Network Security II

Question 1 ***NSEC***

In class, you learned about DNSSEC, which uses signature chains to ensure authentication for DNS results. Recall that in the case of a negative result (the name requested doesn't exist), the nameserver returns a signed pair of domains that are alphabetically before and after the requested name.

For example, suppose the following names exist in `google.com` when it's viewed in alphabetical order:

```
...
a-one-and-a-two-and-a-three-and-a-four.google.com
a1sauce.google.com
aardvark.google.com
...
```

In this ordering, `aaa.google.com` would fall between `a1sauce.google.com` and `aardvark.google.com`. So in response to a DNSSEC query for `aaa.google.com`, the name server would return an NSEC RR that in informal terms states "the name that in alphabetical order comes after `a1sauce.google.com` is `aardvark.google.com`", along with a signature of that NSEC RR made using `google.com`'s key.

(a) DNS attacks we previously saw in class caused victims to unknowingly visit an attacker-controlled domain. Since receiving a negative result back from a nameserver causes a client to raise an error rather than visit a domain, why is a signature still necessary? What attack becomes possible without one?

> **Solution:** This prevents a DoS attack. If signatures weren't provided with the negative result, then object security is lost. An adversary can pretend to be a nameserver and return negative results for every user query to keep them from visiting any websites. This won't be detected as there's no longer a signature to check.

(b) A startup, `ThoughtlessSecurity`, decides to modify DNSSEC to only return a signature of the *requested domain* on a negative result. They claim that this change will drastically reduce the packet-size of a negative result.

A company implements `ThoughtlessSecurity`'s product on their nameserver. What attack is now possible? Specify exactly how an attacker could execute this attack.

Solution:

A DoS attack is now possible. An attacker can query random domains that have a high probability of not existing. This will cause the nameserver to constantly compute signatures on the fly which will lead to server exhaustion if enough queries are sent.

Note that the queries need to be unique, as negative results are generally cached.

(c) Using the originally-described DNSSEC protocol, describe how an attacker can enumerate all domain names

Solution:

An attacker can send any query for a domain that doesn't exist. Upon receiving the request, they learn two domain names. They can then send requests for non-existent domains that are alphabetically directly before and after those domain names to learn two more domain names. They can continue this process to enumerate all the names.

(d) A new startup, `ThoughtfulSecurity` wants to use a hash function to hinder this enumeration process and start by taking the hash of each existing domain. How can they use hashes to provide authenticated negative results?

Solution: Instead of sorting on the domains, the sorting is done on *hashes* of the names. For example, suppose the procedure is to use SHA1 and then sort the output treated as hexadecimal digits. If the original zone contained:

```
  barkflea.foo.com
   boredom.foo.com
    bug-me.foo.com
   galumph.foo.com
   help-me.foo.com
perplexity.foo.com
     primo.foo.com
```

then the corresponding SHA1 values would be:

```
  barkflea.foo.com = e24f2a7b9fa26e2a0c201a7196325889abf7c45b
   boredom.foo.com = 6d0edfd3efa5bf11b094cb26a7c95a3bd5e85a84
    bug-me.foo.com = 649bb99765bb29c379d935a68db2eebc95ad6a29
   galumph.foo.com = 71d0549ab66459447a62b639849145dace1fa68e
   help-me.foo.com = 1ed14d3733f88e5794cd30cbbef8cc32fa47db2a
perplexity.foo.com = 446ac4777f8d3883da81631902fafd0eba3064ec
     primo.foo.com = 8a1011003ade80461322828f3b55b46c44814d6b
```

Sorting these on the hex for the hashes:

```
   help-me.foo.com = 1ed14d3733f88e5794cd30cbbef8cc32fa47db2a
perplexity.foo.com = 446ac4777f8d3883da81631902fafd0eba3064ec
    bug-me.foo.com = 649bb99765bb29c379d935a68db2eebc95ad6a29
   boredom.foo.com = 6d0edfd3efa5bf11b094cb26a7c95a3bd5e85a84
   galumph.foo.com = 71d0549ab66459447a62b639849145dace1fa68e
     primo.foo.com = 8a1011003ade80461322828f3b55b46c44814d6b
  barkflea.foo.com = e24f2a7b9fa26e2a0c201a7196325889abf7c45b
```

Now if a client requests a lookup of `snup.foo.com`, which doesn't exist, the name server will return a record that in informal terms states "the hash that in alphabetical order comes after `71d0549ab66459447a62b639849145dace1fa68e` is `8a1011003ade80461322828f3b55b46c44814d6b`" (again along with a signature made using `foo.com`'s key). This type of Resource Record is called **NSEC3**.

The client would compute the SHA1 hash of `snup.foo.com`:

```
      snup.foo.com = 81a8eb88bf3dd1f80c6d21320b3bc989801caae9
```

and verify that in alphabetical order it indeed falls between those two returned values (standard ASCII sorting collates digits as coming before letters). That confirms the non-existence of `snup.foo.com`.

(e) How does this method help prevent enumeration attacks? Which properties does the hash function need to have?

Solution:

Since the client only receives hashes of the domain names, they can't learn what the original domain names are unless they can break the `one-wayness` of the hash function.

(f) Describe how an adversary with access to a dictionary might still be able to perform an enumeration attack. What conditions must hold true for the domain names?

Solution:

An adversary can conduct a *dictionary attack*, either directly trying names to see whether they exist, or inspecting the hash values returned by NSEC3 RRs to determine whether names in a dictionary (for which the attacker computes hash values offline) indeed appear in the domain. The domain names must be part of the dictionary in this case.

Question 2 *DNS*

(a) Alice wants to access Berkeley's diversity advancement project DARE, `dare.berkeley.edu`. Her laptop connects to a wireless access point (AP).

Alice worries that a hacker attacks the DNS protocol when her laptop is looking for the IP address of `dare.berkeley.edu`. Assume that DNSSEC is not in use.

◇ **Question:** Which of the following can attack the DNS protocol and have Alice's browser obtain an incorrect IP address for DARE? (Select 0 to 8 options.)

- ■ The laptop's operating system.
- ■ The laptop's network interface controller.
- ■ The wireless access point.
- ■ An on-path attacker on the local network.
- ■ The local DNS resolver of the network.
- ■ The root DNS servers.
- ■ `berkeley.edu`'s DNS nameservers.
- ■ An on-path attacker between the local DNS resolver and the rest of the Internet.

Solution: 4 points. 0.5 points per marking.

(b) Now assume that `berkeley.edu` implements DNSSEC and Alice's recursive resolver (but not her client) validates DNSSEC.

◇ **Question:** Which of the following can attack the DNS protocol and have Alice's browser obtain an incorrect IP address for DARE? (Select 0 to 8 options.)

- ■ The laptop's operating system.
- ■ The laptop's network interface controller.
- ■ The wireless access point.
- ■ An on-path attacker on the local network.
- ■ The local DNS resolver of the network.
- ■ The root DNS servers.
- ■ `berkeley.edu`'s DNS nameservers.
- ☐ An on-path attacker between the local DNS resolver and the rest of the Internet.

Solution: Any on-path attacker can see or modify the DNS traffic. 4 points. 0.5 points per marking.

(c) An attacker wants to poison the local DNS resolver's cache using the Kaminsky attack. We assume that the resolver does not use source port randomization, so the attacker will likely succeed.

In the Kaminsky attack, the attacker asks the resolver for a *non-existing* subdomain of UC Berkeley, *e.g.*, `stanford.berkeley.edu`, instead of asking for an *existing* domain like `dare.berkeley.edu`.

⋄ **Question:** What is the advantage of asking for a non-existent domain compared to asking for an existing domain? (answer within 10 words)

____

____

Solution: When you fail, you can keep trying with another nonexistant name/race until win!

(Note, caching alone is not sufficient, because you do have caching of NXDOMAIN too. The big thing is "race until win". (3 points))

Question 3 ***Low-level Denial of Service***

In this question, you will help Mallory develop new ways to conduct denial-of-service (DoS) attacks.

(a) CHARGEN and ECHO are services provided by some UNIX servers. For every UDP packet arriving at port 19, CHARGEN sends back a packet with 0 to 512 random characters. For every UDP packet arriving at port 7, ECHO sends back a packet with the same content.

Mallory wants to perform a DoS attack on two servers. One with IP address A supports CHARGEN, and another with IP address B supports ECHO. Mallory can spoof IP addresses.

i. Is it possible to create a single UDP packet with no content which will cause both servers to consume a large amount of bandwidth?

- If yes, mark 'Possible' and fill in the fields below to create this packet.
- If no, mark 'Impossible' and explain within the provided lines.

● Possible　　　○ Impossible

If possible, fill in the fields:

Source IP: **B**　　Destination IP: **A**

Source port: **7**　　Destination port: **19**

If impossible, why?

____

____

Solution: Source IP: B, port: 7. Destination IP: A, port: 19. Source and destination can be flipped. Notice this will create a chain of CHARGEN and ECHO that will generate a lot of network traffic.

ii. Assume now that CHARGEN and ECHO are now modified to only respond to TCP packets (post-handshake) and not UDP. Is it possible to create a single TCP SYN packet with no content which will cause both servers to consume a large amount of bandwidth? Assume Mallory is off-path from the two servers.

- If yes, mark 'Possible' and fill in the fields below to create this packet.
- If no, mark 'Impossible' and explain within the provided lines.

○ Possible　　　● Impossible

If possible, fill in the fields:

Source IP: ____________ Destination IP: ____________
Source port: ____________ Destination port: ____________
Sequence #: ____________ Ack #: N/A

If impossible, why?

____

____

Solution: Impossible. As seen in previous question, source/destination IP has to be B/A for the chain to work. If you send a SYN packet to A pretending to be B, A will send SYN-ACK to B, which won't respond since it never sent a SYN. The connection won't be established.