Paxson
Spring 2011

CS 161
Computer Security

Discussion 3

February 9, 2011

Question 1 ***DNS*** **(7 min)**

Recall that in a *blind* DNS spoofing attack, the attacker tries to guess the identification number of the DNS request sent by the victim.

(a) For the following, assume that the victim's DNS resolver cache does not contain a record for the domain the attacker is targeting:

i. In a blind spoofing attack, the attacker must know when a DNS request is about to be made by the victim so that he can respond with his attack responses. How might the attacker know when the victim is about to make a DNS request?

ii. What can an attacker do if he successfully gets a victim to believe his bogus DNS mapping?

iii. How can an attacker avoid having to carry out this attack for every request made by the victim?

(b) Now assume that the victim's DNS cache has a genuine `NS` record for the domain the attacker is targeting.

i. Can the attacker still be successful at poisoning the `A` records for some of the names belonging to the domain?

ii. Can the attacker poison the `NS` record of this domain? If yes, how?

Solution:

(a) i. Lure a victim to your web site, which contains a link (or many links) to the site whose DNS record you would like to attack (e.g., `google.com`). When you see that a victim has contacted your site, you know they are about to make a DNS request for `google.com`, so you initiate the attack at this point.

ii. He is in control of the content of any request made to a hostname whose DNS record has been successfully spoofed. For example, if the attacker managed to get the victim to accept a bogus DNS record for `google.com`, then any subsequent request to `google.com` will actually go to a domain of the attacker's choosing. He might get you to reveal your password to the fake `google.com` at that point.

iii. DNS records are cached, so the attacker might set a long TTL (time-to-live) so that the bogus DNS record will live in the cache for a long time. The attack will succeed for as long as the bogus DNS record lives in the cache.

(b) i. The attacker can still poison `A` records of subdomains. For example, if the victim has an `NS` record for `foo.com` in the cache that points to `ns.foo.com`, then the attacker can simply force the client to lookup random subdomains such as `qt0378hr.foo.com`. The victim only knows who to ask, namely `ns.foo.com`, but does not know the answer for that query.

ii. The attacker can also poison the `NS` record using the Kaminsky attack. In this case, the attacker can specify the same `NS` record in the *authority section* of the reply to tell the victim that `ns.foo.com` is responsible for the domain `foo.com`. Moreover, the attacker also provides an `A` record for `ns.foo.com` in the *additional section* pointing to a bogus address. This extra information is also known as the *glue record*.

Question 2 ***IP Spoofing*** **(7 min)**

You are the network administrator for a large company.

(a) Your company will be held liable for any spoofing attacks that originate from within your network. What can you do to prevent spoofing attacks by your own employees?

(b) Is there anything you can do to prevent others from sending your employees spoofed packets?

Solution:

(a) Inspect the source IP address of all outgoing packets. If a packet has an address from outside the range assigned to your network, block the packet. This is called *egress filtering*.

(b) It is highly dependent on how your system is setup. If you have many links to other networks from your company, and you know what IP addresses are associated with those networks, if a packet comes in on a link and it does not match with the IP addresses associated with that network, you can filter out those packets. This is called *ingress filtering*. However, if you only have one link, this does not work. Furthermore, it is not always possible to associate a set of IP addresses with a link connection. ISPs can do this for traffic coming from their edge customers (on separate autonomous systems).

Question 3 ***Sniffer Detection*** **(7 min)**

As the security officer for your company, your network monitoring has observed a download of a "sniffer" tool. This tool passively eavesdrops on traffic, and whenever it sees a web session going to a server in a `*.yahoo.com` domain, it extracts the user's session cookie. It then uses the cookie to create a new connection that automatically logs in as the user and dumps their `*.yahoo.com` settings.[1]

You become concerned that one of your employees may have installed a network "tap" somewhere among the hundreds of links inside your building, and will use it to run this tool. How might you determine whether such a sniffer is in operation?

Solution: One approach is to send customized web traffic along each of the network's links, as follows. The traffic connects to a remote server with the address `A.B.C.D`, which you know none of your systems would normally connect to. Because the sniffer needs to determine whether a given connection goes to a `*.yahoo.com` domain, but the forged traffic only contains an IP address, it will need to perform a *reverse lookup* to find the hostname that corresponds to `A.B.C.D`. By monitoring for such lookups, you can determine that a sniffer appears to be in operation, since none of your normal systems should have reason to make the lookup.

Another approach is again using customized web traffic, this time with connections to a `*.yahoo.com` domain. You "seed" the connections with a unique (fake) session cookie and monitor your outbound traffic for any *additional* connection that uses the fake cookie.

Finally, you can *passively* detect cookie stealing without injecting additional traffic. This works by keeping track of the cookies: if the same cookie is used by more than one machine, someone else reused the cookie. If you are interested in the technical details, you can read Matthias' blog post about this approach [1].

References

[1] Matthias Vallentin. Taming the Sheep: Detecting Sidejacking in Bro, October 2010. http://matthias.vallentin.net/2010/10/taming-sheep-detecting-sidejacking-in.html.

[1] Capturing and reusing session cookies is known as *sidejacking* or *session hijacking*.