

**1. TLS**

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

**Answer:**

- (a) TLS provides secure communication between a client and server, but was not designed specifically 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) 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 <http://www.wired.com/threatlevel/2009/07/kaminsky/> for more details.
- (c) 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.

**2. SYN Cookies** The Wednesday before spring break, Professor Wagner described SYN cookies, a technique for choosing TCP initial sequence numbers (ISN's) that helps mitigate denial of service (DOS) attacks. Specifically, SYN cookies help prevent SYN flooding with spoofed source addresses.

- (a) When using SYN cookies, a host computes part of its ISN as a MAC. Would using digital signatures instead offer any security advantages?
- (b) Why might signatures be difficult to use for this purpose? See if you can come up with more than one reason.
- (c) SYN cookies do not completely prevent the type of storage-based DOS attack they address. Can you quantify the extent to which they mitigate it?

- (d) One type of email spam is known as “bounce spam”. In this case, an attacker sends spam to a non-existent address while forging the “from” address to be that of the victim. The victim later receives the spam when an innocent mail server bounces it to the address listed in the “from” header. Can you think of a technique similar to SYN cookies that could be used to mitigate this type of spam?

**Answer:** Recall how SYN cookies work:

```
Client -----SYN(s)-----> Server
Client <---SYN(x) | ACK(s)---- Server
Client -----ACK(x)-----> Server
```

The value  $x$  is the server’s 32-bit initial sequence number, and it is computed as follows. The five most significant bits are a low resolution time stamp (typically it increments every 64 seconds), the next three bits specify one of eight maximum segment sizes (just another piece of state associated with a TCP connection), and the final twenty four bits are a MAC. The MAC is computed over the two IP addresses and TCP ports and the 5-bit timestamp.

After receiving the ACK packet, the server subtracts one from the sequence number to obtain the original value  $x$ , checks that the timestamp is current, and verifies the MAC. If all this is fine, the server creates a new entry in its table of TCP connections, stores the maximum segment size, and proceeds normally from that point on.

- (a) Not really. Since the party that generated the ISN will also verify it (in the ACK packet), we might as well only have a single key. So the usual advantage of signatures (being able to verify with a public key rather than the signing key) isn’t useful.
- (b) There are at least a couple reasons.  
Most importantly, signatures are too long to fit in 32 bits, much less 24 (the shortest modern signatures are about 160 bits). Of course, typical MAC algorithms will produce a longer result as well. A MAC, however, can be truncated while possibly still offering some measure of security. This will not work with signatures, since they are fed into a separate verification algorithm rather than being regenerated for verification. In general, truncating a MAC could completely compromise its security, but in the case of, say, HMAC, the security can be assumed to scale gracefully as the length is reduced.  
Another reason why signatures would be difficult to use is that they take much longer to generate. This would offer another way of mounting a DOS attack (this time based on computation time rather than storage).
- (c) An attacker can still make up ACK( $x$ ) packets to send to the server by just picking the 24-bit MAC (the only unpredictable part) randomly. If they guess right, the server will store TCP state as before. The attacker will guess right with probability  $\frac{1}{2^{24}}$ , so the magnitude of their attack will be reduced by a factor of about 16 million.
- (d) One way would be for a user’s mail client to keep track of all messages sent, and ignore any bounced messages it receives that don’t correspond to a previously sent message. However, this approach suffers from the problem that users may have several different machines, so it could be complicated to ensure they all know about all messages sent from any location. The solution is to compute a MAC over each sent email and include it as an additional header. The key to the MAC can just be a password that the user remembers, so there is no need to synchronize the state of their machines.

### 3. Limiting program permissions

- (a) What permissions does the calculator application on your computer have?
- (b) What permissions does it need?
- (c) List some powerful permissions that it has but does not need.
- (d) How could this be a problem?
- (e) Imagine you have a way to hook into every system call a program makes. How could you use this to address the problem?
- (f) What problems might you run into when you deploy this?
- (g) How could this capability of hooking into system calls be used for malice instead of good?

**Answer:**

- (a) It can do anything your user account can do.
- (b) It only needs to be able to perform computation and draw to the screen.
- (c) Create network connections, delete files, spawn processes, etc.
- (d) It violates the Principle of Least Privilege. If there is a bug in the application, it can do a lot more than it ever needed to do in the first place. For example, it could read all your sensitive files and send them to an attacker.
- (e) Write a policy that allows only certain system calls for a given program. For example, for the calculator app, allow only system calls related to drawing to the screen. This is called *system call interposition*.
- (f) You have to write a correct policy for every program that you want to sandbox. It is hard to write correct policies.
- (g) Rootkits do this! Rootkits are pieces of malware that hide evidence that they are present on the system. They usually hide themselves by hooking into system calls and cleaning up any evidence of their existence returned by any system calls. For example, if a user lists the contents of the directory that contains support files for the rootkit, the rootkit will modify the results of the system call to remove its support files from the list.

**4. DNSSEC**

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.)
- (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?
- (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.)

**Answer:**

- (a) 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) Revealing this information could aid in other attacks. For example, the names in a zone could be used as probable email addresses for spam.
- (c) 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.