Paxson
Spring 2011

CS 161
Computer Security

Discussion 10

April 6, 2011

Question 1 ***TLS*** **(8 min)**

(a) TLS provides end-to-end authentication, integrity, and confidentiality guarantees. Is that enough to make online commerce safe and secure? Why or why not?

> **Solution:** TLS provides secure communication between a client and server, but was not specifically designed for online transactions. For instance, the browser checks the name in the certificate against the site's domain name, but this gives no assurance that the site is a bona fide merchant. Similarly, the online merchant has no way to check that the person making the purchase is authorized to use the credit card. Customer's can repudiate purchases, claiming their credit card number was stolen. In these cases, the credit card company usually pays the price.

(b) Some client-side implementations of TLS checked the name field of a certificate by reading up to the first null character. How could this be exploited?

> **Solution:** The client can be fooled into thinking an attacker's certificate comes from a legitimate site. For example, `yourbank.com\0.attacker.com` would be read as a certificate from `yourbank.com`. This can be used to launch a MITM attack. See [2] more details.

(c) The TLS protocol includes a closure alert signal (`close_notify`) that can be sent by either side to indicate the end of the connection. Why is this necessary? Couldn't the two parties just stop sending new messages when they are done?

> **Solution:** If the protocol doesn't include the `close_notify` signal, an attacker that can forge a TCP FIN packet can trick the recipient into thinking the communication is over. If the protocol does include the `close_notify` signal, the recipient of a forged TCP FIN signal will know the connection has not been legitimately closed by the other end. SSL v2 suffered from this vulnerability.

Question 2 ***HTTPS issues*** **(7 min)**

(a) HTTPS ensures that a coffee shop attacker cannot steal your cookies and hijack your session. MyBank.com decides to send the authentication cookie only over HTTPS to be secure against this attack. Say user Baldrick accesses his bank account by typing `mybank.com` in the URL bar while he is already logged in. Is he safe against the coffee shop attacker?

Solution: The threat model in this question is no longer a malicious website owner but instead a *network attacker*,[1] which is a threat that HTTPS aims to protect against. Recall that HTTP provides no protection against a network attacker who can passively watch the victim's traffic (violating confidentiality) or actively modify it (violating integrity). HTTPS defends against both of these threats by using TLS.

When Baldrick types `mybank.com` in the URL bar, his browser assumes he wants to open `http://mybank.com`, the *insecure* version of MyBank.com over HTTP. Since he is already logged in, the HTTP request will contain his authentication cookie which can then be seen by the attacker.

You might have hoped that Baldrick is safe because MyBank.com will redirect him to the HTTPS version of the site. But already the *first* request contains the authentication cookie and can then be used maliciously by the attacker.

To protect against this issue, the developers of MyBank.com need to set authentication cookies with the `secure` flag set. This flag ensures that browsers *only* send the cookie over HTTPS—and never over HTTP. Here is an example of the `Set-Cookie` header in a HTTP response with the secure flag:

```
Set-Cookie: auth=5f4dcc3b5aa765d61d8327deb882cf99; secure;
```

You might also think that this attack only works if Baldrick is already logged in, because otherwise he needs to type in his password, which MyBank.com's redirect will ensure only happens over HTTPS. Unfortunately, this does not work. A network attacker can spoof responses from MyBank.com to redirect Baldrick to a fake login page over HTTP. Most users will not notice that the redirect to HTTPS did not occur and will happily type in their password to the attacker.

[1] A network attacker in the coffee shop is typically a passive eavesdropper, aka. Eve, who can see everybody's communication and inject own traffic. Mallory is the active version of Eve is a man-in-the-middle (MITM) and can also block, delay, and drop connections. Unless otherwise noted, we assume that the network attacker has the capabilities of Eve.

(b) The `secure` keyword for cookies allows the server to tell the browser "only send this cookie over an HTTPS request." What should the browser do if it receives a secure cookie over HTTP?

Solution: The browser should ignore such a request. Otherwise, an attacker could set an authentication cookie for an HTTPS session. For example, the attacker can login and hand the received session cookie to the victim (over HTTP). If the browser accepts it, then the victim will be logged in as the attacker. This is a *session fixation* attack.

(c) In class we talked about Google Analytics. If you go to one of the TA's websites, you will notice that the code for including analytics starts is similar to:

```
var gaJshost = "http://www.google-analytics.com";
if ("https:" == document.location.protocol)
    gaJshost = "https://ssl.google-analytics.com";
```

The variable `gaJshost` is then used to decide which domain to download the Google Analytics JavaScript (`ga.js`) from. This JavaScript runs with full privileges of the including page; in this case the TA's home page.

Why is this condition necessary? What can an attacker do if it is not present and HTTP is always used?

Solution: If the script used always HTTP, the attacker could replace the `ga.js` script with his own, even if the TA's homepage was accessed over HTTPS. Since this script runs with the full privileges of the TA's homepage, XSS can occur and most of the advantages of accessing the homepage over HTTPS will be lost.

Since serving pages over HTTPS is more expensive than HTTP, Google Analytics checks if the main web page is served over HTTP and uses the "lighter" version if possible. If the user accesses the TA's page over HTTP in the first place, sending `ga.js` also over HTTP does not reduce the security of this scheme since a network attacker can already tamper with the HTTP session of the enclosing page.

(d) Consider what happens when you connect to a top site like `gmail.com` over HTTPS. GMail presents a certificate stating something similar to "This is the public key of GMail, and the corresponding private key is a secret known only to GMail." But in the real world, key compromise happens: what can happen if someone hacks into GMail servers and gets the private key?

> **Solution:** The whole game is lost. An attacker with the private key can present all the credentials needed by HTTPS and thus create a MITM attack. For example, the attacker can make you connect to his computer instead of GMail, and your browser will not show any warning.

(e) In view of the above problem, most browsers use the OCSP protocol. In essense, browsers ask the CA "Is this certificate still valid, or has it been compromised?" What do you think should happen when the CA responds with a "invalid"? How about when the CA is not reachable (say the request times out)?

> **Solution:** Clearly, if the CA tells you that the certificate is compromised, you should refuse to connect. It gets interesting in the case when you are unable to connect. At first glance, refusing to connect seems the right thing to do. But this has huge implications. It would mean that the availability of a HTTPS website depends on its CA: if the CA goes down, the website goes down with it through no fault of its own. This is unacceptable to most top websites. Do you think Google/Amazon would like to rely on VeriSign for their uptime? Currently, most browsers continue connecting and give a hard-to-notice warning.
>
> Think about what happens when you connect to AirBears. You are not allowed to access *any* website until you have logged in via CalNet. The CalNet page is served over HTTPS. But, your browser has no way to verify that the certificate presented by the CalNet website is not compromised. Can you really trust the CalNet login page?
>
> Adam Langley recently studied this in more detail [1].

Question 3 ***DNSSEC*** **(7 min)**

In class you learned about DNSSEC which uses certificate-style authentication for DNS results.

(a) In the case of a negative result (the name requested doesn't exist), what is the result returned by the nameserver to avoid dynamically signing a statement such as "`aaa.google.com` does not exist"? (This should be a review from lecture.)

Solution: The nameserver has a canonical ordering of all record names in its zone. It creates, off-line, signed statements for each pair of adjacent names in the ordering. When a request comes in for which there is no name, the nameserver replies with the record that lists the two existing names just before and just after where the requested name would be in the ordering. This proves the non-existence of the requested name. The reply is called an NSEC record.

(b) One drawback with this approach is that an attacker can now enumerate all the record names in a zone. Why is this a security concern?

Solution: Revealing this information could aid in other attacks. For example, the names in a zone could be used as targets when probing for vulnerable servers.

(c) How could you change the response sent by the nameserver to avoid this issue?

HINT: One of the crypto primitives you learned about will be helpful.

Solution: The nameserver can create a list of the hash of each name, ordered by hash, and sign pairs of adjacent hash values. When a request comes in, the nameserver will compute the hash of the requested name and return the signed record that lists the two existing hashed names just before and just after where the hash of the requested name would be. The client can then compute the hash of the requested name and see that it would be between the two values returned by the server if it existed. This type of record has been proposed as a replacement to the NSEC record in DNSSEC; it is called an NSEC3 record.

References

[1] Adam Langley. Revocation doesn't work, 2011. http://www.imperialviolet.org/2011/03/18/revocation.html.

[2] Kim Zetter. Vulnerabilities Allow Attacker to Impersonate Any Website, July 2009. http://www.wired.com/threatlevel/2009/07/kaminsky.