Lab Project,
Final report

Group 5
Magnus Krane, Erik Sørensen, Piyush Bajpayee

{eriksore, magnkr, piyushb}@stud.ntnu.no
TTM4135 Information Security
March 13, 2013

Abstract

This paper is a lab report from the "Web Security Lab" which is a term assignment in the course TTM4135 at NTNU.

1 Introduction

The web could be argued to be one of the most important infrastructures in the world. E-trade, banking, governmental web pages, education, management on physical infrastructure and more are done over the Internet and on web pages. It's therefore of the utmost importance to keep a good enough web security on such services to ensure their integrity, availability and confidentiality.

To give a rudimentary understanding of web security this lab is divided into four main parts: Understanding and creating certificates, access control on an Apache web server, understandig Subversion Repositories and creating dynamic webpages with PHP and MySQL.

2 Experimental procedure

During the experiment we followed [1] point by point, therefore only questions asked in the text (other than Q-questions) and points where we changed procedure will be mentioned in this part.

Part 1.4 The purpose of the 'echo' and 'touch' commands were to create a "database" for the certificates.
In the *caconf.cnf* file the default-md and policy-match variables were changed. default-md were set to SHA1, which is more secure than md5 [3], and more matches were set in policy-match. These steps were done to make the certificate more secure.

3 Results

The result of this lab is shown at the following web pages:

http://ttm4135.item.ntnu.no:8121/
https://ttm4135.item.ntnu.no:8122/

One part is an unsecure HTTP page, while the other is a more secure HTTPS page. Most of the lab were done in the HTTPS environment, and follows the form and build up as described in [1].

4 Discussion

4.1 Q1

Comment on security related issues regarding the cryptographic algorithms used to generate and sign your groups web server certicate (key length, algorithm, etc.).

For the web server certificate, the signature algorithm were chosen as SHA1 with RSA encryption, with a public-key length of 2048 bits. At the current time, RSA laboratories (creators of the RSA algorithm) recommends a 2048 bits key size for extremely valuable keys, and 1024 bits key size for corporate use [2]. Even 768 bits key might be regarded as sufficient. The use of 2048 bits key size in this case could therefore be regarded as 'over-the-top'. Ultimately the choice of key length is a trade-off between de-/encryption speed (see [5] page 301) and security. SHA1 (160-bit hash) were chosen over Md5 (128-bit hash) as message digest due to known security problems with Md5 [3]

4.2 Q2

The detached PGP signature of the source code should be verified using GnuPG.

Explain what you have achieved through each of these verications. What is the name of the person signing the Apache release?

Firstly the Apache release (*.qz) were checked against detached signature (*.asc). Through these step we have identified that a "Jim Jagielski" has signed the file, but we can still not trust the file as it is not certified with a trusted signature. A check against the pgpkeys.mit.edu keyserver also shows this. In conclusion, the received public key can't be trusted. To validate the authenticity of a key (and therefore also, the integrity of the file) [4] recomends a face-to-face validation with the signee. This is, of course, in many situations impossible, but one could enter a so called "web of trust" were other people already have validated each other keys.

4.3 Q3

What are the access permissions to your web servers conguration files, server certicate and the corresponding private key? Comment on possible attacks to your web server due to inappropriate file permissions.

The following file permissions have been set.

- Configuration files: 600, -rw- — —
- Server certificate: 400, -r– — —
- Private key: 400, -r– — —

These permissions has been set as restrictive as possible. Especially for the certicate and key files, their persmissions have been set to read only. This is to make sure their integrity is kept. All files have been given read/write access for owner only, i.e. you would need to be user root/gr05 to read these files. The files is also stored in folders not accesible from the Internet.

Possible attacks could be to change certificate requirements in the configuration file to gain access, or if the private key were to be accessed the whole certificate access control would be rendered useless.

4.4 Q4

Web servers offering weak cryptography are subject to several attacks. What kind of attacks are feasible? How did you configure your server to prevent such attacks?

- **Brute force attacks** are a type of attacks were all possibilites are tried (in this case for the private key). This attack will not be feasible on our server as a key space of 2048 is regarded as safe with todays computing power and factoring algorithms [5]. ¡¡¡¡¡¡¡ HEAD
- **Encrypted passwords in database** A possible attack on the database is SQL-injection (see Q5 for description and prevention). A good way to secure the relevant data in the database, in our case the passwords, is to use encryption. If a attack on the database were succesfull, all they would see is the encrypted password. The MySQL database has encryption methods that use the official AES algoritm to encode the data with a 128-bit key length. =======
- **Collision attacks** ¿¿¿¿¿¿¿ efef
- **Password exhaustive/brute force attacks:** The cryptography protecting the server is worth nothing should an attacker gain access to the root user. This could be done through a brute force/dictionary attack against the servers password (see [6] page 276 for details). If such a attack were successfull, an attacker would have full access to the whole server and database. Creating a strong password will be the best defence against such an attack.

4.5 Q5

What kind of malicious attacks is your web application (PHP) vulnerable to? Describe them briey, and point out what countermeasures you have developed in your code to prevent such attacks.

Our application is vulnerable to Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Session Hijacking, and SQL-Injection. XSS uses input fields on a site to upload html that includes sripts, like javascript, which loads to the page on request from other users who then reads and executes the script([8] page 734). CSRF is when the evil doers tricks a user to submit a forged HTTP request through for instance XSS. ¡¡¡¡¡¡¡ HEAD =======

We are also vulnerable to SQL-Injection. This is when the developer passes user input to a back-end database without checking to see whether it contains SQL-code. This is often given away by error messages, from which the evil doer may interpret to mount a attack ([8] page 120). Session Hijacking could executed when the evil doers have gotten hold of a session or session ID from another user and they act as the user.

If we look at the most critical web application security flaws [7], we can see that we didnt acount for many of them. To avoid XSS and CSRF we could have have correctly escaped the input to make sure that they are stored and displayed correctly. To avoid SQL-injection we could have used prepared statements that make sure that input is stored as plain text and not interpreted by the database as SQL-code. Countermeasures against Session Hijacking could be to make sure the user is correcly authenticated on every page, and also to protect agains XSS since it can be used to steal session IDs.

Our application is vulnerable to Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Session Hijacking, and SQL-Injection. XSS and CSRF uses input fields on a site to upload html that includes sripts, like javascript, which loads to the page on request from other users who then reads and executes the script([8] page 734). CSRF is when the evil doers tricks a user to submit a forged HTTP request through for instance XSS. ¿¿¿¿¿¿¿ efef

It is also vulnerable to SQL-Injection. This is when the developer passes user input to a back-end database without checking to see whether it contains SQL-code. This is often given away by error messages, from which the evil doer may interpret to mount an attack ([8] page 120). Session Hijacking could be executed when the evil doers have gotten hold of a session or session ID from another user, and then act as that user.

If we look at the most critical web application security flaws [7], we can see that we didn't acount for many of them. To avoid XSS and CSRF we could have have correctly escaped the input to make sure that they are stored and displayed correctly. To avoid SQL-injection we could have used prepared statements that make sure that input is stored as plain text and not interpreted by the database as SQL-code. Countermeasures against Session Hijacking could be to make sure that the user is correcly authenticated on every page, and also to protect agains XSS since it can be used to steal session IDs.

5 Conclusions

References

[1] ITEM, *Web Security Lab*. Version from Feb 27th 2013, NTNU, 2013.

[2] RSA Laboratories, *How large a key should be used in the RSA cryptosystem?*. http://www.rsa.com/rsalabs/node.asp?id=2218, retrieved Mar 8th 2013.

[3] IETF.org, *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. http://tools.ietf.org/html/rfc6151, downloaded Mar 9th 2013.

[4] Apache.org, *Verifying Apache HTTP Server Releases*. http://httpd.apache.org/dev/verification.html, retrieved Mar 10th 2013.

[5] William Stallings, *Cryptography and Network Security - Principle and Practice*. Fifth edition, Prentice Hall, 2011.

[6] Charles P. Pfleeger, *Security in Computing*. Fourth edition,Prentice Hall, 2006.

[7] Owasp.org *Owasp Top Ten Project*.
https:www.owasp.org/index.php/Category:OWASP_Top_Ten _Project

[8] Ross Anderson *Security Engineering. Second edition*, Wiley Publishing, 2008.