Mittuniversitetet
MID SWEDEN UNIVERSITY

Final exam

DT116G Network Security

Daniel Bosk
daniel.bosk@miun.se
Phone: 060 - 14 8709

Lennart Franked
lennart.franked@miun.se
Phone: 060 - 14 8683

2013-01-09

Instructions

Carefully read the questions before you start answering them. Note the time limit of the exam and plan your answers accordingly. Only answer the question, do not write about subjects remotely related to the question.

Write your answers on separate sheets, not on the exam paper. Only write on one side of the sheets. Start each question on a new sheet. Do not forget to *motivate your answers.*

Make sure you write your answers clearly, if I cannot read an answer the answer will be awarded no points – even if the answer is correct. The questions are *not* sorted by difficulty.

Time 5 hours.

Aids Dictionary.

Maximum points 41

Questions 10

Preliminary grades

E $\geq$ 50%, D $\geq$ 60%, C $\geq$ 70%, B $\geq$ 80%, A $\geq$ 90%.

Questions

The questions are given below. They are not given in any particular order.

1. You have a web server configured with SSL. It was set up before your time, it has just worked so nothing needed to be changed. The SSL setup currently enables one cipher, and there has just been published a paper on a known-plaintext attack on this cipher.

(3p) (a) You should be worried sick and be sleepless in the nights with this server setup running. Why?

(1p) (b) What is your solution to the problem? The server has to be running.

> **Suggested solution** A known-plaintext attack tells you that if you have a known plaintext you can compute the secret key and use it to decrypt other messages, or parts of the same message, which is unknown [1, p. 45]. Having a known-plaintext attack on a cipher protecting the HTTP protocol is not good as you know that all HTTP-sessions are initialized by essentially the same text. This makes it particularly vulnerable to this kind of attack.
>
> The solution to the problem would be to disable this cipher. Since, for some reason, this was the only cipher enabled we have to enable another cipher-suite. There is no reason to have only one cipher enabled, enable all safe ciphers and modes of operation available in SSL.

2. When you communicate with your friends, to be sure that no one is trying to impersonate your friends you always sign your communication. Your friends want you to sign their newly generated keys. They send you these new keys in a message signed in with their old keys. Your friends use a simple MAC where

(1p) (a) MD5,

(1p) (b) SHA1, and

(1p) (c) SHA256,

respectively, are used as hash functions and the message digest is then signed with their old keys. Motivate the level of trust you would assign each new key.

> **Suggested solution** As MD5 is completely broken [1, p. 82], the key signed using MD5 as a hash function would be assigned no trust at all.
>
> SHA1 is safer than MD5, but there are known attacks on SHA1 [1, p. 84]. Thus the key signed using SHA1 should also have lowest possible trust-level.
>
> As there are no known attacks on SHA256, the key signed with SHA256 as the hash function can receive the same trust as the old key. (You cannot trust a key more than the key used to sign it.)

(2p) 3. Why would HMAC, using MD5 as a hash function, be better as a MAC than the usage example from previous question?

> **Suggested solution** According to [1, p. 89] HMAC is defined as
>
> $$\text{HMAC}_K(m) = H(K \oplus opad \,||\, H(K \oplus ipad \,||\, m)), \tag{1}$$
>
> where, in this particular case, the function H will be MD5. In comparison, the MAC used in question 2 was simply $H(m)$. Because the hash value will depend on the key as well as the text, and it will be hashed twice, you cannot easily find a collision.
>
> In the case of question 2 you do know the entire text which generated, and will generate, the MAC. Thus you can find a collision.

In the case of HMAC you do not know the entire text which generated and will generate the MAC. Thus you cannot alter the text by simply finding a collision for the text as in question 2 on the previous page, you would require the key K to do this.

However, in any case, MD5 should not be relied upon for any kind of security.

(6p) 4. Paranoia just struck you. Hence, you are thinking about implementing full-disk encryption on your systems. Because of fear for government trap-doors in already packaged software implementations you have decided to implement your own version. You know that you need to decide upon an encryption algorithm and a cipher block mode.

(a) Give pros and cons for using any of the following algorithms: RSA, 3DES, and AES.

(b) Give pros and cons for using any of the following cipher block modes: CBC, CFB, and CTR.

Suggested solution RSA is supposedly safe for large enough keys. Its encryption and decryption operations, however, are very slow, and the larger the key the slower its operations become. There are some interesting effects with using this algorithm for disk encryption. For instance some sectors may be only signed but not encrypted. Also a system could be run in write-only mode without being able to read from the disk. Similarily it could be run in read-only mode without being able to write comprehensible data to the disk.

3DES is fast in hardware, but slow in software, and a secure symmetric block cipher [1, p. 52]. There are currently no known attacks on this cipher [1, p. 52]. The key size is not a power of two, but the block size is, and hence it would work well with disk blocks with sizes being powers of two.

AES is probably the best choice for disk encryption. It is a strong block cipher which should be unbreakable for any forseeable future. It is fast in both hardware and software [1, p. 52], and should thus give good performance. Its block size is $256 = 2^8$ and thus a divisor of any disk-block size, which is commonly a power of two, just as for 3DES.

In the following discussion, let E be any block cipher encryption function and D the corresponding decryption function. Further, let K be the secret key.

Cipher block chaining (CBC) mode will provide more security as each block depends on all previous blocks [1, p. 65]:

$$C_i = E_K(C_{i-1} \oplus P_i), \text{ and} \tag{2}$$
$$P_i = D_K(C_i) \oplus C_{i-1}, \tag{3}$$

where C_i and P_i are the ith ciphertext and plaintext block, respectively, for $i \geq 1$. Let C_0 be the initialization vector (IV). To decrypt, however, you just need the current and its previous block, as illustrated in equation (3). The entire hard-disk cannot be encrypted from the first disk block (sector) to the last as a write to one block will make the next block undecryptable. A decryption and reencryption will result in that the rest of the hard-disk must be reencrypted due to the dependencies. A disk block by disk block encryption would be a well-working solution. When a disk block is modified the entire disk block is written to disk, hence we do not mind if we have to reencrypt the entire disk block instead of just a few of its ciphertext blocks.

Cipher feedback (CFB) mode would provide as much security as CBC. It works similarly as CBC:

$$C_i = P_i \oplus S_s(E_K(C_{i-1})), \text{ and} \tag{4}$$
$$P_i = C_i \oplus S_s(E_K(C_{i-1})), \tag{5}$$

where C_i and P_i are the ith block of ciphertext and plaintext, respectively, for $i \geq 1$, s is the block size and S_s is a function returning the s most significant bits of a bitstring. (Note that the block size s is not necessarily equal to the block size of the actual block cipher.) Let C_0 be

the IV here as well. The same also applies to CFB as for CBC; for encryption, if one block is changed all the following blocks must be reencrypted. However, the same solution as for CBC can be applied to make it work.

A problem with the solution for CBC and CFB is that it reduced the scheme to an ECB scheme if the IV is chosen as a constant value. Imagine HTML-files, most of them have the same initial text, meaning the initial ciphertext blocks will be the same. This can be solved by using the disk block number, inode number, or similar, as the IV instead of a constant value.

Let us also look at counter (CTR) mode. There are several possibilities. The most obvious one is to let the counter span the whole disk, i.e. you number each cipher block on the disk from the first disk block to the last. Then we have

$$C_i = E_K(i) \oplus P_i, \text{ and} \tag{6}$$
$$P_i = E_K(i) \oplus C_i, \tag{7}$$

where i is the counter, C_i and P_i are the ith ciphertext and plaintext block, respectively. The counter for a specific cipher block in a disk block can be computed and hence both encryption and decryption will work for individual disk, or cipher, blocks. The CTR approach has the advantage, over both CBC and CFB, that it does not have to execute the encryption function all the time. Both CBC and CFB uses the encryption function to encrypt data, whereas CTR uses it to encrypt the counter. Thus the encryption function only has to execute once, then the result can be kept in memory by the operating system while the disk block is actively being used. As the encryption operation encrypts the counters it is highly parallelizable, this is by design, in contrast to CBC and CFB.

However, the bad thing is that if an adversary can copy a block, say C_i – and we must assume this is possible – then if we update said block to C_i' we have the following:

$$C_i = E_K(i) \oplus P_i, \text{ and}$$
$$C_i' = E_K(i) \oplus P_i'.$$

All the adversary has to do now is to compute $C_i \oplus C_i'$ which turns into

$$\begin{aligned} C_i \oplus C_i' &= E_K(i) \oplus P_i \oplus E_K(i) \oplus P_i' \\ &= E_K(i) \oplus E_K(i) \oplus P_i \oplus P_i' \\ &= P_i \oplus P_i'. \end{aligned}$$

Thus we have the two plaintexts, which is highly structured and not random-looking, simply XORed together.

(3p) 5. A user connects to your web server using an SSL connection secured by an X.509-certificate. How can the user be sure that it is the correct server?

Suggested solution The X.509-certificate contains information about the server. It contains the name of the organization, the domain name it is valid for, among other things, and a public key [1, pp. 131–132]. This information is then signed by a certificate authority (CA).

This is accomplished by having the CA generate a message digest of the information in the certificate using a hash function, such as SHA256, and then encrypt this using its private key [1, p. 132]. Then anyone who wishes to verify the information computes the message digest of the certificate, then decrypts the hash using the public key of the CA, and finally compare the digests. As long as the hash function used is collision resistant, unlike some of those in question 2 on page 2, no one can change this information. The user must trust the certificate of the CA for the system to work, or trust a CA which in turn trusts this CA, i.e. has signed the certificate of this CA.

(3p) 6. For computer aided exams where the exam is taken by opening a particular webpage in the web browser, explain how you can use a rule-based NIDS such as Snort to detect cheaters. (Note that you do *not* have to provide syntactically correct Snort-rules which can be loaded into Snort without error.)

> **Suggested solution** First write rules which allow all server addresses required to sign in and do the exam. Then have rules matching all other addresses and report these as cheating.
>
> There is the problem of false positives with this solution. The operating system and other processes might make connections as well in the background. These connections will yield false positives and must be taken into account.
>
> We attempt to counter this by matching content with keywords from the subject matter. There are several problems with this solution as well. We have the problem of encrypted connections (HTTPS) where we cannot match any content, i.e. we have false negatives. We have even more problems with false negatives. Imagine having "secure connection" as a keyword, this will not match a student submitting the search query "connexion sécurisée" (which is French for the same thing).

7. Given the three scenarios below, name *one service* and *one mechanism* that you can use to ensure the correct type of security. Your answer should *only* address that particular scenario.

(2p) (a) You are sending a message to a friend and want to protect your message from passive attacks.

(2p) (b) You have developed a piece of software and published it on the web. After a couple of months you find out that someone is redistributing a modified version of your software that contains a trojan horse. You want the users to use the correct version of your software.

(2p) (c) When you arrive at work one day your boss calls you in to her office and informs you that she suspects that a disgruntled employee have been accessing sensitive information but she can't prove it. You want to prevent this from happening again.

> **Suggested solution** Part one: A passive attack means intercepting and reading the messages [1, p. 23]. Use a service which provides confidentiality, for example GPG, the mechanism used is encryption.
>
> Part two: We want to ensure our users that they have acquired a non-malicious copy of the software. Thus we need some service that provides integrity, such as GPG, where the mechanism is MAC. This MAC can be published anywhere as it is signed using our public–private keypair. Thus it can always be verified, and thus our software can be verified. No one can change the digest without invalidating the signature.
>
> There is an alternative approach, less flexible and scalable though – and strictly speaking also less secure. That would be to generate a message digest using a hash function such as SHA256, see question 2 on page 2 about different hash functions. This message digest can be published using our HTTPS-server, as all connections to this server are secured by a signed certificate indicating this is our server. This assumption is valid as long as our server remains trustworthy, i.e. not breached – as someone may have changed the message digest.
>
> The difference in security is that in the first case we can generate a signature and then lock-up the key in a vault – offline! In the second case we have the signing key online on a webserver. Thus, anyone with access to the webserver can assure our users that the malicious version is the valid one.
>
> Part three: Some service that provides access control and accountability, e.g. operating system access controls or an IPS. The mechanism is also access control and logging, respectively.

(2p) 8. Explain the difference between a circuit-level gateway and an application-level gateway.

Suggested solution The application-level gateway inspects the payload of the connections made to determine whether to allow or disallow the information sent or received [1, pp. 397–398]. This means that the application-level gateway needs to know how the application-level protocol works to be able to interpret it. With this granularity one can disallow certain features of an application protocol and still let the user use the rest of the application's features.

The circuit-level gateway is more or less a stateful packet-inspection (SPI) firewall with a possibility of authentication, and thus different rules may apply to different users – a feature not present in an SPI firewall [see 1, pp. 396–387, 398–399]. It shall be noted that both of these gateways can use authentication of users.

An appliation-level gateway operates on the application layer of the OSI model, whereas the circuit-level gateway operates on the transport layer. Note that they operate *on* these layers, not *in* them. They both require an application-layer protocol to support authentication.

(4p) 9. In a Kerberos realm a server will allow users to access its services on the basis that the user can provide a valid Kerberos ticket. Explain why the server should trust such a user.

Suggested solution The security of the Kerberos system is based on symmetric encryption. Let E_K be the encryption function of any strong symmetric cipher using the key K, and let conversely D_K be the corresponding decryption function also using the key K. In this account we give a simplified version of the Kerberos protocol involving the most important parts. It is covered fully by Stallings [1, pp. 126–128].

The client C generates a key K_C based solely on the client's password. The authentication server (AS) must also know the client C's password, this is to generate K_C to communicate with C. The client then sends a plaintext message, no encryption, to the authentication server (AS) containing its identification and the identification of a ticket granting server (TGS) (as well as a randomly generated value and some more information which we do not cover here), i.e.

$$ID_C \,||\, ID_{TGS}$$

is sent to the AS.

The AS responds with a ticket-granting ticket T_{TGS} to the specified TGS, where

$$T_{TGS} = E_{K_{TGS}}(K_{C,TGS} \,||\, ID_C)$$

is encrypted using the secret key K_{TGS} of the TGS – only the TGS (and the AS) can decrypt this message. In its response, the AS also includes a message

$$E_{K_C}(K_{C,TGS} \,||\, ID_{TGS})$$

encrypted using the client's secret key K_C. The ticket holds essentially a secret session key $K_{C,TGS}$ and the identity of C. As the AS knows the client's password it too can generate the key K_C, thus if C can generate this key it must know the password and C can then be considered authenticated. If C does not know the password it cannot generate the key K_C and hence cannot recover the key $K_{C,TGS}$ which must be used with the ticket T_{TGS} in communication with the TGS. The session key $K_{C,TGS}$ is now shared between the TGS and the client C.

Now C has a ticket-granting ticket T_{TGS} and the key $K_{C,TGS}$. If C is interested in using a service S, then it sends an unencrypted message containing

$$ID_S \,||\, T_{TGS} \,||\, A_C,$$

where

$$A_C = E_{K_{C,TGS}}(ID_C)$$

is called an authenticator (the authenticator also contains timestamps to counter replay-attacks). The TGS now reply to C with an unencrypted message containing

$$ID_C \,||\, T_S \,||\, E_{K_{C,TGS}}(K_{C,S} \,||\, ID_S),$$

where $T_S = E_{K_S}(K_{K_{C,S}} \,||\, ID_C)$ is the service-granting ticket. Note that only the TGS and service S can decrypt the ticket T_S.

Finally, C will send the ticket T_S and the authenticator A_C to the service S and they can now communicate using the session key $K_{C,S}$ – as S gets this key from the ticket and C already has it from the TGS. As only the TGS can create a valid ticket containing the identity ID_C of the client C and a uniquely generated session key $K_{C,S}$, then the client can be trusted to actually be C – and that C is authorized to use the service S.

10. In the context of IPsec, explain the application of the following functions:

(2p) (a) AH/ESP

(2p) (b) Tunnel-mode

(2p) (c) Transport-mode

(2p) (d) SA/SAD

Suggested solution

- Authentication Header (AH) provides message authentication in IPsec; Encapsulating Security Payload (ESP) provides authentication and payload encryption.
- Tunnel-mode means that the entire packet is encrypted and encapsulated within another packet.
- Transport-mode means that the payload is encrypted but the headers remain unencrypted.
- For each IPsec connection a Security Association (SA) is created. The SA contains information about what kind of encryption is used, keys, versions etc. The Security Association Database (SAD) contains all SAs.

References

[1] William Stallings. *Network security essentials : applications and standards*. Prentice Hall, Upper Saddle River, N. J., 4. ed. edition, 2010. ISBN 0-13-706792-5 (pbk.).