DNS cache poisoning attacks return due to Linux weakness

Researchers from Tsinghua University and the University of California have identified a new method that can be used to conduct DNS cache poisoning attacks.

The new discovery revives a 2008 bug that had once been thought to have resolved for good.

What is DNS spoofing or cache poisoning?

Domain Name System (DNS) can be best understood as a phonebook for the internet.

Much like, when you want to call your friend *Alex,* you'd need to look up their phone number through a system called the phonebook.

Likewise, when you browse to a domain, your web browser attempts to identify its IP address by looking it up through an internet directory system called DNS.

The actual process happens in a series of steps and isn't always so straightforward.

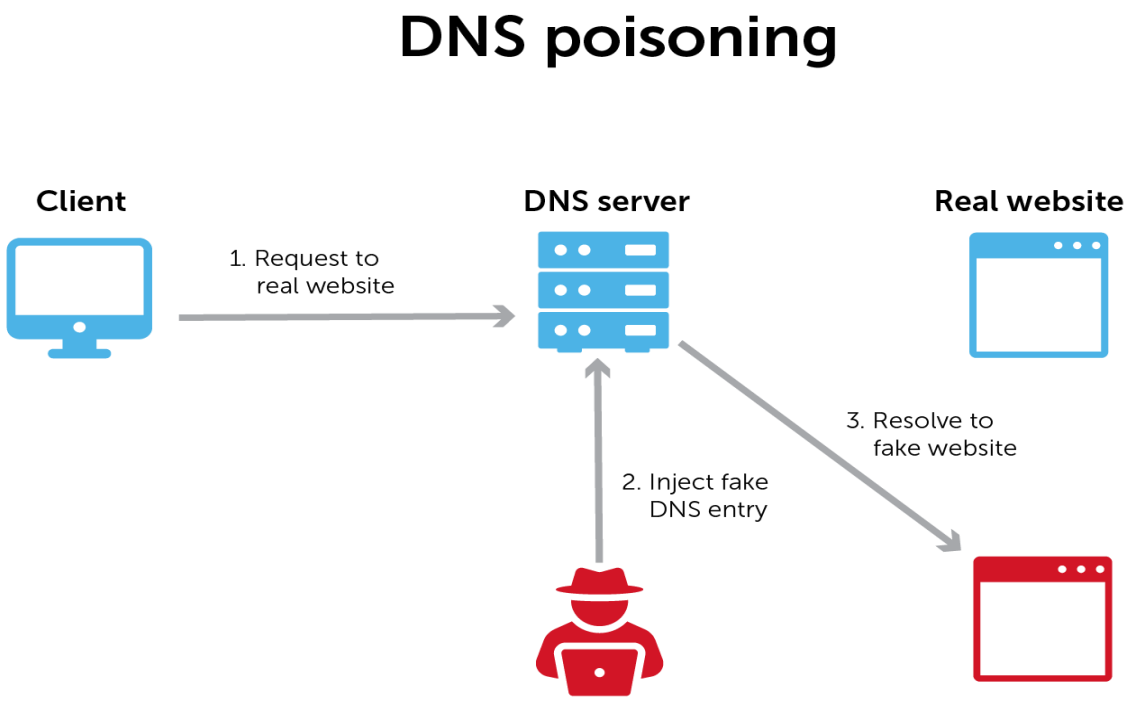
For example, had you or someone on your network previously visited *bleepingcomputer.com*, our IP address would get cached either somewhere on your computer or on intermediary servers.

This means the next time you visit *bleepingcomputer.com,* another DNS lookup won't be necessary. Your computer or web browser would already know where to locate us.

DNS cache poisoning attacks refer to polluting this very cache existing on intermediary servers.

Imagine if a DNS cache your computer (the client) had been relying on to lookup *bleepingcomputer.com*'s IP, returned to you an incorrect IP address instead of ours?

Attackers could wreak havoc on the internet should they be able to poison DNS caches.



DNS Spoofing or cache poisoning attack illustrated

Such a bug was discovered by security researcher Dan Kaminsky in 2008.

When a device lookups up a domain name's IP address using DNS, it includes a unique 'transaction ID' number in the request to the DNS server.

When the server responds to the device with an answer (e.g. *bleepingcomputer.com* can be found at 104.20.60.209), the device will *only accept* the response as valid if it also includes that original transaction ID.

The reason for this is simple: to prevent a rogue DNS server from flooding your device with malicious, invalid DNS entries.

Should these checks not have been in place, a rogue DNS server could easily give the user a spoofed IP address, and when connecting to a website, the user would now get redirected to the attacker's server instead of the legitimate one—thanks to the spoofed DNS response.

However, Kaminsky had discovered there were only 65,536 possible transaction IDs.

An attacker could therefore send multiple fake DNS responses with IDs from 0 to 65,535, and at the same time *prevent* the first response from getting cached.

To prevent caching of the first DNS response, the attacker would send responses with slight variations of a domain, such as every response containing a different subdomain: *1.bleepingcomputer.com, 2.bleepingcomputer.com*, and so on.

Eventually, the attacker would be able to guess the correct transaction ID of a DNS request and simultaneously provide their malicious server IP via DNS response.

Next time, the user goes to *bleepingcomputer.com,* the domain would resolve to the attacker's server, should such an attack succeed.

How has DNS cache poisoning returned?

To prevent DNS cache poisoning attacks source port randomization had been implemented.

This means, even as an attacker even if I could eventually guess one of the 65,536 transaction IDs specified by your device in a DNS request, I wouldn't know *where* to send the DNS response—because now your device making a DNS lookup is doing so from a randomized port (which in theory has 65,536 values too) instead of port 53.

This solution had made it virtually impossible for DNS cache poisoning attacks to be carried out via Kaminsky's discovered method, given the billions of possibilities.

But researchers at Tsinghua University and the University of California published a method which takes advantage of a side-channel attack to deduce the source port number of the DNS client.

With the source port being out of the bag, it becomes once again possible to conduct Kaminsky's DNS cache poisoning attacks by guessing the transaction IDs as described above.

Guessing the source port becomes possible because of how the Linux kernel handles ICMP requests (think *ping* or *tracert*).

To save bandwidth, the rate limiter built into Linux defaults the number of incoming requests to 1,000 per second and uses a counter to keep track of these requests.

For every request received at a closed port on a Linux-based server, the counter would decrement by 1 *and* the server would respond with "unreachable."

Meaning, in a second, if you sent 1,000 packets to different random ports on a server, all of which were closed, the server would cut you off for that second.

But, this would also tell you that all of your 1,000 guesses for which port could be open were incorrect.

Interestingly, the counter does not decrement for every request that is received at a valid, open port. And, further, "unreachable" would obviously not be sent by the server.

This means, **every second**, an attacker could flood a DNS resolver with 1,000 spoofed packets destined for random ports.

In this manner, in a matter of seconds, the attacker will be able to deduce what all ports are open on the DNS resolver that they are trying to poison.

With the knowledge of the right port, they can then re-exploit Kaminsky's bug to cause DNS poisoning attacks.

The 2020 DNS cache poisoning vulnerability impacting multiple DNS resolves (e.g. BIND, Unbound, dnsmasq) has been nicknamed **SAD DNS** (Side-channel AttackeD DNS) and assigned **CVE-2020-25705**.

So is there a solution to this?

Much like source port randomization had added some complexity for the attackers, the Linux kernel randomizing the rate limiter's maximum value as opposed to always using 1,000 could prove useful.

Such a fix would make it again hard for an attacker to deduce the correct port to target for DNS spoofing attacks.

David Maxwell, software security director at BlueCat offered a suggestion:

> "The change being introduced by Linux to automatically randomize the ratelimit value was included in the Linux kernel on Oct 16th, meaning it will be a while before most systems are running that. It's not in the release yet. In the meantime, you could use a small shell script to constantly vary the ICMP ratelimit. Not as clean as it being built into the kernel, but a viable

> workaround." So, we're working on making a sample script like that available to the community at the moment."

The fix made to the Linux kernel's *icmp.c* file now assigns a randomized value to the counter (called 'credit'), but it may be a while before Linux servers around the world catch up as Maxwell explains.

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

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

1 files changed, 5 insertions, 2 deletions

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

Fix made to ICMP ratelimiter of the Linux kernel

Other solutions proposed by the researchers include destroying the side-channel itself by, for example, disabling outgoing ICMP or randomizing the rate limit value (as Linux developers did), reducing the DNS query timeout to minimize the attack window, and using technologies like DNSSEC, DNS cookies, or 0x20 bit encoding to add secrets to DNS messages.

However, Maxwell does not recommend blocking ICMP traffic completely as the protocol has legitimate use-cases, such as in IPv6 fragmentation.

> "If a DNS server has ICMP blocked completely, zone transfers could fail, if there is a hop with a smaller MTU (blocking ICMP causes PMTUD black hole) between it and the other server," Maxwell told BleepingComputer.

The discovery of this side-channel attack puts extra pressure on the internet engineers to step up DNS security.

DNS is already a relatively insecure protocol designed to favor speed over performance and security.

Even building enhancements such as DNS-over-HTTPS (DoH) has not deterred attackers from abusing DNS for their malicious activities.

Update 13-Nov-2020: *Added, this vulnerability is being tracked as CVE-2020-25705. Added solutions proposed by the researchers.*

Additionally, according to the researchers, the vulnerability also impacts other operating systems shown below for which a patch has not yet been issued.

"In addition to Linux, we have verified that other major OS kernels are vulnerable as well, albeit with lower global rate limit — **200** in Windows and FreeBSD, and **250** in MacOS," state the researchers in their paper.

> *Linux 3.18-5.10*
> *Windows Server 2019 (version 1809) and newer (we did not test older versions)*
> *macOS 10.15 and newer (we did not test older versions)*
> *FreeBSD 12.1.0 and newer (we did not test older versions)*
>
> *The patch for Linux is integrated into 5.10 and backported to many stable versions. However, we don't know how and when Windows/macOS/FreeBSD will patch this vulnerability.*