ACM CCS 2020 DNS Cache Poisoning Attack Reloaded

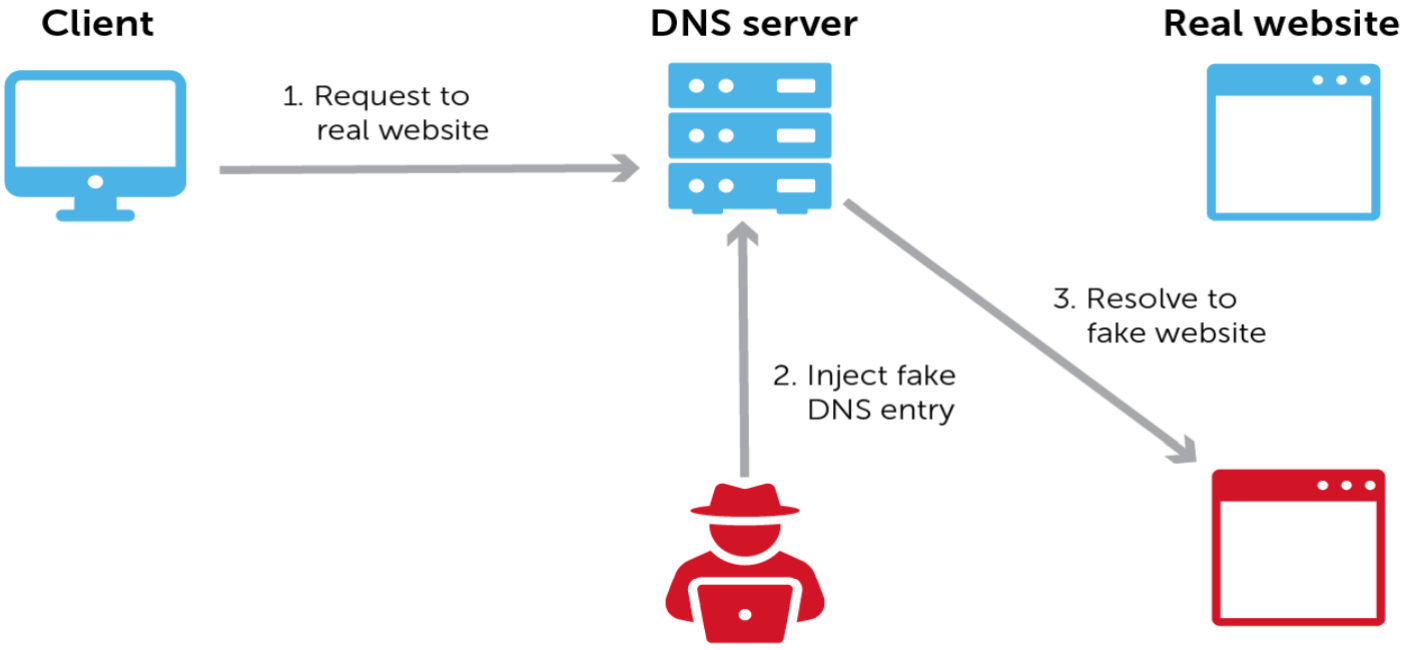
Hey everyone, I'm Paul. Today I'm excited to bring you the strong revival of the classic DNS cache poisoning attack.

The title of the paper is "DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels" and this is the joint work between UC riverside and Tsinghua University.

Here is the outline of today's presentation. Let's start by talking about the classic DNS cache poisoning attack.

The idea behind the DNS cache poisoning attack is to pollute the cache of a DNS resolver so that the traffic to a domain will be handed to a malicious host, and the attacker can therefore use this to conduct several crimes like scamming.

Let's go through it step by step. To begin with, we have a victim resolver, a name server (NS) and an Off-path attacker. We assume the attacker is a legitimate user of the resolver and cannot sniff[1] or modify the traffic between servers.



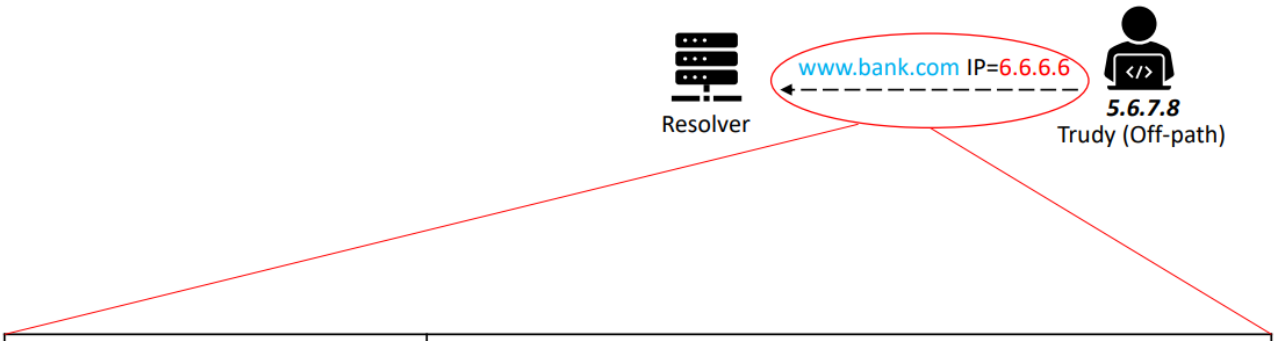

Usually the resolver will catch the mappings between the domain and the IP address, after it receives the response of the name server (NS). Based on this mechanism, the attacker first sends the query to the resolver to trigger the open query on it.

As expected, the resolver queries the name server (NS). However, the attacker then injects the packet to the resolver by spoofing[2] the identity of the name server (NS), right before a response

[1] Sniffing Traffic is the process of capturing and viewing traffic as it is passed along the network.

[2] Spoofing Attack is a situation in which a person or program successfully identifies as another by falsifying data, to gain an illegitimate advantage.

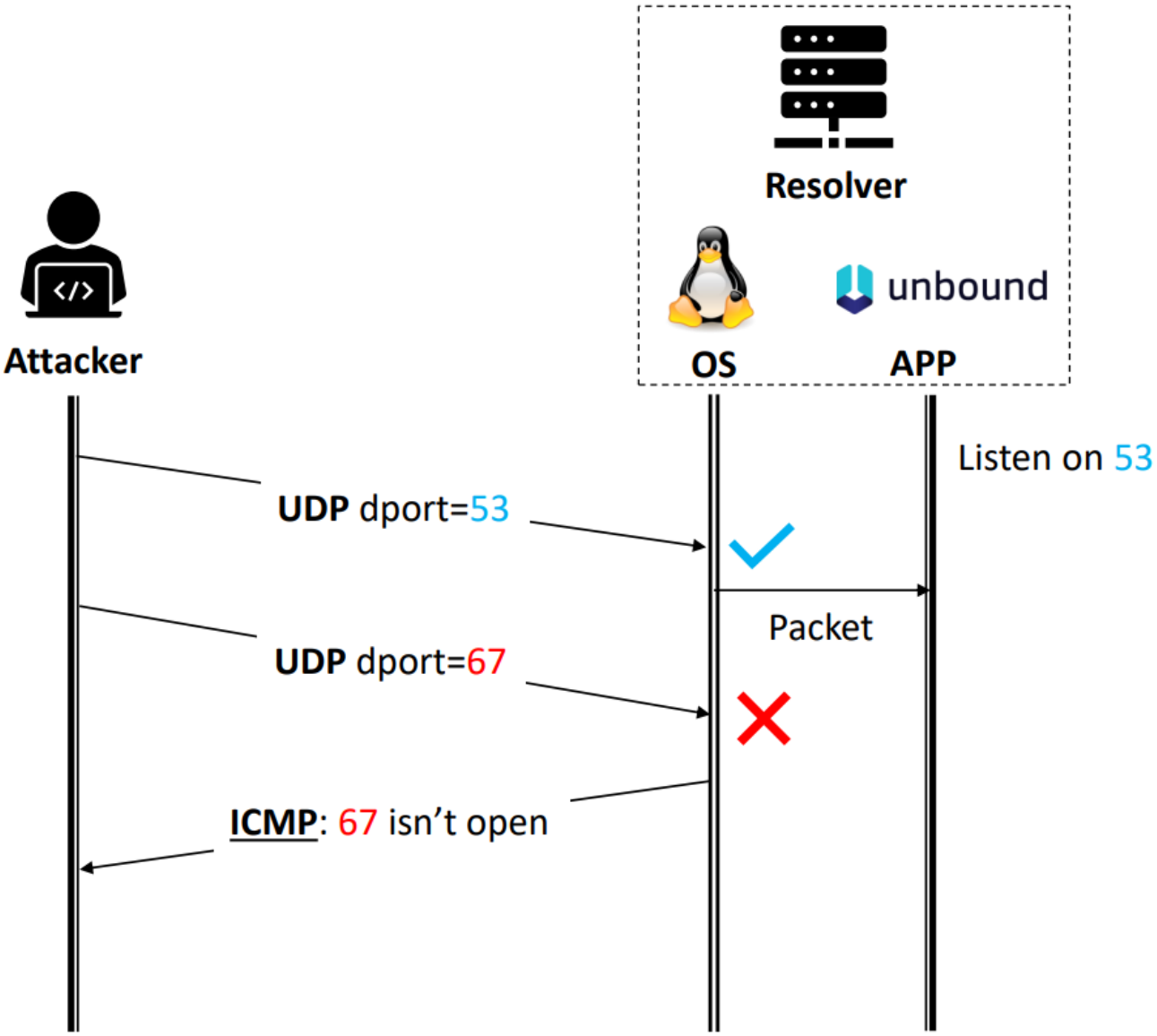
to the query. The resolver then accepts and caches the fake response. Now the authentic response arrives, but it's too late and they're dropped by the resolver. Since the resolver has cached the fake response, all future queries will be responded with the wrong answer.
So how to craft a validated fixed DNS as an injection packet. From the packet disassembly, we can see there are two unknown fields. The first one is the UDP destination port a.k.a. the source port, which is the same as the ephemeral port[3] number and transaction ID, both are randomly selected by the operating system (OS) of the victim resolver. By the way, this kind of defense is called source port randomization and it is perhaps the most effective and widely deployed defense.



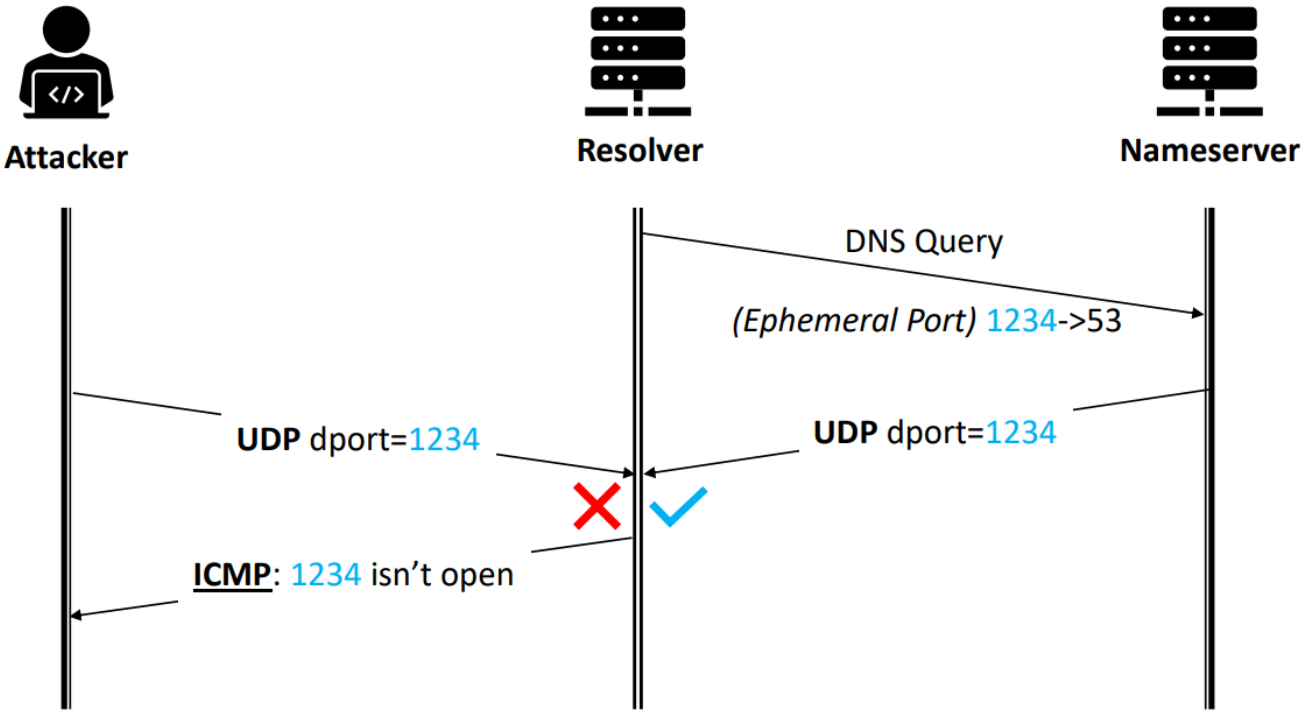

| | | |
|---|---|---|
| IP Layer | **Src:** 5.6.7.8 | |
| | **Dst:** *(resolver)* | |
| UDP Layer | **Src Port:** 53 | **Dst Port:** |
| DNS Layer | **TxID:** | |
| | **Question:** www.bank.com A ? | |
| | **Answer:** www.bank.com A 6.6.6.6, TTL=**99999** | |

The attacker would need to send an impractical number of 2^32 spoofed responses simultaneously guessing the correct source port (16-bit) and transaction ID (16-bit). Because the source port and transaction ID each as 16-bit long.
This requires the attacker to guess both values in a very short time before the resolver gets the legitimate answer which is quite challenging and has been thought to be impossible since 2008. Surprisingly, the researchers discover weaknesses that allow an attacker to "divide and conquer" the space by guessing the source port (ephemeral port) number first and then the transaction ID, leading to only 2^16 + 2^16 (2^16 each) spoofed responses, so there are two problem to solve, infer the correct ephemeral port number and reduce the responsiveness of the name server.

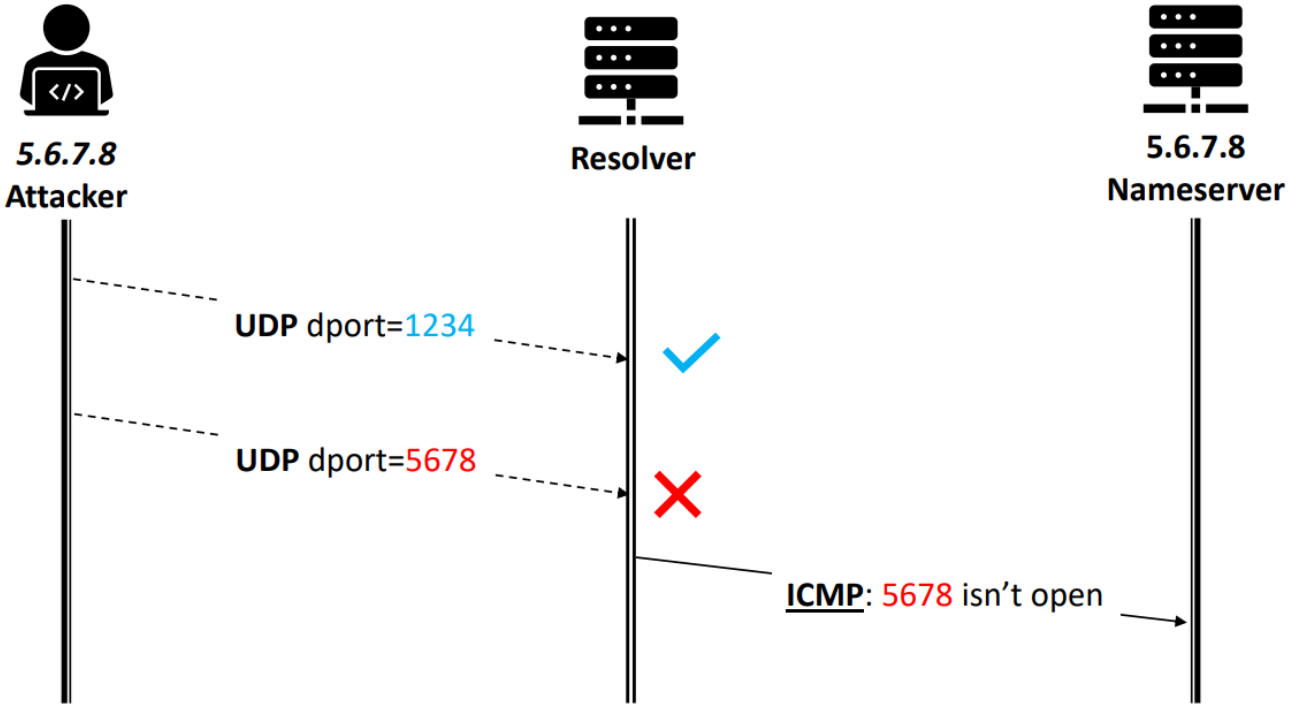
[3] Ephemeral Port is a communications endpoint (port) of a transport layer protocol of the Internet protocol suite that is used for only a short period of time for the duration of a communication session.




Let's now solve the first one. In general, inferring an open UDP port is pretty easy, just send the UDP approval packet with its destination port set to the port you want to probe. And if nothing happens which means we are right, the proper port is indeed open. Or if we receive an ICMP reply, then unfortunately that port is closed, as the operating system (OS) senses that no application is listed on that port. Ephemeral ports or source ports in UDP are like client ports in TCP and they are usually only open to a specific remote host. Therefore, the direct port scan will not work in discovering ephemeral ports.



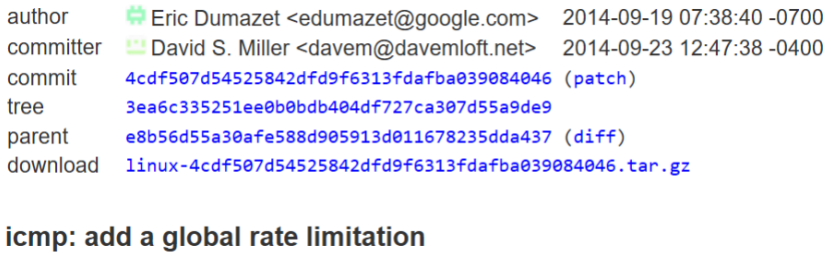

In this figure we can see, even if the attacker's probing UDP has guessed the correct port number, 1234. He still gets an ICMP back because that port is an ephemeral port, it's only open to the name server (NS) and not open to the attacker.




To deal with this problem, we can use IP spoofing, meaning we can send UDP proper packets by setting the source ID address to the same IP as of the name server (NS). This action will solicit ICMP packets but sadly as shown in the figure, the ICMP will be sent to the name server (NS) and the attacker cannot receive any feedback. Whether the proper port is open or not.

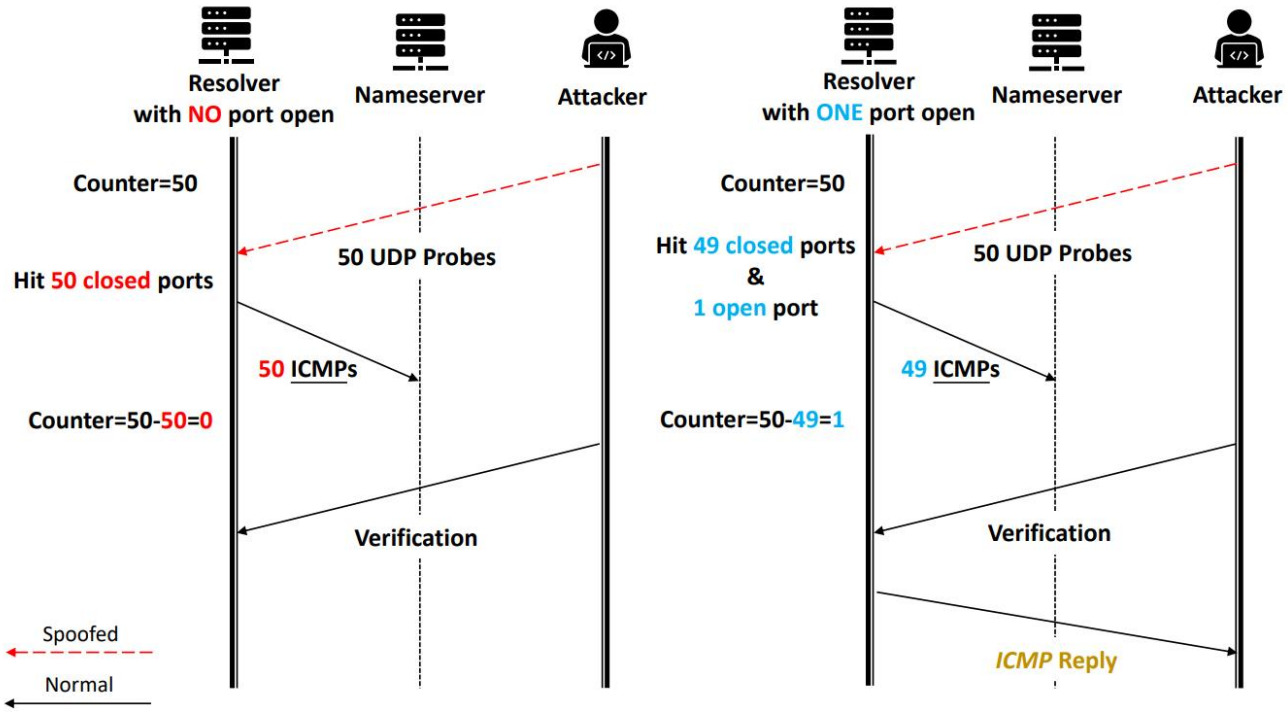
So, can we receive any other feedbacks when sending with the spoofed IP address? The answer is yes, but not directly and this is called the side channel. Anytime when there are shared resources, side channel can potentially arise, like the TCP side channel[4] discovered by same researchers in 2016. Where the TCP global rate limit counter is used as the shared resource between the attacker's connection and the victim's connection.

- ICMP Global Rate Limit:
  - Limit sending rate
  - Shared by all IPs




This side channel uses ICMP global rate limit counter as a shared resource, the counter limits the maximum amount of ICMP that could be send out during a period of time, to mitigate the denial of service (DoS) attacks. It is also shared by all remote IPs including both the attacker's IP and the name server's (NS) IP. Specifically, in Linux the counter has a maximum value of 50. Here is how they creatively exploit such a shared global counter for ephemeral port scan.
Initially the resolver has 50 permits in its counter.
To check open ports, the attacker sends 50 spoofed probe packets to 50 different ports. The resolver then generates ICMP for the probes based on whether the attacker has the correct port, the resolver may reply with 50 or 49 packets, and the counter is deducted accordingly. Since no ICMP is allowed to send if the counter hits zero.
To reveal the status of the global counter the attacker then sends a verification packet to closed port on the victim's server using his real IP address to try to solicit an ICMP reply. If he really gets a reply meaning the counter is non-zero and thus he found an open port. Otherwise, it means the counter has already hits zero and thus no port is open.

[4] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. 2016. Off-path TCP exploits: global rate limit considered dangerous. In Proceedings of the *25th USENIX Conference on Security Symposium (SEC'16)*. USENIX Association, USA, 209–225.




This side channel will give the attacker up to 1000 packets per second per scan rate while using the spoofed IP. In the paper, the researchers have engineered their exploit carefully so that it could work even on poor networks a.k.a. the realistic network conditions that with losses, delays and jitters[5].

[5] Jitter is the variation in time delay between when a signal is transmitted and when it's received over a network connection.

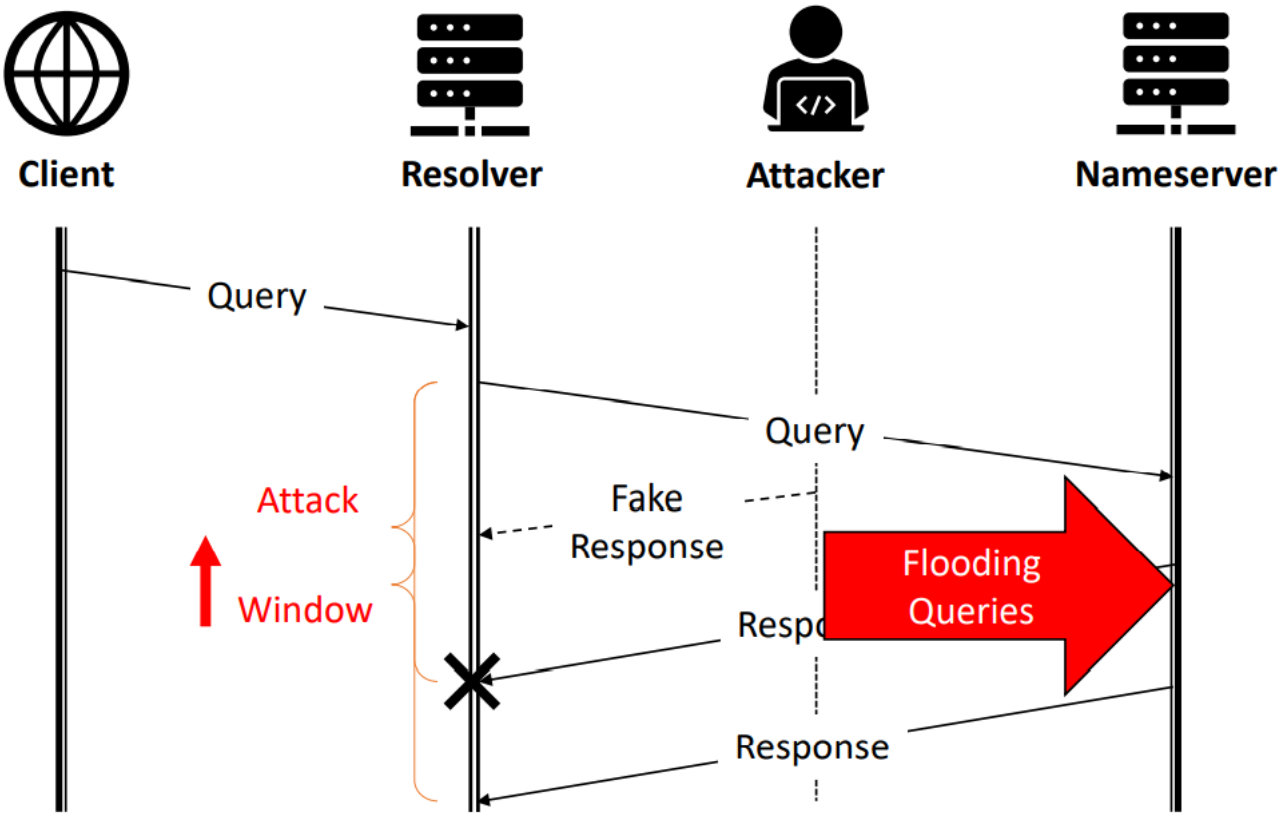
- Open Resolvers:
  - **34%** Vulnerable
- Well-known Public Resolvers:
  - **12**/14 Vulnerable

| | |
|---|---|
| Google | 8.8.8.8 |
| Cloudflare | 1.1.1.1 |
| OpenDNS | 208.67.222.222 |
| Comodo | 8.26.56.26 |
| Dyn | 216.146.35.35 |
| Quad9 | 9.9.9.9 |
| AdGuard | 176.103.130.130 |
| CleanBrowsing | 185.228.168.168 |
| Neustar | 156.154.70.1 |
| Yandex | 77.88.8.1 |
| Baidu DNS | 180.76.76.76 |
| 114 DNS | 114.114.114.114 |
| Tencent DNS | 119.29.29.29 |
| Ali DNS | 223.5.5.5 |

| Name | Address | Example Backend Addr. | # of Backends | ICMP | Global Rate Limit | Using `connect()` | Vulnerable |
|---|---|---|---|---|---|---|---|
| Google | 8.8.8.8 | 172.253.2.4 | 15 | Y | Y | N | Y |
| CloudFlare | 1.1.1.1 | 172.68.135.169 | 2 | Y | Y | Y | Y |
| OpenDNS | 208.67.222.222 | 208.67.219.11 | 107 | Y | Y | Y | Y |
| Comodo | 8.26.56.26 | 66.230.162.182 | 2 | Y | Y | N | Y |
| Dyn | 216.146.35.35 | 45.76.11.166 | 1 | Y | Y | N | Y |
| Quad9 | 9.9.9.9 | 74.63.16.243 | 11 | Y | Y | Y | Y |
| AdGuard | 176.103.130.130 | 66.42.108.108 | 3 | Y | Y | N | Y |
| CleanBrowsing | 185.228.168.168 | 45.76.171.37 | 1 | Y | Y | Y | Y |
| Neustar | 156.154.70.1 | 2610:a1:300c:128::143 | 2 | Y | Y | N | Y |
| Yandex | 77.88.8.1 | 77.88.56.132 | 19 | Y | Y | Y | Y |
| Baidu DNS | 180.76.76.76 | 106.38.179.6 | 16 | Y | Y | Y | Y |
| 114 DNS | 114.114.114.114 | 106.38.179.6 | 11 | Y | N | N | Y |
| Tencent DNS | 119.29.29.29 | 183.194.223.102 | 45 | Y | N | N | N[1] |
| Ali DNS | 223.5.5.5 | 210.69.48.38 | 160 | N | N/A | N/A | N |

[1] Though meeting the requirements, it is not vulnerable due to interference of fast UDP probing encountered (likely caused by firewalls).

Without leveraging the global counter the researchers would not be able to perform any kind of scan. Since this turns out to be a design flaw of the operating system (OS), it affects a large population of the machines on the internet including 34% of open resolvers and 12 out of 14 public resolvers according to their measurements.




So till now the port inference problem is solved, we could finally jump into about how to extend the attack window.

As previously mentioned, the attacker only has hundreds of milliseconds to infer the ephemeral port number which is usually the same length as the Round-Trip Time (RTT) between the resolver and the name server (NS), and we called this time duration the attack window.

Although with the help of this advanced scanning method, the attacker still needs at least 30 seconds to finish scanning all possible ports. Even if the port is correctly identified, it will still take time to inject rogue DNS records. So, the larger the attack window is, the more port the attacker can scan and the more chance he can succeed.

Additionally, the attacker can repeat the attack multiple times if he failed in the first attempt and as long as he finally succeeded the cache will be poisoned for quite a long time, like a few days or so.

To increase the attack window, the attacker can simply flood the query traffic, to overwhelm the name server (NS). In fact, the researchers even found that a BIND[6] name server (NS) can be easily overwhelmed by flooding random queries.

[6] Berkeley Internet Name Domain, BIND is currently the most common DNS software on the internet.

| | Setup | | | | | Result | |
|---|---|---|---|---|---|---|---|
| Attack | # Back Server | # NS | Jitter | Delay | Loss | Total Time | Success Rate |
| Tsinghua | 2 | 2 | 3ms | 20ms | 0.2% | 15 mins | 5/5 |
| Commercial | 4 | 1 | 2ms | 30ms | 0.6% | 2.45 mins | 1/1 |

| Exp. | RTT range | Probe loss | Name sever mute level | Average time taken | Success rate |
|---|---|---|---|---|---|
| Base(D) | 0.2-1.2ms | ~0% | 80% | 504s | 20/20* |
| Base(M) | 0.2-1.2ms | ~0% | 80% | 410s | 20/20* |
| Mute Lv. | 0.2-1.2ms | ~0% | 75% | 1341s | 18/20* |
| Mute Lv. | 0.2-1.2ms | ~0% | 66.7% | 2196s | 20/20# |
| Mute Lv. | 0.2-1.2ms | ~0% | 50% | 8985s | 9/20# |
| Altered | 37-43ms | 0.20% | 80% | 930s | 5/5* |

*: 1-hour threshold. #: 3-hour threshold. D: Day. M: Midnight

Finally, let's discuss about the possible defenses.

- DNSSEC
- 0x20 encoding
- DNS cookie
  - Only 5% open resolvers deployed
- Disable ICMP port unreachable
- Randomize ICMP global rate limit

First, we can add more secrets to defeat the off-path attacker. Although these techniques have been standardized, but in fact they are rarely deployed due to the compatibility and incentives issues.

Second, we can remove the side channel to prevent effective port scanning.

**Diffstat** (limited to 'net/ipv4/icmp.c')

-rw-r--r-- net/ipv4/icmp.c 7

1 files changed, 5 insertions, 2 deletions

```
diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 07f67ced962a6..005faea415a48 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -239,7 +239,7 @@ static struct {
 /**
  * icmp_global_allow - Are we allowed to send one more ICMP message ?
  *
- * Uses a token bucket to limit our ICMP messages to sysctl_icmp_msgs_per_sec.
+ * Uses a token bucket to limit our ICMP messages to ~sysctl_icmp_msgs_per_sec.
  * Returns false if we reached the limit and can not send another packet.
  * Note: called with BH disabled
  */
@@ -267,7 +267,10 @@ bool icmp_global_allow(void)
        }
        credit = min_t(u32, icmp_global.credit + incr, sysctl_icmp_msgs_burst);
        if (credit) {
-               credit--;
+               /* We want to use a credit of one in average, but need to randomize
+                * it for security reasons.
+                */
+               credit = max_t(int, credit - prandom_u32_max(3), 0);
                rc = true;
        }
        WRITE_ONCE(icmp_global.credit, credit);
```

All in all, let's conclude my presentation.

First, we introduced a novel side channel based on ICMP global limit counter.

Then by using the side channel the researchers developed the DNS poisoning attack.

And finally, the results showed that this kind of attack is very powerful and can succeed in real-world resolvers in just few minutes.