Introduction to Security

Jelena Mirkovic
USC CSCI 430

Module 21 – DNS security
How DNS Works

• Ask local resolver first about name->IP mapping
  – It returns info from cache if any

• If info not in cache, resolver asks servers in DNS hierarchy that are authoritative for a given domain www.usc.edu
  – Start from root, root sends it to the next point in hierarchy
  – End at authoritative server for the domain you are asking about (auth)
  – Resolver caches the replies
DNS Hijacking

• Wait for resolver (target) to ask about a name from the victim’s domain
  – Provide your own reply faster than auth server
  – Provide some extra information (IP of auth server)
• This poisons the cache at that resolver, not globally
  – But if the resolver is at the important/large network the victim can lose a lot of traffic to the attacker
• Attacker’s goal: impersonate the victim (phishing, stealing cookies)
DNS Hijacking

1. Who has www.victim.com?
2. It’s at IP 1.2.3.4
3. It’s at IP 5.6.7.8
   Too late!
Short-Circuiting Waiting

• Ask the target resolver about some name from the victim’s domain
  – Doesn’t have to be the name you intend to hijack since DNS will take extra info in the reply
  – Asking about non-existing names guarantees they are not in resolver’s cache already

• Can ask about different domains too
Cache Poisoning 1

1. www.attacker.com?
2. www.attacker.com?
3. don’t know ns.victim.com is auth for victim.com ns.victim.com has IP 1.2.3.4

VictimNS
5.6.7.8

Target

Attacker

AttackNS
1.2.3.4

Hijack entire domain

• IP for www.attacker.com (attacker asks target, target asks attackNS)
• Don’t know, ns.victim.com is auth for victim.com, its IP is 1.2.3.4 (attackNS’s reply)
• Target stores ns.victim.com -> 1.2.3.4 in cache
Defenses

• Only accept auth if its domain is same as the domain you asked about
• Don’t accept extra info in the reply if you did not ask for that info
Cache Poisoning 2

- IP for `www.victim.com` (attacker asks target, target asks victimNS)
- Attacker replies faster than the victimNS giving 1.2.3.4
- Target stores `www.victim.com`-`1.2.3.4` in cache
Cache Poisoning 3

- IP for madeup.victim.com
- Attacker replies faster than the victimNS
  - ... ns.victim.com is auth for victim.com, its IP is 1.2.3.4
- Target stores ns.victim.com->1.2.3.4 in cache

Hijack entire domain
Being Faster is Not Enough

• DNS requests come from a random source port (UDP header) and have a random request ID (DNS header)

• Target will only accept replies that
  – Come to the port from which requests are sent
  – Come with the replyID same as the requestID

• Attacker does not see target’s requests
  – Can try to sniff the info
  – Can try to guess the info

• If the resolver accepts the reply, it will accept extra info in reply too even if it has a different version in cache
ARP vs DNS Hijacking

- ARP maps IPs to MAC addresses, DNS maps names to IPs
- ARP will take unsolicited replies, DNS will not
- ARP replies just need to be faster
- DNS replies need to:
  - Be faster
  - Match source port and request ID of the requester
- Both ARP and DNS will take gratuitous info into cache:
  - ARP takes it from unsolicited replies
  - DNS takes it from extra information in the matching replies
- ARP will not overwrite info in the cache
  - DNS also will not overwrite info in the cache with main info from reply, but it will overwrite info in cache with ADDITIONAL info from reply
Hijacking With Sniffing

• Because the attacker needs to know the right port and request ID to create winning DNS responses
  – He may try to sniff these from the local network if sharing this network with the target

• Attacker can try to ARP spoof to make itself MITM between the target and the switch
  – Attacker must be on the same subnet as the target
  – Position itself on the path of target’s requests
  – DNS traffic uses UDP and is not encrypted, easy to see port and request ID and provide appropriate reply
Hijacking With Guessing

• If target uses predictable numbers for source port and request ID attacker can perform many attacks, each time trying to guess the right port/replyID combination
  – Having randomness in one is not enough ($2^{16}$ tries)
  – Need randomness in both ($2^{32}$ tries) for the right combo
  – Kaminsky attack (cache poisoning 3 + no randomness in source port)

    • Worked because some (lazy) implementations did not randomize ports
Defenses

• Randomize source port
  – Makes guessing harder but not impossible

• Use DNSSEC
  – Replies must be signed by auth
  – Everyone can check signatures
  – Auth servers for zones sign certificates when they delegate a sub-zone (e.g. .com for example.com)
  – Requires clients to implement DNSSEC to verify replies
DNSSEC

- Auth stores two more record types
  - RRSIG – signature for each A record
  - DNSKEY – public key to verify the signature
- Auth inserts DS record into parent in DNS hierarchy
  - Like a certificate
  - says who is authoritative for a given subdomain and gives a signed hash of the public key (DNSKEY of auth)
- Resolver sets a bit asking for DNSSEC replies
  - Verifies each DNSKEY using DS from the level higher in hierarchy
  - Verifies RRSIG of the record
DNSSEC

The Global Chain of Trust

Web Server

Web Browser

DNS Resolver

DNS Svr

.example.com

.com root

DNS Svr

.example.com

DNS Svr

.example.com

10.1.1.123

www.internetsociety.org/deploy360/
NSEC

• NSEC and variants: prove a record does not exist
  – What happens if we just say “Not here” every time? (Replay problems)
  – Intent was to just provide “does not exist” reply that is specific to a given query to avoid replay
  – Thus DNSSEC/NSEC provides a list of records before and after that would house such record if it existed

• Weaknesses
  – Zone walking: Attacker learns all the names and info for a domain
    • NSEC3 adds one-way hashes to avoid this
    • NSEC3 + Salt.... Does what? Exactly what you think.
  – DDoS: When the name exists, but specific records for that name do not, the list to prove this info does not exist can be large! (Asymmetry: small request results in big response)
  – Used in DDoS reflector attacks: spoofed requests cause legit servers to send data to the DDoS victim