LAPRAD: LLM-Assisted PRotocol Attack Discovery

R. Can Aygun
rcaygun@cs.ucla.edu
UCLA

Yehuda Afek
afek@tauex.tau.ac.il
Tel-Aviv University

Anat Bremler-Barr
anatbr@tauex.tau.ac.il
Tel-Aviv University

Leonard Kleinrock
lk@cs.ucla.edu
UCLA

***Abstract*—With the goal of improving the security of Internet protocols, we seek faster, semi-automatic methods to discover new vulnerabilities in protocols such as DNS, BGP, and others. To this end, we introduce the LLM-Assisted Protocol Attack Discovery (LAPRAD) methodology, enabling security researchers with some DNS knowledge to efficiently uncover vulnerabilities that would otherwise be hard to detect.**

LAPRAD follows a three-stage process. In the first, we consult an LLM (GPT-o1) that has been trained on a broad corpus of DNS-related sources and previous DDoS attacks to identify potential exploits. In the second stage, a different LLM automatically constructs the corresponding attack configurations using the ReACT approach implemented via LangChain (DNS zone file generation). Finally, in the third stage, we validate the attack's functionality and effectiveness.

Using LAPRAD, we uncovered three new DDoS attacks on the DNS protocol and rediscovered two recently reported ones that were not included in the LLM's training data. The first new attack employs a bait-and-switch technique to trick resolvers into caching large, bogus DNSSEC RRSIGs, reducing their serving capacity to as little as 6%. The second exploits large DNSSEC encryption algorithms (RSA-4096) with multiple keys, thereby bypassing a recently implemented default RRSet limit. The third leverages ANY-type responses to produce a similar effect. These variations of a cache-flushing DDoS attack, called SigCacheFlush, circumvent existing patches, severely degrade resolver query capacity, and impact the latest versions of major DNS resolver implementations.

I. Introduction

Internet protocols are inherently complex and vulnerable, requiring researchers to quickly detect and fix weaknesses. Several studies, such as GRoot [1], and the work by Siu et al. [2], have explored formal modeling approaches for DNS, aiming to systematically analyze the protocol and uncover vulnerabilities. However, these approaches require substantial manual effort to model the protocol and result in only a partial protocol model, limiting the verification of protocol logic, as is the case in the DNS protocol which is complex with 297 RFCs [3].

In recent years, large language models (LLMs) have shown success in tasks like natural language understanding [4], text generation [5], and code generation [6].

In this paper we harness LLMs technology and prompt engineering to develop LAPRAD, a methodology to assist in the fast and efficient discovery of vulnerabilities in Internet protocols. We show how LAPRAD enables a security researcher familiar with the DNS protocol to identify potential DNS protocol DDoS vulnerabilities, and test these ideas in a controlled testing environment in an automated fashion. It follows three key steps: (1) the researcher provides an example attack to the LLM, which generates similar attack ideas and refines them through iterations. (2) another LLM automatically generates a suitable zone file for the discovered attack idea using the ReACT approach; and (3) the researcher tests the attack using the generated zone file in a real testing environment.

This methodology leverages LLMs' extensive knowledge of DNS protocol features and security warnings from RFCs and public sources (e.g., DNS-OARC, IETF DNSOP mailing lists), offering a faster alternative to the labor-intensive process of manually analyzing protocol documents and security forums. Moreover, LLMs retain information from remote, hard-to-find sources that may no longer be available, such as public mailing lists and conference websites. Thus serving as a valuable resource for future vulnerability research.

We use LAPRAD (our main contribution), to make the following additional contributions:

- Identified three distinct new variants of the CacheFlushing vulnerability based on DNSSEC. These variants bypass recently issued CacheFlushing patches [7] affecting the latest versions of major DNS resolver implementations (BIND 9.18.31 [8], Unbound 1.22.0 [9], and Knot 5.7.4 [10]), significantly degrading their throughput.
- Automatically generate attack zone files (an error-prone process), along with a high-level execution guide to comprehensively illustrate the discovered vulnerabilities (Section IV-B2).
- Assisting in pinpointing exact RFC sections and sentences that represent the root cause of the discovered vulnerabilities, while providing clear reasoning explaining why these elements contribute to the vulnerabilities (Section IV-B3).
- Re-discovering two recent DNS attacks, KeyTrap [11] and CacheFlush [12], without these attacks being part of the LLM's training data.

ISBN 978-3-903176-72-0

To develop effective mitigations for the newly discovered attacks, we conducted measurements on DNSSEC-enabled domains within the Cloudflare Top 100,000 list [13]. Based on these observations, we identified practical limits and proposed mitigation preventing abuse of the DNSSEC mechanism.

Section 2 provides background information on DNSSEC and the role of LLMs in this research, and related work. Section 3 outlines the threat model. In Section 4, we introduce the LAPRAD methodology, for discovering both known and zero-day attacks. Section 5 details the experiments conducted on the newly discovered attacks. Section 6 discusses findings and future work, and Section 7 concludes the paper.

II. Background and Related Work

LLMs are AI systems that generate human-like text by analyzing patterns with deep learning. Trained on massive datasets, they capture context and meaning to produce coherent responses [5].

Prompt engineering structures inputs to improve LLM responses. For our prompts, we applied techniques such as in-context learning [14], where prompts include examples (e.g., network protocol RFCs); one-shot learning, where a single example guides the model [5]; role-based prompting (e.g., "You are a DNS expert") [15]; Chain of Thought prompting, which helps the model outline logical steps [16]; ReAct, which allows LLM agents to interact with the external environment, retrieve outputs, and adjust their actions accordingly [17]; and the Multi-LLM agents approach, which involves multiple distinct LLMs collaborating to complete a task.

DNSSEC esures the autheticity of DNS responses, protecting from unauthorized responses, e.g., cache poisoning attacks. Each *RRSet* is signed by the domain's *DNSKEY* (which is protected by a chain of trust-keys starting at the root NS), ensuring response authenticity, with signatures stored in *RRSIG* records. DNSSEC is inherently complex, as it incorporates redundancy, high availability, and strong security mechanisms.

DNS DDoS Attacks NRDelegation exploits non-responsive authoritative responses to trigger excessive retries, depleting server CPU resources [18]. KeyTrap targets DNSSEC by sending malicious packets that force intensive cryptographic computations, resulting in CPU exhaustion [11]. CacheFlush [12] attack exploited large resource record sets, packing up to 2,000 NS records into a single response, 65kB, per malicious query. An attacker sending 1K qps (queries per second) easily flushes a 100MB resolver cache, reducing throughput by 80%. Different recent patches mitigate these including the latest, by capping RRSets at 100 records, rendering the CacheFlush attack infeasible with standard DNS RRSets.

DNS Protocol Modeling and Attack Discovery GRoot [1] introduced a formal model for DNS resolution to detect configuration errors such as rewrite loops, blackholing, and missing glue records. Liu et al. [2] improved DNS modeling by formalizing resolver logic, including query state, and caching. However, these models provide a limited view of the actual DNS protocol.

Applications of LLMs in Network Security LLMs have been applied to identify security issues in both general applications and network protocols. For example, they have been used to enhance fuzzing for IoT devices [19] and to identify inconsistencies in 4G/5G documentation [20]. Eywa [21] used LLMs to generate partial DNS resolver functions for testing and finding vulnerabilities. In contrast, our approach uses protocol knowledge directly, making vulnerability identification easier without relying on implementation. Google used LLM agents to identify a previously unknown stack buffer underflow in SQLite by leveraging a relevant code change—and guiding the system to search for similar issues [22]. This approach relates to our work as it trains LLMs on past examples to find similar vulnerabilities though it focuses on code-level issues.

III. Threat Model

In our threat model, the attacker controls a client to send malicious DNSSEC queries and an authoritative server to host domains that return crafted replies with excessive bogus data. An attacker can purchase a single domain for about $1 and create unlimited subdomains (e.g., sub1.attacker.com, sub2.attacker.com) at no extra cost. A single malicious domain can inject around 200KB of junk data into the cache, meaning 500 subdomains could flush a 100MB cache, while a 2GB cache may require 10,000. Using 20,000 subdomains is even more effective, as it forces the resolver to fetch new domains from the attacker's authoritative server. Bogus DNSSEC *RRSIGs* can be generated on the fly without cryptographic operations, requiring no extra memory per *RRSet*, so adding subdomains has minimal impact on cost or cpu/memory usage. This attack model is well-known and used in *CacheFlush* [12] and *KeyTrap* [11].

An attacker can run an authoritative server on a budget in two ways: hosting it on the cloud, with outgoing traffic costing around $0.60 per minute for a 2GB cache-flushing attack, or using managed DNS services like GoDaddy's premium plan, which offers unlimited subdomains and requests for $15 per month [12].

IV. LAPRAD methodology for DNS

Here we present the LAPRAD methodology for discovering DNS and DNSSEC DDoS vulnerabilities. We used GPT-o1 for attack discovery workflows and GPT-v4o for basic queries and secondary tasks [23]. We uncover new DNSSEC CacheFlushing vulnerabilities, and re-discovered recently published DNSSEC and DNS attacks — KeyTrap [11] and CacheFlush [12] — which were not part of GPT's training data. Both papers were published after the October 2023 training cutoff date of the models used. Due to space limitations, we outline only the key steps of the newly discovered SigCacheFlush attacks and briefly cover the Keytrap discovery. The full LLM conversations for all discovered vulnerabilities are available online [24].

A. LLM-Assisted Protocol Attack Discovery (LAPRAD)

LAPRAD is a conversational, multi-step prompting-based strategy for attack investigation, designed to systematically leverage LLM assistance to discover vulnerabilities, as illustrated in Figure-1.

Step-1: Attack Idea Investigation We instruct the LLM to act as a DNS security expert, providing a detailed example DNS-based DDoS attack description to guide its understanding. The LLM is then prompted to generate alternative DNS-based DDoS attacks targeting CPU, memory, or network bandwidth. An example initial prompt in this ***Step-1*** is shown in Prompt-1. The LLM produces a list of attack ideas for a human researcher to review:

- The researcher manually selects one of the proposed attack ideas generated by the LLM.
- The researcher instructs the LLM to elaborate on the selected attack idea, refining it to make the attack more advanced. The goal in this step is to uncover protocol features that an attacker could exploit to enhance the effectiveness of the attack.
- This feedback loop is repeated for several rounds until the attack idea is sufficiently interesting to the researcher for testing.

Step-2: Attack Configuration Generation The selected attack idea, shown as **Output-1** in Figure-1, is combined with a benign zone generation script to automatically create an attack zone generation script tailored to the attack requirements.

Step-3: Testing Perform a test on the LLM-proposed attack using the DNS zone file generated in ***Step-2***(**Output-2**). If the attack is invalid or the researcher opts to try a different idea, the process returns to ***Step-1*** to select another attack.

Prompt-1

You are a DNS security expert. Here is an example of a DNS attack that affects the cache of the resolver and reduces its performance:
NS Cache Flush Attack: An attacker owns an authoritative server and controls the example.com domain. The attacker issues A-type queries for different unique names under this domain, such as test1.example.com, ..., test10000.example.com. The attacker's authoritative server responds to these queries with approximately 1500 NS records, making the DNS reply reach the maximum allowed size of 65 KB. Each of these records is cached by the resolver, leading to the eviction of benign cached records. This forces the resolver to re-query these domains, significantly reducing its performance.
Task: Come up with another cache-flushing attack based on DNSSEC. Utilize your DNSSEC knowledge and explore potential mechanisms that could be used to target a resolver for cache flushing purposes.

B. Discovering New DNS Vulnerabilities

Here we use LAPRAD to discover new DNSSEC-based cache-flushing vulnerabilities and analyze their implications in three steps, see Figure-1. ***Step-1*** described below, ***Step-2*** in Section IV-B2, and ***Step-3*** in Section V.

1) *Step-1: Attack Idea Investigation*

We used the recently discovered *CacheFlushing attack* [12] as an illustrative example in Prompt-1 and tasked the LLM with identifying novel methods to flush a DNS resolver's cache through DNSSEC features, ultimately uncovering the *SigCacheFlush* vulnerabilities.

GPT responded to Prompt-1 by suggesting an attack that combines multiple DNSKEYs and signatures, multiple DS records, and NSEC3 records. We focused on the aspect involving large DNSKEY records and multiple signatures, as shown in (GPT-Response-1). Then we used this answer to follow up with GPT using Prompt-2.

GPT-Response-1

The attacker's domain is signed with multiple large DNSKEY records. For instance, it can include: Multiple RSA-4096 keys, ..., or A mix of RSA/ECDSA/EdDSA algorithms... Each DNSKEY RRset might be tens of kilobytes when combined (especially if multiple RRSIGs are present).

Prompt-2

Focus on GPT-Response-1. How many RSA 4096 keys can fit into a 65KB DNS response, including all the RRSIGs as well?

GPT-Response-2

Each DNSKEY record and its associated RRSIG consume: Per-key size=600 bytes (DNSKEY)+400 bytes (RRSIG)=1,000 bytes per key ...The total number of DNSKEY records is: Number of keys=Available space/Per-key size= 65035/1000=65(max-rsa packet- [24])

The CacheFlushing patches [7] set a default limit of 100 records per RRSet(BIND's maximum-records-per-type setting). GPT-Response-2 enables an attacker to stay within the 100-record limit while generating large responses. We named this variation, ***Multiple RSA-4096 Signatures-based Cache Flushing*** (multi-rsa [24]).

To evaluate GPT's reasoning ability regarding this patch [7], we introduced a stricter limit and aimed to reach the maximum DNS packet size of 65KB using only two DNSKEYs instead of 100 (Prompt-3):

Prompt-3

BIND has set a maximum limit of 2 records for any RRSet type. We need to be more creative to generate large responses containing 65KB answers in a single response, rather than

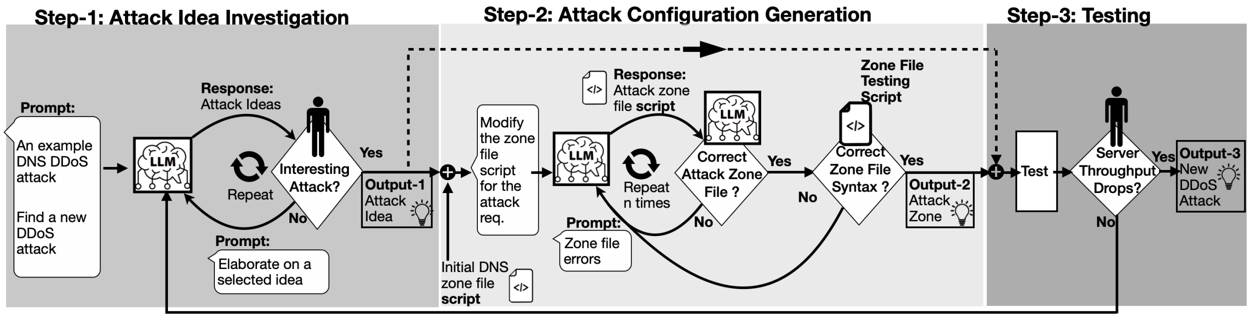

Fig. 1. Attack Investigation Workflow. The human figure indicates where human assistance is required to complete the task. The dashed line connects Output-1 (the attack idea) from ***Step-1*** to ***Step-3***, illustrating that it will be tested using the attack zone configuration (Output-2).

multiple responses. Focus on any unbounded field that can help achieve this goal.

In GPT-Response-3, GPT notices that DNSKEY and RRSIG RDATA fields are unbounded and proposes using two large DNSKEYs as a workaround for the Patch challenge.

GPT-Response-3

...nothing in the DNS wire-format truly prevents using absurdly large keys (e.g., 32k- or 64k-bit RSA). RRSIG RDATA: With a gigantic RSA key, the signature can also be thousands of bytes. In total, these 4 records can easily approach or exceed 60 KB if you size them aggressively. The DNS packet can reach the 65,535-byte maximum ... Resolvers can impose strict upper bounds on DNSKEY or RRSIG sizes and either truncate or refuse them. ... Some validating resolvers already treat extremely large keys as misconfigurations and fail validation.

GPT-Response-3 states DNSSEC protocol specifications do not strictly limit signature size, though extremely large signatures may be rejected by resolvers. Then, we asked Prompt-4.

Prompt-4

Would resolvers accept very large DNSKEYs or signatures?

GPT-Response-4

...Even if a resolver does not explicitly reject large keys, it might run out of memory or trigger other internal limits (e.g., "message size," "maximum memory usage," or "maximum TTL for suspicious data").

GPT-Response-4 states that large signatures are often impractical due to query timeouts from excessive memory overhead. As validation fails in such cases, we asked GPT the question in Prompt-5. Finally, GPT proposed an approach that bypasses the resolver validation and allows the resolver to cache the bogus signature in GPT-Response 5.

Prompt-5

If we want to have a resolver accept a very large RRSIG without actually validating it, how can we achieve this?

GPT-Response-5

... At least one of your DNSKEYs uses a mainstream algorithm (e.g., RSA/SHA-256) that the resolver recognizes. That legitimate key signs your zone normally, producing standard-sized RRSIGs that fully validate.
Using that huge unknown-algo DNSKEY, generate an enormous signature (the RRSIG).
...at least one RRSIG from a recognized algorithm, the zone passes DNSSEC validation overall. The "unknown algorithm" RRSIG is effectively ignored for validation. The resolver sees it, doesn't know how to validate it, but does not treat the zone as bogus.
Typically, the resolver caches all RRs returned from the authoritative server—even those it did not use in successful validation

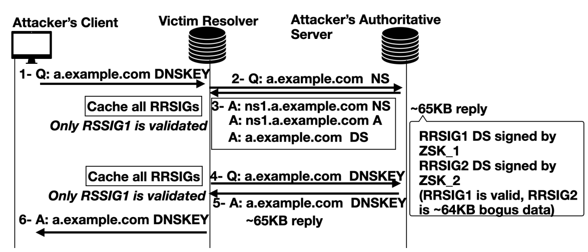

Fig. 2. Bait & Switch Cache Flushing Attack

The idea described by GPT in GPT-Response-5 suggests that a very large DNSSEC signature (RRSIG) can be cached by the resolver, provided the response includes at least one valid RRSIG for the corresponding RRSet. We named this attack ***Bait & Switch CacheFlushing Attack*** (bait-discovery- [24]). Figure 2 illustrates the attack mechanism. In this example, the attacker responds with two RRSIG records: RRSIG1, a valid signature that the resolver successfully verifies, and RRSIG2, a bogus signature approximately 64KB in size. Although only RRSIG1 is validated, the resolver caches both signatures. Each query by the resolver—for the DS and DNSKEY RRSets—receives a 65KB response containing similar bogus data. As a result, a single malicious DNSKEY query causes roughly 130KB of bogus data to be inserted into the resolver's cache. The *SigCacheFlush* attack we discovered using LAPRAD bypasses the CacheFlush patches [12], which limit the number of NS records returned. Our attack leverages large DNSSEC signatures instead, making it unaffected by these mitigations.(*Prompts for the ANY Type Cache Flushing Attack are omitted due to space, available at (any-discovery- [24])*).

The number of prompts to discover an attack (NPD) and unique GPT-generated ideas per prompt (NIP) (formatted as **(**# of ideas for 1st prompt, 2nd prompt,...**)**) in ***Step-1*** are: **Multiple RSA** (4, 1), **Bait and Switch** (4, 1, 4, 4, 1, 1), **ANY Type attack** (4, 8, 1, 1), **Cache Flush-CNAME** (3, 4, 4, 4, 1), **KeyTrap** (5, 1, 1, 1, 8, 5, 1), with NPD ranging from 2 to 7.

*2) **Step-2: Attack Configuration Generation***

We used GPT to generate detailed attack steps and configurations, including the attacker's zone file. Although GPT successfully outlined the steps, the generated zone files were inconsistent and too abstract for direct implementation. To address this issue, we implemented a ReACT-based script to automatically generate zone files without human intervention (Figure-1, Step-2).

(2.1) The attack idea's zone requirements from Figure-1, Step-1, are passed to the ReACT script, which then calls the *zone generation script* to create a basic DNS zone (e.g., sub.example.com). The LLM then checks if the zone file meets attack requirements, modifying the script iteratively for up to n = 5 iterations until the requirements are satisfied.

(2.3) Once the attack zone is structured, the ReACT script runs named-checkconf for syntax and rule validation. If errors occur, they are reported to the LLM, which updates the *zone generation script* to fix them.

(2.4) After validation, the generated zone file is uploaded to a virtual machine. The ReACT script then checks the BIND server startup logs for errors. If issues arise, the *zone generation script* is sent back to the LLM with the BIND errors for further corrections. These steps are combined in Step-2, as the *Zone File Testing Script*.

(2.5) The final version of the *zone generation script* is run in a loop to generate thousands of zone files, each with a given domain name as a parameter, ensuring sufficient attack subdomain configurations based on the victim resolver's size.

*3) **Source of Information***

We investigated the sources of information for the newly discovered vulnerabilities in three steps. First (①), we asked GPT how it formulated a specific idea, such as using a small valid signature alongside a large bogus one, explicitly requesting the exact RFC number and a verbatim quote. GPT provided reasoning and accurate RFC numbers, but the quotes were hallucinated. Next (②), we combined this initial response + (`Attack Summary`) + (`Full RFC Text`) → Instructed GPT to find relevant sentences. Finally (③), GPT correctly identified the sentences within the given RFC.

*4) **Proposed Mitigations***

Bait & Switch Cache Flushing Attack mitigation: Prevent oversize responses by limiting RRSIG signature size, with the default cap set to the maximum signature size of supported algorithms, RSA-4096 (744 bytes).

Multiple RSA-4096 Signatures-based Cache Flushing Attack mitigation: Lower the default max-records-per-type for DNSKEYs, and for RRSIGs per RRSet, from 100 to 20. Analyzing Cloudflare's top 100,000 domains, we found 9,336 DNSSEC-enabled domains. As shown in Figure-3 b), none of which require more than 16 DNSKEYs. Setting a default limit to 20 would considerably reduce the effect of the attack.

ANY Type Cache Flushing Attack mitigation: ANY-type queries can trigger large cumulative responses because each RRSet type's RRSIG is treated as a distinct RRSet, with limits applied separately. This allows numerous signatures (e.g., 100 for NS RRSIG and 100 for MX RRSIG). Setting separate limits for ANY-type queries can prevent this.

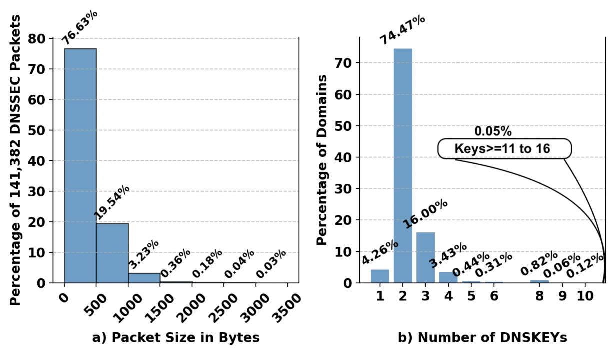

Fig. 3. For the top 9,336 DNSSEC-enabled domains and their TLDs, panel (a) presents the DNSSEC response packet size distribution, while panel (b) shows the DNSKEY distribution (up to 16 keys per domain, with values from 11 to 16 keys omitted as they account for only 0.05% of domains).

C. Re-discovering Existing DNS Vulnerabilities

KeyTrap attack [11] exploits DNSSEC's KeyTag collusion flaw, where multiple keys match the same KeyTag. Since resolvers must try all matching keys until one validates the signature, attackers can force excessive computations by triggering these collisions.

LAPRAD re-discovered the KeyTrap attack (keytrap-discovery [24]) in 7 prompts. We replace the example in Prompt 1, in Step-1, with the NRDelegation attack [18], following the refinement process in Figure 1, Step-1. Due to space limitation, we only show the final GPT response below.

GPT-Response-6

Repeated Signature Validation Attempts: ... scenario with 100 DNSKEYs and 100 corresponding RRSIGs, a resolver could be driven to perform up to 10,000 individual signature validation checks 100 DNSKEYs:Each DNSKEY has the same key tag or otherwise appears to be a valid candidate (e.g., due to multiple DS records referencing all of them).

100 RRSIGs: Each RRSIG claims to sign the DNSKEY RRset (or some relevant RRset), Trying Each DNSKEY Against Each RRSIG:In a naive (or maliciously forced) validation process, the resolver sees RRSIG 1 and tries all 100 DNSKEYs before determining which one (if any) actually matches. Then it moves on to RRSIG 2, tries all 100 DNSKEYs again, and so on through RRSIG 100. Multiplying the Checks: 100 RRSIGs x 100 DNSKEYs = 10,000 signature validations....

V. Experiments

We tested the three newly discovered attacks in a Docker environment to verify that they generate 65KB DNSSEC responses accepted by BIND, Unbound, and Knot. All three produced responses close to 65KB, which were successfully cached by the tested resolvers, confirming their feasibility.

We analyzed DNSSEC responses from 9,336 domains and found an average packet size of 449 bytes(Figure-3, a). SigCacheFlush attacks amplify this by 144×. Since all three attacks aim to flush the resolver's cache using large responses, we focused our performance evaluation on the ***Bait-and-Switch attack***, which is the most intricate one.

Step-3: Testing (Figure-1) A human expert gets a; detailed attack idea, high-level testing guide from the LLM, and an attack zone file to complete the test.

The performance impact of the ***Bait-and-Switch attack*** was evaluated in an Azure cloud environment. The setup included an attacker's authoritative server, a benign authoritative server, a recursive resolver, an attacker client, and a benign client. Each machine had an Intel Xeon E5-2673 v4 @ 2.30GHz (2 vCPUs), 8 GiB RAM, and Ubuntu 20.04.

To evaluate resolver performance under a ***Bait-and-Switch attack***, we use the Resperf [25] tool, a widely accepted standard for measuring resolver throughput. In this test, we measured the resolver's maximum throughput by using two query files: the attacker's Resperf query file contained 10,000 unique subdomains under a single purchased domain (see Section III for cost calculation), while the benign query file contained 100,000 unique domain names. Both clients began with an empty cache: the benign client gradually increased its query rate until the resolver reached its capacity, while the attacker client maintained a steady query rate.

We tested the attack using three attacker query rates: 300 qps, 1,000 qps, and 3,000 qps. The tests were performed on the latest versions of three resolvers—BIND 9.18.31, Unbound 1.22.0, and Knot 5.7.4—all configured with a 100MB cache. Each test was repeated three times, and the average results are shown in Figure-4, a) . Resolver throughput is reported as the maximum number of benign queries per second.

During the BIND tests, we observed a transient peak of approximately 4,000 benign QPS for the 1,000 QPS attacker. However, once the attack took effect, all benign responses from the resolver resulted in 'SERVERFAIL' (Figure-4, b)). In contrast, both Unbound and Knot experienced a significant drop in benign QPS under attack. Notably, Unbound never returned 'SERVERFAIL,' while Knot occasionally did, but at a rate of less than 1%.

Table-I presents the caching behavior of open resolvers for the newly discovered ***SigCacheFlush*** vulnerabilities. First, we queried the resolver for large DNSSEC records of a domain under our control, then terminated the domain's authoritative server. If the resolver continued responding to queries for this domain as before, it indicated successful caching; otherwise, it was merely relaying responses. Each resolver was tested only once to minimize disruption to legitimate services.

Open Resolver	**B&S**	**MK**	**ANY**
Google Public DNS (8.8.8.8)	✓	✓	✓
Cloudflare DNS (1.1.1.1)	✓	✓	×
Quad9 (9.9.9.9)	✓	✓	✓
OpenDNS -Cisco (208.67.222.222)	✓	✓	×

TABLE I
COMPARISON OF OPEN RESOLVERS FOR BAIT AND SWITCH (B&S), MULTIPLE RSA KEYS (MK), AND ANY-TYPE SIGCACHEFLUSH ATTACKS. (✓) INDICATES THAT THE RESOLVER SUPPORTS (OR IS VULNERABLE TO) THE FEATURE; (×) MEANS THE OPPOSITE.

VI. Discussions and Future Work

Currently, human operators drive the attack investigation process by focusing on answers they believe might lead to new attacks. This approach limits the scope of attack discovery, as it is influenced by the operator's intuition. Despite this limitation, we demonstrated that a viable attack could be achieved using a small number of general questions (2-7), with the only bias being the selection of target answers. This underscores the potential for further automation in the attack discovery process.

In future work, we aim to eliminate human involvement in the attack discovery loop by automating Step 1 of LAPRAD using ReACT [17] and multi-LLM agents. This will enable dynamic decision-making and iterative refinement during attack idea generation and RFC-based fact-checking.

VII. Conclusion

This paper introduced LAPRAD methodology which has shown the effectiveness of LLMs in identifying, analyzing, and addressing vulnerabilities in the DNSSEC protocol. Given a DNS attack targeting the resolver cache, LAPRAD led to the proposal of three new variants of the initial example attack, each exploiting completely different protocol features than the original. By leveraging detailed protocol features and adapting to constraints, LAPRAD successfully generated sophisticated attack ideas, such as the bait-and-switch and KeyTrap attacks.

We identified previously unknown DNSSEC-related vulnerabilities and confirmed our approach by testing LLMs on existing vulnerabilities not included in their

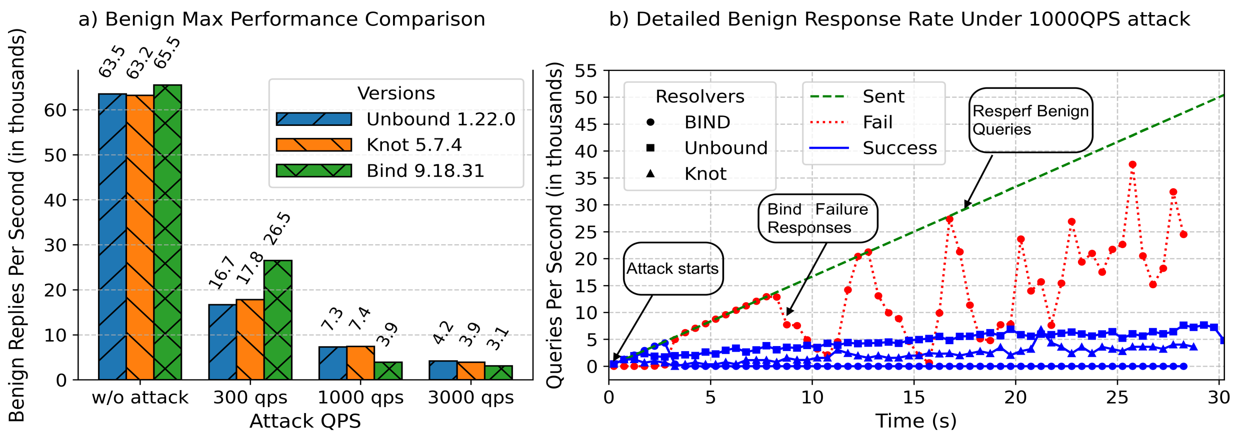

Fig. 4. a)Resolver Benign Max QPS Performance with Bait&Switch Attack (excluding SERVERFAIL responses) under constant attack rates of 0, 300, 1,000, and 3,000 QPS. b) Detailed benign response rates of resolvers benign max QPS under a bait-and-switch attack at constant 1,000 QPS. The attack starts at 0s, marked by the black arrow.

training set. The methodology we outlined can be applied to any network protocol or software system with well-documented standards, such as protocol RFCs.

Our findings demonstrate that LLM-assisted network vulnerability investigation is a promising foundation for further automation. Moving forward, we aim to create a highly automated LAPRAD framework that minimizes human involvement while maximizing the discovery of vulnerabilities. (An extended version of this study is available at [24].)

Acknowledgments

We thank George Varghese (UCLA) for helpful discussions and Shoham Danino for assistance with the testing environment. We thank Petr Špaček (ISC), Yorgos Thessalonikefs (Unbound), Otto Moerbeek (PowerDNS), and other DNS-OARC Mattermost members for their role in responsible disclosure and helpful discussions. This work was supported in part by a grant from the Sunday Group to The UCLA Connection Lab, as well as by ISF Grant No. 1527/23, a grant from the Blavatnik Interdisciplinary Cyber Research Center (ICRC) at Tel Aviv University, and the Blavatnik Family Fund.

References

[1] S. K. R. Kakarla, R. Beckett, B. Arzani, T. Millstein, and G. Varghese, "Groot: Proactive verification of dns configurations," in *Proceedings of the ACM SIGCOMM '20*, 2020, pp. 310–328.
[2] S. Liu *et al.*, "A formal framework for end-to-end dns resolution," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 932–949.
[3] "PowerDns." [Online]. Available: https://www.powerdns.com/
[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805
[5] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
[6] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.
[7] "Cve-2024-1737." [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-1737
[8] "BIND 9 - ISC." [Online]. Available: https://www.isc.org/bind/
[9] "Unbound." [Online]. Available: https://www.nlnetlabs.nl
[10] "Knot DNS." [Online]. Available: https://www.knot-dns.cz/
[11] E. Heftrig, H. Schulmann, N. Vogel, and M. Waidner, "The harder you try, the harder you fail: The keytrap denial-of-service algorithmic complexity attacks on dnssec," in *Proceedings of the ACM CCS 2024*, 2024, pp. 497–510.
[12] Y. Afek, A. Bremler-Barr, S. Danino, and Y. Shavitt, "A flushing attack on the {DNS} cache," in *USENIX Security 24*, 2024, pp. 2299–2314.
[13] "Top100kDomains." [Online]. Available: https://radar.cloudflare.com/
[14] Q. Dong *et al.*, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.
[15] A. Kong *et al.*, "Better zero-shot reasoning with role-play prompting," *arXiv preprint arXiv:2308.07702*, 2023.
[16] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
[17] S. Yao et al., "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.
[18] Y. Afek, A. Bremler-Barr, and S. Stajnrod, "{NRDelegationAttack}: Complexity {DDoS} attack on {DNS} recursive resolvers," in *USENIX Security 23*, 2023, pp. 3187–3204.
[19] X. Ma, L. Luo, and Q. Zeng, "From one thousand pages of specification to unveiling hidden bugs: Large language model assisted fuzzing of matter {IoT} devices," in *USENIX Security 24*, 2024.
[20] M. M. Rahman, I. Karim, and E. Bertino, "{CellularLint}: A systematic approach to identify inconsistent behavior in cellular network specifications," in *USENIX Security 24*, 2024, pp. 5215–5232.
[21] S. K. R. Kakarla and R. Beckett, "Oracle-based protocol testing with eywa," *arXiv preprint arXiv:2312.06875*, 2023.
[22] "Google project zero, from naptime to the big sleep," accessed: 2025-01-17. [Online]. Available: https://googleprojectzero.blogspot.com/2024/10/from-naptime-to-big-sleep.html
[23] "GPT-4." [Online]. Available: https://openai.com/index/gpt-4/
[24] "All investigation conversations." [Online]. Available: https://tinyurl.com/gptconversations
[25] "Resperf." [Online]. Available: https://www.dns-oarc.net/tools/dnsperf