Questions from past exams

April 24, 2019

In this document, you will find the questions from passed exams we discussed in class during revision week. Please note that the solutions are not fully worked out.

1 Coin flipping

Alice and Bob are responsible for marking the Computer Security exam, but none of them wants to do it. They decide to toss a coin to determine who will be marking the exam. One problem is that one of them is travelling, so they only communicate over the telephone. Alice proposes that she flips the coin and telephone Bob with the result. This proposal is of course not acceptable to Bob since he has no way of knowing whether Alice is telling the truth when she says that the coin landed tails.

In this question we will examine 5 different protocols for distributed coin tossing. Nor Alice nor Bob should be able to cheat and force the coin to show heads or tails. For each of these schemes, say if it is secure or not. If you think it is secure justify your answer. If you think it is insecure give the attack which allows one of the two parties to obtain heads with probability more than 50%.

In the following, we let p be a 2048-bit prime number, g a generator modulo p, and H a secure (in the usual sense) cryptographic hash function know to all parties.

1. Alice randomly picks a such that $0 < a < p$ and sends to Bob $g^a \mod p$. Bob randomly picks b such that $0 < b < p$ and sends to Alice $g^b \mod p$. Both Alice and Bob can now compute $g^{ab} \mod p$. If $g^{ab} \mod p$ is even, then the coin flip is heads; otherwise, the coin flip is tails.

   Solution

   This protocol is unfair. Once Bob receives $g^a \mod p$ he can chose b such that $g^{ab} \mod p$ is even and thus the coin flip is heads. The probability of this attack is 1.

2. Alice randomly picks a such that $0 < a < p$ and sends to Bob $g^a \mod p$. Bob randomly picks b such that $0 < b < p$ and sends b to Alice. Alice then sends a to Bob who checks that it corresponds to what Alice sent in her first message. If $H(a||b)$ is even, then the coin flip is heads; otherwise, the coin flip is tails.

Solution

This protocol is fair. Due to the the hardness assumption on CDH. Alice commits to a without revealing it and Bob chooses b without knowing a. So none of the two parties has any advantage.

3. Alice randomly picks a such that $0 < a < 2^{128}$ and sends $H(a)$ to Bob. Then, Bob randomly picks b such that $0 < b < 2^{128}$ and sends $H(b)$ to Alice. Then, Alice reveals a to Bob and Bob checks that it matches what Alice previously sent. Bob also reveals b to Alice and Alice checks that it matches what Bob previously sent. If a and b have the same parity (ie. both parties picked an odd number, or both parties picked an even number) then the coin flip is heads. Otherwise, the coin flip is tails.

Solution

This protocol is unfair. Bob can just send back to Alice what he received from her. With probability 1 the coin toss will be heads.

4. Alice randomly picks a such that $0 < a < 2^{128}$ and sends $H(a)$ to Bob. Then, Bob randomly picks "even" or "odd" and sends that to Alice. Then, Alice reveals a to Bob who checks that it corresponds to what Alice sent earlier. If Bob did guess a's parity (ie. Bob said "even" and a is even, or Bob said "odd" and a is odd), then the coin toss is heads; otherwise, it is tails.

Solution

This protocol is fair. Similarly to (b), once Alice has sent $H(a)$, she has committed to a. She won't be able to later lie about that value because that would require finding a collision in a. Therefore, there is no way for Alice to cheat.
The situation for Bob is similar to (b). The preimage resistance of H means that, after seeing $H(a)$, Bob cannot learn a, and Bob cannot predict whether a is even or odd so he has 50% chance of guessing right.

5. Alice randomly picks a and k such that $0 < a, k < 2^{128}$. Alice computes $c = a \oplus k$ [the one-time-pad encryption of a, under key k] and sends c to Bob. Then, Bob randomly picks "even" or "odd" and sends that to Alice. Alice then reveals a and k to Bob who checks that it corresponds to what Alice previously sent. If Bob did guess a's parity (ie. Bob said "even" and a is even, or Bob said "odd" and a is odd), then the coin toss is heads; otherwise, it is tail.

Solution

This protocol is unfair. After Alice receives Bob's guess, Alice can check whether the outcome will be heads if she continues honestly. If yes, she can reveal her a and k. If not, she can pick $k' = k \oplus (0^{127} \cdot 1)$ and $a' = a \oplus (0^{127} \cdot 1)$ and reveal a' and k'.

2 TLS protocol

In class we discussed the TLS handshake protocol.

- The handshake begins when a client connects to a TLS-enabled server and sends a `client-hello` message requesting a secure connection and presenting the list of supported ciphers and hash functions.
- From this list, the server selects a cipher and a hash function that it also supports and notifies the client of the decision with the `server-hello` message.
- The server then provides identification in the form of a digital certificate. The certificate contains the server name, the trusted certificate authority (`T`) that vouches for the authenticity of the certificate, and the server's public encryption key. The certificate is signed under `T`'s signing key.
- The client confirms the validity of the certificate before proceeding (recall that `T`'s verification key is installed on the client's computer).
- To generate the session keys used for the secure connection, the client then encrypts a random number with the server's public key and sends the result to the server in the `client-key-exchange` message (which only the server can decrypt with its private key); both parties then use the random number to generate a unique session key for subsequent encryption and decryption of data during the session.
- The server sends a `finish` message which contains the hash of all messages sent in steps 1-5.

- The client compares the received hash and its own hash on his view of the messages sent in steps 1-5. If the hashes are different the client aborts the session, otherwise the client sends a `finish` message which contains the hash of all messages sent in steps 1-6. The session is now established.

1. Consider a company network that connects to the Internet through a border gateway. The administrator wants to spy on the company's employees and decrypt all their HTTPS communications (which goes through the gateway). To this aim, the administrator (`A`) generates a CA signing and verification key, and installs the verification key on all employees' computers. All employees' browsers will now trust any certificate signed with `A`'s signing key. Explain how the administrator can now decrypt any HTTPS connection established by an employee's browser, to an external HTTPS website. Explain this man-in-the-middle attack at each step of the protocol.

Solution

(a) Assume the employee tries to connect to `facebook.com`. His browser will send the `client-hello` message according to the protocol.

(b) The border gateway will just forward this to the server of `facebook.com`.

(c) The Facebook server will respond with the `server-hello` message according to the protocol.

(d) The border gateway will just forward this to the employee's browser.

(e) The server will send his public key together with a certificate for it signed by a trusted CA (let's say `T`'s key).

(f) The border gateway will intercept this, and instead send to the employees browser his own public key with a certificate signed under his own signing key.

(g) The employee's browser will check the certificate, and accept it as the administrator has taken care to install the corresponding verification key on the employee's computer as a trusted CA key.

(h) The employee's browser will generate a random number and encrypt it under the border gateway's public key received at the previous step.

(i) The border gateway will intercept this message, decrypt it as it knows the corresponding private key, and retrieve the random number. At this point the border gateway can compute the sessions keys that will be used to protect subsequent communications between the employee's browser and Facebook.

(j) The border gateway reencrypts this random number under the server's public key and sends it to Facebook.

(k) Facebook computes the hash of all messages sent so far

(l) The border gateway can compute the hash expected by the employees browser.

(m) The employee's browser checks that the received hash matches what he expects. It now computes the hash of his view of the handshake execution so far and sends it out in the finish message

(n) The border gateway can compute the hash expected by Facebook's server.

(o) The remote server checks that the received hash matches what he expects.

2. Assume an employee of the company connects to `acmebook.com`, and that his connection is being intercepted as described in question 1. Can Facebook's server detect this attack? Explain your answer.

   Solution

   No. This version of the TLS protocol only intends to authenticate the server and not the client. There is an extension of the protocol which allows mutual authentication but we do not consider this here.

3. Assume an employee of the company connects to `acmebook.com`, and that his connection is being intercepted as described in question 1. Can the user detect this attack? Explain your answer.

   Solution

   The user can look on a different network which CA signs Acmebook's public key. It can then check on his company computer the CA signing the public key of Acmebook when he connects to it from the comapny network. He will realise that the CA is different depending on the network he is on, and detect the administrator's Man-in-the-Middle attack.

4. To mount the man-in-the-middle attack you described in your answer to question 1, the border gateway issues and signs a certificate under `A`'s signing key for the external domain (for example for `acmebook.com`). Suppose the certificate issued by the gateway is identical to the one received by `acembook.com` in step 3, except for the CA's name, the public key of `acamebook.com`, and the CA's signature. Suppose also that the border gateway does not do any checks on the certificate sent by `acmebook.com`. Explain how an attacker outside the company can now mount man-in-the-middle attacks on the company's employees. Describe your attack at each step of the protocol.

   Solution

   An attacker sitting on the network between the border gateway and Acmebook's server can try and impersonate Acmebook. In step 3 it will provide a fake self-signed certificate for Acmebook's domain. Because the border does not make any check on the attacker's certificate, but replaces the attacker's certificate by his own which will be accepted by the employee's browser, the attacker can do the same attack as described in question 1.

5. A previous version of the TLS protocol (called SSL2) is subject to an attack called ROBOT: an attacker can iteratively query a server running SSL2 to decrypt previously captured sessions of the SSL/TLS handshake protocol. But many servers support both SSL2 and more recent versions of TLS, to allow connections from older clients. Explain how an active attacker can exploit this, to learn the session key established through the more recent TLS protocol. Describe your attack.

   Solution

   If the server uses the same public key for the executions both of SSL2 and the more recent TLS, he can use the SSL2 sessions as an oracle to decrypt TLS messages - in particular the one containing the random pre-master session key.

6. Suppose a remote server that supports both SSL2 and the more recent TLS 1.2, to allow to older clients which only implement SSL2 to connect to it. How could the remote server defend against the attack from question 5?

   Solution

   The attacker just needs to maintain SSL2 and TLS executions disjoint. He needs for this different public keys and certificates for the two protocols.

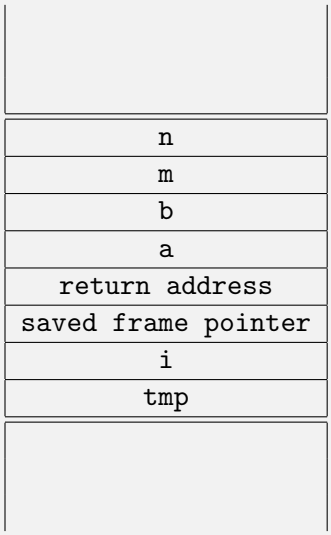
3 Memory safety

Consider the following C code.

```
void vuln(int a[], int b[], size_t m, size_t n) {
     for (size_t i = 0; i < m; i++) {
         int tmp = a[i];
         a[i] = b[n-i];
         b[n-i] = tmp;
     }
}
```

1. Depict the stack frame of a call to `vuln` before the execution of the `for` loop. You will assume that the code is executed on a 32-bit machine, with the size of the int data type being 4 bytes, and the size of the `size_t` data type being 4 bytes. `size_t` is defined by the C standard to be the **unsigned integer** return type of the `sizeof` operator.

Solution



Note that `a` and `b` may be pointing somewhere higher up in the stack, or somewhere in the heap.

2. This code is not memory-safe. Can you explain why and how could an attacker exploit this?

Solution

This code is vulnerable to a smashing the stack attack by overwriting a portion of the stack beyond the bounds of the `a` and `b` buffers. It can be exploited to mount a control hijacking attack, that executes arbitrary malicious code by taking the control flow of the application. In particular if `n` and `m` are greater than the size of `a` and `b` respectively.

3. Which of the following conditions on `a`, `b`, `m`, and `n` would ensure that `vuln()` be memory safe? If it is sufficient explain why. If not give an example of an input that would satisfy that condition but would be unsafe.

   (a) `a != NULL && b != NULL`

   Solution

   No memory-safe. Consider `a = [0]`, `b = [0]`, `m = 2`, `n = 1`.

(b) `a != NULL && b != NULL && m == 0 && n == 0`

Solution

Memory-safe. The loop is never executed, so nothing is read from or written in memory.

(c) `a != NULL && b != NULL && m == n`

Solution

Not memory-safe. Consider `a = [0]`, `b = [0]`, `m = 2`, `n = 2`.

(d) `a != NULL && b != NULL && m < size(a) && n < size(b)`

Solution

Not memory safe. Consider `a = [0,1,2]`, `b = [0]`, `m = 2`, `n = 0`.

4. Suggest a better condition. Your solution should ensure that `vuln` is safe, and be as general as possible. Explain your solution.

Solution

`a != NULL && b != NULL && m <= size(a) && n < size(b) && m <= n+1`.

4 CSRF attacks and defenses

1. In class we discussed Cross Site Request Forgery (CSRF) attacks. Explain what a CSRF attack is.

Solution

This is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.
Target: user who has an account on vulnerable server
Main steps of attack:

(a) build an exploit URL

(b) trick the victim into making a request to the vulnerable server as if intentional

Attacker tools:

(a) ability to get the user to "click exploit link"

(b) ability to have the victim visit attackers server while logged-in to vulnerable server

Key ingredient: requests to vulnerable server have predictable structure

2. Can HTTPS prevent CSRF attacks? Explain your answer.

Solution

No. The malicious code can issue https request. The victim browser will establish a TLS connection with the vulnearble server, and the malicious HTTP request will then be sent by the victim browser causing the unwanted state change.

3. Can session cookies solely prevent CSRF attacks? Explain your answer.

Solution

No. If the attacker tricks his victim to issue an HTTP request to the vulnerable server, the victim's browser will include all cookies for the vulnerable server in any HTTP request.

4. The `Referer` HTTP header contains the URL of the previous page visited. If you click on a link on a page, a GET/POST request is issued with the URL of this page as the value for the referer header. Explain how the `Referer` HTTP header can be used to prevent CSRF.

Solution

The remote server verifies that the hostname in the Referer header matches the target origin. This will preven CSRF attacks as the attacker cannot manipulate the referer header, and thus malicious requests will not have the target origini as hostname in the referer header. This method does not require any per-user state. This makes Referer a useful method of CSRF prevention when memory is scarce.

5. A common defense against CSRF attacks is the introduction of anti-CSRF tokens in the DOM of every page (often as a hidden form elements) in addition to the cookies. An HTTP request is accepted by the server only if it contains both a valid cookie and a valid anti-CSRF token in the POST parameters. Why and when does this prevent CSRF attacks? Explain your answer.

Solution

- Any state changing operation requires a secure random token (e.g., CSRF token) to prevent CSRF attacks
- Characteristics of a CSRF Token
  - Unique per user session
  - Large random value
  - Generated by a cryptographically secure random number generator
- The CSRF token is added as a hidden field for forms or within the URL if the state changing operation occurs via a GET
- The server rejects the requested action if the CSRF token fails validation

6. One way of choosing the anti-CSRF token is to choose it as a fixed random string. The same random string is used as the anti-CSRF token in all HTTP responses from the server. Does this prevent CSRF attacks? If your answer is yes, explain why. Otherwise describe an attack.

Solution

No. The attacker can just run an honest session with the remote website to learn the value of the anti-CSRF token, and include it in the malicious URL.

7. Suppose `bank.com` has a 'Transfer money' link which links to the following page:

```
<p> Transfer details:
<form action="/transfer" method="post">
   <input type="hidden" name="user" value="{{username}}">
   <input type="hidden" name="CSRFToken" value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEw">
   Recipient's account: <input type="input" name="account"/><br>
   Amount to transfer: <input type="input" name="amount"/><br>
   <input type="submit" value="Transfer now"/>
</form>
```

The web server replaces {{`username`}} with the username of the logged-in user who wants to do the transfer. The implementation of /transfer is given by the following pseudocode:

```
if validate_login_cookie(request.parameters['user'], request.cookies['login_cookie'])
then transfer_money(request.parameters['user'],
                    request.parameter['account'],
                    request.parameters['amount']);
     return '<p>Money successfully transfered</p>'
else return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
```

where `validate_login_cookie()` checks that the cookie sent by the browser is authentic and was issued to the specified username. Assume that `login_cookie` is tied to the user's account and difficult to guess. Is `bank.com` vulnerable to a CSRF attacks?

Solution

Yes. The server only checks the validity of the cookie and not the anti-CSRF token value sent with the HTTP request. The attacker can thus choose any value for the anti-CSRF token in his malicious code as it will not get checked by the server. This is exactly the twitter 2013 attack saw in class.