Raluca Ada Popa
Spring 2018

CS 161
Computer Security

Discussion 8

Week of March 12, 2018

Question 1 ***DNS Walkthrough*** **(15 min)**

Your computer sends a DNS request for "www.google.com"

(a) Assume the DNS resolver receive 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.

> **Solution:** The IP address for "www.google.com" is not known. However, the "a.gtld-servers.net" is a name server for .com, and that is where the resolver should ask next at IP address 192.5.6.30.

(b) If an off-path adversary wants to poinson the DNS cache, what values do the adversary need to guess?

> **Solution:** The adversary will need to guess the identification number (16 bits). Some resolvers even randomize source ports.
>
> The reason an off-path attack is difficult is because the ID (and port numbers) have to match exactly, but once the legitimate reply reaches the resolver and cached, the server is no longer vulnerable to the poisoning attempts.

(c) Why can't we use TLS to secure DNS?

> **Solution:** DNS is designed to be lightweight and TLS will add lots of overhead.
>
> TLS also works very poorly with DNS caching, which is required for scalability.
>
> But the biggest issue is that we do not know which name servers to trust and TLS provides no protection against that. This is a fundamental difference between object security and channel security.

Question 2 ***DNSSEC*** **(15 min)**

In class, you learned about DNSSEC, which uses certificate-style authentication for DNS results.

(a) Suppose the case of a negative result (the name requested doesn't exist). Why can't the nameserver just return a signature on a statement such as "`aaa.google.com` does not exist"? What should the nameserver return instead?

Solution: The nameserver uses a canonical alphabetical ordering of all record names in its zone. It creates (off-line) signed statements for each pair of adjacent names in the ordering. When a request comes in for which there is no name, the nameserver replies with the record that lists the two existing names just before and just after where the requested name would be in the ordering. This proves the non-existence of the requested name. The reply is called an **NSEC** resource record.

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.

The signature allows the recipient to verify the validity of the statement, and by checking that `aaa.google.com` would have fallen between those two names, the recipient has confidence that the name indeed does not exist.

(b) One drawback with this approach is that an attacker can now enumerate all the record names in a zone. Why is this a security concern?

Solution: Revealing this information could aid in other attacks. For example, the names in a zone could be used as targets when probing for vulnerable servers.

(c) How could you change the response sent by the nameserver to avoid this issue?

HINT: One of the crypto primitives you learned about will be helpful.

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` but without indicating what names do exist, just what hashes exist.

By using a cryptographically strong hash function ~~like SHA1~~[1], it's believed infeasible to reverse the hash function to find out what name(s) appear in the zone (there's more than one potential name because hash functions are many-to-one). Note though that an attacker can still 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.

Question 3 ***Detection Strategies*** **(20 min)**

Suppose you are responsible for detecting attacks on the UC Berkeley network, and can employ host-based monitoring (a HIDS) that can inspect the keystrokes that users enter during their shell sessions. One particular attack you are concerned with is malicious modification or deletion of files in the directory `/usr/oski/config/`.

(a) One method of detection is called signature matching. This involves looking for particular well-defined patterns in traffic that are known to represent malicious activity. Give a couple of examples of signatures you can use to detect these attacks. What are some limitations of this approach?

Solution: Example signatures:

1. Look for the string `/usr/oski/config/` in requests
2. Look for `rm -rf`
3. Wait until a particular attack occurs. Afterwards, look for the same packets as occurred during that attack.

Problems with this approach:

1. It is prone to false positives, as it lacks context. It could be that access to /usr/oski/config occurs frequently for benign reasons, and without ensuing modifications. Similarly, users might often use rm -rf to manipulate directories other than the one youre observing.
2. It can be prone to false negatives or evasion. For example, an attacker could issue `cd /usr/oski; cd config` followed by `rm -f -r`. and easily evade detection.
3. If you only create signatures based on known (= previously seen) attacks, then the approach is purely reactive; if youre looking for a threat unique to your site, you cannot inoculate yourself from it until you have suffered it. On the other hand, (1) if the threat is one faced by other sites, they might have written signatures for it after having experienced it, and (2) one can adapt signature technology to write vulnerability signatures (signatures that match a known potential problem, rather than a known specific attack), which can be proactive.

(b) Another approach is to search for behaviors. Instead of looking for known attacks,

[1] As we know, SHA1 is no longer considered secure for many use cases. Using stronger hash functions for DNSSEC is therefore recommended. That said, the property we need from the hash function is one-way-ness, which to date is not an identified weakness of SHA1 (nor of MD5, in fact).

the detector might use knowledge of the system to look for suspicious sets of actions. Give two examples of host-based behavioral detection. Be specific as to how your examples differ from signature matching that looks for known attacks. What are some problems with this approach?

Solution: Examples:

1. Look for changes to files in `/usr/oski/config/` after multiple attempts at logging in as root. Here, rather than looking for a specific attack were looking for a pattern associated with likely-attack activity.
2. Dont even look for attacks; look for related suspicious activity indicative of a compromise. For example, look for login sessions that immediately issue rm commands, based on knowledge that benign users dont start off their sessions by removing files, but those who want to mess with Oski very well might. This approach can potentially detect a wide range of compromises for which the attacker obtains login access to the target system.

Issues:

1. Relies on the assumption that benign users will very rarely exhibit the behavior we key off of.
2. While potentially more general than signature matching, can still miss a wide range of attacks that dont happen to include (or for which the attacker consciously avoids including) the behavior for which we monitor.

(c) Suppose now we aim to detect modifications to any files in `/usr/oski/config/` using the following procedure. Each night, we run a cron job that checksums all of the files in the directory using a cryptographically strong hash like SHA256. We then compare the hashes against the previously stored ones and alert on any differences. (This scheme is known as Tripwire.) Discuss issues with false positives and false negatives.

Solution: False positives can occur any time that the files are changed for a legitimate purpose. Given a single change to a file, false negatives should not be a direct problem: due to the properties of a hash function like SHA256, if an attacker makes any modification to a file, the hash will change; they will not be able to find any alternative value for the file that yields the same hash. However, if the attacker gains administrative privileges then they could modify the OS to return the old content of the file whenever the nightly job runs; or modify the nightly job directly to always report nothing has changed; or modify the stored hashes to reflect the new content of the file. In addition, if the attacker makes a change to the file to their benefit, but then changes the file back prior to the run of the nightly job, then they will escape detection (false negative).