CS161
Fall 2025

Introduction to Computer Security

Discussion 11

Q1 *DNS Walkthrough* (3 points)

Your computer sends a DNS request for www.google.com'‘

Q1.1 (1 point) Assume the DNS resolver receives back the following reply:

```
com. NS a.gtld-servers.net
a.gtld-servers.net A 192.5.6.30
```

Describe what this reply means and where the DNS resolver would look next.

Q1.2 (1 point) If an off-path adversary wants to poison the DNS cache, what values does the adversary need to guess?

Q1.3 (1 point) Why do we not use TCP for DNS? Why not use TLS to make the DNS connection secure?

Q2 *DNS* (3 points)

Q2.1 (1 point) 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?

- ☐ 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.
- ☐ A MITM between the local DNS resolver and the rest of the Internet

Q2.2 (1 point) 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?

- ☐ 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.
- ☐ A MITM between the local DNS resolver and the rest of the Internet

Q2.3 (1 point) 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.

Using the Kaminsky attack, the attacker wants to poison the resolver's cache for the domain **`www.berkeley.edu`**.

The attacker constructs a website with lots of images as follows:

```
<img src="http://fake1.berkeley.edu">
<img src="http://fake2.berkeley.edu">
<img src="http://fake3.berkeley.edu">
<img src="http://fake4.berkeley.edu">
                ...
```

◇ **Question:** If the attacker wants the resolver to accept their spoofed response, the transaction id (16-bit value) must be the same as the resolver's request transaction id. If the attacker can only send k responses before the name sever responds, what is the probability that at least one of the attacker's responses will be accepted by the resolver?

◇ **Question:** If the attacker can send multiple responses which use each possible transaction id, why would the attacker want to load many, even thousands of images on their website rather than just a few.

◇ **Question:** Assuming that the attacker has constructed a website with the image tags given above, can the attacker poison the resolver's cache for the domain www.google.com?

Q3 *NSEC3* **(3 points)**

In class, you learned about DNSSEC, which uses signature chains to ensure authentication for DNS results. Recall that when using NSEC3, in the case of a negative result (the name requested doesn't exist), the name server returns a signed pair of hashes that are alphabetically adjacent to the requested name's hash.

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
 galumph.foo.com
   primo.foo.com
```

then the corresponding SHA1 values would be:

```
barkflea.foo.com = e24f2a7b9fa26e2a0c201a7196325889abf7c45b
 galumph.foo.com = 71d0549ab66459447a62b639849145dace1fa68e
   primo.foo.com = 8a1011003ade80461322828f3b55b46c44814d6b
```

Sorting these on the hex for the hashes:

```
 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).

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`.

Q3.1 (1 point) How does NSEC3 help prevent enumeration attacks? Which properties does the hash function need to have?

Q3.2 (1 point) 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?

Q3.3 (1 point) Is there a modification we can make to prevent the dictionary based enumeration attack that was constructed in subpart 2?