April 25, 2016

Question 1  Videogames Cheating  

(a) Suppose the client of an online game calculates the player’s current position locally and sends it back to the server as a pair (x, y). What can go wrong?

**Solution:** You can teleport to any place in the game.

(b) How can you avoid this sort of problem?

**Solution:** Let the server be authoritative. The client should only send back user inputs.

Question 2  Certificate Transparency

One problem with current certificate management structure is that it lacks transparency. We don’t know what certificates are issued by the CAs, so it is hard to spot a problematic certificate.

(a) Suppose Mallory compromises a certificate authority, so he can issue any certificate that he wants (Or maybe the certificate authority is just lazy). What can he do to pretend to be google.com?

**Solution:** He can issue a certificate for google.com with his own public-key. Even though google.com already has a certificate, he can still generate another valid certificate for it.

(b) What are some ways to mitigate this problem? (Hint: We want to make it easy for people to know exactly what certificates have been published, so we should use something like a log here.)

**Solution:** We can require every CA to append the certificates it creates to a public accessible certificate log. When the user receives a certificate, besides the usual check of validity, the user can also check that the certificate is part of the certificate log. Now since all the certificates are accessible through the log, it is easy to spot fraudulent certificates and certificates for phishing sites.
(c) The solution you thought of above is actually called Certificate Transparency. It relies on certificate logs. What properties should the logs have in order to be secure?

Solution:

1. They should be append only. It shouldn’t be possible to modify or delete existing certificates.
2. It should be easy to check for the existence of a certificate.
3. It should be hard for attacker to tamper with the content of the log.
4. It should be easy to verify the validity of the log.

(d) How can we construct the certificate logs to have these properties? (Hint: Think about project 2.)

Solution:

We can use a hash tree for the log. Periodically, the log server appends all of the newly submitted certificates to the log. It does this by creating a separate tree hash with the newly acquired certificates, and then combining this separate hash tree with the old hash tree to form a new hash tree. For someone monitoring the log server, they can easily check the later version includes everything in the earlier version.
Question 3  Detecting packers  (25 min)

Most antivirus softwares today are still heavily signature-based. This means that they can only detect malware with known malicious code. To make malware harder to detect, malware authors often “pack” their program. This question will help you understand what “packers” are as well as some ways to detect them.

(a) Given a database of existing malicious code, what is the simplest way of detecting malware?

Solution: You can compare the similarity between known malicious codes against the contents of other files.

(b) Suppose Company Unsafe releases their new antivirus software Insecure. Insecure adopts the naive approach is the previous part. Mallory immediately realizes that he can easily make any piece of malware undetectable by Insecure. What is Mallory thinking of? (Hint: Think about buffer overflow).

Solution: Given a piece of existing malware, Mallory can first use compression to obfuscate it. Then Mallory puts the obfuscated code inside another program he wrote. When this new program executes, it automatically decompresses the content of the original malware and loads it into memory. After this, his new program can directly execute the loaded code. The term packing refers compressing an executable into another self-decompressing executable and a packer is a program that automates this process.

(c) From the above example, we can see that packed programs usually have write-then-execute sequences. If we can detect this sequence, we can potentially detect the newly generated code inside memory. After taking CS161, Alice decides that she can detect this sort of packers herself. She decides to do it by page management. The malware expects the memory to be both writable and executable. However, in her system, when an executable runs, it would mark the executable region as read-only and executable, and everything else as read-only and non-executable. She also swapped out the page fault handler in her system to her custom program. How can she use this setup to detect write-then-execute sequences without damaging the functionality of normal programs?

Solution: When a packed program executes, it would write into read-only memory and cause a page fault. Her page fault handler would record the page number of this page to a list and change the permission of this page to writable but non-executable. This allows the write to continue. Then when the packed program executes the decompressed code, it would raise another page fault. Her page fault handler would find that the page number of this page matches the list of written pages. Her page fault handler then could record the content of
that page and only continue execution if it is safe.