CS 161 **Spring 2016**

Homework #2

Nikhil Unni

1. (a) Since the config servers are open to the world, an attacker can try logging into Cheaporouters with the default user/pass ("admin" and "password"). From there, he can just set the primary DNS server to a malicious DNS server that the attacker owns. From there, the attacker can conigure his DNS server to route calls to google.com to hackrzsrch.com. Because the Google queries are sent via http, there's no certificate validation needed. And because the actual search is baked into the URL via GET requests, we can map it to the correct search on Hackrzsrch.

   Concretely, the attacker would go to the link : http://yourrouter/set?dns=⟨ IP OF MALICIOUS DNS SERVER ⟩.

   (b) The attacker can merely connect to the open WiFi network once they drive within range of the network. Once they're on the network, they can connect to the config server and execute the attack as described in part a. The attacker could even do this attack with a mobile phone if he or she wanted.

   (c) A computer infected with malware could potentially be able to make network calls. If this is the case, the moment Joe physically connects his infected laptop to the default router, the malware can now send requests from the internal network, and can immediately set the default DNS server on the Cheaporouter to the malicious DNS server.

   (d) The website can send HTTP requests to the config server (and try to log in with the default user/pass), and change the default DNS server. This does not violate the cross-domain policy of most browsers – it is merely Javascript run **from** Sam's computer (perhaps some AJAX call from JQuery). So from the perspective of the Cheaporouter, it just looks like the request is originating from Sam, who is on the internal network.

2. (a) $\mathcal{A}$ can just send one ping to $\mathcal{P}$ at the start, and another ping at the end of the one-minute window. If the two responses have an ID field that's greater than one apart, then $\mathcal{A}$ knows that $\mathcal{P}$ has been sending packets to hosts other than $\mathcal{A}$ in that window. In fact, if the two IDs that come back to $\mathcal{A}$ are m, n respectively, then $\mathcal{A}$ knows that the precise number of packets sent to other hosts is $m - n - 1$ (assuming that $\mathcal{A}$ is not sending anything else to $\mathcal{P}$ in that window).

   (b) $\mathcal{A}$ can send a SYN packet with a spoofed IP header to $\mathcal{V}$. The packet will have an IP header with the source IP address of $\mathcal{P}$. If n is a closed port, then $\mathcal{V}$ will send RST to $\mathcal{P}$, who will in turn not respond, and thus its global ID will remain unchanged. If n is an open port, then $\mathcal{V}$ will send a SYN/ACK to $\mathcal{P}$, who will respond with an RST, thus incrementing the global ID by 1.

   $\mathcal{A}$ can exploit this by pinging $\mathcal{P}$ before and after sending the SYN to $\mathcal{V}$, to find the global ID of $\mathcal{P}$ before and after, to see if it increased by one or not. There's a chance that $\mathcal{P}$ might be sending out packets to other sources in that time frame... but this technique can be replicated 1000s of times until there's a timeframe where this is the only network activity.

   (c) $\mathcal{P}$ can simply maintain different ID namespaces for different destination IP addresses. For example, a hash of the dest IP could be the first 8 bits, and the last 8 bits could just be a running counter. This way servers from one IP address could not query $\mathcal{P}$ for information about other IP addresses.

3. (a) Right after resetting Brewed Awakening's network, it's assumed that all devices' ARP caches are flushed. At this point, devices in the WiFi network will send out ARP requests to one another. If we can get on the Brewed Awakening's network (by first buying a cup of coffee), we can figure out the IP/MAC of the wireless Access Point (AP). Also, once we're on the network, we can send out malicious ARP responses. Since we don't know which specific device is Nick's, we can just send malicious ARP responses that point to our own laptop's MAC address to any ARP request.

   Since all devices in the network think that my laptop is the AP, we can passively sniff packets, and subsequently send them to the actual AP or device to allow for normal traffic. This positions us to be a MITM of all traffic on the Brewed Awakening network. Now, we simply wait until we find HTTP Piazza traffic, and we can parse out the unencrypted login/password. This will eventually give us Nick's login info, which we can use to post bogus answers.

   (b) On my laptop, I can maintain a table of Piazza post ids that I've bogus-ly answered so far (posts have id's in the DOM that are different than the cid in the URL, but I believe they have a 1-1 correspondance). Then, whenever I'm sending back Piazza traffic to devices on the network (or just to Nick, since we would've gotten his IP address back when we scraped his login info), I can edit the actual unencrypted contents by wiping out any information in the packets about posts that are in my blacklist. If I exclude all such posts in the sidebar, hopefully Nick will not be able to find them.

   (c) I can alter the routing table to forward all requests to 23.23.249.136 to my own laptop's MAC address. If I passively sniff all http packets to Piazza from the Brewed Awakening WiFi this way, I can wait until Nick logs in with his credentials to Piazza, which I can use to impersonate him.

   (d) From part c., I can sniff all outgoing packets to Piazza. If I wait long enough, I can find HTTP traffic that comes from Nick's Piazza account (either from the HTTP requests themselves, or by noting whenever Nick posts on Piazza through my own Piazza account) and I can get his MAC address/IP as well. With this, I can be a full MITM between Piazza and Nick, and I can alter the routing table so that any packets that are meant for Nick's IP are forwarded instead to my own laptop.

   (e) Even though I've started sniffing the packets in the middle of Nick's TCP session, I can still examine the ingoing/outgoing packets to find the current sequence number, relevant ports, checksum, etc. With this information, I can construct my own TCP packet with the correct sequence number that I can inject into the session that the server will accept. From here, we can simply send an HTTP request to log out (/logout perhaps), and then Nick will once again have to send his login credentials through HTTP, which we can extract.

   Alternatively, if we're not allowed to inject any packets, perhaps we could steal Nick's session cookie, which amounts to the same thing as stealing his login credentials. With the session cookie, all our requests would look like they're coming from Nick, although the session would eventually expire.