

Homework 2

Due: February 11, 2009, 10:00 PM EST

Problem 1

1. In order to spoof the cs.brown.edu domain, an attacker would first send a DNS lookup request for a fake address in that domain (e.g. ldf234.cs.brown.edu) to a nameserver. Since the nameserver wouldn't have an IP cached for that address, it would send out a request to another nameserver. The attacker would then try to flood the original nameserver with forged responses from the second nameserver which essentially say "I don't know where ldf234.cs.brown.edu is, but you should ask [attacker's IP]". These responses come with random transaction IDs. The attacker then repeats this process using a new random address (e.g. def456.cs.brown.edu) until it manages to get one of its forged responses accepted by the first name server (this happens when the randomly chosen transaction ID matches the original one). Once the original nameserver accepts one of the forged responses, it then asks the attacker's computer for information about the fake cs.brown.edu address. The attacker's computer responds (it doesn't matter with what) and the nameserver then stores the attacker's IP address as the DNS server to ask about cs.brown.edu. The attacker will then receive all requests for cs.brown.edu addresses until the cache's TTL removes it. This gives the attacker the power to redirect all cs.brown.edu address wherever he chooses.
2. A larger value for the TTL of replies to DNS queries does not prevent the above attack, because the TTL only applies to the specific domain originally requested. Namely the TTL is re-initialized whenever there is a request for a new domain.
3. The attacker has $1,024/65,536 = 1/64$ chance of getting his answer in. Therefore the probability of success for k fake page requests is $1 - (63/64)^k$. This is 99% for $k = 293$.
4. Possible answers are as follows:
 - (a) Creating fake links to attackers domains over the Internet.
 - (b) Injecting some javascript code, which sends requests to his domain, to web pages.
 - (c) Generating spam mails from attackers domains, the email clients will make DNS requests for these names.
 - (d) Including links as any kind of externally referenced objects (e.g., pictures)
5. Two possible answers for this question are as follows: (a) Apart from the transaction ID, we can also randomize the port of the request. That increases the message space that makes guessing of transaction IDs very difficult; (b) authenticate the answers that are received, by using some kind of certification authority. That however changes the architecture of the system.

Problem 2

1. You can create a series of fake sessions by going through the steps that legitimate

hosts controlled by your network would have. The steps would be:

- Create a syn message from a host in your network and send it to the target. It helps if this is an IP address in your network that you know is not in use.
 - Intercept the SYN-ACK and construct an appropriate ACK, i.e., increment the sequence number and send it back.
 - The server allocates resources for this spoofed connection and keeps it open until the connection times out (traditionally about 3 minutes).
2. The connection queue is only 8 slots so you need to maintain that many spoofed connections at all times. If your connection timeout is 3 minutes then you need to open up 8 connections every 3 minutes. This parameter is configurable, so the real answer is 8 per s where s is the configured timeout for the server. It may require an experiment to determine what s is for your target.
 3. Possible defenses for this could include limiting the number of concurrent connections from a specific range of IP addresses or subdividing the queue (i.e. addresses from 1-64.*.* can only occupy slots 1-2, 65-128.*.* can only reside in 3-4 on the queue, etc.). Another thing you can do to potentially limit the damage of this attack is lower the connection timeout of servers that have created unused connections in the past (although this goes against most quality-of-service recommendations).

Problem 3

1. When AIMD is used to increase the congestion window size it calculates the new congestion window with the following formula:

$$w = w + b/w$$

This makes it so that as the congestion window gets bigger it grows at a smaller rate per ACK. b is usually a very small number such as 0.1. So each time the congestion window expands it will increase by a successively smaller number. Thus, even with repeated optimistic ACKs this attack still takes many iterations to be effective.
2. This approach will identify bad receivers by tempting them to send ACKs when they should not. Once a bad receiver is identified we can cut him off from service. The downside to this approach is that we have to diminish quality of service to really fast clients because they will get bad transmissions once in a while from dropped segments.
3. Traffic shaping would be good for limiting the damage of the worst case. If a server is given an artificial cap to how fast it can transmit to a given user this will prevent it from choking other clients. On the downside this all limits how fast it can go to servers that are close to it and a hard-coded limit would be difficult to arrive at (trading off between servers that are far away and attacking by only sending the ACKs and servers that are closer and will have fast request rates). This primarily defends against attackers that have access to a limited number of hosts.
4. One could decide what a client's transmission speed should be by pinging its gateway or (if that fails) a nearby server (i.e. one that shares the first 3 octets). From this we could divide the round trip time by two and determine that this is the maximum speed we want to transmit at (or slightly less). There may be other solutions to this problem with various degrees of correctness.

Problem 4