is important to note here the necessity of better error handling, usability improvements for the entire application, better input sanitization and proper isolation of restricted areas. As a general assessment, we agree our application is in its incipient form and is just the basis of a potentially production level system. Nevertheless, it provided us with the opportunity to experiment and consider various architectural and security principals in a hands-on manner.

We have allocated some time towards the end of the project period for testing our own application and others, against some of the more common exploits such SQL injection, magic URLs, attempted XSS and scripted CSRF. Unfortunately, it turned out to be slightly less exciting than we hoped, since we were unable inflict any significant trauma to our competitors.
On the other hand we were also unable to penetrate our own system using these methods, which was the goal.
We decided against using blackbox software to test our system due to lack of time for it.
McGraw also states that too much faith should not be put into these blackbox tests and as such it should not constitute a bigger loss than the fact that we now do not have experience with these types of software.

8 Conclusion

From an analysis and design perspective, the project highlighted most of the issues with software security and designed a strategy to deal with the majority of issues involved in software security.
The project followed the guidelines from McGraws three pillars as well as The Department of Homeland Securities guidelines for software security. Even if some of the aspects are only used in a limited form, such as the third pillar knowledge, it was used in the form the limited time allowed.

During the encounter process a number of issues were encountered in terms of the product. the project still managed to include the required parts of the product in a minimalistic form. This also means that some predicted weaknesses are active in the project, even though a mitigation plan was planned. The process of creating the product was still a very valuable lesson in proper designing and knowledge of tools relating to security.

These weaknesses have been considered and in the report a number of mitigation strategies for the majority of issues do exist. As such if the project was to continue, a plan already exists to deal with the issues that we felt was important in the project.

9 Appendix

9.1 Group Contract - Group 5

Members:
Jonatan Rueløkke (jrue@itu.dk)
Pedram Hamishe Javan (pjav@itu.dk)
Casper Falck Hansen (cfal@itu.dk)
Radu Alexandru Miu (radm@itu.dk)

GitHub repository:
https://github.com/radu-m/SAS_code_war

Disclaimer
By accepting into the group the following rules apply. As such, no formal signature is required as all members mentioned above are subject of the following group work contract. Breach of its stipulations may result in exclusion from the group. Problems in general will be discussed internally, and if no solution found the course manager will be contacted.

Course Manager
Agata Przybyszewska (agpr@itu.dk)

Communication
Communication will happen through Skype and SMS messages. One of these options should generally be available when possible.

Rules for group work
The rules described below are a formality. Generally any and all issues will be discussed and dealt with logically by the group, however, the formality below is here for the sake of organisation and disagreements.

- Members will be required to meet at maximum 2 times per week if project work is on schedule. This will usually be on Fridays and Tuesdays – the time will vary to fit the majority of members.
- ○ If project work is behind schedule – more hours will be planned accordingly by the members.
- ○ Working on the project does not have to be on campus, however on scheduled meeting hours, meeting on campus is required.
- ○ If members are considerably late for project work (usually 15 minutes or more) – they are required to compensate the group with cake. Strawberry pie is considered the best possible compensation.
- ○ If a member cannot meet for any reason – the member will simply be at the group's mercy when it comes to work distribution. If a member continually does not show up, or fails to inform the group of this – the group can discuss this with the course manager.
- All members are responsible for their own sufficient participation beyond what is assigned to them by the group as a whole.
- ○ As such, if a member is not assigned enough work to pass for the exam, it is his/her own responsibility to be assigned more work by the group.

    - All members are also required to read the diary and make sure their participation is well documented.
    - If a member do not show up for a meeting, it is his job to stay apprised of the design choices/changes that occurred in the meeting - however another group member has to be available to respond to questions. The group is, however, obligated to inform the missing member of any changes that directly affects his current work.
- Usually work will be distributed internally by the members, however if serious disagreements occur to whom will be performing what work – the following options occur in order.
    - The group will either vote on who will be able to perform the work to a higher quality and that person will be assigned the work.
    - If no distribution fit either person’s skill-set better, a coin flip will be performed.
- If the member assigned to some work is unable to perform this work, the member should go through the following list:
    - Contact the group, asking for assistance before the next meeting.
    - If this does not help, find a group member that is confident in their ability to perform the given work who will then be assigned to do that.
    - If none feel confident they can perform the work on their own, call for a meeting to deal with the issue on campus as a group.
- If a member cannot be contacted on either Skype or SMS for more than a week, the group can discuss a solution for this with the course manager.

9.2 Week by week diary

Week 3 (Friday 13/09-13 to Friday 20/09-13)
Jonatan: Setting up Linux server end such as users, installing initial programs.
Pedram: Reading up on suggested programs (such as log4j) to create an informed decision on choices.
Casper: Taking care of formalities involving group work, setting up shared space for report drafts and such.
Radu: Setting up github as online repository for source code.

Weekly goal:
This week will mainly be about getting everything set up. On the members end, it is setting up each member's computer with access to the source code repository and a set-up to start implementing.
On the server end it is securing an initial user. Setting up firewall and a program to log the server. Lastly setting up tools that will be required later on, such as SQL database.

Week 4 (Friday 20/09-13 to Friday 25/09-13)
Jonatan: Create database and work on the linking part with the web client
Pedram: Worked on the server tool side - installing security, mysql support, tomcat and so on.
Casper: Worked on introduction and formalizing product requirements discussed in text - also keep an overview of report structure, as well as doing diary.
Radu: Worked on the web client side as well as considering linking to database.

Weekly goal:
Do about a third of the required work for the hand-in! This means forwarding both code/system side and report side equally. All decisions was discussed through before starting to work on them.

Week 5 (Friday 25/09-13 to Friday 04/10-13)
Jonatan (Absent: Sick): Work in conjunction with Radu to - link up database with web client - create simple log in, user profile, friends and so on (Todays slides relevant). Focus on parts of recieving database information, editing information. Think about privileges and multiple validation steps.
Pedram: Get remaining server side things to work. Put up the database created on the server (priority) - may write design chapter concerning choices of tools and other installed programs server side.
Casper: Work further on report - work with architecture principles and system design. This will be in conjunction with every other member - as such every member has worked on report and Casper have been a part of every implementation part in some small way.
Radu: Work in conjunction with Jonatan to - link up database with web client - create simple log in, user profile, friends and so on (Todays slides relevant). (Think about/set up way to give github

access to course supervisor). Assist Casper on structure of ITU style reports - and write web application security part.

Weekly goal:
Get the majority of hand-in work done, so that the remaining can be done the 4th of October. Mainly getting everything up and running, so that all minor additions and such can be addressed along with the report the following Friday on the hand-in date.

Week 6 (Friday 04/10-13 to Friday 11/10-13)
Jonatan: Mainly been working with report software documentation part. But also in conjunction with Casper on both system architecture and testing parts as well.
Pedram:.Been dealing with issues relating to setting up database on the slice environment due to port issues.
Casper: Mainly been working with re-evaluating parts of product requirements and doing risk analysis sections. Also been working in conjunction with Jonatan on software documentation and testing parts.
Radu: Mainly been working on implementing web client and database integration parts - basically front end work.

Weekly goal:
Working on finishing up the hand-in. Focus is on implementation of the main functionality that is still causing problems, meanwhile half the group (Jonatan and Casper) is working to push the report to be available for hand-in.

As everything except the Web-client was ready, Radu was instructed to hand-in the report with git-hub access for our code. This was done on monday the 7th.

Week 7 (Friday 11/10-13 to Friday 18/10-13)
Jonatan: Review report feedback
Pedram:.Slice still has issues (connecting to stuff) - User file issue was fixed this week (hours on hours of work spend here).
Casper (Sick): Review report feedback
Radu: Do the web client (Still not done, not uploaded or anything).

Weekly goal:
Catch up week - work on things that was not completely ready. Then start prepping for the Presentation - Command Injection was chosen.

Week 8 (Friday 18/10-13 to Friday 25/10-13)
Jonatan: Vacation!
Pedram:.Vacation!
Casper: Vacation!
Radu: Vacation!

Weekly goal:
Vacation!

Week 9 (Friday 18/10-13 to Friday 25/10-13)
Jonatan: Do “command injection” redemption steps
Pedram:.Do “command injection” testing for the issues and extra defence measures.
Casper: Do the “command injection” introduction (what is, related sins, affected languages, etc) (Also, brought cake - due to no presence last meeting)
Radu: Do “command injection” examples.

Weekly goal:
Focus on presentation. Also Radu still working on web client.
Project part 2 released + “have to get project up on slice before next friday” requirement.
Everyone keeps working on previous parts.

Week 10 (Friday 25/10-13 to Friday 1/11-13)
Jonatan: Work on SQL binding, queries, etc.
Pedram:.Setup users for SQL (allow IP’s etc), redo some earlier Slice settings to increase security (iptables).
Casper: Reviewing code (due to prior focus on report), also spend time helping others with code related issues almost exclusively this week.
Radu: Still working on web client. Was asked to finish it and upload to slice before deadline.

Weekly goal:
Slice was supposed to be up the 25.! It unfortunately was not.

Week 11 (Friday 1/11-13 to Friday 8/11-13)
Jonatan: Had to redo web client from scratch with the help from Casper. Included Log4J this time around.
Pedram:.Spend time figuring out how to access SQL from server side and how to use the queries in relating to JSF
Casper: Did web client from scratch with Jonatan.
Radu: Spend all day trying to figure out how to upload his to the slice.

Weekly goal:
So at the 1/11-13 we met on the school, Radu was asked to upload the slice for the 25. and he did not report any issue to us relating to this. Upon meeting we are told by Radu that the web client is not up (all of our fault for not checking).
This initiates panic mode. Radu spends all day trying to get his to work - however as he used a number of packages and tools that the group had not planned to use, not did any in the group have any experience with any of these made us completely unable to assist Radu in this.
This meant Jonatan and Casper started from scratch to create a login system in JSF without

any use of additional tools or packages.
Meanwhile Radu spend all day trying to get his uploaded (and it was not feature complete as the features he was asked to implement over the past 4 weeks).
Pedram spend the day fixing the slice so it would cooperate with a database.
The day ended out with a simple web login (without database integration) started from scratch.

Week 12 (Friday 8/11-13 to Friday 15/11-13)
Jonatan: Spend all week getting SQL integration and database integration. - Then did work on report.
Pedram:.Did database integration with Jonatan (along with a pressing slice SQL issue - something with login from server having anonymous non-password users added automatically, that had to be removed). - Then did feature implementation to dating site.
Casper: As time for hand-in is pressing went back to focussing on report.
Radu: Asked to look at risk analysis and do some simple features (error messaging, using already existing search SQL query to create friend-search function)

Weekly goal:
At 8/11-13, still incredibly pressured. Pedram and Jonatan spend all day on campus fixing out all of the database integration such that the project actually has basic functionality required by the hand-in.
Radu did not work on campus but was assigned a few tasks outlined above.
Casper asked to focus on the report.
From home during the week Jonatan was also required to look at the report.
Pedram was asked to do a bunch of relatively simple features from home (home page, edit information, etc).
At 14/11-13 the group met on campus to work on the pending issues. Casper has been working on the major report parts, meanwhile Jonatan and Pedram are working on web client side, setting up all of the information and adding friends and so on.
Meeting on the 15/11-13 should conclude the required parts of web client implementation. After this focus will switch to distributing tasks relating to penetration tests and the test section in general. This should be concluded over the weekend where the hand-in part 2 should be done on Sunday.

Week 12 (Friday 15/11-13 to Friday 22/11-13)
Jonatan: Worked with Casper to plan the remainder of time to make sure we finished all required implementation.
Pedram: Pedram worked on feature implementation, main making GUI and so on work. This was required due to none of Radu's assignments from previous week was added to the site.
Casper: Spend time together with Jonatan to plan the rest of the time, in terms of report and product to make sure everything is remembered. Was assigned some report work. Mainly working on changing some main report parts from feedback. Casper was only one present at class to make sure a cooperation was created for the REST framework.
Radu: Radu was asked to work simultaneous with his implementation of the project.

Weekly goal:
At 18/11-13 Pedram and Radu was supposed to meet up before the hand-in time to merge their parts of the project. Radu called in the evening apologizing not being able to meet - as such Pedram uploaded what he had and sent the report in it's current state. The parts Radu was supposed to have done at this point never saw the light of day.
Radu said in the meeting he was not comfortable working in the existing working code (something with not being used to JSF/JSTL) so he wanted to work side by side on his own implementation, which apparently is an edited version of a friends software?).
Due to Pedram being the most inclined in the code parts and had finished most of the previous implementation is asked to look at the REST framework.

Week 12 (Friday 22/11-13 to Friday 29/11-13)
Jonatan: Continued work on the report. Jonatans focus was on business and use cases.
Pedram: Pedram finished up the last feature implementation, and started working on the REST framework in cooperation with Jesper from Team 8.
Casper: Continued work on the report. Focused on McGraw's Pillars and principles. Also spend time being a passenger programmer to assist Pedram when required.
Radu: Radu was asked to work simultaneous with his implementation of the project.

Weekly goal:
Get the rest framework up and running.
Pedram got the REST framework up and running very quickly with a string output. The cooperating Team 8 asked if it could be changed to a Java object.
Report progress slowly this week due to focus on other projects.

Week 12 (Friday 29/11-13 to Friday 6/12-13)
Jonatan: More report.
Pedram: Pedram did not have a lot of time this week due to work related obligation. Promised to finish up the REST framework part regardless.
Casper: More report.
Radu: Was asked to work mainly with writing the testing section of the report, as well as attempting penetration tests on our own slice and other groups slices.

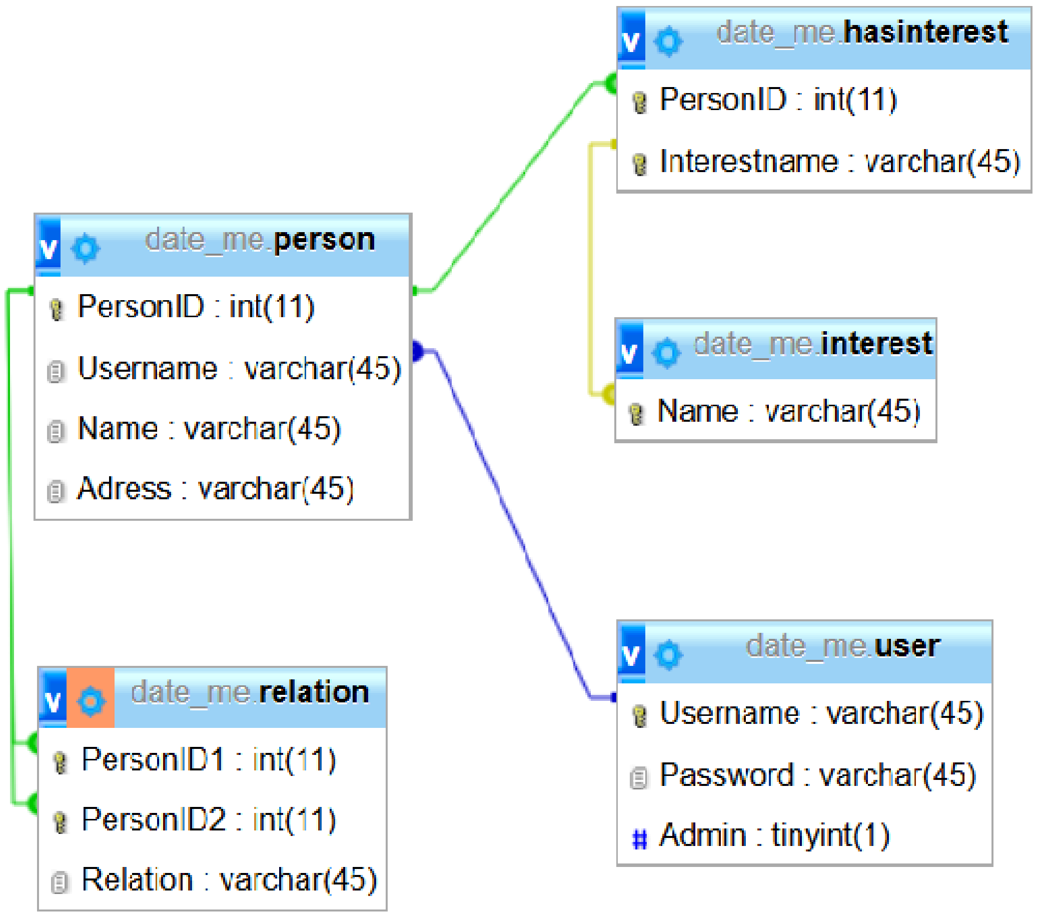
Weekly goal:
The goal of this week was basically to get pretty much to get the entire report out of the way.
At 6/11-13 most of the report is complete. Minor changes required here and there, but mainly only testing section and discussion/conclusion missing.
A number of product problems were identified and put on the to-do list.

Week 13 (Friday 6/12-13 to Friday 13/12-13)
Jonatan: Some slice side polishing. Mainly Report
Pedram: Web service polish work and report polish.

Casper: Polish report.
Radu: Penetration testing and report polish.

Weekly goal:
As the report had mostly all of the “meat” written, the goal is to just finish up everything and have it ready for hand-in by the 13th.

9.3 DB Model



date_me.hasinterest
PersonID : int(11)
Interestname : varchar(45)
date_me.person
PersonID : int(11)
Username : varchar(45)
Name : varchar(45)
Adress : varchar(45)
date_me.interest
Name : varchar(45)
date_me.relation
PersonID1 : int(11)
PersonID2 : int(11)
Relation : varchar(45)
date_me.user
Username : varchar(45)
Password : varchar(45)
Admin : tinyint(1)