Analysis of DNS DoS Attacks and Defenses with Focus on TuDoor attack

Mausam Piyush Vora
SUTD
PhD-ISTD
Singapore
1009891@mymail.sutd.edu.sg

Abhilasha Mohapatra
SUTD
MTD (Cybersecurity)
Singapore
1009708@mymail.sutd.edu.sg

Likitha Balaji
SUTD
MTD (Cybersecurity)
Singapore
1009639@mymail.sutd.edu.sg

***Abstract*—This report explores denial-of-service (DoS) attacks targeting the Domain Name System (DNS) by analyzing and replicating techniques from two key studies: the TuDoor attack and anomaly-based filtering of application-layer DNS DoS attacks. The primary focus is on the TuDoor attack, which exposes logic vulnerabilities in DNS response pre-processing that can be exploited to cause DNS cache poisoning, denial-of-service, and resource exhaustion with minimal traffic. We conducted an in-depth technical analysis of DNS DoS to understand how it exploits flaws in resolver state machines. We have replicated the anomaly-based filtering defense, which mitigates application-layer DNS DoS attacks by identifying abnormal query volumes based on traffic history and filtering suspicious sources. Using Python implementations on a MacOS system, we evaluated the defense's performance across multiple attack datasets, generated allowlists, optimized parameters, and visualized filtering outcomes. Our findings show that while anomaly-based filtering effectively reduces high-rate DNS query floods, it is ineffective against low-volume, protocol-level attacks like TuDoor's DNSDoS. This contrast highlights the need for a multi-layered defense combining traffic anomaly detection with robust protocol validation to protect DNS infrastructure against diverse DoS threats. Through replication and analysis, this report demonstrates the importance of addressing both traffic-based and logic-based attack vectors in DNS security.**

***Index Terms*—DNS security, TuDoor attack, DNS denial-of-service, cache poisoning, anomaly-based filtering, application-layer DoS**

I. Introduction

The Domain Name System (DNS) is a fundamental part of the Internet, acting as the "phonebook" that translates human-readable domain names into machine-readable IP addresses. Every time a user accesses a website, sends an email, or uses an online service, a DNS query is made in the background to resolve the domain name to its corresponding IP address. Because of this main role, the availability, integrity, and security of DNS infrastructure are critical to make sure the Internet functions reliably.

However, DNS is also a common target for attackers. One of the most serious threats facing DNS is the Denial-of-Service (DoS) attack, which aims to make DNS services unavailable by overwhelming servers with malicious traffic or exploiting weaknesses in DNS implementations. Successful DoS attacks on DNS can disrupt access to websites, online services, and entire networks, with potentially widespread impact on users and businesses.

This report focuses on understanding and replicating the "TuDoor attack: Systematically Exploring and Exploiting Logic Vulnerabilities in DNS Response Pre-processing with Malformed Packets." The TuDoor attack demonstrates how an attacker can **exploit logic vulnerabilities in DNS response processing** to cause denial-of-service, using only a small number of carefully crafted packets. This makes it a powerful method for disabling DNS services.

In addition to replicating the TuDoor attack, this report also examines a defense mechanism presented in the paper "Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives." This defense proposes an upstream filtering system that detects and blocks abnormal DNS traffic patterns to protect authoritative DNS servers from application-layer DoS attacks. By exploring both the TuDoor attack and this anomaly-based defense, the report connects the offensive and defensive perspectives in the context of DNS security.

The main objectives of this report are:

1) To demonstrate a deep technical understanding of the TuDoor attack, including how it exploits vulnerabilities in DNS response pre-processing.
2) To replicate the DNS DoS attack described in the TuDoor paper and observe its effects in a controlled environment.
3) To explore and evaluate the anomaly-based filtering defense approach, including an attempt to replicate its implementation.
4) To analyze how the offensive and defensive approach relate to each other, and to discuss their implications for securing DNS infrastructure.

By investigating both the attack and the defense, this report aims to provide insights into the challenges of protecting DNS systems against denial-of-service attacks that target both traffic patterns and underlying protocol logic.

II. Background and Related Work

A. Overview of DNS Infrastructure

Domain Name System (DNS), as explained in section I, is a hierarchical and distributed naming system that translates

human-readable domain names into machine-readable IP addresses. This translation is very important for locating and accessing resources on the Internet [1].

The DNS resolution process involves three primary components:

1) **Stub Resolver**: Typically part of a user's operating system, it initiates DNS queries on behalf of applications and forwards them to a recursive resolver.
2) **Recursive Resolver**: Receives queries from stub resolvers and performs the necessary lookups by querying other DNS servers. It caches responses to improve efficiency and reduce latency for repeated queries.
3) **Authoritative Name Server**: Holds the official records for domain names and responds to queries with their corresponding IP addresses.

The stub resolver sends a query to the recursive resolver. If the resolver already has a cached answer, it responds immediately. Otherwise, it queries the root server, which directs it to a top-level domain (TLD) server (example - .com), and from there to the authoritative name server. The authoritative server provides the answer, which the resolver caches and returns to the stub resolver. [1] Caching plays a key role in reducing load and latency but also introduces vulnerabilities such as cache poisoning. Query processing at each stage represents a potential attack surface that attackers can target to disrupt DNS availability or integrity.

B. DoS Attacks Targeting DNS

Denial-of-Service (DoS) attacks aim to disrupt normal services by overwhelming servers with traffic or exploiting weaknesses. In DNS, these attacks are typically classified as:

1) **Volumetric Attacks**: Flooding DNS servers with high volumes of traffic to exhaust bandwidth or processing capacity. A common example is the DNS amplification attack, where attackers exploit open resolvers to send large responses to a victim, magnifying attack traffic [2].
2) **Application-Layer Attacks**: Target specific DNS functionalities by sending semantically valid queries at high rates, exhausting computational resources rather than network capacity [3].

DNS is inherently vulnerable because it is designed as a publicly accessible, open protocol with distributed trust across many servers [4].

Common attack types include:

1) **DNS Amplification**: Using small queries to generate large responses directed at a victim [2].
2) **NXDOMAIN Flooding**: Sending large numbers of queries for non-existent domains, forcing resolvers to process and cache negative results [5].
3) **DNS Tunneling**: Embedding data within DNS queries/responses to evade security controls [6].
4) **Cache Poisoning**: Injecting false DNS records into resolver caches, redirecting users to malicious sites [1].

These attack vectors show why DNS needs both traffic-volume defenses and protocol-level integrity protections.

C. Review of Key Papers

*1) **TuDoor Attack**:* TuDoor attack [1] identifies logic vulnerabilities in DNS response pre-processing. Their analysis showed that many DNS software implementations process malformed DNS response packets incorrectly, leading to exploitable state machine flaws. The paper describes three novel attack vectors:

1) **DNSPoisoning**: Exploiting side channels to inject false records without needing to guess transaction IDs or source ports.
2) **DNSDoS**: Sending malformed responses that prematurely terminate legitimate DNS queries, causing denial-of-service.
3) **DNSConsuming**: Forcing resolvers into resource-consuming loops that drain CPU and memory.

These attacks are dangerous because they require only a few crafted packets to cause significant disruption, bypassing traditional volume-based detection.

*2) **Anomaly-Based Filtering of Application-Layer DDoS Against DNS Authoritatives**:* Bushart & Rossow (2023) proposed an anomaly-based filtering defense to protect authoritative DNS servers from application-layer DoS attacks. [7] Their method uses two main strategies:

1) **Allowlisting**: Learning and maintaining a list of legitimate recursive resolvers based on historical query behavior.
2) **Low-Pass Filtering**: Applying rate limits to traffic from unknown or non-allowlisted sources.

Their approach allows upstream providers to filter malicious traffic patterns while preserving access for well-behaved clients. This defense focuses on traffic behavior rather than packet content, offering a scalable solution for application-layer DoS mitigation.

D. Contrast Between the Two Approaches

The TuDoor attack and anomaly-based filtering defense target different layers of DNS security:

- The TuDoor attack exploits **protocol-level logic flaws** in DNS response handling at the resolver, making it effective even with low traffic volumes.
- The anomaly-based filtering approach detects **abnormal traffic patterns** at the network level, effective against high-volume application-layer DoS but potentially blind to stealthy protocol exploitation.

Understanding both offensive and defensive mechanisms highlights the need for multi-layered DNS protection combining protocol validation with anomaly-based traffic filtering.

III. TuDoor Attack

A. Threat Model

The TuDoor attack [1] requires an attacker who is able to send DNS queries to a target resolver and who can figure out the external IP address that the resolver uses to send its queries. This can be done, for example, by getting the resolver

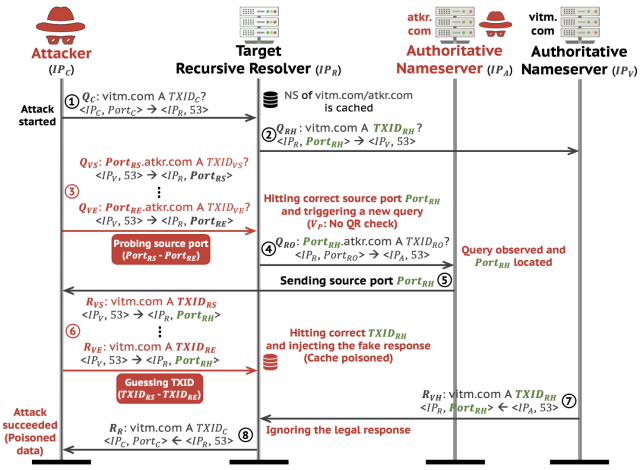

Fig. 1: Steps for DNS Cache Poisoning Attack

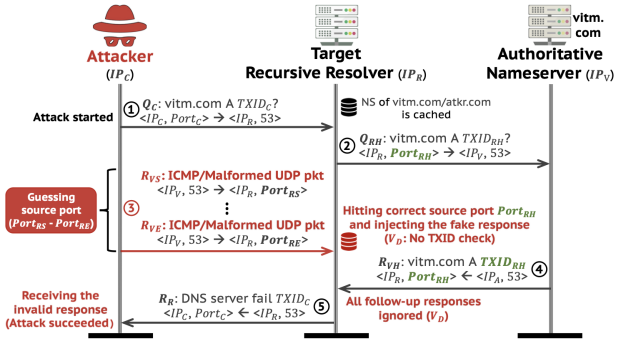

Fig. 2: Steps for DNS Dos Attack

to query a domain the attacker controls, and watching which IP address shows up on their own authoritative server.

Once the attacker knows the resolver's IP, they can use **IP spoofing** to send fake DNS responses that appear to come from legitimate authoritative nameservers. This ability to send spoofed packets is key for two of the TuDoor attack variants: DNS cache poisoning (DNSPOISONING) and DNS denial-of-service (DNSDOS). For the third variant, DNS resource consumption (DNSCONSUMING), the attacker needs to control an authoritative nameserver or be on-path to intercept responses.

What makes TuDoor different from earlier DNS attacks is that it doesn't depend on overwhelming traffic or brute-forcing both transaction IDs and ports. Instead, it focuses on **logic vulnerabilities** in how DNS software processes responses. By sending malformed packets that trigger specific weaknesses in the DNS response handling state machine, the attacker can poison caches, cause failures, or make resolvers waste resources with just a few carefully crafted packets.

B. DNS Cache Poisoning Attack (DNSPOISONING)

The DNS cache poisoning attack in TuDoor, as shown in figure 1 , uses a clever side channel to discover the source port that a resolver is using for a DNS query. This is critical because traditional DNS cache poisoning attacks require the attacker to guess both the transaction ID (TXID) and the source port to inject a forged response successfully. As it discovers the source port in advance, TuDoor reduces the guessing effort from 32 bits (16 bits for port, 16 for TXID) to just 16 bits (TXID only). The attack begins when the attacker initiates a query from the resolver for the victim domain (Step 1). During this, the attacker sends a series of **spoofed DNS queries** to the resolver, each targeting a different potential source port and embedding the port number inside the query name (e.g., 12345.atkr.com). This step is shown as **probing source ports** (Steps 2 - 3).

When one of these spoofed queries accidentally hits the correct source port being used by the resolver's pending query, the resolver **mistakenly forwards that query to the attacker-controlled authoritative nameserver** (Step 4). This happens because of the vulnerable pre-processing behavior (no QR flag check). As soon as the attacker sees which query name arrives at their authoritative server, they immediately know which port the resolver used ("query observed and port located" in the diagram).

Now that the attacker has the correct port, they can begin flooding the resolver with **spoofed DNS responses**, each guessing a different TXID (Step 5). If any of these forged responses matches the actual TXID used in the resolver's outstanding query, it gets accepted and cached by the resolver (Step 6), even before the real response arrives from the legitimate authoritative server.

Once the resolver caches this forged response, all future queries for the victim domain return the attacker's malicious answer (Step 8). Any subsequent legitimate responses arriving from the real authoritative server are ignored because the query is already considered resolved.

So, the attacker has poisoned the cache by racing a forged response to the resolver, helped by first leaking the source port. This attack is extremely fast and stealthy. It requires no brute-force traffic across the entire port space and needs only a single spoofed response to succeed once the TXID is guessed.

The key innovation here is the use of an active side channel (leveraging the resolver's faulty handling of QR=0 queries on the response socket) to reveal the port, drastically simplifying the poisoning effort. Microsoft DNS, for example, binds thousands of ports simultaneously, making passive port guessing ineffective, but TuDoor evades this with the probing method shown in the diagram.

C. DNS Denial-of-Service Attack (DNSDOS)

The DNSDOS attack, as shown in figure 2, exploits a different flaw in the resolver's pre-processing logic. Instead of trying to poison the cache, this attack aims to make the resolver prematurely **abort its resolution process** and treat the authoritative nameserver as unreachable.

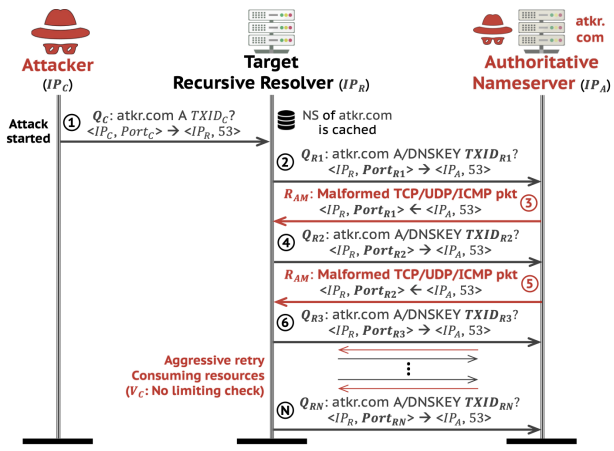

Fig. 3: Steps for DNS Resource Consumption Attack

The attack starts with the attacker forcing the resolver to query the victim domain (Step 1). As the resolver forwards this query to the authoritative server, the attacker sends **spoofed malformed packets** targeting different source ports of the resolver (Steps 2 - 3). These malformed packets can be empty UDP payloads, packets with invalid DNS headers, or even ICMP error messages. The attacker is mainly guessing ports in the same way as in DNSPOISONING but instead of valid queries, they send broken responses.

Once a malformed packet hits the correct source port (Step 4), the vulnerable resolver processes it and triggers a failure inside its state machine. The resolver **closes the socket, terminates the query, and sends a SERVFAIL error back to the client,** assuming that the upstream nameserver is broken or unreachable.

Many DNS resolvers also apply **negative caching** to SERVFAIL responses. This means that for some time, the resolver will refuse to retry that query and will return SERVFAIL to all clients asking for that domain. This amplifies the denial-of-service impact, where a single malformed packet sent to the correct port can block access to a domain for all users relying on that resolver, at least until the negative cache expires.

In the diagram, after hitting the correct source port with a malformed packet, all follow-up legitimate responses from the authoritative nameserver are **ignored** (Step 4). The client ultimately receives a SERVFAIL error (Step 5), completing the attack.

This attack doesn't require flooding traffic or high volumes of malformed packets, it just requires a few carefully timed spoofed packets targeted at the resolver's port range. Abusing the pre-processing error, the attacker can silently disable resolution for a domain or even entire zones (if targeting NS records), with almost no detectable anomaly in traffic patterns.

D. DNS Resource Consumption Attack (DNSCONSUMING)

The DNSCONSUMING attack, as shown in figure 3, takes advantage of a flaw where the resolver fails to enforce retry limits when handling malformed responses. In normal DNS behavior, a resolver retries a query a limited number of times if it doesn't get a valid response, and then gives up. However, in some DNS implementations, malformed responses don't count toward this retry limit because they fail parsing checks before incrementing retry counters.

In this attack, the adversary controls an authoritative nameserver for a malicious domain (e.g., atkr.com). The resolver sends an initial query for this domain (Step 1), expecting a response. The attacker responds with a malformed packet(Step 3), which fails validation but doesn't increment the retry count.

As a result, the resolver issues a new query (Step 4). Again, the attacker replies with a malformed response (Step 5). This loop continues indefinitely (Step 6 and beyond), with the resolver repeatedly sending new queries and getting malformed responses, never reaching the configured retry limit because the counter isn't incremented.

Each query and socket allocation consumes CPU cycles, memory, and networking resources inside the resolver. Over time, the resolver's query handling queue fills up, its sockets exhaust, and CPU usage increases. Eventually, legitimate queries for other domains are delayed or dropped because the resolver is stuck retrying the attacker's domain endlessly.

The diagram shows this continuous query-and-malformed-response loop. Every query sent by the resolver gets a malformed reply, causing an "aggressive retry" cycle that consumes resources without ever progressing to completion.

The fact that makes this attack dangerous is that it doesn't require spoofing or packet injection from off-path. It only requires control over the authoritative server. It works even if the network enforces IP spoofing protections. Also, by using large malformed responses via fragmentation or multiple malicious domains, the attacker can amplify resource exhaustion, forcing the resolver into a degraded or non-functional state.

E. Impact of TuDoor Attacks

Together, these three attacks show how subtle logic flaws in DNS software can have serious consequences. TuDoor attacks are so powerful as they don't need to flood the resolver with traffic or rely on obvious brute-force guessing. Instead, they use carefully designed malformed packets that trigger bugs in how resolvers handle responses before they even check whether the data is valid.

The TuDoor paper found that 24 different DNS implementations were vulnerable to at least one of these attacks, including popular software like BIND (which we try to replicate as shown in section 4).

The real danger of TuDoor is that it bypasses common defenses. Firewalls and anomaly detection systems that look for query floods or unusual traffic patterns will not catch these attacks because they use normal looking traffic at low volumes. The attacks exploit how the resolver interprets packets but not how many packets it gets.

In short, TuDoor shows that defending DNS resolvers is not just about blocking bad traffic, but also about making sure that the resolver's internal logic is sound and robust against unexpected or malformed inputs.

IV. Replication of DNS DoS Attack and Defense

A. Replication of TuDoor DNSDOS Attack

This section presents the complete step-by-step replication of the TuDoor DNSDoS (Denial-of-Service) attack, originally described by Xing et al. The simulation involves setting up a BIND DNS resolver on the victim VM, discovering its UDP source port via ICMP side channels, and then launching a brute-force spoofing attack to inject invalid responses, ultimately leading to a denial-of-service condition. Screenshots and command output are used to validate each phase.

Environment Setup and Verification: 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 DNSDoS simulation.

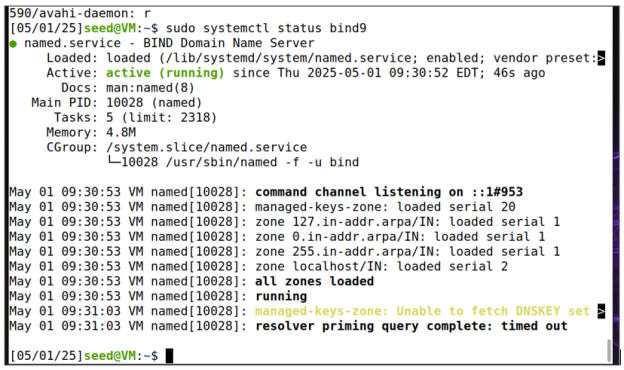

Fig. 4: BIND9 DNS service active and ready.

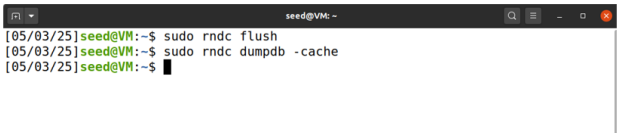

Fig. 5: Flush any existing DNS cache entries.

Verifying Internal Network Communication Between VMs

- **Command:** `ping 10.9.0.30` and `ping 10.9.0.20` These ping commands confirm bidirectional connectivity between the attacker and the victim over the internal VirtualBox network. Victim IP address is 10.9.0.20 and Attacker IP address is 10.9.0.30

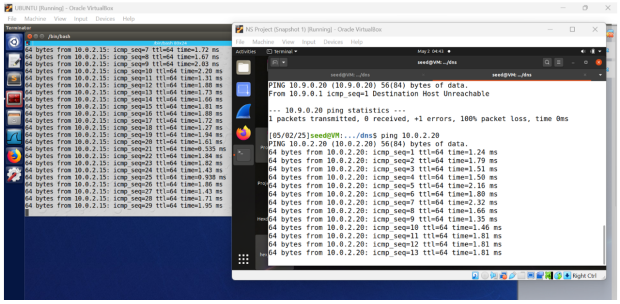

Fig. 6: Ping from attacker to victim successful Vice Versa.

Triggering the DNS Query (Initial Step):

- **Command:** `dig @10.9.0.20 vitm.com A` This command triggers the recursive DNS resolver to initiate a fresh external query (not cached), required for the spoofing to succeed. SERVFAIL is initially expected.

Fig. 7: dig command sent to victim DNS resolver.

Discovering the Source Port via Spoofed ICMP packets:

- **Script:** `spoof_icmp_port_discovery.py` Sends spoofed ICMP packets over a UDP port range from fake authoritative server to identify which port victim DNS resolver used.
- **Command:** `sudo tcpdump -n -i any port 53 or icmp` Captures outgoing queries to identify which port caused SERVFAIL, indicating the correct source port used.

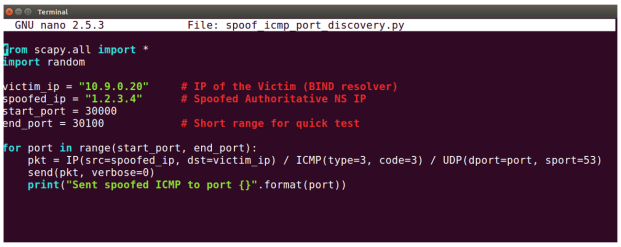

Fig. 8: Python ICMP spoof script

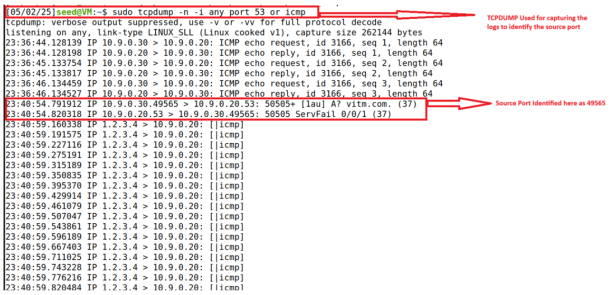

Fig. 9: tcpdump output reveals correct resolver source port.

Brute-Forcing TXID and Delivering Spoofed DNS Responses:

- **Script:** `spoof_dns_response.py`
  Sends spoofed DNS replies using fake authoritative IP and all possible TXIDs to match the victim's expected transaction ID.
- **tcpdump:** Captures a flood of spoofed replies that arrive at the victim using the guessed port and random TXIDs.

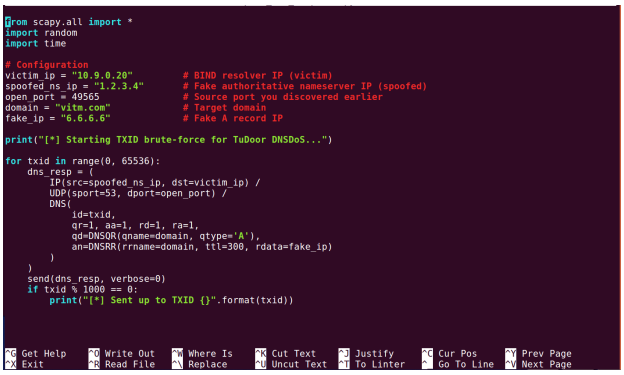

Fig. 10: Spoofed DNS response TXID brute-force script.

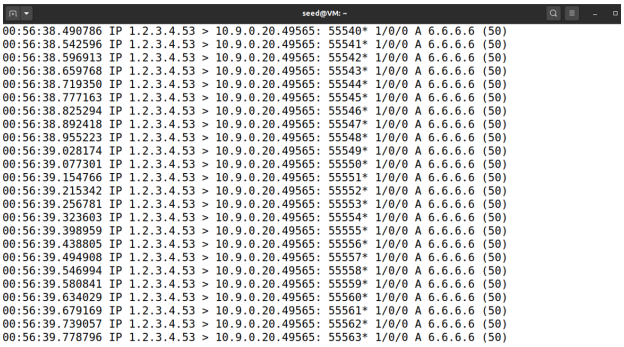

Fig. 11: Spoofed A records observed using tcpdump.

Validating the DoS Condition:

- **Command:** `dig @10.9.0.20 vitm.com A` (repeated) Even after flushing and retrying, queries return SERVFAIL due to spoofed or corrupted state.
- **Command:** `sudo grep vitm.com /var/cache/bind/named_dump.db`
  Confirms whether the spoofed entry was accepted or not cached at all.

Fig. 12: Multiple dig attempts returning SERVFAIL.

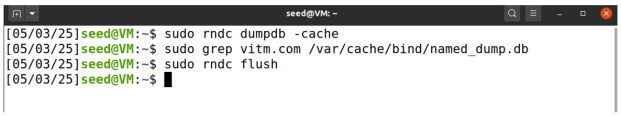

Fig. 13: DNS cache inspection via named_dump.db.

TABLE I: Step-by-step Mapping: TuDoor Paper vs Our Simulation

Step	TuDoor Paper Description	Our Simulation Equivalent
1	Q_C: Attacker sends DNS query to resolver	`dig @10.9.0.20 vitm.com A` to trigger recursive lookup
2	R_{VS}, R_{VE}: Sends spoofed malformed UDP packets over a port range	`spoof_icmp_port_discovery.py` sends ICMP with fake authoritative IP
3	R_B: Resolver responds with SERVFAIL, leaking the UDP source port	`tcpdump` output used to observe correct resolver port
4	R_F: Brute-force TXID via spoofed DNS replies	`spoof_dns_response.py` script floods TXID responses from spoofed NS
5	R_D: Resolver poisoned or enters DoS state, ignores legit replies	`SERVFAIL` confirmed and no record in cache indicates DoS success

Comparison Table: Our Simulation vs. Author's TuDoor DNSDoS Steps:

Analysis and Discussion: In this work, we successfully replicated the TuDoor DNSDoS attack using a virtualized environment comprising an attacker and a victim resolver running BIND9. The attack was conducted by first triggering a DNS resolution to force the resolver to send an outbound query. Using a custom packet spoofing script, we exploited BIND's pre-processing logic flaw to discover the resolver's UDP source port by observing anomalous SERVFAIL responses caused by spoofed probes. Once the port was identified, we executed a brute-force TXID spoofing attack that flooded the resolver with fake DNS responses crafted to appear as legitimate replies from an authoritative server. The persistent SERVFAIL replies and the absence of valid records in the resolver's cache confirmed that the resolver had entered a denial-of-service state. This simulation effectively validates the TuDoor paper's core claim: that improper validation during the pre-processing phase in BIND permits attackers to bypass source port randomization and inject spoofed DNS responses, resulting in a disruption of service. Through this replication, we empirically demonstrated that the vulnerability remains

exploitable in practical settings, thereby reinforcing the critical need for defensive mechanisms in recursive DNS resolvers.

B. Replication of DNS DoS from Anomaly-based Filtering Context

In this section, we present the replication of the anomaly-based filtering defense proposed in "Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives. [7]" The goal of this experiment was to simulate DNS query-based DoS attacks and evaluate how anomaly-based filtering can mitigate such attacks by learning traffic patterns and applying rate-based filtering.

All experiments were conducted locally on a MacOS laptop, running Python implementations derived from the original paper's approach. The replication workflow followed key steps: generating synthetic DNS traffic, building an allowlist of trusted sources, evaluating the defense using different attack datasets, and visualizing defense performance.

1) Replication Workflow and Setup: The environment was set up by installing necessary Python packages (numpy, pandas, matplotlib) and organizing the project files. A JSON configuration defined the evaluation parameters: test time windows, low-pass filter thresholds, and tolerance factors. The evaluation used pre-extracted datasets (mirai, sality, and open-resolver traffic) representing real-world DNS DoS traffic patterns, without generating new synthetic traffic which was taken by their github link [8].

An allowlist was generated using create_allowlist.py, and these datasets were used to test the anomaly-based filtering defense.

To evaluate defense performance, we ran the evaluate_defense.py script on three attack datasets representing different types of application-layer DNS DoS traffic:

- mirai-2022-all-ips.json (botnet-based query floods)
- sality-20220828.json (another botnet variant)
- open-resolver-equal-encoded-traffic.json (open resolver abuse)

Each evaluation printed a summary including total attack packets, packets passing the filter, filtering efficiency, and estimated false positive rate. Figure 14 shows the terminal output logs after running the evaluation for all three datasets.

The defense performance was summarized in a CSV file and visualized using visualize_results.py, which generated key plots and a summary table. The defense summary (Figure 17, 15) reported filtering efficiency of 46.75% for mirai, 25.90% for resolver abuse, and 0% for sality, with corresponding false positive rates of 1.75%, 2.80%, and 3.91%, as evident by the terminal output shown in Figure 18.

2) Parameter Optimization: A parameter optimization script was also implemented to explore how different configuration settings would impact defense performance. The parameter_optimization.py script systematically tested various combinations of test time windows (8 or 24 hours) and low-pass filter thresholds (128, 512, 2048, 8192 pph), while keeping the tolerance factor at 2.

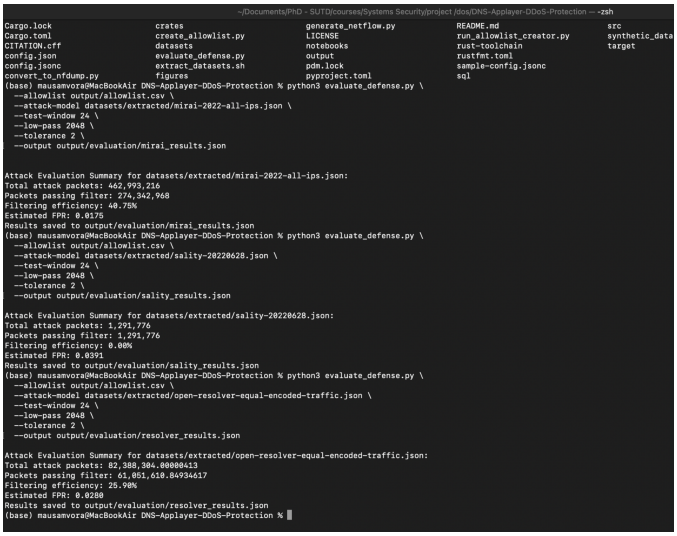

Fig. 14: Attack evaluation results after running evaluate_defense.py for mirai, sality, and open-resolver datasets.

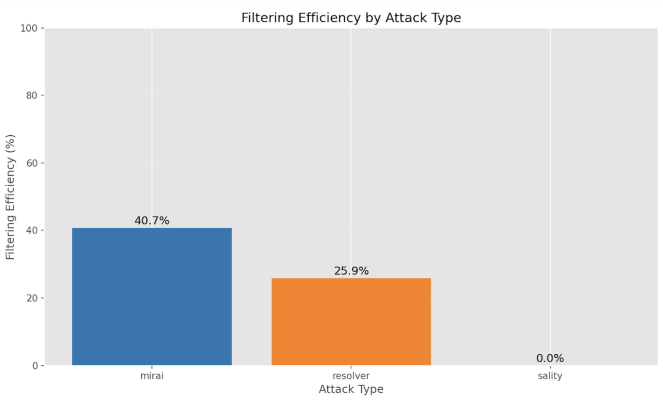

Fig. 15: Defense Performance Summary (Generated plot)

Each parameter set was evaluated independently for each attack dataset. The optimal configuration for each dataset was determined by selecting the setting that kept attack traffic below 100,000 packets per second while minimizing FPR.

The results were visualized in the scatter plot shown in Figure 16.

Each dot in Figure 16 represents a unique combination of test_window and low_pass filter for a given dataset. Points located towards the lower-left corner (low FPR and low filtered traffic) indicate better-performing configurations. From the plot, it is evident that configurations with test_window = 8 hours and low_pass = 512 packets per hour provided a favorable balance between low traffic and low FPR.

The script (parameter_optimization.py) printed the optimal configuration for each dataset at the end of its execution. In our replication, figure 19 shows sample optimization outputs for the mirai dataset across parameter settings.

Similarly, Figure 20 shows the output for another dataset, for which the optimal configuration was determined as:

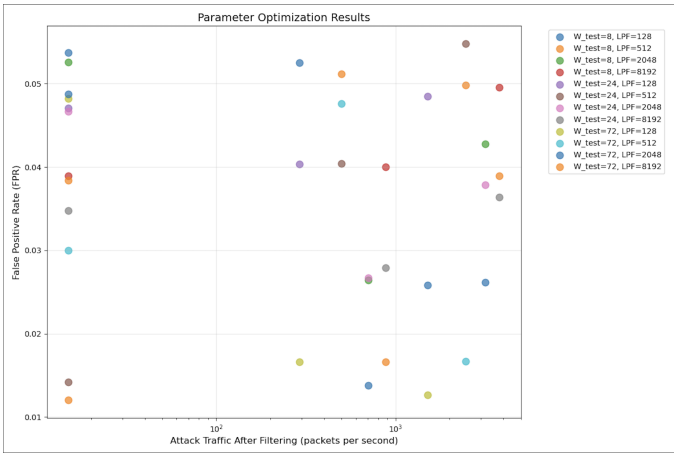

Fig. 16: Parameter Optimization Results showing False Positive Rate vs Attack Traffic After Filtering for different parameter settings.

- Test window: 8 hours
- Low-pass filter: 512 packets per hour
- Tolerance factor: 2

This configuration achieved a filtering efficiency of 71.87% and an estimated false positive rate of 1.21% for the open-resolver-equal-encoded-traffic dataset.

3) Observed Impact and Interpretation: The anomaly-based filtering defense was able to reduce incoming attack traffic significantly under the mirai and resolver attack datasets, keeping post-filter traffic well below operational thresholds (i.e., ¡100,000 pps). However, its filtering efficiency was lower for the resolver attack and negligible for sality, proving limitations when faced with certain traffic patterns.

The parameter optimization results, as shown in Figure 16, confirmed that tuning the test window and low-pass filter had a noticeable impact on the trade-off between filtering aggressiveness and false positives. The selected optimal configuration reduced attack traffic to a manageable level while maintaining an acceptable false positive rate.

4) Comparison to TuDoor DNSDOS: Unlike the anomaly-based filtering defense, which mitigates DoS attacks by filtering abnormal query volumes, TuDoor's DNSDOS attack targets the protocol behavior itself rather than query rates. TuDoor bypasses traffic-based defenses by injecting malformed packets that trigger resolver state machine failures, resulting in premature SERVFAIL errors.

This replication shows that anomaly-based filtering is effective against volumetric application-layer DoS attacks but not against protocol-level logic exploits like DNSDOS. Therefore, deploying anomaly-based filtering is necessary for controlling traffic floods but must be complemented with protocol-hardening measures to defend against more sophisticated attacks like TuDoor.

V. Key Proposed Improvements

Focusing on the most critical defenses, we recommend the following measures to harden DNS resolvers against TuDoor-style exploits and related floods.

A. Strict DNS Response Validation

- **Discard Malformed or Misflagged Packets:** Immediately drop any response with QR = 0, unexpected flag combinations, or header inconsistencies to eliminate the basic pre-processing vulnerability leveraged by both VDS and VCP variants [1].
- **Anomaly Logging:** Record and rate-limit sources that trigger validation failures, enabling rapid identification of probing or low-rate stealth attacks.

B. Universal DNSSEC Enforcement

- **Reject Unsigned Responses:** Enforce full DNSSEC validation on all outgoing and recursive queries so that any attempt at cache poisoning is negated at the protocol layer [1].
- **Fallback Controls:** If validation fails, serve only locally cached positive answers and short-TTL negative responses to avoid opening a denial window.

C. Adaptive Rate Limiting

- **Sliding-Window Profiling:** Build per-resolver query baselines over short intervals, then adjust caps dynamically—allowing genuine spikes while blocking outliers that match DDoS patterns [9].
- **Lightweight Classifiers:** Augment simple volume checks with small ML models that catch unusual header-field combinations or malformed sections, improving detection of stealthy attack traffic.

D. Real-Time Monitoring and Red-Team Validation

- **Live Telemetry Dashboard:** Instrument query rates, malformed-packet counts, and ICMP event metrics into a dashboard that raises alerts when deviations exceed normal baselines.
- **Quarterly Stress Tests:** Conduct controlled injection of malformed DNS packets and high-rate query floods against a staging resolver cluster to validate both protocol patches and upstream filters under realistic conditions.

Conclusion

This report focused on understanding, replicating, and analyzing denial-of-service (DoS) attacks targeting the DNS infrastructure, with the help of two papers: the TuDoor attack and the anomaly-based filtering defense against application-layer DNS DoS.

We conducted a study of the TuDoor attack, which introduced novel ways to exploit logic vulnerabilities in DNS resolver state machines. We also explained the three attack vectors DNS cache poisoning, DNSDOS, and DNS resource consumption, and we saw how an attacker can bypass traditional DNS security measures not by sending large volumes of traffic, but by exploiting weaknesses in how resolvers process malformed or misleading packets.

Alongside this, we replicated the anomaly-based filtering defense proposed for mitigating application-layer DNS DoS

attacks. Our replication tested the defense against multiple attack datasets (mirai, sality, open-resolver), showing how query rate filtering based on traffic history can reduce incoming attack traffic at authoritative servers. The defense successfully lowered traffic below operational thresholds for some attack types but was less effective against others.

As we compare these two approaches, we understood their fundamentally different focuses: anomaly-based filtering addresses traffic volume anomalies, while TuDoor's DNS DoS exploits protocol logic flaws independent of traffic rates. This contrast reveals an important takeaway that traffic filtering defenses are essential for mitigating large-scale query floods but are insufficient on their own to stop logic-based attacks like those demonstrated in TuDoor.

Our replication efforts validated the core findings of both papers and demonstrated the challenges of defending DNS systems against diverse attack strategies. Defending DNS availability requires a layered approach, combining traffic anomaly detection with protocol-level input validation and robust state machine design to prevent both volumetric and logic-based denial-of-service attacks.

This project deepened our technical understanding of DNS security threats by bridging attack replication, defense evaluation, and comparative analysis. We also understood that modern DNS security cannot rely solely on filtering traffic anomalies but must also address subtle protocol vulnerabilities that attackers can exploit with minimal traffic.

REFERENCES

[1] X. Li, W. Xu, B. Liu, M. Zhang, Z. Li, J. Zhang, D. Chang, X. Zheng, C. Wang, J. Chen, H. Duan, and Q. Li, "Tudoor attack: Systematically exploring and exploiting logic vulnerabilities in dns response pre-processing with malformed packets," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4459–4477.

[2] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "DNS amplification attack revisited," *Comput. Secur.*, vol. 39, pp. 475–485, Nov. 2013.

[3] N. Tripathi and N. Hubballi, "Application layer denial-of-service attacks and defense mechanisms: A survey," *ACM Comput. Surv.*, vol. 54, no. 4, May 2021. [Online]. Available: https://doi.org/10.1145/3448291

[4] S. Ariyapperuma and C. J. Mitchell, "Security vulnerabilities in dns and dnssec," in *The Second International Conference on Availability, Reliability and Security (ARES'07)*, 2007, pp. 335–342.

[5] N. Peleg, "Measuring negative caching behaviour in dns public resolvers," Ph.D. dissertation, 2023, copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2025-01-03. [Online]. Available: https://www.proquest.com/dissertations-theses/measuring-negative-caching-behaviour-dns-public/docview/2904089510/se-2

[6] M. Sammour, B. Hussin, M. F. I. Othman, M. Doheir, B. AlShaikhdeeb, and M. S. Talib, "Dns tunneling: A review on features," *International Journal of Engineering & Technology*, vol. 7, no. 3.20, pp. 1–5, 2018.

[7] J. Bushart and C. Rossow, "Anomaly-based filtering of application-layer ddos against dns authoritatives," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroSP)*, 2023, pp. 558–575.

[8] ——, "Anomaly-based filtering of application-layer DDoS against DNS authoritatives," in *8th IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Jul. 2023. [Online]. Available: https://github.com/cispa/DNS-Applayer-DDoS-Protection

[9] ——, "Anomaly-Based Filtering of Application-Layer DDoS Against DNS Authoritatives," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, 2023, pp. 558–575.

VI. APPENDIX

defense_summary.csv:

	A	B	C	D	E
1	Attack Type	Original Attack Traffic (pps)	Filtered Attack Traffic (pps)	Filtering Efficiency	Estimated FPR
2	mirai	5358.717778	3175.265833	0.40745791	0.01749
3	resolver	953.5683333	706.6158663	0.258977211	0.027975
4	sality	14.95111111	14.95111111	0	0.039135
5					
6					

Fig. 17: Defense Performance Summary (CSV showing filtering efficiency and false positive rate for each attack type)

```
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % touch visualize_results.py
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % nano visualize_results.py
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % chmod +x visualize_results.py
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % python3 visualize_results.py
Created figures/fpr_vs_traffic.png
Created figures/filtering_efficiency.png
Created figures/defense_summary.csv

Defense Performance Summary:

mirai Attack:
  Original Traffic: 5,358.72 pps
  Filtered Traffic: 3,175.27 pps
  Filtering Efficiency: 40.75%
  Estimated FPR: 0.0175

resolver Attack:
  Original Traffic: 953.57 pps
  Filtered Traffic: 706.62 pps
  Filtering Efficiency: 25.90%
  Estimated FPR: 0.0280

sality Attack:
  Original Traffic: 14.95 pps
  Filtered Traffic: 14.95 pps
  Filtering Efficiency: 0.00%
  Estimated FPR: 0.0391
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection %
```

Fig. 18: Terminal Output showing summary of defense performance.

```
~/Documents/PhD - SUTD/courses/Systems Security/project /do
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % touch parameter_optimization.py
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % nano parameter_optimization.py
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % chmod +x parameter_optimization.py
(base) mausamvora@MacBookAir DNS-Applayer-DDoS-Protection % python3 parameter_optimization.py

Optimizing parameters for mirai-2022-all-ips...
Running: test_window=8, low_pass=128, tolerance=2

Attack Evaluation Summary for datasets/extracted/mirai-2022-all-ips.json:
Total attack packets: 154,331,072
Packets passing filter: 43,410,000
Filtering efficiency: 71.87%
Estimated FPR: 0.0258
Results saved to output/parameter_optimization/mirai-2022-all-ips_w8_lp128_tol2.json
Running: test_window=8, low_pass=512, tolerance=2

Attack Evaluation Summary for datasets/extracted/mirai-2022-all-ips.json:
Total attack packets: 154,331,072
Packets passing filter: 71,074,128
Filtering efficiency: 53.95%
Estimated FPR: 0.0498
Results saved to output/parameter_optimization/mirai-2022-all-ips_w8_lp512_tol2.json
Running: test_window=8, low_pass=2048, tolerance=2

Attack Evaluation Summary for datasets/extracted/mirai-2022-all-ips.json:
Total attack packets: 154,331,072
Packets passing filter: 91,447,656
Filtering efficiency: 40.75%
Estimated FPR: 0.0428
Results saved to output/parameter_optimization/mirai-2022-all-ips_w8_lp2048_tol2.json
Running: test_window=8, low_pass=8192, tolerance=2

Attack Evaluation Summary for datasets/extracted/mirai-2022-all-ips.json:
Total attack packets: 154,331,072
Packets passing filter: 109,452,360
Filtering efficiency: 29.08%
Estimated FPR: 0.0496
Results saved to output/parameter_optimization/mirai-2022-all-ips_w8_lp8192_tol2.json
Running: test_window=24, low_pass=128, tolerance=2

Attack Evaluation Summary for datasets/extracted/mirai-2022-all-ips.json:
Total attack packets: 462,993,216
Packets passing filter: 130,230,000
Filtering efficiency: 71.87%
Estimated FPR: 0.0485
Results saved to output/parameter_optimization/mirai-2022-all-ips_w24_lp128_tol2.json
Running: test_window=24, low_pass=512, tolerance=2

Attack Evaluation Summary for datasets/extracted/mirai-2022-all-ips.json:
Total attack packets: 462,993,216
```

Fig. 19: Terminal output from parameter optimization showing how filtering efficiency and false positive rates vary with window size and low-pass filter.

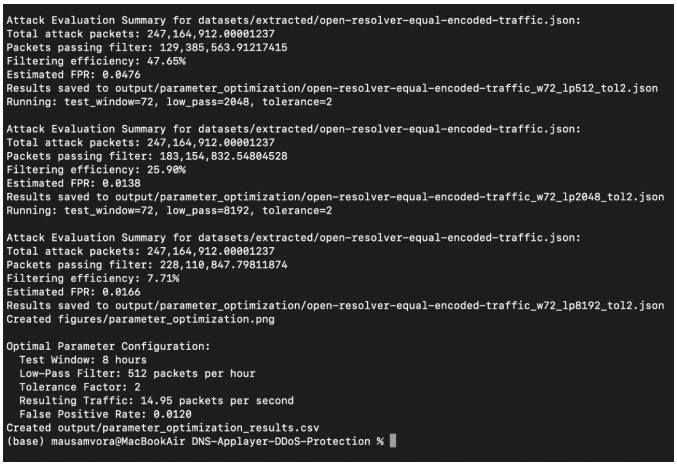

Fig. 20: The optimal configuration found in the replication applied to the open-resolver dataset