DNS Cache Poisoning Attack: Resurrections with Side Channels

Keyu Man
kman001@ucr.edu
University of California, Riverside
Riverside, CA, USA

Xin'an Zhou
xzhou114@ucr.edu
University of California, Riverside
Riverside, CA, USA

Zhiyun Qian
zhiyunq@cs.ucr.edu
University of California, Riverside
Riverside, CA, USA

ABSTRACT

DNS is one of the fundamental and ancient protocols on the Internet that supports many network applications and services. Unfortunately, DNS was designed without security in mind and is subject to a variety of serious attacks, one of which is the well-known DNS cache poisoning attack. Over the decades of evolution, it has proven extraordinarily challenging to retrofit strong security features into it. To date, only weaker versions of defenses based on the principle of randomization have been widely deployed, *e.g.,* the randomization of UDP ephemeral port number, making it hard for an off-path attacker to guess the secret. However, as it has been shown recently, such randomness is subject to clever network side channel attacks, which can effectively derandomize the ephemeral port number.

In this paper, we conduct an analysis of the previously overlooked attack surface, and are able to uncover even stronger side channels that have existed for over a decade in Linux kernels. The side channels affect not only Linux but also a wide range of DNS software running on top of it, including BIND, Unbound and dnsmasq. We also find about 38% of open resolvers (by frontend IPs) and 14% (by backend IPs) are vulnerable including the popular DNS services such as OpenDNS and Quad9. We have extensively validated the attack experimentally under realistic configuration and network conditions and showed that it works reliably and fast.

CCS CONCEPTS

• **Security and privacy → Network security**; *Operating systems security*; • **Networks → Cross-layer protocols**; **Naming and addressing**.

KEYWORDS

DNS, cache poisoning, side channel, attack, ICMP, fragment

ACM Reference Format:
Keyu Man, Xin'an Zhou, and Zhiyun Qian. 2021. DNS Cache Poisoning Attack: Resurrections with Side Channels. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3460120.3486219

1 INTRODUCTION

Domain Name System (DNS) is one of the most important infrastructures of the modern Internet. It translates the human-readable domain names into machine-readable IP addresses. This basic functionality has also now been used by various security services such as email authentication [35], routing security (*e.g.,* RPKI [42]), and even certificate issuance where proof of domain ownership is the common method to acquire certificates [5] . As a result, compromising DNS can lead to catastrophic security failures with a wide range of consequences [20] (*e.g.,* man-in-the-middle attacks and fake TLS certificates being issued [13]).

Despite its critical role, DNS has been a fragile part of the security chain. Historically, efficiency was the primary consideration of DNS, leading to the design of a single query and response over UDP, which is still the primary mechanism used today. Although security features like DNSSEC and DNS cookies have been standardized, they are not widely deployed due to backward compatibility. This led to a series of DNS cache poisoning attacks [33, 36, 45] that allow an off-path attacker to poison a DNS cache with a malicious record to map a domain to an arbitrary IP address. The earliest such attack dates back to 1997 [58]. In 2008, Dan Kaminsky identified a way to bypass the standard bailiwick checks [36]. Recently, a side-channel based DNS cache poisoning attack [45], dubbed SADDNS [1, 45], was developed that can effectively derandomize the ephemeral port in a DNS query.

In SADDNS, the key insight is that a shared resource, *i.e.,* ICMP global rate limit shared between the off-path attacker and victim, can be leveraged to send spoofed UDP probes and infer which ephemeral port is used. Unfortunately, it is unclear how many more such side channels exist in the network stack. In this paper, we explore a non-conventional type of port scan packets, i.e., ICMP packets which are by design error messages and cannot solicit any explicit response. This is distinct from SADDNS where it has considered UDP packets which are conventional port scan packets. Even though it is known that ICMP can interact with UDP/TCP [4, 48], e.g., shutting down a socket (with an ICMP port unreachable message), it is not immediately obvious how ICMP probes can allow an off-path attacker to infer the ephemeral port number selected for a UDP socket. Surprisingly, we uncover novel side channels that have been lurking in the Linux network stack for over a decade and yet were not previously known.

The successful exploitation of these side channels in the context of DNS hinges on the subtle interactions among three different layers, *i.e.,* ICMP, UDP, and application. Interestingly, due to the lack of documentation and awareness, such interactions are often neglected and misconceived, leading to many exploitable scenarios. In addition to novel side channels, we also find that ICMP messages can be used to DoS DNS transactions, indirectly assisting the cache poisoning attack.

We have comprehensively characterized the impact of the side channels. They affect the most popular DNS software including BIND, Unbound, and dnsmasq running on top of Linux. In addition,

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11.
https://doi.org/10.1145/3460120.3486219

we estimate that they affect 13.85% of open resolvers. Finally, we evaluate the end-to-end attack on the latest BIND resolver and a home router and find that it is reliable and takes only minutes to succeed. To mitigate the attack, we suggest setting proper socket options, randomizing the caching structure, and rejecting specific ICMP messages when possible.

We summarize our contributions as the followings:

- We discovered novel side channels that allow us to use ICMP probes to scan UDP ephemeral ports.
- We thoroughly analyzed the root cause of the discovered side channels and developed powerful DNS cache poisoning attacks based on that.
- We measured their impact in the real world and proposed corresponding mitigations.

2 BACKGROUND

In this section, we will introduce the necessary background regarding the two types of UDP ephemeral ports that an attacker would want to scan to conduct the DNS cache poisoning attack. We will then introduce the ICMP messages that interact with UDP in interesting ways.

2.1 Public-facing and Private-facing UDP Ports

Traditionally, port scans refer to scanning *server* ports as the intention is to infer which services are running. However, in the context of DNS cache poisoning attacks, the goal is to scan ephemeral ports instead (more details are provided in §3). Interestingly, as well summarized in [45], unlike TCP, UDP ephemeral ports can be further divided into two types: (1) public-facing and (2) private-facing. This is due to the stateless nature of UDP, as stated in RFC 8085 [41]. Specifically, if a client sends a UDP packet by invoking `sendto()` with a specific remote IP as an argument, the client OS will in fact accept packets from "any IPs" when it subsequently invokes `recvfrom()`. Therefore, by default, any UDP ephemeral port will become public-facing. Only if the client explicitly invokes `connect()` will the OS reject packets from all but the one "connected" remote IP [41]. This effectively makes the ephemeral port private-facing.

As shown in the prior work [45], public-facing ephemeral ports are generally easier to scan. Interestingly, whether an ephemeral port is public-facing also has ramifications with regard to the new side channels we identify. We will describe them later in §4.4.

2.2 ICMP Messages and Impact on UDP

As first introduced in RFC 792 [48], ICMP is a diagnostic protocol used to signal errors during the delivery of IP packets. This can happen, for example, when a router discards the packet and return an ICMP TTL expired message back to the source after it detects that the TTL of the forwarded packets reaches zero. To allow the source to distinguish which packets have encountered errors, a partial copy of the packet is embedded in the ICMP message, which includes the source and destination address, source and destination port. According to recent RFCs [27], the source should accept such messages only if the wrapped four-tuple matches an existing socket. Upon validating the correctness of such an ICMP message, depending on the nature of the error and the socket options set by the application, the source may ignore the error, remedy the situation by taking actions in the OS kernel (*e.g.,* updating routing entries) and/or reporting the error to the application layer through the socket interface.

Below we describe a few relevant ICMP message types that have interesting interactions with UDP:

- **Fragment Needed** Such messages are typically sent by a router to signal the source that the size of its packet has exceeded the MTU of the next hop [4, 47]. Specifically, they are called "fragmentation needed and DF set" or "packet too big" for IPv4 and IPv6 respectively. The desired MTU is included in the message so that the source OS can take actions, *e.g.,* updating its PMTU cache for the corresponding destination, and reducing the size of all future packets with the same destination address.
- **Redirect** Redirect messages [48, 56] are usually sent back to the source by the next-hop router (*e.g.,* gateway) to signal a shorter route to a destination. After the source receives such a message, it will update its routing table and route all future packets to that destination through the new gateway, which is specified in the redirect packet. This message is only supposed to be sent by the gateway, and therefore, the OS of the source usually checks the source IP of the ICMP message before accepting the redirection [56].
- **Host/Port Unreachable** Such messages are used to signal the source that the original packet was sent to the wrong host or port and thus cannot be delivered [4, 48]. According to RFCs [4, 12], upon receiving such messages, the OS must notify the application as long as a socket is found based on the embedded four-tuple in the ICMP message.

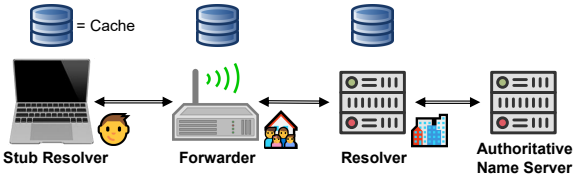


Figure 1: DNS Hierarchy

3 THREAT MODEL AND WORKFLOW

In this section, we will describe the general threat model used in DNS cache poisoning attacks.

DNS Hierarchy and Attack Targets. Figure 1 shows a typical DNS Hierarchy. The stub resolver (usually provided by OS) runs on an individual client and acts as a proxy—it only forwards the query to the upstream DNS server without resolving the query itself. The sole purpose of the stub resolver is to provide the local cache to speed up DNS queries from the same host. In the next layer up, DNS forwarders are also caching proxies—common in home and business gateways (*e.g.,* Wi-Fi router) [6, 40, 54], but they serve multiple clients in a LAN. At the highest layer, DNS resolvers finally perform the real name resolution task by recursively consulting the name servers, where the actual DNS records are stored. Resolvers are usually operated by ISPs or tech companies (*e.g.,* 8.8.8.8 operated by Google) and generally serve many more clients. As a result, DNS resolvers are the most prolific and impactful attack

targets. Furthermore, some resolvers, *e.g.,* those offered by Google and Cloudflare, are even open to the public and are accessible by everyone, making them more accessible to attackers as well.

Nevertheless, since stub resolvers, forwarders and resolvers are all equipped with DNS caches, they are all potentially subject to DNS cache poisoning attacks. [9, 32, 59] proposed the cache poisoning attacks that only work against DNS forwarders instead of resolvers, because they exploited the unique position or design goals of the forwarder. Specifically, [9, 32] assume the attacker is under the same NAT gateway as the forwarder, however However, resolvers are not usually behind NAT. [59] is based on the fact that the forwarders rely on the resolvers to perform the bailiwick check and thus become vulnerable. To our knowledge, there are only two practical attacks [33, 45] that can work against the resolver in the past decade and SADDNS [45] is the only one using side channels to launch poisoning attacks (which no longer works as the vulnerability is already patched). In this paper, we will introduce novel side-channel-based cache poisoning attacks that affect all DNS servers in the hierarchy. Our discussion focuses on DNS forwarders and resolvers.

Assumptions. Generally, an attacker needs two main capabilities to launch the attack:

(1) The ability to trigger one or more queries from the target DNS server (forwarder or resolver). This is trivially satisfiable if the DNS server is publicly accessible. In practice, there are hundreds of thousands of them (see §5.2), including the popular ones such as 1.1.1.1 and 8.8.8.8. If the DNS server is private, the attacker would need to join the network directly or indirectly. For example, there are various open networks in coffee shops and airports which allow an attacker to easily join. It is also possible that an attacker can trick a victim client in a private network to visit a malicious website where malicious scripts can execute and trigger DNS queries.

(2) The ability to send packets with spoofed IP addresses. This is because the goal of the attack is to inject malicious records to either the forwarder or resolver, and such rogue responses have to come from a host that they contacted before, *i.e.,* either the resolver or the name server. This requirement is also not difficult to satisfy. As shown in a 2019 report [44], there are still 30.5% and 32.1% of ASes in the world that do not block packets with spoofed source IPv4 and IPv6 addresses respectively, which renders the attack still feasible today.

Workflow. Taking the resolver attack as an example, the attacker is off-path, *i.e.,* unable to modify or eavesdrop on the traffic between the resolver and the name server. The first step of the attack is to turn the resolver into a state where it is willing to accept responses from the name server. This can be achieved by simply sending a query to the resolver. After that, the attacker tries to forge a response packet and send it back to the victim resolver to poison the cache. However, in order for the rogue response to be accepted by a modern DNS resolver, several things have to match (there are additional defenses that may be optionally and rarely deployed as we will discuss in §8.3): (1) The source IP of the response should be the name server. Since the attacker controls the domain name in the query, the corresponding name server can be easily looked up ahead of time. (2) The 16-bit destination port number in the response has to match the ephemeral port that is typically randomly generated on the resolver. (3) The 16-bit transaction ID in the DNS payload has to match the one randomly selected by the resolver [46].

If a rogue response with all the matching fields arrives before the legitimate one sent by the name server, then the resolver will accept and cache the rogue results. This can be an insurmountable hurdle as an attacker needs to effectively enumerate all possible 32-bit values (equivalent to ~4 billion values) within a small time window, *i.e.,* the RTT between the resolver and name server. Even if an attacker can repeat the attempts over many queries, it is still a largely infeasible attack.

In summary, there are 6 steps of the attack.

(1) Identify the victim resolver, the domain to poison, and its name server.
(2) Slow down name servers and prevent them from responding to the victim resolver (§6.1); this gives the attacker more time.
(3) Start triggering the query on the resolver.
(4) Infer the ephemeral port of the query using our new side channels (§4).
(5) Once the port is known, inject 65,536 rogue responses with different TxIDs to the victim resolver by spoofing the name server's IP.
(6) Check if the cache is poisoned. If not, go back to (3).

4 ICMP-BASED EPHEMERAL PORT SCANS

4.1 Prior Methods of UDP-Based Port Scans

Traditionally, UDP probes are used to determine whether a UDP port (specified as the destination port number in the probe) is open or closed. According to the RFCs [4, 48], if the destination replies with an ICMP port unreachable message, it indicates that the port is closed. This is traditionally used to probe server ports as shown in Figure 2(a). In addition, as mentioned in §2.1, this can also be used to discover public-facing ephemeral ports. It is obvious that the presence or absence of the ICMP response is explicit feedback on the UDP probe.

SADDNS. To scan private-facing ephemeral ports, UDP probes must be sent using the source IP address of the remote peer, forcing an off-path attacker to find an indirect way of performing the scan. Based on this, in 2020, Man *et al.* [45] identified a global rate limit on the ICMP responses to UDP probes, enabling an attacker to send spoofed UDP probes and indirectly infer whether they have solicited responses. Specifically, if a guessed port number (in a spoofed UDP probe) happens to match the correct ephemeral port, the resolver will not generate an ICMP message (otherwise it would). This results in either a stationary limit counter or a decrement of the counter. An attacker can then check whether the counter has been drained by attempting to solicit ICMP responses with a UDP probe from his real/non-spoofed IP. Fundamentally, this is a variant of the traditional UDP-based scan because the goal is still to indirectly infer the presence or absence of ICMP responses.

4.2 ICMP-Based Port Scans

In contrast with the traditional methods of UDP-based port scans, in this paper, we investigate the ICMP-based port scans. As mentioned in §2.2, an ICMP message embeds the header of the original packet from the source, including the source and destination port

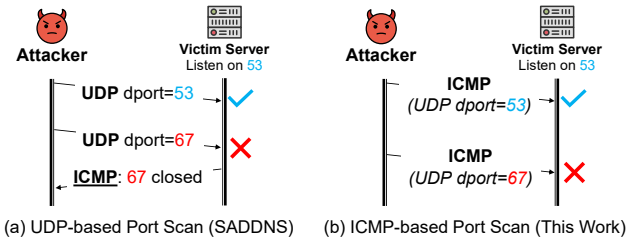


Figure 2: Ephemeral Port Scan

information. This opens up an opportunity to craft an ICMP message embedding a guessed port number, which is used to match a specific socket on the receiver end [4, 48]. However, the challenge is that ICMP messages are by design error messages useful for diagnostic purposes only, which do not solicit explicit responses [12]. This means that regardless of whether a port number is guessed correctly, the receiver will not provide any response, as shown in Figure 2(b), making the ICMP-based port scans seem infeasible.

Surprisingly, *we observe that an attacker does not necessarily have to rely on the explicit feedback from an ICMP probe. Instead, even if the processing of ICMP probes is completely silent, as long as there is some shared resource whose state is influenced, we may find ways (other probes) to observe the changed state of the shared resource.* This is a generalization of the prior probing methods that rely on spoofed probes that by design can solicit responses from the victim. In addition to SADDNS whose probes are designed to solicit ICMP responses, it is also the case for the series of TCP side channels [14, 17, 43]. Specifically, [14] leveraged TCP probes that can solicit challenge ACKs; [17, 43] required TCP probes that can solicit any response. In summary, it requires a leap of faith to realize the potential of the ICMP-based probes to scan UDP ports.

In this project, we systematically investigated all types of ICMP and narrowed them down to two that are useful for port scans: *ICMP fragment needed (or ICMP packet too big in IPv6)* and *ICMP redirect*. Next, we will describe their processing logic in the Linux kernel and the corresponding shared resources that form side channels.

4.3 Analysis of ICMP Error Processing Logic

We use the ICMPv4 (ICMPv6 is similar) in the latest Linux kernel (5.11.16 at the time of writing) as an example to illustrate this (the logic is largely the same since 3.6). When the OS receives an ICMPv4 message with an embedded UDP packet, it will invoke `__udp4_lib_err()` to handle the error. Here the four-tuple in the wrapped UDP packet is first checked with the socket table (`__udp4_lib_lookup()`) to verify the legitimacy of the ICMP packet, *i.e.,* it is indeed triggered by the packet the host sent before. If it passes the check, the ICMP error will be handled according to the type of error. Additionally, the ICMP error may optionally be delivered to the application if the OS has received the proper socket options (which will be described in §5.1).

To handle the ICMP frag needed and redirect, two corresponding kernel functions are invoked respectively: `ipv4_sk_redirect()` and `ipv4_sk_update_pmtu()`. Both of them will update a global resource maintained in the routing module, called the **n**ext **h**op **e**xception (f**nhe**) cache. We refer to it as "exception cache" in short from here on. It stores various states including the non-default MTU for specific remote IPs (updated by ICMP frag needed messages), and the non-default gateway IP for specific remote IPs (updated by ICMP redirect messages). These exception cache entries affect the routing decisions for all future outgoing packets destined to the remote IPs in the entries. These entries are cached for some time unless explicitly evicted due to a limit on the total number of entries (details are provided in §4.5).

One thing worth noting is that the OS does not check the source IP address of the ICMP frag needed messages. This is by design as such messages can be generated by any router along the path. And due to the dynamic nature of the Internet, the victim resolver cannot easily verify if a given IP belongs to the routers along the path. This has an interesting implication that the attacker's probes of ICMP frag needed messages, which we will describe next, do not need to spoof the source IP address at all.

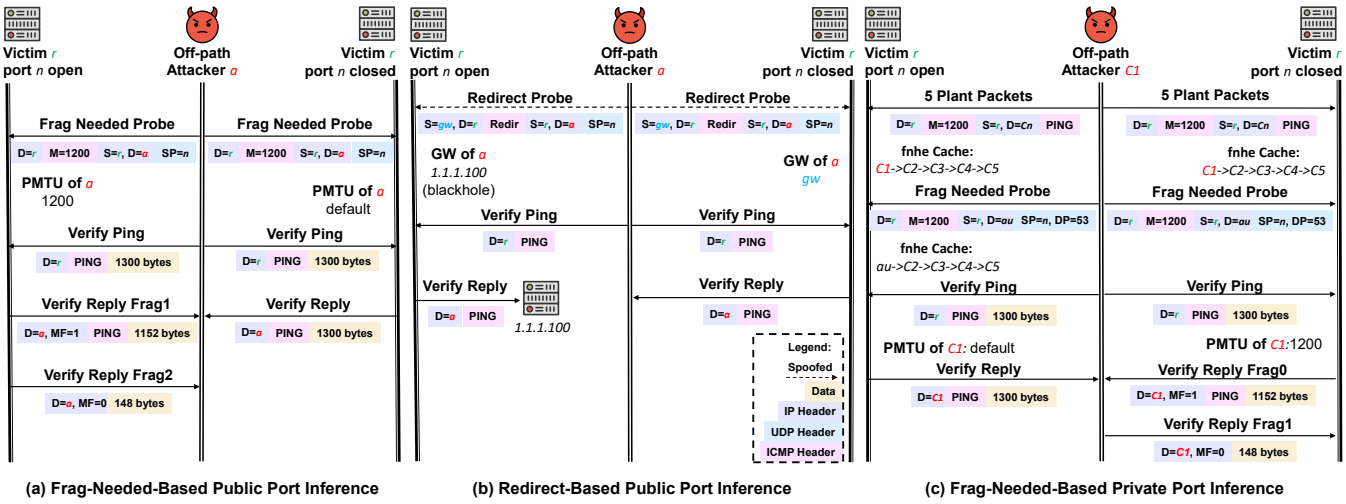
4.4 Public-Facing Port Number Inference

We illustrate the basic idea of public-facing ephemeral port scan in Figure 3(a) & 3(b). For ICMP frag needed, all we need to do is to send an ICMP frag needed message with the attacker's own IP address (which is unchecked by the resolver as mentioned above). The message embeds a UDP header with a guessed source port and a destination port of 53. It is also supposed to contain the source and destination IP addresses, which should be the resolver's IP and name server's IP respectively. However, some popular DNS software such as Unbound (IPv4 only) and dnsmasq produce public-facing ephemeral ports as introduced in §2.1 (also called wildcard sockets in the kernel terminology). It turns out that Linux (and other OSes) treat such public-facing ports much more liberally and accept any inner destination IP address in an ICMP message, as long as the inner source address matches the resolver's IP and inner source port matches the ephemeral port. This effectively means that against such public-facing ports, one can easily trick the resolver to update the MTUs for any remote IPs (even though the resolver may not have even talked to them before). Therefore, in the attacker's probe packets, we will use its own IP address to fill the destination IP of the embedded packet such that the MTU for the attacker's IP will be lowered if the guessed ephemeral port is correct.

To observe the change in the cache, the attacker can simply send a PING or any other packet (verification packet) that will trigger a reply (verification reply) from the resolver, and observe if the response will be fragmented as a result of the lowered MTU. As shown in Figure 3(b), if ICMP redirect is used for probing, the effect is that the victim resolver becomes unresponsive because the traffic to the attacker will now be redirected to a wrong gateway IP (potentially black hole) set in the redirect message.

4.5 Private-Facing Port Number Inference

Most DNS software (*e.g.,* BIND) will produce private-facing ephemeral ports, rendering the previous method invalid. The first adjustment we have to make is to set the inner destination IP address to the IP of the name server. This is because `__udp4_lib_lookup()` will check the complete four-tuple of the embedded UDP packet to locate the socket that has previously been "connected" to a specific remote IP and port (See §2.1). The exception cache state change is therefore also "private" to the name server and not directly observable by the attacker. For example, even if the MTU for the name server is reduced, an off-path attacker cannot directly observe the



(a) Frag-Needed-Based Public Port Inference (b) Redirect-Based Public Port Inference (c) Frag-Needed-Based Private Port Inference

Keys: D=Destination IP, S=Source IP, M=PMTU, SP=Source Port, DP=Destination Port, MF=More Fragment, Cn(C1-C5)=Colliding IPs, au=Authoritative Name Server

Figure 3: Port Number Inference

change because fragments will go towards the name server directly. Interestingly, it turns out that there is another method to indirectly observe the state change.

The key idea is to leverage the limited number of total slots in the global exception cache. By default, Linux organizes such a global exception cache as a 2048-bucket hash table which uses the destination IP address as the key and has a linked list of length 5 and 6 slots (for IPv6 and IPv4 respectively) to solve collisions for each bucket. When the linked list reaches the limit, the oldest exception will always be evicted and replaced with a newly inserted exception.

The requirement is that the attacker needs to create hash collisions with the name server's IP. As shown in Figure 3(c), the attacker first needs to find 5 IPs (in the case of IPv6) that can be hashed into the same bucket as the name server's IP on the victim resolver's exception cache and control at least 1 IP `C1` (the other 4 IPs can be spoofed). For now, we assume the attacker can find the 5 colliding IPs but will describe our tested strategy in §4.6.

As shown in Figure 3(c), once the colliding IPs are collected, the attacker first fully occupies the 5 allowed slots in the linked list using the 5 different IPs. This can be done by sending a series of ICMP frag needed or ICMP redirect packets wrapping a PING reply packet [26]. The kernel blindly accepts ICMP errors caused by PING replies because they are sent by the kernel with no sockets and therefore matching the socket before accepting is not possible. Subsequently, the attacker would proceed with the ephemeral port scan by probing different source ports with ICMP messages. If a probe happens to hit the correct ephemeral port, a new exception regarding the name server is to be inserted into the linked list and evict the first exception (*i.e.,* `C1`) prepared by the attacker. The attacker can observe this by a verification packet, in the case of MTU caches, checking the current MTU for `C1`.

4.6 Finding IPs that Cause Hash Collisions

Finding IP collisions has been studied before when leveraging IPID side channels [8, 25], where they needed to find a single IP address that collides within the same IPID bucket as the victim. [8] states owning 10,000 IPs would bring the colliding rate to an arbitrary IP over a 2048-entry hash table to more than 98%. Unfortunately, this naive brute force does not transfer well to our attack. Specifically, in order to observe a collision in the case of the exception cache, we know that we need 5 or 6 IP addresses to fully occupy a bucket entry. This means that we need to find at least 50,000 to 60,000 IPs to have a good chance. This is still easily achievable in IPv6 because ISPs often assign a /64 address block by default. However, for IPv4, we consider it possible but a very strict requirement. We therefore come up with an alternative strategy as follows.

Instead of finding the collision set directly, we choose to infer the secret used in the keyed hash function that computes the index into the 2048 buckets. First of all, the hash function is public (listed in the kernel source code). Secondly, since the secret is only 32-bit and persists until reboots, it is possible to crack it once and use it subsequently to check which IPs collide with a given name server's IP. This allows us to target a resolver and potentially poison an arbitrary domain name after a single cracking. To infer the secret, the basic idea is to find some collision set (of 6 IPs in the case of IPv4) that allows us to test which secret can produce the collision set. The key is that in this process we no longer require a collision with a specific IP, i.e., the IP of a name server, and therefore we can benefit from the birthday paradox [55] — it is much more probable to observe a collision at any bucket rather than a given bucket. Based on our empirical evaluation, we only need 3,500 IPv4 addresses to reliably find one or more collision sets on some buckets. In particular, we rented 3,500 AWS EC2 instances to acquire 3,500 different random public IPs. Given that each tiny instance only costs less than one cent per hour, renting instances for sending probing packets is cheap. In practice, we found that one round of probing with 3,500 IPs is usually sufficient to find enough collision sets that allow us to uniquely pinpoint the secret — this takes only minutes computationally with 3,500 tiny CPU cores. In the rare event that we fail, we can simply re-acquire another set of 3,500 IPs and redo the probing. Finally, we also tested the same methodology with IPv6 where only 1,500 addresses were needed to achieve the

same result because an IPv6 hash bucket has only 5 slots instead of 6.

4.7 High-Speed Scans

As one can expect, for either public-facing or private-facing ports, an attacker can probe multiple source ports simultaneously to learn if any of the guesses match the correct ephemeral port. We confirmed with small-scale experiments that both ICMP frag needed and redirect messages are not rate limited on the Internet (see Appendix B). We consider two options below.

Batch scan. We can probe many ports at once, and check whether any of them has hit the correct port. If it does, we can then re-probe a smaller sub-range (e.g., a binary search) to narrow down on the exact port. In this strategy, every round of probes will incur at least one round trip time between the attacker and victim (as is the case in SADDNS [45]). Note that we will need to somehow reset the exception cache state once we hit the correct port in a batch. This is because we have already evicted one of the exceptions we planted earlier. We will describe the methods in detail in Appendix C.

Single packet scan. An alternative strategy is to scan only a single port in each batch (batch size equal to 1). This means that every scan will be accompanied by an additional verification packet. Even though this sounds like a sub-optimal strategy, we point out that the probes can in fact be initiated in a pipeline, without having to wait for feedback for previous probes. This is because our verification packet can encode a unique ID (*e.g.,* ping ID) that can differentiate after which batch of probes, an update in the exception cache has taken place. Of course, we can also use a larger batch size. However, as mentioned, it will incur additional round trips to narrow down the search. In contrast, the single packet scan (a batch size of 1) will allow us to precisely pinpoint which port is open without the additional round trips. The tradeoff is that for every ephemeral port we scan, two packets need to be sent, *i.e.,* one is the probe, the other is the verification packet.

As the attack is highly time-sensitive, we favor fewer round trips over higher bandwidth consumption. We wish to point out that this allows us to scan at a much higher speed than 1,000 per second which was the limit in SADDNS [45].

5 VULNERABLE POPULATION

In this section, we will first study the necessary conditions for the vulnerability to be present and exploitable. Then we study the vulnerable combination of OS and DNS software. Interestingly, the outcome is determined by both the OS and DNS software (sometimes either one). In addition, we also explored historical versions of OS and DNS software because a large fraction of resolvers on the Internet may not be running the latest software. We then conduct a measurement study to measure the vulnerable population of open resolvers on the Internet that satisfy the vulnerable conditions. Due to measurement constraints, we also conduct a small-scale experiment on ICMP redirect attack (see Appendix A).

5.1 Conditions of Successful Attacks

Below we summarize the key necessary conditions for a resolver to be considered exploitable.

- $C1$: Must check the port number in the embedded UDP packet of an ICMP error before processing it. **[OS]**
- $C2$: Must cache the MTU or next-hop information. **[OS]**
- $C3$: Must not ignore the ICMP fragment needed or ICMP redirect messages in the kernel. **[APP/OS]**
- $C4$: Must not shutdown or retransmit the query after receiving ICMP messages. **[APP/OS]**

For $C1$&$C2$, they form the basis of side channels in the kernel. As mentioned earlier, the latest Linux kernel satisfies both conditions.

For $C3$, interestingly the latest Linux kernel allows applications to pass special socket options (either `IP_PMTUDISC_OMIT` or `IP_PMTUDISC_INTERFACE`) which will cause the kernel to ignore the frag needed messages for the corresponding sockets. However, this feature was introduced in Linux kernel 3.15. Therefore, whether or not the condition is satisfied depends on both the kernel and DNS application. Nevertheless, ICMP redirect messages are not affected by any socket option and are always processed in the kernel.

For $C4$, it is a necessary condition because the port scan assumes the ephemeral port stays the same after it is successfully detected. If an application decides to shutdown the connection or retransmit the query after receiving an ICMP message (embedding the correct ephemeral port), then the detected ephemeral port will be effectively forfeited. Interestingly, this is again determined by the OS kernel as well as the application. First of all, the OS kernel has to expose the ICMP error messages to the application layer (again ICMP redirect never gets exposed). Secondly, an application may choose to react to such errors in different ways.

In Table 1, we summarize the vulnerable combinations of Linux kernel and DNS software according to the above conditions. We break down the Linux kernel versions into three groups, representing three major changes that affect the above conditions. Similarly, we break down BIND into two groups because of some key changes in behaviors. As we can see, $C1$ is always satisfied in all recent kernel versions. Regarding $C2$, the Linux kernel since 3.6 is vulnerable in IPv4 because of the introduced exception cache. It took Linux some time until 4.15 to port the same exception cache to IPv6. Therefore, IPv6 redirect attacks, which only require $C1$&$C2$ to work, are only exploitable on kernel versions newer than 4.15. Regarding $C3$, Since Linux 3.15, the socket options mentioned above become available and BIND decides to use `IP_PMTUDISC_OMIT` since 9.12 for IPv4 sockets, leaving the condition satisfied for IPv6 sockets only. For $C4$, since Linux 3.15 and BIND 9.12, `IP_PMTUDISC_OMIT` on IPv4 sockets similarly causes the kernel to notify the application regarding ICMP frag needed errors for sockets that have private-facing ports (therefore does not apply to older Unbound versions and dnsmasq). Furthermore, BIND will retransmit the query (with a different ephemeral port) upon receiving such a notification. As we can see, the interactions between the kernel and application layer are very much inconsistent and evolving constantly. We will discuss the reasoning behind them in §8.2.

In summary, for the latest versions of BIND and Unbound on the latest kernels, their IPv6 sockets can be exploited for the ephemeral port scan. In contrast, dnsmasq is always vulnerable as it does not set any special socket option. Nevertheless, in practice, IPv6 is gaining significant traction in deployment [28]. In fact, as we will

Table 1: Exploitability of Different DNS Software and Kernel Versions

Kernel Version	3.6-3.14				3.15-4.14				>4.15						
DNS Software Version	BIND 9.3-9.11		BIND >9.12		BIND 9.3-9.11		BIND >9.12		BIND 9.3-9.11		BIND >9.12		Unbound >1.5.2		dnsmasq ANY
IP Version	4	6	4	6	4	6	4	6	4	6	4	6	4	6	4/6
$C1$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$C2$	✓	✗	✓	✗	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓
Redir Vuln.	V_{priv}	✗	V_{priv}	✗	V_{priv}	✗	✗	✗	V_{priv}	V_{priv}	✗	V_{priv}	✗	V_{priv}[1]	V_{pub}
$C3$	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✗	✓	✓
$C4$	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓
Frag Vuln.	V_{priv}	✗	V_{priv}	✗	V_{priv}	✗	✗	✗	V_{priv}	V_{priv}	✗	V_{priv}	✗	V_{priv}[1]	V_{pub}
Vuln. in Any	✓		✓		✓		✗		✓		✓		✓		✓

1: V_{pub} before 1.13.0. Note: V_{pub} and V_{priv} indicate vulnerable to public-facing or private-facing port scans respectively.

show in §5.2, half of the popular public DNS resolvers support IPv6. Furthermore, our attack is fully capable of exploiting a dual-stack (IPv4/IPv6) resolver, combined with techniques such as name server muting (as will be discussed in §6.3).

Due to space constraints, we did not show the analysis results of historic versions of dnsmasq and Unbound in Table 1. For dnsmasq, it is vulnerable on all kernel versions since 3.6. For Unbound, it has a similar road map as BIND and starts to use `IP_PMTUDISC_OMIT` since 1.5.2. The only difference is that it used public-facing ports in the past. This leads Unbound to be not only vulnerable in the IPv4 of kernel versions between 3.15 and 4.14, but also IPv6 in the same kernel ranges. This is because the public-facing ports can be successfully scanned (as shown in Figure 3) as long as the MTU or redirect information is stored somewhere in the kernel. In practice, for kernel version 3.15 to 4.14, such info is stored in a tree which can only time out as opposed to being forcefully evicted.

Other Operating Systems. We have additionally analyzed FreeBSD (whose networking stack is also used by macOS) and Windows with regard to the previously described conditions.

For FreeBSD, it is not vulnerable because $C1$&$C2$ are broken for ICMP frag needed and redirect respectively. For ICMP frag needed messages, even though the OS will check the embedded four-tuple and act accordingly, it does not store any PMTU information in any kernel-maintained data structure and thus breaking $C2$. Instead, it simply forwards the error to the application layer. This is actually not compliant with RFC1191 [47] which explicitly states that "the IP layer should associate each PMTU value that it has learned with a specific path" and "it (a host) should be able to cache a per-host route for every active destination". For ICMP redirect packets, surprisingly, FreeBSD will blindly accept them without checking the embedded four-tuple and therefore breaks $C1$.

For Windows, we reverse-engineered `tcpip.sys` and `ntoskrnl` of a Windows 10 copy. We found that there is a similar hash table storing the path information (including the MTU). However, we did not find any eviction algorithm and it will only stop inserting new exceptions after the kernel runs out of memory. Although the attacker can still leverage this as a side channel, due to the large and different memory configurations, it is hard to do so in practice. However, lacking a cap on memory consumption of the hash table would lead to a potential DoS attack on the entire system. Interestingly, although this breaks the private-facing port scan, a public-facing port scan is nevertheless feasible on Windows because it does not rely on being able to evict exception entries in any shared resources (See §4.4). On Windows Server 2019, the built-in Microsoft DNS server uses public-facing ports which makes it vulnerable.

5.2 Open Resolvers

Now we move on to measure the vulnerable population in the real world. Note in this section we focus on the attack leveraging ICMP frag needed messages only. This is because ICMP redirect based attacks require IP spoofing even for port scans, and we are concerned that it is invasive to conduct such a large-scale IP spoofing experiment. Instead, we defer to Appendix A for a small-scale measurement of the conditions of the redirect-based attacks.

Setup and Dataset. Open resolvers represent hosts that provide recursive DNS lookup services to the public. We obtain a list of open resolvers from Censys.io [22], which contains 1.84M IPv4 addresses, serving as the dataset used in our measurement. Unfortunately, the list does not contain IPv6 open resolver addresses. Nevertheless, these IPv4 addresses only correspond to the frontend IPs. In practice, most open resolvers will go through backend servers that conduct the actual DNS query on behalf of the frontend. Therefore, we design a method to solicit queries from IPv6 backend servers. Specifically, we control two domain names whose NS records point to an IPv4 and an IPv6 address respectively. For each frontend IP, we always send two queries asking for the IPv4 and IPv6 domain names respectively. For the domain where its NS record points to an IPv6-only address, it will force a backend server to use its IPv6 address to contact our name server. In the end, we are able to receive 129,196 queries from IPv4 addresses and 27,541 from IPv6 addresses.

Methodology. When a backend server (either IPv4 or IPv6) contacts our name server, we will perform the following four tests that approximately correspond to the four conditions we discussed earlier.

T1: The rejection of the ICMP error when the embedded source port is incorrect. To verify $C1$ in §5.1, we first send a PING to the resolver and record the reply. Then we craft an ICMP fragment needed packet wrapping the DNS query we received to signal that the PMTU is lowered. Before we send it, we deliberately change the source port of the embedded UDP packet to a different random value to check whether the resolver will blindly accept ICMP packets without checking the port number. After sending that forged packet, we send another PING and check if the ICMP is accepted. If the

PING reply is not fragmented, we consider the resolver rejects the ICMP error and thus meets $C1$.

T2: The existence of the next hop exception cache. To verify $C2$ in §5.1, ideally we would want to directly test the existence of an exception cache. However, as described in §4.6 this will require us to find 5 or 6 IPs that would be hashed into the same bucket, causing the hash collision. Although it is a one-time effort, targeting every single open resolver will require sending a large amount of traffic which can be overly invasive. Therefore, we decide to resort to `nmap` to fingerprint the OS version of the resolver and check whether the cache exists according to the OS version discussed in §5.1. Note that `nmap` may not be perfect, especially when considering backend servers may not always have open TCP ports, through which most of the fingerprints are extracted by `nmap`. Nevertheless, we can use the distribution obtained from resolvers that do have open ports and extrapolate to those that do not. To minimize the impact, we sampled 20 out of 8,141 backend resolver IPs that have a valid `nmap` signature and performed the collision test using 3,500 rented IPs following the methodology described in §4.6. Note that this is still an intrusive test (we do slow down the packet speed to about 1,000 pps to minimize any disruption) and thus cannot scale. The results show 16 out of 20 servers support `nmap`'s conclusion and therefore we estimate the accuracy of `nmap` 80%.

T3: The acceptance of the ICMP error. To verify $C3$ in §5.1, we use a similar test to *T1* but without modifying the port number to verify if the resolver is willing to accept the ICMP packet at all. Additionally, if there is no PING reply at all, we will send a truncated DNS response to solicit the TCP query from the resolver. If the MSS in the TCP header is decreased according to the PMTU value indicated in our ICMP packet (which we verify to be the behavior of modern Linux kernels), it also means the resolver has accepted the PMTU value inside the ICMP packet. Besides, we will conduct another test by changing the destination IP address in the wrapped IP packet if we find the resolver accepts the original ICMP. If the resolver also accepts the modified ICMP, it means its port is open to the public, and otherwise, we consider its ephemeral port as private-facing.

T4: The open-port status after receiving the ICMP error. To verify $C4$ in §5.1, after the ICMP fragment needed is sent during *T3*, we follow up with a "truncated response" (if it is not sent in *T3*) indicating the response is too big which will cause the resolver backend to switch to TCP. If we observe a TCP handshake, it indicates that the ICMP error did not cause the resolver to close the original ephemeral port, therefore supporting the attack. In the more rare cases, even if we did not observe any TCP connection attempt, it is still possible that the ephemeral port is open and it is simply due to the resolver not supporting DNS over TCP. In such cases, we will check whether the name server will receive a retransmitted query (with a different ephemeral port) from the resolver immediately, which potentially indicates that the ICMP has induced the DNS software to close the ephemeral port and transmit another query. To distinguish between the ICMP-induced retransmission and the timeout-induced retransmission, we record the time delay between the ICMP transmission and the time we received the retransmitted query. Specifically, if the delay is close to RTT, which we collect in *T1* by measuring the time delay between the PING response and the request, *i.e.,* within a 10% margin of difference, we consider the retransmission to be caused by the ICMP. Otherwise, if the delay is larger than RTT, we will consider the retransmission to be timeout-induced (and thus still supporting the attack).

Results. Overall, out of the 156,737 backend resolver IPs that reach our name servers, 13.85% of them are estimated to be vulnerable. If we count by frontend resolver IPs, out of the 1.84M, 37.72% are estimated vulnerable. This is because a large number of frontend IPs share the same backend. To further break down the total 13.85% vulnerable population in the backend, we find that 13,914 (8.9%) are clearly vulnerable to public-facing port scans. However, when we count the vulnerable population regarding the private-facing port scans, it requires a more accurate estimate of the Linux kernel version from `nmap`. Unfortunately, as mentioned earlier, we find `nmap` has a relatively low success rate of OS fingerprinting: only 63.26% for IPv4 addresses and 1.06% for IPv6 addresses. We therefore use the distribution of kernel versions observed from the 63.26% IPv4 hosts to estimate the total vulnerable population. In particular, within these IPv4 hosts, we find that 58.66% of them have the IPv4 exception cache only or also the IPv6 exception cache. We then apply the 58.66% to the 13,277 resolver backends that are suspected to be vulnerable (passing all other tests), resulting in an estimate of 7,788 backends being vulnerable to private-facing port scans.

The results indicate that the majority of the vulnerable population is not actually running BIND. Instead, they could be running an older Unbound, dnsmasq, or other DNS resolver software that we have not explicitly tested. Among the servers that are not vulnerable, most of them are simply because they do not accept the ICMP frag needed messages (including cases that we cannot tell) and fail in *T3*.

Public Resolvers. We also highlight the results of a few well-known public DNS services and summarize the result in Table 2. Overall, we find 6 out of 12 to be definitely vulnerable at the time we performed the test, 3 in IPv4 and 3 in IPv6, including famous providers such as OpenDNS and Quad9. Interestingly, although the most popular DNS software BIND is not vulnerable in IPv4 in its latest releases, there are still 3 public resolvers vulnerable in IPv4, indicating that they are either running an older BIND version or a different DNS software (we know Cloudflare runs Knot [2]). Note that currently only 6 providers support IPv6 (others are marked as N/A) and we expect more DNS services to be impacted as they start supporting IPv6.

The most common reason for not being vulnerable is again because they failed *T3*, *i.e.,* the ICMP fragment needed messages do not appear to trigger the MTU to decrease. As we can see in Table 2, there are still a few cases where we are unable to fingerprint the kernel versions even after we tried testing a few custom fingerprints in addition to nmap (marked with "?" in the T2 column). For such cases, we simply mark them as "Possibly Vulnerable" ($P_{priv/pub}$) when they pass all other tests, since it is likely their public servers are well-maintained and using a newer Linux kernel.

6 PRACTICAL CONCERNS

In this section, we will describe a few practical considerations which will influence the success and reliability of the attack.

Table 2: Vulnerable Status of Public Resolvers

Name	Frontend IP	IPv4 Backend					IPv6 Backend				
		T1	T2	T3	T4	Vulnerable	T1	T2	T3	T4	Vulnerable
Google	8.8.8.8	✓	✗	✗	✓	✗	✓	?	✗	✓	✗
Cloudflare	1.1.1.1	✓	✓	✓	✓	V_{priv}	✓	✗	✓	✓	✗
OpenDNS	208.67.222.222	✓	?	✓	✓	P_{pub}	✓	✓	✓	✓	V_{priv}
Comodo	8.26.56.26	✓	✓	✗	✓	✗	N/A				N/A
Quad9	9.9.9.9	✓	✓	✗	✓	✗	✓	?	✓	✓	V_{pub}
AdGuard	94.140.14.14	✓	✓	✗	✓	✗	✓	✓	✓	✓	V_{priv}
CleanBrowsing	185.228.168.168	✓	✓	✗	✗	✗	N/A				N/A
Neustar	156.154.70.1	✓	✓	✓	✓	V_{pub}	✗	?	✓	✓	✗
Yandex	77.88.8.1	✓	✓	✗	✓	✗	N/A				N/A
Baidu	180.76.76.76	✓	✓	✓	✓	V_{priv}	N/A				N/A
114	114.114.114.114	✓	✓	?	✓	?	N/A				N/A
Ali	223.5.5.5	✓	✓	✗	✓	✗	N/A				N/A

6.1 Small Attack Window

By default, the attack window is only a round trip time (ranging from tens to hundreds of milliseconds) between a resolver and a name server, forcing the attack to finish both the port scan and the injection of 65,536 fake DNS responses (brute-forcing the TxID) in a small amount of time. Nevertheless, this does not represent a fundamental hurdle as the attacker can simply repeat the attack multiple times; as long as one of the attempts succeeds, the cache will be poisoned. Specifically, in practice, we find an attack attempt more likely to succeed if the correct ephemeral port is located at the beginning of the port scan range (see numbers in §7). To circumvent the wait of TTL for the legitimate record to time out in case of a failed attack attempt, we use a previously proposed method [38, 45] to improve the speed of the attack. The basic idea is to issue queries with random subdomains and forging a response containing an NS record, causing the resolver to cache the wrong name server such that all future queries (including the target domain and all subdomains) will be directed to the malicious name server. This method is well documented in [38, 45] and works against both BIND and Unbound.

To increase the attack window, an attacker can attempt to mute a name server, *i.e.,* preventing the name server from responding to a resolver's query. If successful, a resolver will keep increasing its wait time, i.e., attack window, to typically 1-2s for BIND and potentially larger than 30s for Unbound [45]. Specifically, it was reported that the response rate limit (RRL) feature on name servers can be abused for this purpose [45] where 18% of the Alexa top 100k websites were shown to be affected. Alternatively, a DoS attack can be launched to mute the name server.

Coincidentally, one of the ICMP messages, ICMP redirect, can be also used for name server muting. The idea is to send the malicious ICMP redirect to either the victim resolver or the name server to reroute the traffic destined to each other to a black hole. Since the query/response is lost after it reaches the wrong next hop, the victim resolver would keep the ephemeral port open for responses until the query timeouts (can be several seconds [45]) and therefore creates a huge attack window.

6.2 Multiple Name Servers & Backend Servers

Multiple name servers. It is also quite common for domains to have multiple name servers. Resolvers may choose to query these name servers in a round-robin fashion (where the order is randomized). In fact, this is considered a defense against DNS cache poisoning attacks [45]. However, this defense has little impact on our attacks for the following reasons.

For resolvers with private-facing ephemeral ports, we can infer the ports specific to different name servers simultaneously by running multiple scanning instances. Since it is unlikely the name servers' IPs will share the same hash bucket given that most second-level domains (*e.g.,* acm.org) only have three or fewer name servers [45], the side channels can be independently leveraged without self-interference.

For resolvers with public-facing ports, the attacker can just scan the port as if there was only one name server since the kernel does not check the destination IP address wrapped in the ICMP probe. The only difference lies in the TxID brute-forcing, where the attacker would inject multiple groups of 65,536 fake response packets, where each group uses a spoofed IP of a different name server. Due to the low number of name servers typically configured, this additional load of packets is not really a fundamental hurdle.

In addition to the above, there is an optional step called "name server pinning" [45] that can further improve the success rate when multiple name servers are encountered. In addition to previously proposed techniques [45], we propose two new methods again based on ICMP messages, *i.e.,* either host/port unreachable or redirect. In the case of BIND resolvers, every time when a query is initiated, we can immediately flood 65,536 (representing the worst cases. BIND uses only 23,232 ports by default) ICMP host/port unreachable messages containing all possible ephemeral ports with a specific name server's IP as the destination IP address in the embedded IP header. This will cause BIND to give up a particular name server in the duration of a query session (up to 10 seconds by default [45]). This is because the OS will pass the host/port unreachable messages to BIND, which will make the subsequent decision to forgo the name server (one of the 65536 guessed ports will match the ephemeral port). Alternatively, we can apply targeted name

server muting as mentioned in §6.1 and targeted ICMP redirect to achieve a similar effect.

In the case of Unbound, ICMP redirect can be used as described above to mute specific name servers. This is because Unbound has special logic to "blacklist" name servers that are non-responsive repeatedly [45]. Therefore, the ICMP redirect will have a prolonged pinning effect beyond a single query session.

Multiple backend servers. Finally, large DNS resolvers tend to have multiple backend servers behind a single frontend IP — usually an anycast one, *e.g.,* 8.8.8.8. These backend servers are the actual workers that talk to the name servers and they are the ones that maintain DNS caches. Therefore the backend servers should be the actual attack target. An attacker can map out the IPs of the backend servers by setting up an attacker-controlled name server and issuing a query of the attacker-controlled domain. This will create an additional challenge to the attacker, as a particular query may get routed to a randomly selected backend IP not known to the attacker. This will mean that the attacker needs to target $m \times n$ pairs of resolver backends and name servers, where m is the number of backend IPs and n is the number of name servers. Otherwise, if the attacker picks only a single backend server to attack, it will have a reduced probability of $\frac{1}{m}$ (assuming the probability of choosing backend servers is uniformly distributed) to succeed. Fortunately, when m is large, it is typically a heavily distributed system that the selection of the backend IPs is actually not random at all. Instead, [45] indicates that it is typically based on location. In other words, backend servers that are located closer to a name server will be more likely to be picked for a given query (destined to the name server). In such cases, the attacker only needs to target a small number of backend servers simultaneously or even a single one and is still able to achieve a decent success rate.

6.3 Dual-Stack Resolvers

As mentioned earlier in §5.1, the latest BIND and Unbound will instruct the Linux kernel to ignore ICMP frag needed messages for IPv4 sockets. Therefore, the vulnerability applies to only IPv6 sockets against them. In practice, both IPv4 and IPv6 are enabled by default in recent Linux distributions (*e.g.,* Ubuntu 20.04 and Red Hat 7). Therefore, we need to understand how to target their IPv6 sockets in the presence of IPv4 sockets. Specifically, BIND and Unbound by default will query different name servers in a round-robin fashion regardless of whether the IP address is IPv4 or IPv6. As a result, we can apply the same strategy as outlined in §6.2 to handle them. Specifically, we can apply name server pinning to cause the IPv4 name server to become non-responsive and never (or rarely) used by a resolver.

6.4 Noises

Background traffic. There are two potential sources of background traffic at the resolver that can influence the ephemeral port scan. First, the victim resolver may have multiple outstanding queries at the same time. During the port scan, it is possible that the ephemeral port we find belongs to a different query. It is not a serious concern for private-facing ports as they are "visible" to only specific name servers, and there are typically few, if any, outstanding queries towards the same name server (in addition to the one triggered by the attacker). However, it can affect the public-facing ports because the ephemeral port of *any* outstanding query to *any* name server can show up during a scan. Nevertheless, we point out that any of the strategies described in §6.1 that can extend the attack window will automatically mitigate this concern. This is because the outstanding query triggered by the attacker would then last for much longer (possibly seconds) while other ordinary queries will only last for hundreds of milliseconds at most. Therefore, we can simply confirm that the port lives long enough before deciding to brute force the TxID.

Another type of background traffic is the benign ICMP error messages a resolver may receive during a port scan. They can create additional entries in the exception cache. This has little impact on public-facing ports because the attack requires only one entry to be created in the cache and it is highly unlikely that there are many naturally-occurring ICMP errors that will hash into the same bucket as the attacker's entry and evict it, during a short time frame of an attack. For private-facing ports, the attack does require all five exception entries in the same hash bucket to be intact during the scan. However, it is still unlikely to have a hash collision from benign ICMP messages during a short time period. Even if it does occur in practice, it will just interfere with one attack attempt (triggering a false positive) and the next attack attempt will follow immediately.

Packet Losses. Although unlikely, if the probing ICMP containing the correct ephemeral port happens to be lost, false negatives can arise. In such cases, the attacker simply moves on to the next attempt. If the loss is on the verification or verification reply packets, it will not affect the attack since the attacker can easily notice and retransmit the verification packet. This is because a verification reply is always supposed to come back either fragmented or not (depending on whether the ephemeral port is guessed correctly).

Packet Reordering. Reordering can cause false negatives on public-facing port scans and both false positives and false negatives on private-facing port scans. Specifically, if the verification packet accidentally arrives before the ICMP probe containing the correct ephemeral port, it will fail to detect the exception cache change and lead to false negatives. Furthermore, if the private-facing port is being scanned, such a false negative would mislead the attacker into continuing the scan despite the fact that one of the planted exceptions has already been evicted. This is guaranteed to lead to a false positive in the scanning of the next batch of ports, as the eviction will be detected by the next verification packet. To mitigate such problems, a small time gap can be inserted between the probing and the verification packets. To mitigate the risk of false positives and flooding the resolver with too many packets, we always double-check whether a detected port is a true positive before deciding to brute force the TxID.

7 EVALUATION

To evaluate the efficiency of our attacks without causing real-world damage, we tested the attack in a controlled environment with different server configurations and simulated network conditions. Overall, our attacks can succeed in minutes and have a near-perfect success rate. Note that inferring private-facing ephemeral ports

requires inferring the colliding IPs as described in §4.6. However, since it is only a one-time effort for each resolver, the time used for the attack does not include the time for inferring colliding IPs.

7.1 Resolver Attack

Attack setup. In this attack, we evaluate the power of the fragment needed attack based on the private-facing port scan. There are 3 hosts involved in the attack: the attacker host, the victim resolver and the name server, all of which are controlled by us. The attack program is executed on the attacker host, which is a MacBook running macOS (Darwin 19.6.0) and is connected to the victim resolver via a wired router (1Gbps). The victim resolver is a PC (with a single CPU of Intel Core i7-9700) running BIND 9.16.13 on Ubuntu 20.04 (Linux 5.11.16). The name server, where our domain's records are kept, is hosted on AWS and also running BIND 9.16.13. The attacker's host, and the victim resolver are at home and connected to the name server via residential Internet and all of the traffic is sent in IPv6. The goal of the attack is to poison the cache of the victim resolver so that our own domain's A record will be altered in the cache.

We conducted 9 groups of experiments to evaluate the impact of the different server configurations, network conditions, and levels of background query traffic on our attack as shown in Table 3. Specifically, we first performed a baseline (*Base*) attack, where the attacking conditions are ideal. Then we changed one configuration or network condition at a time to check how they would influence the attack. Then, we tested the performance of our attack against a more realistic configuration and network condition to simulate a real-world scenario (*Real*). Finally, we introduced the background query traffic to the resolver and evaluate how the interfering query traffic affects our attack. Specifically, in *Real*1, we followed the workload on a production resolver reported in SADDNS [45] with 70M queries per day, averaging at 810 queries per second. To simulate the worst-case scenario, the domains in these queries are randomly sampled from the Alexa top 1M to reduce the cache hit, leading to more open ports. In *Real*2, we added another 10 queries per second asking for the same domain that the attacker is trying to poison (which would cause confusion to our port scan).

To stay stealthy, we limit the rate of our packets to 7k pps (including both the probes and verification packets), which is 3.5k ports scanned per second. Note that 7k pps applies to the port scan phase only. During the TxID brute-forcing phase, we limit our brute force speed to 40 kpps and 70 kpps for *Real*1 and *Real*2 (to compete with the background traffic). We simulate varying degrees of packet losses, jitters, and delays according to the representative numbers reported on the Internet [19, 24]. Besides, we also evaluated how the name server muting level and the number of name servers affect our attack. Although the name server can be completely muted (*i.e.,* 100% muting level) using ICMP redirects as mentioned in §6.1, we also evaluate the scenario where it is difficult to completely mute a name server (*e.g.,* leveraging response rate limit). As mentioned in §4.7, we also studied the impact on the attack performance when using different batch sizes (*i.e.,* the number of ports scanned in a batch).

Results. Overall, we find our attacks can succeed on average in 1.3 to 15.6 minutes, depending on the setup. Note that we consider a test failed if it still does not succeed after an hour. In both baseline (*Base*) and packet loss (*Loss*) experiments, the attack succeeds in around 80s, indicating the minimal impact of moderate packet losses. This is expected as discussed in §6.4. In the *RTT* experiments, we found the delay and jitter do affect our attack. Under such unstable networks, the attack may experience false positives as the verification packet may be received before the probe. Fortunately, our attack can still succeed because we have inserted time gaps to minimize reordering (see §6.4).

Table 3: Resolver Attack Results

Exp.	Pkt. Loss	RTT range /ms	NS Mute Level	# of NS	Batch Size (N)	Bg. Noise	Avg. Time /s	Succ. Rate
Base	0%	0.3-1.2	100%	1	1	0	80	20/20
Loss	0.20%	0.3-1.2	100%	1	1	0	83	20/20
RTT	0%	37-43	100%	1	1	0	149	20/20
ML	0%	0.3-1.2	50%	1	1	0	713	5/6
NS	0%	0.3-1.2	100%	3	1	0	347	20/20
Batch	0%	0.3-1.2	100%	1	1024	0	496	5/5
Real	0.20%	37-43	80%	2	1	0	410	20/20
Real1	0.20%	37-43	80%	2	1	810	659	10/10
Real2	0.20%	37-43	80%	2	1	810+10	933	10/10

For name server muting levels, we find they do have a significant impact on our attack but are much smaller compared to the impact on SADDNS [45]. Under the same muting level (50%), our attack (*ML*) is 10x faster than SADDNS. This should be attributed to the substantially faster scan speed and the fact that we do not need to perform iterative probes to narrow down the search space. As a result, this allows our attack to fare better under smaller attack windows. Experiment *Batch* further confirms this. With N=1024, the average success time increased by five times compared to the baseline where N=1. Note in *ML*, there is one attack attempt that failed (after an hour) likely due to a link-layer issue that we are unable to reproduce.

We also notice it would take ~4x the amount of time to poison a domain with 3 name servers (*NS*). This is due to the limit of 7k pps packet sending rate, which forces us to scan for each name server at 1/3 of the total rate. However, if an attacker scans with 3 times the bandwidth, the result would have been close to the baseline.

In the real world scenario experiments (*Real*), we succeeded in 410s on average, which is 2x the speed of SADDNS with the same setting, despite the fact that our test is against BIND which is known to have a much smaller attack window (about only 2s as experienced in our experiments) than Unbound (more than 30s as reported in SADDNS [45]).

Finally, for the background query traffic experiment *Real*1, we found random domain queries do not significantly impact the attack performance. As expected, we do not find our scan being confused by the additional open ephemeral ports because they are all private ports and not visible to the name server which hosts the target domain name (see §6.4). Instead, we find that the increase of time-to-succeed is mostly attributed to the machine being slowed down in processing these query packets. Compared to *Real*1, *Real*2 experienced worse results because the additional 10 queries per second can generate ephemeral ports that are visible to the target name server, therefore creating confusion to our scan. Looking into the

detailed logs, we see that *Real*2 experiences 22 failed TxID brute force attempts on average whereas *Real*1 experiences only 11. The majority of the additional failed brute force attempts are due to the failure in inferring the correct port number.

In general, we make two additional general observations on the results. First, the overall attack time is spent predominantly on repeated port scans (starting from the smallest port to the largest), accounting for 96% to 98% of the time. The remaining time is spent on brute-forcing the TxIDs. Second, the time-to-succeed varies significantly depending on how close the correct port is to the beginning of the port scan. In many cases, we see the time-to-succeed being a few seconds, whereas in the worse case (especially when noise is introduced), it can take 30 minutes to find the port and succeed in brute-forcing the TxIDs.

7.2 Other Attacks

Forwarder Attack. To evaluate the performance of the public-facing port scan, we launched the attack against an ASUS AX6600 Wi-Fi router which has a built-in DNS forwarder. We used a similar setup as the *Base* experiment in the resolver attack where the attacker is a LAN machine that can trigger DNS queries on the forwarder. In this attack, we used the IPv4 network and set the upstream resolver as 8.8.8.8, which the attacker needs to spoof when brute-forcing the TxIDs. Finally, the attack succeeded in 13s.

Redirect Attack. Similar to *Base*, we launched the redirect-based attack under the same settings, with the only change of replacing IPv6 with IPv4, to demonstrate the private port scan under different IP versions. Finally, the attack succeeded in ~150s.

8 DISCUSSION

8.1 Comparison with SADDNS

Ephemeral port inference method. As mentioned in §4, the first and foremost difference is the use of ICMP probes in our attack. By design, ICMP messages are considered errors that should not solicit any responses [12]. This makes them an unlikely avenue to probe any secret. Nevertheless, we demonstrate a superior understanding of the nature of side channels, making ICMP probes a successful entry point in UDP ephemeral port scans.

Side channel type. Our side channel leverages the space resource limit (*i.e.,* the space for storing the next hop exception cache is limited) while SADDNS' side channel leverages the time resource limit (*i.e.,* ICMP error generating rate is limited). Moreover, our side channel arises when processing incoming ICMP packets (and this is why we can still infer the ephemeral port despite no reply to the ICMP probing packet is sent) while SADDNS' side channel arises when processing outgoing ICMP packets.

Port scan speed. Thanks to the novel space-constraint side channel arising in the packet receiving path, the ICMP-based ephemeral port scan rate can be theoretically unlimited. In practice, the attacker can also adjust the scan rate and strategy flexibly to achieve a higher success rate according to different network conditions. SADDNS, however, only allows the fixed 1000 pps slow port scan due to the nature of the time-constraint side channel it uses. The slow scan rate leads directly to a lower success rate when racing against legitimate DNS responses.

Resistance to the noise. Unlike the global counter used in SADDNS, which is shared across all remote IPs, the exception cache used in our side channel is a hash-based structure and is only shared with a smaller range of IPs, which reduces the noise level of our side channel — it is less likely to be interfered with by background traffic associated with random IPs. Besides, SADDNS requires a strong 50-ms time block synchronization, which can be hard to achieve with noise. In contrast, our attack does not have such a strict synchronization requirement.

Preparation of the attack. Compared to SADDNS, our attack requires an additional step of inferring colliding IPs that hash into the same bucket. Nevertheless, as described in §4.6, it is only a one-time effort for each resolver we target.

8.2 PMTUD and DNS

It has been a controversial decision to enable Path MTU Discovery (PMTUD) on DNS packets. Historically, [10] indicates ICMPv6 packet too big messages could benefit the responsiveness of DNS queries while [30] argues the opposites claiming that it could lead to fragmentation-based DNS cache poisoning attacks. As a result, we see DNS software (especially BIND) changing back and forth regarding its socket options related to PMTUD.

Recently, there appears to be a convergence as both BIND and Unbound start to set the socket option of `IP_PMTUDISC_OMIT`, which instructs the kernel to never reduce the MTU. This is mostly in fear of the fragmentation-based DNS cache poisoning attacks that rely on tricking the name server to fragment its responses [30]. Interestingly, this option is now enabled for the sockets on both the name servers and resolvers (even though the concern was mostly on name servers). In addition, both BIND and Unbound decide to enable this option for IPv4 sockets only and leave IPv6 unchanged.

The reason for leaving IPv6 sockets unchanged is likely that fragmentation can be avoided most of the time as the minimum MTU is increased to 1280. This means that any link carrying IPv6 datagrams must be able to handle at least 1280 bytes of payload. This is large enough to transmit most DNS packets and makes the fragmentation-based attacks unlikely to succeed.

8.3 Existing Defenses

There are already a number of additional DNS security solutions in addition to the randomization of ephemeral ports that can defend against DNS cache poisoning attacks. However, they are not widely deployed due to various reasons.

DNSSEC adds the data origin authentication and data integrity to DNS [52] and therefore by design prevents any attacker without holding the correct key to inject any records. However, only 1.85% of Alexa Top 10k websites enable the DNSSEC, and only 12% of open resolvers actually validate the record integrity if provided [18]. During our experiment, we found famous websites like Google, Facebook, and Twitter do not even have DS records on the parent zone, which is a necessary record for DNSSEC to function.

0x20 Encoding is proposed to randomize the upper and lower case of the letters in the domain name (of both the query and response), thus introducing additional entropy beyond the TxID and ephemeral port [57]. The amount of entropy increases as the length of the domain name increases. Unfortunately, it is found

recently [45] that the 0x20 encoding has compatibility issues (since it requires support from the name server) and 12 out of 14 popular public DNS resolvers tested do not use it (which we also confirm to remain true at the time of writing). Famous resolvers like 8.8.8.8 only enable 0x20 encoding for whitelisted domains.

DNS Cookie is yet another secret exchanged between a resolver and name server, designed to defeat any form of off-path response injection [3]. Similar to DNSSEC, DNS cookie requires support from both the resolver and name server to work properly. However, only 5% of open resolvers enable this by default and some may even reject queries with DNS cookie, as reported in recent studies [21, 45], indicating compatibility is still a serious concern. Moreover, DNS cookie is only a solution during the transitioning period into DNSSEC (as it is strictly weaker than DNSSEC), which can be another reason why it is not being widely deployed.

8.4 New Defenses Against Our Attack

In addition to the existing defenses, we also propose a set of orthogonal and near-term solutions to mitigate our attack. We will further discuss the generalized defense against the network side channels in Appendix D.

Set proper socket options. The most direct way is to use the socket option `IP_PMTUDISC_OMIT`, which instructs the OS not to accept the ICMP frag needed messages and therefore eliminates the side-channel related processing in the kernel. However, legitimate ICMP frag needed messages can be sent by a router which will be ignored also. In such cases, we recommend that the application can retransmit the query using TCP to avoid failing to transmit a UDP query due to real problems with the MTU.

Randomize the caching structure. Similar to the solutions to other network side channel attacks [14–16, 45, 51], sufficiently randomizing the shared resource would make the side channel practically unusable. With regard to the exception cache, we recommend a few places where randomization can take place: (1) the max length of the linked list used for solving hash collisions, (2) the eviction policy (currently the oldest will always be evicted), (3) the secret of the hash function, *i.e.,* we can re-key periodically (every few seconds or tens of seconds).

Reject ICMP redirects. Redirects are originally designed for a network with multiple gateways (similar to a router with multiple next-hop options). If a DNS server has only one default gateway, the administrator should consider ignoring ICMP redirect messages to prevent redirect-based attacks, which can be configured via `sysctl` (see Appendix A).

8.5 Ethical Concerns

We conduct our experiments with ethics as a top concern. During the measurement of the vulnerable population in the wild, we attempt to minimize the impact of our probes by (1) querying our own domain and (2) at a mild speed for each resolver (under 1,000 packets per second). Also, we avoid sending suspicious-looking packets, e.g., an excessive number of ICMP packets or packets with spoofed IPs that can potentially trigger firewall alerts.

In the evaluation section, since it requires flooding fake DNS responses to finish the end-to-end attack, we refrain from attacking any real resolver and performed the attack in the local setup instead.

Responsible disclosure. We have reported our findings to the key stakeholders in the DNS community, including BIND, Cloudflare, and Linux. Linux has applied two patches on both IPv4 and IPv6 stacks to randomize the depth of the linked list storing the exceptions. BIND also begins to set `IP_PMTUDISC_OMIT` on IPv6 sockets from 9.16.20 concurrent to our study.

9 RELATED WORK

DNS Cache Poisoning Attacks The off-path DNS cache poisoning attack was first popularized by Dan Kaminsky in 2008 [36]. After the ephemeral port number was randomized, fragmentation attacks [13, 33] were invented to eliminate the need for guessing the source port by replacing the second fragment of the original DNS response. However, these attacks usually have some strong assumptions like predicting the IPID of the packet or running a puppet on the victim resolver. Port exhaustion attacks [9, 32] derandomize the ephemeral port number by exhausting all but one available ephemeral port, leaving the DNS resolver only one fixed port to use. These attacks also require the puppet to work. In 2020, SADDNS [45] was invented as the first side-channel based DNS cache poisoning attack. However, this attack is slow and usually takes tens of minutes to finish. At the same time, Zheng *et al.* ports the fragmentation attacks to the forwarder and found the attack much easier when using the attacker-controlled name server to force the fragmentation [59]. Jeitner *et al.* present a novel way to poison DNS caches by exploiting domain parsing ambiguities [35]. Amit Klein uses the IPID value to predict the random UDP ephemeral port number by exploiting the cryptographic properties of the shared random number generator [37].

Overall, unlike other works, our attacks are based on another unique side channel in ICMP and provide another way to poison the DNS cache using the fast ICMP-based port scan.

Side Channel Attacks Side channels have been leveraged in network attacks [7, 39, 43, 50]. Specifically, [23] leverages IPID global counter to probe open ports. [49] utilizes the stateful firewall to infer the TCP sequence number. [14] uses global challenge ACK counter to infer the TCP sequence number and hijack the TCP connection off-path. Besides, Cao *et al.* presents an automated tool for finding side channels in the TCP stack using model checking and found several other side channels inside the Linux TCP stack [16].

10 CONCLUSION

This paper presents novel side channels during the process of handling ICMP errors, a previously overlooked attack surface. We find that side channels can be exploited to perform high-speed off-path UDP ephemeral port scans. By leveraging this, the attacker could effectively poison the cache of a DNS server in minutes. We show that side channels affect many open resolvers and thus have serious impacts. Finally, we present mitigations against the discovered side channels.

ACKNOWLEDGMENTS

We wish to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the National Science Foundation under Grant No. 1652954.

REFERENCES

[1] [n.d.]. SADDNS website. https://www.saddns.net/.

[2] 2018. Introducing DNS Resolver, 1.1.1.1 (not a joke). https://blog.cloudflare.com/dns-resolver-1-1-1-1/.

[3] D. Eastlake 3rd and M. Andrews. 2016. *RFC 7873: Domain Name System (DNS) Cookies.* Technical Report. https://tools.ietf.org/html/rfc7873

[4] S. Deering A. Conta and Ed. M. Gupta. 2006. *RFC 4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification.* Technical Report. https://tools.ietf.org/html/rfc4443

[5] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: An automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2473–2487.

[6] Akamai. 2021. Configure split-DNS forwarding on Cisco routers. https://learn.akamai.com/en-us/webhelp/enterprise-threat-protector/enterprise-threat-protector/GUID-5916F532-2D1D-4CC0-B926-9D6CC44BEF33.html.

[7] G. Alexander and J. R. Crandall. 2015. Off-path round trip time measurement via TCP/IP side channels. In *2015 IEEE Conference on Computer Communications (INFOCOM)*.

[8] Geoffrey Alexander, Antonio M Espinoza, and Jedidiah R Crandall. 2019. Detecting TCP/IP Connections via IPID Hash Collisions. *Proceedings on Privacy Enhancing Technologies* 2019, 4 (2019).

[9] Fatemah Alharbi, Jie Chang, Yuchen Zhou, Feng Qian, Zhiyun Qian, and Nael Abu-Ghazaleh. 2019. Collaborative Client-Side DNS Cache Poisoning Attack. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1153–1161.

[10] Hanieh Bagheri, Victor Boteanu, Willem Toorop, and Benno Overeinder. 2013. Making do with what we've got: Using PMTUD for a higher DNS responsiveness.

[11] F. Baker. 1995. *RFC 1812: Requirements for IP Version 4 Routers.* Technical Report. https://tools.ietf.org/html/rfc1812

[12] R. Braden. 1989. *RFC 1122: Requirements for Internet Hosts – Communication Layers.* Technical Report. https://tools.ietf.org/html/rfc1122

[13] Markus Brandt, Tianxiang Dai, Amit Klein, Haya Shulman, and Michael Waidner. 2018. Domain validation++ for MitM-resilient PKI. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2060–2076.

[14] 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 *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 209–225.

[15] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. 2018. Off-Path TCP Exploits of the Challenge ACK Global Rate Limit. *IEEE/ACM Transactions on Networking (TON)*.

[16] Yue Cao, Zhongjie Wang, Zhiyun Qian, Chengyu Song, Srikanth V. Krishnamurthy, and Paul Yu. 2019. Principled Unearthing of TCP Side Channel Vulnerabilities. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 211–224. https://doi.org/10.1145/3319535.3354250

[17] Weiteng Chen and Zhiyun Qian. 2018. Off-Path TCP Exploit: How Wireless Routers Can Jeopardize Your Secrets. In *27th USENIX Security Symposium (USENIX Security 18)*. 1581–1598.

[18] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2017. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1307–1322.

[19] European Commision. 2014. Quality of Broadband Services in the EU. http://ec.europa.eu/newsroom/dae/document.cfm?action=display&doc_id=10816.

[20] Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. 2021. From IP to Transport and beyond: Cross-Layer Attacks against Applications. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 836–849. https://doi.org/10.1145/3452296.3472933

[21] Jacob Davis and Casey Deccio. 2021. A Peek into the DNS Cookie Jar. In *Passive and Active Measurement*, Oliver Hohlfeld, Andra Lutu, and Dave Levin (Eds.). Springer International Publishing, Cham, 302–316.

[22] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A Search Engine Backed by Internet-Wide Scanning. In *22nd ACM Conference on Computer and Communications Security*.

[23] Roya Ensafi, Jong Chun Park, Deepak Kapur, and Jedidiah R. Crandall. 2010. Idle Port Scanning and Non-Interference Analysis of Network Protocol Stacks Using Model Checking. In *Proceedings of the 19th USENIX Conference on Security* (Washington, DC) *(USENIX Security'10)*. USENIX Association, USA, 17.

[24] FCC. 2018. Eighth Measuring Broadband America Fixed Broadband Report. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-eighth-report.

[25] Xuewei Feng, Chuanpu Fu, Qi Li, Kun Sun, and Ke Xu. 2020. Off-Path TCP Exploits of the Mixed IPID Assignment. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1323–1335. https://doi.org/10.1145/3372297.3417884

[26] Linux Foundation. 2021. net/ipv4/icmp/c. https://elixir.bootlin.com/linux/v5.11/source/net/ipv4/icmp.c#L1218.

[27] F. Gont. 2010. *RFC 5927: ICMP Attacks against TCP.* Technical Report. https://tools.ietf.org/html/rfc5927

[28] Google. 2021. IPv6 Adoption Statistics. https://www.google.com/intl/en/ipv6/statistics.html.

[29] Hang Guo and John Heidemann. 2018. Detecting ICMP rate limiting in the Internet. In *International Conference on Passive and Active Network Measurement*. Springer, 3–17.

[30] Matthias Göhring, Haya Shulman, and Michael Waidner. 2018. Path MTU Discovery Considered Harmful. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 866–874. https://doi.org/10.1109/ICDCS.2018.00088

[31] Brendon Harris and Ray Hunt. 1999. TCP/IP security threats and attack methods. *Computer communications* 22, 10 (1999), 885–897.

[32] Amir Herzberg and Haya Shulman. 2012. Security of Patched DNS. In *ESORICS 2012*, Sara Foresti, Moti Yung, and Fabio Martinelli (Eds.).

[33] Amir Herzberg and Haya Shulman. 2013. Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org. In *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 224–232.

[34] J. Mogul J. McCann, S. Deering and Ed. R. Hinden. 2017. *RFC 8201: Path MTU Discovery for IP version 6.* Technical Report. https://tools.ietf.org/html/rfc8201

[35] Philipp Jeitner and Haya Shulman. 2021. Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association.

[36] Dan Kaminsky. 2008. Black ops 2008: It's the end of the cache as we know it. *Black Hat USA* (2008).

[37] Amit Klein. 2020. Cross Layer Attacks and How to Use Them (for DNS Cache Poisoning, Device Tracking and More). arXiv:2012.07432 [cs.CR]

[38] Amit Klein, Haya Shulman, and Michael Waidner. 2017. Internet-wide study of DNS cache injections. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[39] Jeffrey Knockel and Jedidiah R. Crandall. 2014. Counting Packets Sent Between Arbitrary Internet Hosts. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*. USENIX Association, San Diego, CA.

[40] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. 2015. Going wild: Large-scale classification of open DNS resolvers. In *Proceedings of the 2015 Internet Measurement Conference*. ACM, 355–368.

[41] R. G. Fairhurst L. Eggert and G. Shepherd. 2017. *RFC 8085: UDP Usage Guidelines.* Technical Report. https://tools.ietf.org/html/rfc8085

[42] M. Lepinski and S. Kent. 2012. *RFC 6480: An Infrastructure to Support Secure Internet Routing.* Technical Report. https://tools.ietf.org/html/rfc6480

[43] lkm. 2007. Blind TCP/IP Hijacking is Still Alive. http://phrack.org/issues/64/13.html.

[44] Matthew Luckie, Robert Beverly, Ryan Koga, Ken Keys, Joshua A. Kroll, and k claffy. 2019. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 465–480. https://doi.org/10.1145/3319535.3354232

[45] Keyu Man, Zhiyun Qian, Zhongjie Wang, Xiaofeng Zheng, Youjun Huang, and Haixin Duan. 2020. DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1337–1350. https://doi.org/10.1145/3372297.3417280

[46] P. Mockapetris. 1987. *RFC 1035: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION.* Technical Report. https://tools.ietf.org/html/rfc1035

[47] J. Mogul and S. Deering. 1990. *RFC 1191: Path MTU Discovery.* Technical Report. https://tools.ietf.org/html/rfc1191

[48] J. Postel. 1981. *RFC 792: INTERNET CONTROL MESSAGE PROTOCOL.* Technical Report. https://tools.ietf.org/html/rfc792

[49] Zhiyun Qian and Z Morley Mao. 2012. Off-path TCP sequence number inference attack-how firewall middleboxes reduce security. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 347–361.

[50] Zhiyun Qian, Z. Morley Mao, Yinglian Xie, and Fang Yu. 2010. Investigation of Triangular Spamming: A Stealthy and Efficient Spamming Technique. In *2010 IEEE Symposium on Security and Privacy*. 207–222. https://doi.org/10.1109/SP.2010.42

[51] Alan Quach, Zhongjie Wang, and Zhiyun Qian. 2017. Investigation of the 2016 Linux TCP Stack Vulnerability at Scale. *SIGMETRICS Perform. Eval. Rev.* (2017).

[52] M. Larson D. Massey R. Arends, R. Austein and S. Rose. 2005. *RFC 4035: Protocol Modifications for the DNS Security Extensions.* Technical Report. https://tools.ietf.org/html/rfc4035

[53] Riccardo Ravaioli, Guillaume Urvoy-Keller, and Chadi Barakat. 2015. Characterizing ICMP rate limitation on routers. In *2015 IEEE International Conference on Communications (ICC)*. 6043–6049. https://doi.org/10.1109/ICC.2015.7249285
[54] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2014. DNS Record Injectino Vulnerabilities in Home Routers. http://www.icir.org/mallman/talks/schomp-dns-security-nanog61.pdf.
[55] Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota. 2006. Birthday paradox for multi-collisions. In *International Conference on Information Security and Cryptology*. Springer, 29–40.
[56] W. Simpson T. Narten, E. Nordmark and H. Soliman. 2007. *RFC 4861: Neighbor Discovery for IP version 6 (IPv6)*. Technical Report. https://tools.ietf.org/html/rfc4861
[57] P. Vixie and D. Dagon. 2008. *Use of Bit 0x20 in DNS Labels to Improve Transaction Identity*. Technical Report. https://tools.ietf.org/html/draft-vixie-dnsext-dns0x20-00
[58] Neil Wright. 2012. DNS in Computer Forensics. *Journal of Digital Forensics, Security and Law* 7 (2012), 11–42. https://doi.org/10.15394/jdfsl.2012.1117
[59] Xiaofeng Zheng, Chaoyi Lu, Jian Peng, Qiushi Yang, Dongjie Zhou, Baojun Liu, Keyu Man, Shuang Hao, Haixin Duan, and Zhiyun Qian. 2020. Poison Over Troubled Forwarders: A Cache Poisoning Attack Targeting DNS Forwarding Devices. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 577–593.
[60] Pengxiong Zhu, Keyu Man, Zhongjie Wang, Zhiyun Qian, Roya Ensafi, J. Alex Halderman, and Haixin Duan. 2020. Characterizing Transnational Internet Performance and the Great Bottleneck of China. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 1, Article 13 (May 2020), 23 pages. https://doi.org/10.1145/3379479

A ICMP REDIRECT ATTACKS

We performed the following small-scale experiments to measure the four conditions (outlined in §5.1) for redirect-based attacks.

University network experiment. We verified the conditions of successful attacks against resolvers in a university network. Since we are able to craft ICMP redirect messages with the spoofed IPs inside the university network, we target 9 resolvers by redirecting the packets destined to our test machine to an IP that is considered nearby of the resolver. The result shows 3 out of 9 resolvers are vulnerable, (*i.e.,* meeting $C1-C4$). Most resolvers are not vulnerable because they do not accept ICMP redirect packets at all, which breaks $C3$. In practice, the acceptance of redirects can be configured via `sysctl` on Linux and the default value varies on different Linux distributions. Two resolvers are not vulnerable because they run FreeBSD which blindly accepts redirects and invalidates $C1$.

Delivery of ICMP redirect on the Internet. Since ICMP redirects are potentially dangerous [31], one concern is that such messages may be dropped on the Internet and only work in local networks. We therefore performed a small-scale experiment by having 8 vantage points (corresponding to 8 ASes) distributed across the world (*i.e.,* in five continents) to send ICMP redirect messages to each other. Specifically, our vantage points reside in AWS (multiple continents), Google Cloud Platform, China educational network, US university campus network, and China residential network. The result shows ICMP redirects can successfully traverse the Internet in all pairs of experiments.

B ICMP RATE LIMIT

ICMP traffic is generally considered as control-plane traffic and it has been proposed that the source should rate-limit the generation of such packets [11, 53]. If such traffic is rate limited not only at the source but also during transit (for ICMP PING [29]), the port scan speed can be significantly hampered. As a result, we conduct a small-scale experiment using the same setup as mentioned in Appendix A and send ICMP frag needed or redirect messages to each other at a rate of 10k pps. We find that none except one Chinese residential host showed packet losses, which confirms rate-limiting in the transit network is not a popular policy. Even for the Chinese residential host, we find that the losses seem to be affected by the nationwide slowdown effect as reported recently [60]. We had the suspicion because UDP packets destined to the same residential host experienced similar losses also.

C RESETTING THE EXCEPTION CACHE STATE

Since the search of the ephemeral port we conduct requires multiple rounds of probes, the attacker has to reset the cache state after getting a positive response (*i.e.,* a probing packet in a batch hitting the correct open ephemeral port or the false positive caused by noises). Generally speaking, this can be done similarly to the cache planting phase in the private-facing port scan where the attacker finds 5 hash-collision IPs (note these can be done via IP spoofing instead of direct ownership) to evict the cache entry containing his primary scanning IP. Note that an easier method exists specifically for the public-facing port scans using ICMP frag needed messages. This is because when a correct port is hit, the resolver will reduce the MTU for the attacker's host to that specified in the ICMP frag needed message. The attacker can continue to lower the MTU in future rounds of probes. Each time the MTU is decreased, an attacker can simply send a PING verification packet to infer if the new MTU is now in effect. Note that it is not possible to raise the MTU using this method according to the specification [34, 47]. As a result, if the minimum MTU is reached, the attacker would have to fall back to the general method (*i.e.,* replanting the cache).

D SYSTEMATIC MITIGATIONS ON NETWORK SIDE CHANNEL ATTACKS

Both this work and SADDNS [45] showed a significant threat against DNS security. Since they arise from the kernel network stack, other protocols (e.g., QUIC or RTP) could suffer from the side-channel-based port scan as well.

To mitigate unknown side channels, we suggest a careful design on any use of shared resources and minimize sharing unless it is absolutely necessary. To verify if a specific sharing is safe, we need to model the side channel threat properly and can apply automated reasoning techniques, e.g., static analysis and model checking [16] to verify whether any leakage of information can occur.

Beyond the above analysis, which can be tricky to do correctly, a universal best practice is to randomize the limit or use of the shared resource. This can substantially increase the difficulty of a side channel attack even if there is a vulnerability. Indeed, this is exactly what the patches do against prior TCP side channels [15] and SADDNS [45].