SCALE: Automatically Finding RFC Compliance Bugs in DNS Nameservers

Siva Kesava Reddy Kakarla[1] Ryan Beckett[2] Todd Millstein[1,3] George Varghese[1]
[1]*University of California, Los Angeles* [2]*Microsoft* [3]*Intentionet*

Abstract

The Domain Name System (DNS) has intricate features that interact in subtle ways. Bugs in DNS implementations can lead to incorrect or implementation-dependent behavior, security vulnerabilities, and more. We introduce the first approach for finding RFC compliance errors in DNS nameserver implementations, via automatic test generation. Our SCALE (Small-scope Constraint-driven Automated Logical Execution) approach *jointly* generates zone files and corresponding queries to cover RFC behaviors specified by an executable model of DNS resolution. We have built a tool called FERRET based on this approach and applied it to test 8 open-source DNS implementations, including popular implementations such as BIND, POWERDNS, KNOT, and NSD. FERRET generated over 13.5K test cases, of which 62% resulted in some difference among implementations. We identified and reported 30 new unique bugs from these failed test cases, including at least one bug in every implementation, of which 20 have already been fixed. Many of these bugs existed in even the most popular DNS implementations, including a critical vulnerability in BIND that attackers could easily exploit to crash DNS resolvers and nameservers remotely.

1 Introduction

The Domain Name System (DNS) plays a central role in today's Internet, as it allows users to connect to online services through user-friendly domain names in place of machine-friendly IP addresses. Organizations across the Internet run DNS *nameservers*, which use DNS configurations called zone files to determine how to handle each query, either returning an IP address, rewriting the query to another one, or delegating the responsibility to another nameserver. There are many popular nameserver implementations of the DNS protocol in the wild, both open-source [21, 23, 25, 76] and in public or private clouds [2, 39, 85, 97].

Over time DNS has evolved into a complex and intricate protocol, spread across numerous RFCs [41, 80, 86, 96]. It is difficult to write an efficient, high-throughput, multithreaded implementation that is also bug-free and compliant with these RFC specifications. As a result, nameserver implementations frequently suffer from incorrect or implementation-specific behavior that causes outages [34, 103, 106], security vulnerabilities [74, 94], and more [15, 19, 22].

This paper presents the first approach for identifying RFC compliance errors in DNS nameserver implementations, by automatically generating test cases that cover a wide range of RFC behaviors. The key technical challenge is the fact that a DNS test case consists of both a query and a zone file, which is a collection of *resource records* that specify how queries should be handled. Zone files are highly structured objects with various syntactic and semantic well-formedness requirements, and the query must be related to the zone file for the test even to reach the core query resolution logic.

Existing standard automated test generation approaches are not suitable for our needs, as illustrated in the top of Figure 1. Fuzz testing is scalable but has well-known challenges in navigating complex semantic requirements and dependencies [13, 36], which are necessary to generate behavioral tests for DNS. As a result, fuzzers for DNS only generate queries and hence are used only to find parsing errors [10, 32, 89, 99]. Symbolic execution [72] can, in principle, generate DNS tests that achieve high code coverage but, in practice, suffers from the well-known problem of "path explosion" [9, 13, 36] that limits scalability and coverage. As a result, symbolic execution has only been used to identify generic errors like memory leaks in individual functions within nameserver implementations, again avoiding the need to generate zone files [93].

Our approach to automated testing for DNS nameservers, which we call **SCALE** (Small-scope Constraint-driven Automated Logical Execution), *jointly* generates zone files and the corresponding queries, does so in a way that is targeted toward covering many different RFC behaviors, and is applicable to black-box DNS nameserver implementations. The key insight underlying SCALE is that we can use the existing RFCs to define a model of the logical behaviors of the DNS resolution process and then use this model to guide test generation. Specifically, we have created an *executable* version of a recent formal semantics of DNS [71], which we then symbolically execute to generate tests for black-box DNS nameservers — each test consisting of a well-formed zone file and a query that together cause execution to explore a particular RFC behavior. Intuitively, tests that cover a wide variety of behaviors in our executable model will also cover a wide variety of behaviors in DNS nameservers since they have the same goal, namely to implement the RFCs.

Symbolic execution of our logical model is still fundamentally unscalable — there are an unbounded number of possible

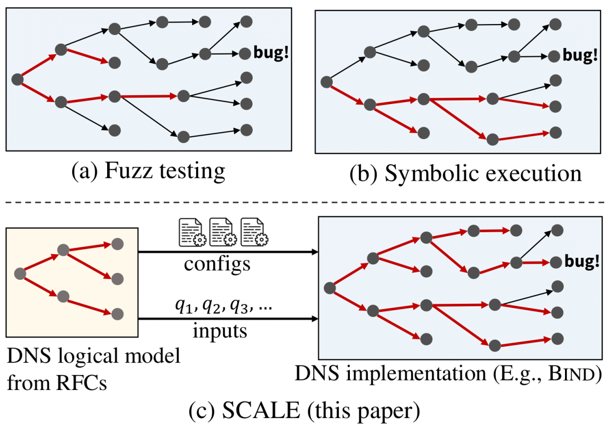


Figure 1: Overview of different automated testing approaches. Tested implementation paths are shown in red. (a) Fuzz testing is scalable but is often unable to navigate complex input requirements. (b) Symbolic execution can solve for input conditions but suffers from path explosion and has difficulty with complex data structures and program logic, and will thus only typically explore a small subset of possible program paths. (c) SCALE uses a logical model of the DNS RFCs to guide symbolic search toward *many* different *logical behaviors*.

execution paths, they grow exponentially in the size of the zone file, and expensive constraint solvers must be used to generate a test case for each path. We therefore bound the generated zone files to contain a very small number of resource records and short domain names — a maximum of 4 for each of these in our experiments, which is much smaller than real-world zone files. However, we provide experimental evidence of the existence of a *small-scope* property [43], meaning that many interesting behaviors can be covered with small tests. First, each return point in our logical model can be reached with a test where the length of domain names and the number of records in the zone file is at most 3. Each return point represents a distinct RFC-specified scenario for DNS resolution (e.g., a particular flavor of query rewrite). Second, while increasing this constant from 2 through 4 increased the number of errors that our tool identified, no new errors were found in a sample of paths that required size 5. This finding makes sense because, while zone files can contain a large number of records, the number of records that are relevant to any particular query tends to be small.

We have used the SCALE approach as the basis for a tool called FERRET[1] for automated testing of DNS nameserver implementations (Figure 2). FERRET generates tests using our logical model, which we have implemented in a modeling language called Zen [4] that has built-in support for symbolic execution. FERRET then performs *differential* testing by running these tests on multiple DNS nameserver implementations and comparing their results to one another. In this way FERRET can identify RFC violations, crashes, as well as situations where the RFCs may be ambiguous or underspecified, leading

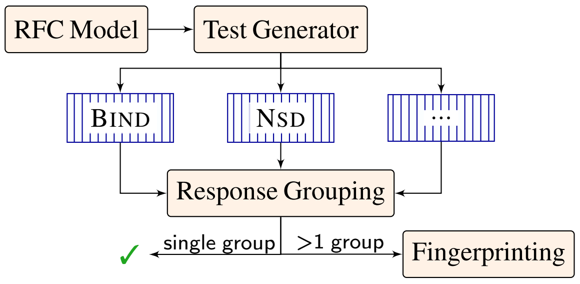


Figure 2: FERRET system architecture.

to implementation-dependent behavior. Because DNS implementers strive for behavioral consistency among their implementations [92], any test that produces divergent results among the implementations represents a likely error. However, there can be orders-of-magnitude fewer root causes than divergent tests, so as a final step we provide a simple but effective technique to help users with bug deduplication. We create a *hybrid fingerprint* for each test, which combines information from the test's path in the Zen model with the results of differential testing, and then group tests by fingerprint for user inspection.

Using FERRET, in just a few hours we generated over 12.5K valid test cases[2] with a maximum zone-file size of 4 records. Running these tests on 8 different open-source DNS nameserver implementations, we found that the implementations' behaviors only completely agreed on 35% of the tests. Our fingerprinting technique reduced the remaining cases to roughly 75 groups. Because our executable model includes a specification of the well-formedness conditions for zone files, we also leveraged Zen to systematically generate zone files that violate one of these conditions. We generated 900 invalid zone files of which 184 resulted in some difference among implementations. Inspecting tests from each fingerprinted group resulted in the discovery of 30 unique bugs across the different implementations. Developers have confirmed all of them as actual bugs and fixed 20 of them, at the time of writing. The most severe bug FERRET found was a subtle combination of zone file and query that an attacker could easily use to crash both BIND nameservers *and* resolvers remotely. We engaged in a secure disclosure process, after which the developers fixed the issue and then publicly disclosed the vulnerability, through a CVE (CVE-2021-25215) [26, 38] rated with high-severity.

Contributions: This paper's contributions are:

- The first automated approach to identify RFC violations in black-box DNS nameservers. A unique feature of our approach, SCALE, is the joint generation of zone files and queries to produce high-coverage behavioral tests.
- An implementation of our approach in FERRET that combines SCALE with differential testing.
- A novel fingerprinting approach for bug deduplication that takes advantage of our RFC model to help triage bugs.
- An evaluation from testing 8 different open-source DNS nameserver implementations with tests generated by FER-

[1]FERRET: https://github.com/dns-groot/Ferret

[2]Test cases: https://github.com/dns-groot/FerretDataset

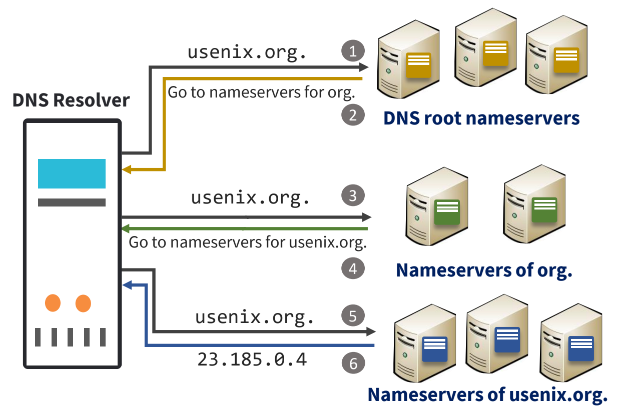


Figure 3: The resolution process for the domain name `usenix.org` (with no caching).

RET consisting of over 13.5K zone files, which resulted in the discovery of 30 new unique bugs and no false positives.

2 Background And Motivation

In this section, we first give a brief overview of DNS and then motivate FERRET through two previously unknown errors that it found in the popular BIND software for DNS [23].

2.1 Overview of DNS

The Domain Name System (DNS) is the phone book of the Internet. Its primary role is to translate domain names (like `usenix.org`) into various pieces of information, IP addresses being the most common. A domain name is represented as a sequence of labels joined by the `.` character. These labels form a tree-like hierarchy with the root as `.` and `org` as a child of it and so on. Each label at any level in the hierarchy can contain information, and the user obtains that information by querying the domain name formed by joining the labels from that node to the root. Data is stored as DNS *resource records* where each record has a domain (owner) name, a type for its information, and the content, among other things.

The namespace database tree is divided into a large number of *zones*. A zone is a collection of records that share a common end domain name. For example, the `usenix.org` zone has only records ending with `usenix.org`. All the resource records of a zone are available to the user through a set of authoritative nameservers, which are in turn identified by a domain name. For example, the `usenix.org` zone is available from servers like `dns1.easydns.com`, `dns2.easydns.net` and `dns3.easydns.ca`. The same zone is served by multiple servers to ensure redundancy and availability.

To resolve a domain name like `usenix.org` to its IP address, a client will traverse the tree from one of the root nameservers. The root nameserver checks its local zone file and either provides the IP record or returns a set of authoritative nameservers to ask instead. The client continues

Example Record			Description
`a.exm.org.`	`A`	`1.2.3.1`	IPv4 record
`*.exm.org.`	`AAAA`	`1:db8::2:1`	Wildcard IPv6 record
`s.exm.org.`	`NS`	`ns.dns.com.`	Delegation record
`c.exm.org.`	`DNAME`	`cs.org.`	Domain redirection
`w.exm.org.`	`CNAME`	`a.exm.org.`	Canonical name

Table 1: Examples of common DNS record types.

by querying the new set of nameservers either until the query is resolved or gets a non-existent domain name error. The process or the software that performs this traversal on the client side is called a *resolver*. The resolution process for the domain `usenix.org` is shown in Figure 3.

A nameserver can serve multiple zones. When a query comes to the nameserver, it first checks whether the query ends with any of the zone domains; otherwise, it sends a refusal message to the resolver. After picking a zone, the nameserver will look up the query name's closest matching records. It then creates a response based on the query type and the records selected. DNS supports many record types, including records for IP addresses, pointers to other records, domain aliases, delegation records, and more. Table 1 shows a few example records.

2.2 Finding DNS Errors with FERRET

The goal of FERRET is to automatically generate high-coverage query and zone file inputs to find behavioral errors in DNS nameserver implementations. In this subsection we illustrate both the challenges in doing so and FERRET's capabilities through two example errors that it automatically found in BIND.

Bug #1: BIND sibling glue records bug. FERRET generated the following test case, which identified a previously unknown performance bug in BIND [47].[3]

```
    campus.edu.  SOA   ...
foo.campus.edu.  NS    ns1.campus.edu.
ns1.campus.edu.  A     1.1.1.1
```

Query: ⟨`anything.foo.campus.edu., A`⟩

In this test case, the query matches the `NS` record in the zone file, which delegates the query to another nameserver, `ns1.campus.edu`. However, that nameserver happens to be a sibling of `foo.campus.edu` (as they are both directly under `campus.edu`), and the zone file contains an `A` record, called a *glue record* [41], for the nameserver's IP address. NSD, KNOT, and POWERDNS correctly return the `NS` record along with the glue record, avoiding extra round-trips to determine the nameserver's IP address, while BIND returns only the `NS` record. Returning the sibling glue record is not compulsory, but our test case exposed two unrelated errors that can negatively affect the performance of many queries.

[3]Note that we have renamed the labels for all the example bugs for clarity.

After we filed the issue the BIND developers confirmed the bug saying, "This report turns out to be very interesting..." Briefly, BIND uses a "glue cache" that had two bugs. First, if the cache lookup fails, then glue records are supposed to be searched for in the zone file, but this was not happening. Second, glue records for siblings domain nameservers were accidentally never searched for at all.

This example illustrates the challenges of identifying nameserver behavior errors. Even though the zone file has only a few records, they have complex dependencies. First, there must be a delegation of the query to another nameserver. Second, that nameserver must be in the same zone. Third, that nameserver must be a sibling domain. Fourth, there must be a glue record for that domain in the zone. Given these dependencies, it is understandable that prior testing techniques did not uncover these bugs. Further, by comparing the outputs from multiple implementations, FERRET is able to identify this test case as potentially buggy behavior despite receiving a valid response from BIND.

Bug #2: BIND crash. As another, more dire example, consider the following zone file that FERRET generated. The zone file is invalid due to having two identical records, but BIND, NSD, and KNOT accept the zone file and make it valid by ignoring the duplicate record.

```
      attack.com.   SOA    ...
      attack.com.   NS     ns1.outside.com.
      attack.com.   NS     ns1.outside.com.
 host.attack.com.   DNAME  com.
```

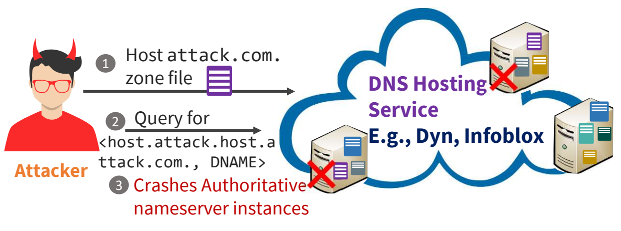
Query: ⟨`host.attack.host.attack.com.`, `DNAME`⟩

FERRET generated multiple queries for this zone file (§ 3.6) and the one showed above caused BIND to crash.

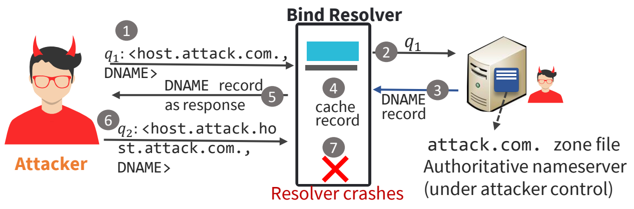
In this test case, the `DNAME` record is applied to rewrite any queries ending with `host.attack.com` to end with just `com`, so the query that FERRET generated is rewritten to the new query `host.attack.com`. The nameservers add the `DNAME` record and rewritten query to the response before resolving the new query. The new query exactly matches the same `DNAME` record, so implementations are expected to return the current response. All implementations except BIND behaved as expected. BIND did not respond, and the query timed out. Inspecting the logs, we found that the server crashed with an assertion failure due to an attempt to add the same `DNAME` record to the response twice.

This error constitutes a critical security vulnerability. We next describe two scenarios to show how this failed assertion check can be exploited remotely by an attacker.

Scenario 1 - Attack on a DNS hosting service that uses Bind: DNS hosting services using BIND's authoritative nameserver implementation (e.g., Dyn [42]) are vulnerable to this attack. An attacker can upload the above zone file to the authoritative server instances through the hosting service. Then, when the above query is requested, the server instances will crash as shown in Figure 4(a). Since a server instance


Host attack.com. zone file
Query for <host.attack.host.attack.com., DNAME>
Crashes Authoritative nameserver instances
Attacker
DNS Hosting Service
E.g., Dyn, Infoblox

(a) Attack on a DNS hosting service using Bind


Bind Resolver
q_1: <host.attack.com., DNAME>
q_1
DNAME record
DNAME record as response
cache record
q_2: <host.attack.host.attack.com., DNAME>
Resolver crashes
Attacker
attack.com. zone file
Authoritative nameserver (under attacker control)

(b) Attack on a public Bind DNS Resolver

Figure 4: `DNAME` attack targeting the DNS hosting services (a) and the public BIND based recursive resolvers (b).

will generally be serving zone files from multiple customers, such a crash will take down the zones for all customers hosted at that nameserver. This provides a method for attackers to trivially and remotely initiate a denial of service attack against customers hosted by such a service.

Scenario 2 - Attack on a public Bind DNS resolver: In this second scenario, the attacker can crash any public DNS resolver based on BIND, thereby constituting, as stated by the BIND security team, an "easily-weaponized denial-of-service vector." As illustrated in Figure 4(b), the attacker purchases, registers, and controls the `attack.com` zone and its authoritative servers. The attacker then simply requests the `DNAME` record from a public recursive resolver running BIND, which attempts to fetch the result from the attacker's authoritative server. This record is cached, and then the test query is sent to the resolver. The resolver uses the cached `DNAME` record and ultimately crashes as described earlier. In some estimates, BIND accounts for over half of all DNS resolvers in use [75], which means that attackers could effectively initiate a simple distributed denial of service (DDoS) attack against the numerous ISPs and public resolvers available to end users.

Disclosure: After discovering the `DNAME` attack, we initiated a responsible disclosure procedure with the BIND maintainers. Understanding the attack severity, they requested that we keep the issue confidential until they worked through their process to patch and then disclose the bug to the relevant parties in a controlled manner. BIND released a Common Vulnerabilities and Exposure (CVE-2021-25215) [26, 38], with a "high severity" rating and asked developers and users to upgrade to the patched version. The attack affected all maintained BIND versions, which in turn affected RHEL, Slackware, Ubuntu, and Infoblox.

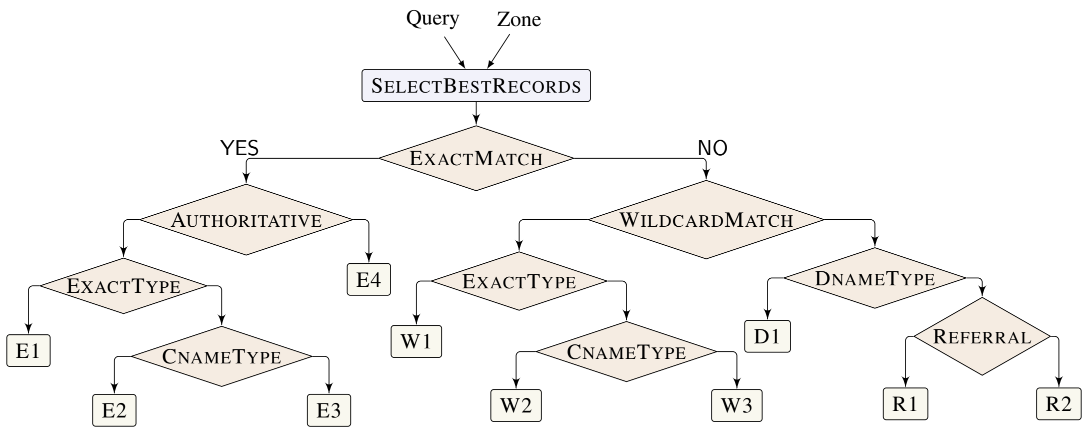


Figure 5: Abstract representation of the Authoritative DNS decision tree used to respond to a user query.

3 Methodology

In this section we overview our methodology for generating high-coverage tests for DNS nameserver implementations and discuss how we address several technical challenges.

3.1 SCALE Approach

As illustrated by the examples in the previous section, the inputs to a DNS nameserver — a query and a zone file containing a set of records — are highly structured. Further, records can be of many different types and have many different kinds of dependencies among them. Therefore, an effective approach to automatically identifying RFC violations must be able to generate *valid* inputs that meet the required structural and semantic constraints of the domain, and it must also be able to explore different combinations of record types and features in a systematic way. To solve this joint generation problem, our approach, SCALE (Small-scope Constraint-driven Automated Logical Execution) leverages a specification of the DNS nameserver logic to drive test generation. Specifically, we have created an *executable* version of an existing DNS specification [71] and generate tests through *symbolic execution* [72] on this executable specification. Symbolic execution is a static analysis technique that enumerates execution paths in a program and uses automated constraint solvers to produce an input that will take each enumerated path, thereby generating tests that cover many different program behaviors.

While the end-to-end behavior of a DNS query lookup can require contacting many nameservers, we employ a *compositional* approach that only generates tests for a single nameserver in isolation. Because our formal model considers the space of all inputs to the nameserver that could be produced by the rest of the system, and because the "next step" delegation of the resolution process is captured in the output at a single nameserver, this approach still allows us to generate tests for all behaviors of the end-to-end DNS. In other words, any implementation bug that exists in a DNS nameserver implementation can be found using our approach. In general, a downside of compositional testing is that it can lead to false positives if the tester considers input states that are, in reality, unreachable with respect to the rest of the system. However, in the case of DNS, nameservers keep no internal state — the response they provide is based only on the supplied query and configuration. This stateless nature implies that compositional testing will not incur any false positives.

Hence our formal semantics focuses on query lookup at a single nameserver, which we model as a stateless function that takes a user query and a zone file and produces a DNS response. Figure 5 shows an abstract view of this function. Given the input query and zone, DNS will first select the closest matching records in the zone for the query using the SELECTBESTRECORDS function and then follow the decision logic laid out in the figure using these records. Each leaf node represents a unique case in the DNS. For example, the tree shows four different cases of exact matches, labelled E1 through E4. Symbolic execution of our query-lookup function generates inputs that drive the function down different execution paths, thereby enabling us to systematically explore the space of DNS behaviors and feature interactions.

Example: Consider the path in Figure 5 to the leaf labelled R1. In order to reach that leaf, the selected records must not contain one with either an exact match or a wildcard match on the query domain name. Further, there should not be a DNAME match but should be one of type NS (REFERRAL). Finally, while not shown in the figure, when preparing a response to the query the function will also search for a glue record if the NS target is in the same zone. Solving all of these constraints caused symbolic execution to automatically generate the first test case shown in § 2.2, which identified two errors in BIND.

3.2 An Executable Model of DNS

We have created an implementation of the formal semantics of query lookup [71] as a program in a modeling language called Zen [4], a domain-specific language (DSL) embedded in C#.

```
 1  Zen<Response> QueryLookup(
 2    Zen<Query> q,
 3    Zen<Zone> z)
 4  {
 5    var records = SelectBestRecords(q, z);
 6    var rname = records.At(0).Value().Name();
 7    var types = records.Select(r => r.Type());
 8
 9    return If(
10      rname == q.Name(),
11      ExactMatch(records, q, z),
12      If(
13        IsWildcardMatch(q.Name(), rname),
14        WildcardMatch(records, q, z),
15        If(
16          types.Any(t => t == RType.DNAME),
17          Rewrite(records, q),
18          If(
19            And(types.Any(t => t == RType.NS),
20                Not(types.Any(t => t == RType.SOA))),
21            Response(Tag.R1,
22              Delegation(records, z), Null<Query>()),
23            Response(Tag.R2, empty, Null<Query>())
24      ))));
25  }
```

Figure 6: Record lookup model in C# using Zen.

To illustrate this approach, we show several components of our model. Figure 6 shows the model's main query-lookup function, as depicted in Figure 5. The function first selects the best records (Line 5) and then tests if the query domain name is equal to the records' domain name (Line 10). If so, then this is an exact match and the model calls out to a helper function to specifically handle the `ExactMatch` subcase (Line 11). Similarly, if the query domain name is a wildcard match for the record domain name (Line 13), then we invoke the `WildcardMatch` subcase (Line 14). We show the implementation of wildcard matching in Figure 7. This function implements the case where the best matching record is a wildcard, properly handles interactions with CNAME records, and synthesizes the correct records for use in the resolver cache.

Our complete executable model consists of 520 lines of C# code. The model can also easily extend to new DNS RFCs that would be added in the future. Similarly, if an organization has a particular way of resolving RFC ambiguities or purposely deviates from the RFCs in specific ways, the organization can modify the logical model to reflect that intent.

We chose to implement our formal model in Zen because it has built-in support for symbolic execution. In Zen, certain inputs can be marked as *symbolic*, and the tool will then leverage SMT solvers [27] to produce concrete values for these inputs that drive the program down different execution paths. In our code examples, the `Zen<T>` type for inputs has the effect of marking them as symbolic. The tests produced by symbolic execution can then be used to test any DNS nameserver implementation. However, making symbolic execution effective required us to address several challenges, which we describe in the rest of this section.

```
26  Zen<Response> WildcardMatch(
27    Zen<IList<ResourceRecord>> rrs,
28    Zen<Query> q,
29    Zen<Zone> z)
30  {
31    var exact = rrs.Where(r => r.Type() == q.Type());
32    var record = rrs.At(0).Value();
33    var newQuery = Query(record.RData(), q.Type());
34    var exactSyn = RecordSynthesis(exact, q.Name());
35    var cnameSyn = RecordSynthesis(rrs, q.Name());
36
37    return If(
38      exact.Length() > 0,
39      Response(Tag.W1, exactSyn, Null<Query>()),
40      If(
41        rrs.Any(r => r.Type() == RType.CNAME),
42        Response(Tag.W2, cnameSyn, Some(newQuery)),
43        Response(Tag.W3, empty, Null<Query>())
44      ));
45  }
```

Figure 7: Wildcard match model in C# using Zen.

3.3 Generating Valid Zone Files

The first challenge that we encountered is that zone files must satisfy several constraints in order to be considered well-formed. For instance, if there is a DNAME record in a zone file for `math.uni.edu`, then no other records below this domain name may exist, for any record type (e.g., an A record for `fun.math.uni.edu` is not allowed). The DNS RFCs define many such constraints as a way to eliminate ambiguous or useless zones, as shown in Table 2. Naively performing symbolic execution will produce many zone files that are not well formed. Further, DNS implementations typically preprocess zone files to reject ill-formed zones, thereby failing to test the intended execution path of the query lookup logic.

Fortunately, our SCALE approach admits a natural solution to this problem. We have formalized all of the DNS zone validity conditions as predicates in Zen. Whenever Zen's symbolic execution engine produces a constraint representing the conditions under which the query lookup function takes a particular execution path, we conjoin these predicates to that constraint before Zen passes it off to an automated constraint solver. In this way we ensure that all test cases will have well-formed zone files by construction.

3.4 Data Representation

In our Zen model, we represent zone files as a list of resource records, where each resource record contains a domain name, record type, and data fields. We represent user queries similarly as consisting of a domain name and a query type. Record and query types are represented using enums, which Zen translates to integer values.

One challenging decision we ran into was how best to represent and model domain names, for both zone records and record data, in a manner that permits fully automatic and scalable analysis. For instance, a natural way to encode domain names

	Validity Condition	RFC Document
i.	All records should be unique (there should be no duplicates).	2181 [28]
ii.	A zone file should contain exactly one `SOA` record.	1035 [87]
iii.	The zone domain should be prefix to all the resource records domain name.	1034 [86]
iv.	If there is a `CNAME` type then no other type can exist and only one `CNAME` can exist for a domain name.	1034 [86]
v.	There can be only one `DNAME` record for a domain name.	6672 [96]
vi.	A domain name cannot have both `DNAME` and `NS` records unless there is an `SOA` record as well.	6672 [96]
vii.	No `DNAME` record domain name can be a prefix of another record's domain name.	6672 [96]
viii.	No `NS` record can have a non-`SOA` domain name that is a prefix of another `NS` record.	1034 [86]
ix.	Glue records must exist for all `NS` records in a zone.	1035 [87]

Table 2: Summary of DNS zone file validity conditions specified in various RFCs.

would be as string values (a domain name is just a '`.`' separated string). Indeed, modern SMT solvers like Z3 [27] support the logical theory of strings, so this is a natural approach to consider. However, the theory of strings is in general undecidable [14, 35]. Moreover, this encoding would require us to define complex predicates for manipulating domain names, including extracting each of the labels of a domain name and checking whether one domain name is a prefix of another.

Therefore, rather than model domain names as strings, we take advantage of the observation that the particular character values in a domain name label string do not matter for DNS lookup. Instead, all that matters is whether two labels are equivalent to one another and whether a label represents a wildcard. As such, we encode a domain name in Zen as a list of integers and use a specific integer value to represent the wildcard character '`*`'. This allows us to use simple, efficient integer operations and constraints to manipulate domain names according to our formal model.

3.5 Handling Unbounded Data

A final challenge associated with symbolic execution for our formal model is the fact that there are several sources of *unboundedness*. For example, a zone file can contain an unbounded number of records, and a domain name can contain an unbounded number of labels. Our Zen model contains an unbounded number of paths, since the number of resource records in a zone file is unbounded and the function to select the best records must examine all of them and compare them to one another. SMT constraint solvers have limited support for unbounded data structures such as lists, and in general, reasoning about such constraints requires quantifiers, which lead to undecidability [95]. Therefore, in our Zen implementation we only consider inputs that have a bounded size, e.g., at most N records in a zone file, and hence only produce test cases that respect these bounds. The size of inputs is a parameter that is configurable by the user. While the SCALE approach can therefore fail to detect some errors, we provide experimental evidence of the existence of a *small-scope* property [43], meaning that many interesting behaviors, and behavioral errors, can be exercised with small tests (§ 5.1).

3.6 Generating Tests for Invalid Zone Files

While it's critical to be able to generate well-formed zone files for testing, bugs can also lurk in implementations' handling of ill-formed zones. Many DNS implementations use zone-file preprocessors to perform syntactic and semantic checks. For example, BIND uses `named-checkzone` [24], KNOT uses `kzonecheck` [18], and POWERDNS uses `pdnsutil` [20]. The implementations either reject an ill-formed zone or accept it but convert it to a valid one by ignoring certain records that cause it to be semantically ill-formed.

Many security vulnerabilities for software lie in the incorrect handling of unexpected inputs (e.g., in parsers [1]), and DNS software should be no different. Since our executable model includes a formulation of the validity conditions for zone files, we leverage Zen to systematically generate zone files that violate one of these conditions. For example, we ask Zen to generate a zone file in which all but the 7th condition in Table 2 is violated and the rest are satisfied.

If an invalid zone is rejected, then there is no issue, but if it is accepted, then there can be errors in how the zone is used for DNS lookups. To test for such errors we must also be able to generate queries for these zones. However, our formal model is only well defined for valid zone files so we cannot use it to generate queries. Instead, we use a technique from our prior work on zone-file verification [71] to partition queries into equivalence classes (ECs) relative to a given zone file. An equivalence class is a set of queries with the same resolution behavior, assuming a correct underlying DNS implementation, and the ECs are generated through a simple syntactic pass over a zone file. FERRET generates these ECs and then uses one representative query from each EC as a test. Though the number of ECs can vary widely, depending on the records in a zone file, in practice a zone containing four records will typically induce tens of ECs.

4 System Overview

FERRET is divided into several components, which are depicted in Figure 2. First it uses our Zen model described above to generate test inputs. Because domain names are encoded in

Zen using lists of integer labels (see § 3.4), FERRET includes a shim layer that translates the generated zone files and queries into meaningful domain names by mapping these labels to a collection of predefined strings (e.g., `com`). FERRET uses the equivalence-class (EC) generation algorithm of GROOT [71] to generate test queries for invalid zone files (§ 3.6).

FERRET uses Docker [83] to construct a working container image of each implementation. We cloned the implementations' code as of October 1st, 2020 [16, 21, 23, 25, 29, 33, 76, 102], from their open-source repositories on GitHub [84] and GitLab [100]. FERRET starts a container for each image, and each container serves one zone file at a time as an authoritative zone. FERRET uses a Python library `dnspython` [17] to construct queries and send them to each implementation's container. For each test case, the Python script prepares the container by stopping the running DNS nameserver, copying the new zone file and the necessary implementation-dependent configuration files to the container, and then restarting the DNS nameserver.

Finally, FERRET performs response grouping followed by fingerprinting to deduplicate errors that are likely to have the same root cause. For each test case, two DNS responses are considered equivalent, and hence in the same group, if they have the same response flags, return code, answer, and additional sections. FERRET only compares the authority section in two responses when their answer sections are empty. We do this because implementations are free to add additional records like a zone's `SOA` or `NS` records along with the requested records. We then fingerprint tests that result in more than one group and thereby represent a likely error. The fingerprint for a valid test is a tuple consisting of **(1)** the case in the formal model (the leaf label in the decision tree from Figure 5) as well as **(2)** the response groupings. An example fingerprint is $\langle$`R1`, $\{\{$NSD, KNOT, POWERDNS, YADIFA$\}$, $\{$BIND, COREDNS$\}$, $\{$TRUSTDNS, MARADNS$\}\}\rangle$. The fingerprint for an ill-formed test is similar but we use the validity condition being violated instead of the model case.

5 Results

5.1 Testing Using Valid Zone Files

Using FERRET, we generated thousands of tests and used them to compare the behavior of 8 popular open-source authoritative implementations of DNS. Table 3 shows the 8 implementations, the languages they are implemented in, and a brief description of their focus or how they are used. We constrained FERRET to generate tests where the length of each domain name and the number of records in the zone was at most 4. We ran FERRET on a 3.6GHz 72 core machine with 200 GB of RAM and it generated a total of 12,673 valid test cases, one per path in our Zen model that is consistent with the length constraints, in approximately 6 hours. Users can run the tests in parallel, so the runtime depends heavily on the user resources for parallelization. Each test takes around 10 seconds to run on average, and most of the time is spent setting up the zone file and necessary configuration files.

Implementation	**Language**	**Description**
BIND [23]	C	*de facto* standard
POWERDNS [21]	C++	popular in N. Europe
NSD [76]	C	hosts several TLDs
KNOT [25]	C	hosts several TLDs
COREDNS [16]	Go	used in Kubernetes
YADIFA [29]	C	created by EURid (.eu)
TRUSTDNS [33]	Rust	security, safety focused
MARADNS [102]	C	lightweight server

Table 3: The eight open-source DNS nameserver implementations tested by FERRET. FERRET can test implementations implemented in any language.

Model Case	**#Tests**	**#Tests Failing**	**#Fingerprints**
E1	3180	239	7
E2	12	10	5
E3	96	12	3
E4	6036	5312	11
W1	60	33	8
W2	24	21	9
W3	18	16	1
D1	230	65	4
R1	2980	2529	27
R2	37	3	1

Table 4: Test generation statistics for $n = 4$. The model case refers to the leaves in Figure 5. Even though the number of failed tests is higher, the number of fingerprints is small.

As described in § 4, FERRET runs each test against all 8 implementations and groups their responses. Out of 12,673 tests, FERRET found more than one group in the majority (8,240) of tests. Table 4 shows the number of tests generated for each case in the model (Figure 5), the number of tests where there was more than one group, and the number of unique fingerprints formed for each model case.

In total the 8,240 tests with more than one group were partitioned into 76 unique fingerprints, for a reduction of more than two orders of magnitude. For 24 of these fingerprints there exists only a single test case, while one fingerprint has 1892 corresponding tests. These 76 fingerprints can over-count the number of bugs since a single implementation issue can cause errors on multiple model paths. For example, YADIFA, TRUSTDNS, and MARADNS do not support `DNAME` records; so any generated test containing this feature will cause them to give the wrong answer or fail to respond. However, two tests can also have the same fingerprint despite different implementation root causes; so the number of fingerprints can also under-count the number of bugs.

For these reasons, we manually examined the test cases matching each fingerprint, examining them all when the fingerprint has 4 or fewer tests and otherwise examining a small random sample. By doing this we identified 24 unique bugs, as summarized in Table 6 (all except the ones marked with ✧). All of these have been confirmed as actual bugs (no false positives) and developers have fixed 14 of them at the time of writing.

5.2 Testing Using Invalid Zone Files

FERRET generated 900 ill-formed zone files, 100 violating each of the validity conditions in Table 2, in 2.5 hours. We used these zone files to test the four most widely used DNS implementations — BIND, NSD, KNOT, POWERDNS— as these have a mature zone-file preprocessor available.

There is no practical limit on the number of invalid zone files the tool can generate. We limited it to 100 for each violation in our experiments, but one could use FERRET to generate many more such tests if desired. Similarly, though we only explored violations of single well-formedness rules, it is straightforward to use FERRET to generate tests that violate a combination of rules. As a first step, FERRET checked all of the zone files with each implementation's preprocessor: `named-checkzone` [24] for BIND, `kzonecheck` [18] for KNOT, `nsd-checkzone` [77] for NSD, and `pdnsutil` [20] for POWERDNS. Each implementation can either reject or accept the invalid zone file and Table 5 shows the statistics of how different implementations treat the zone files.

All together there are 573 invalid zone files (the first five rows in the table) that are accepted by more than one DNS implementation and so are amenable to differential testing. Our formal model relies on zones to be well-formed: so we cannot use it to generate queries for these zones. Instead we leverage GROOT [71], which generates query equivalence classes (ECs) of the form $\langle$`example.com`$, t\rangle$ for a given zone file, one for each DNS record type t, and does not require the zone to be semantically well-formed. We used 7 query types: `A`, `NS`, `CNAME`, `DNAME`, `SOA`, `TXT`, `AAAA`. We excluded 19 zone files as GROOT generated over 200 ECs for each of them due to multiple interacting `DNAME` loops. For the remaining 554 zone files, the average number of ECs is 21*7 i.e., 21 domains names and each domain name is paired with the 7 types, and we chose one representative query from each EC.

The last column in Table 5 shows the results of differential testing. For example, 106 out of the 201 zone files in the first row exhibited differences among the three implementations during testing. We manually inspected all differences for the zone files that violated conditions of i, ii, iii, vi, and ix, as there were 12 or fewer such differences in each category, and we inspected a random sample for the others. By doing this we identified 6 new errors as shown in Table 6 with the ✧ symbol and all of them are fixed. Some of the errors identified earlier were also present here but are not double-counted.

BIND	NSD	KNOT	PDNS	#Zones	Condition violated	#Zones with a difference
A	A	A	R	100 + 100 + 1	i or viii or ix	11 + 94 + 1
A	A	R	R	100 + 61	vi or ix	8 + 3
A	R	A	R	17 + 100	ii or iii	1 + 6
A	R	R	A	60	vii	53
R	A	R	A	34	ix	7
A	R	R	R	39	vii	-
R	A	R	R	4	ix	-
R	R	R	A	95 + 1	v or vii	-
R	R	R	R	83 + 100 + 5	ii or iv or v	-

Table 5: Invalid zone file statistics. The second row shows that 100 (61) zone files that violate condition vi (ix) are accepted by only BIND and NSD, and 8 (3) of them resulted in some difference between the two implementations.

5.3 Example Bugs

We now provide a detailed description of some of the bugs from Table 6. Two of them were already described in § 2.2.

Bug #3: COREDNS Crash. FERRET generated the following test that causes COREDNS, the recommended nameserver for Kubernetes, to crash. It was subsequently confirmed and fixed by the COREDNS developers.

```
  example.   SOA     ...
 *.example.  CNAME   foo.example.
```

Query: $\langle$`baz.bar.example.`, `CNAME`$\rangle$

In this example the zone file has a wildcard `CNAME` record that rewrites any query ending with the label `example` to `foo.example`. This rewritten query will then match the wildcard record again and so on, causing COREDNS to loop and consume resources until, eventually, the server crashes with the following message:

```
runtime: goroutine stack exceeds 1000000000-byte limit
runtime: sp=0xc03c6c0378 stack=[0xc03c6c0000, ...]
fatal error: stack overflow
```

Interestingly, COREDNS correctly guards against `CNAME` loops that do not involve wildcard; so only a test that combines `CNAME` and wildcards will trigger the bug. After our bug report, the developers fixed the issue by adding a loop counter and breaking the loop if the depth exceeds nine. They commented: "Note the answer we're returning will be incomplete (more cnames to be followed) or illegal (wildcard cname with multiple identical records). For now it's more important to protect ourselves than to give the client a valid answer."

Crashes like this represent serious security vulnerabilities, particularly in multi-tenant settings such as the attack described earlier in Figure 4(a).

Bug #4: Wrong `RCODE` for synthesized `CNAME`. FERRET generated a zone that violates condition vii in Table 2:

Implementation	Bugs Found	Bug Type	Status
Bind	Sibling glue records not returned [47]	Wrong Additional	✓
	Zone origin glue records not returned [45]	Wrong Additional	✓
	`DNAME` recursion denial-of-service✧ [44]	Server Crash	☑
	Wrong `RCODE` for synthesized record✧ [46]	Wrong `RCODE`	☑
Nsd	`DNAME` not applied recursively [65]	Wrong Answer	☑
	Wrong `RCODE` when $*$ is in Rdata [64]	Wrong `RCODE`	☑
	Used `NS` records below delegation✧ [67]	Wrong Answer	☑
	Wrong `RCODE` for synthesized record✧ [66]	Wrong `RCODE`	☑
PowerDns	`CNAME` followed when not required [62]	Wrong Answer	✓
	`pdnsutil` check-zone `DNAME`-at-apex✧ [63]	Preprocessor Bug	☑
Knot	Incorrect record synthesis [58]	Wrong Answer	☑
	`DNAME` not applied recursively [61]	Wrong Answer	☑
	Used records below delegation [59]	Wrong Answer	☑
	Error in `DNAME`-`DNAME` loop Knot test [60]	Faulty Knot Test	☑
	Wrong `RCODE` for synthesized record✧ [91]	Wrong `RCODE`	☑
CoreDns	`NXDOMAIN` for existing domain [53]	Wrong `RCODE`	☑
	Wrong `RCODE` for `CNAME` target [55]	Wrong `RCODE`	☑
	Wildcard `CNAME` loops & `DNAME` loops [52]	Server Crash	☑
	Wrong `RCODE` for synthesized record [57]	Wrong `RCODE`	☑
	`CNAME` followed when not required [56]	Wrong Answer	☑
	Sibling glue records not returned [54]	Wrong Additional	✓
Yadifa	`CNAME` chains not followed [70]	Wrong Answer	☑
	Wrong `RCODE` for `CNAME` target [69]	Wrong `RCODE`	☑
	Used records below delegation [68]	Wrong Answer	☑
MaraDns†	AA flag set for zone cut `NS` RRs	Wrong Answer	✓
	Used records below delegation	Wrong Answer	✓
TRustDns†	Wildcard match only one label [49]	Wrong Answer	✓
	Used records below delegation [51]	Wrong Answer	✓
	AA flag set for zone cut `NS` RRs [50]	Wrong Flag	✓
	`CNAME` loops crash the server [48]	Server Crash	✓

Table 6: Summary of the bugs found by Ferret across the eight implementations. Status column represents whether the developers responded and acknowledged (✓) and also fixed (☑) to the filed bug report. The † symbol denotes implementations with unreported issues due to missing or unimplemented features. The ✧ symbol denotes the bugs found exclusively using testing with invalid zone files. We reported all the bugs Ferret identified to the respective developers before publishing this paper.

```
     test.com.   SOA     ...
 foo.test.com.   DNAME   bar.test.com.
cs.foo.test.com. AAAA    1:db8::2:1
```

Query: ⟨`www.foo.test.com.`, `CNAME`⟩

Bind and PowerDns accepted the zone file but Nsd and Knot did not. Ferret chose the above query as the representative from the query EC ⟨α`.foo.test.com.`, `CNAME`⟩ generated by GRoot, where α represents any sequence of labels that does not start with `cs`. Bind responded with:

```
"rcode NXDOMAIN",
";ANSWER",
"foo.test.com. 500 IN DNAME bar.test.com.",
"www.foo.test.com. 500 IN CNAME www.bar.test.com.",
```

The response from PowerDns was the same but with a `NOERROR` `RCODE`. The `RCODE` is important as resolvers can use QNAME minimization (RFC 7816 [6]) to wrongly conclude domain (non-)existence if an incorrect `RCODE` is returned. However, since the RFCs do not describe this subtle case, the intended behavior is unclear. Since the query is not relevant to the `AAAA` record, which violates the validity condition, to further investigate this issue we decided to remove that record and check the responses from Nsd and Knot. Both responded with the same response as Bind, leading us to (wrongly) conclude that the issue was with PowerDns.

To our surprise, after reporting the issue to PowerDns they responded: "The PowerDNS behavior looks correct to me. Are you sure BIND, NSD, and Knot all return NXDOMAIN on a CNAME query in this context?" Bind and Knot noticed the issue we filed on PowerDns's GitHub and fixed the bug almost immediately, even before we filed reports on their repositories. After some back and forth with the Nsd developers they concurred saying: "If you are right that the other implementations do this, then we can do that too; that makes less unexpected surprises in packet responses."

Max Length (n)	2	3	4	5
No. of Tests	52	618	12673	646K (51K tested)
Test generation time	10m	40m	6h	14d
No. of Tests Failing	12	224	8240	41173
No. of Fingerprints	9	22	76	115
No. of Bugs	4	14	24	27

Table 7: Results summary for different bounds.

Bug #5: POWERDNS pdnsutil bug. FERRET generated the following test case and POWERDNS returned an incorrect response, exposing a bug in its zone-file preprocessor.

```
     dept.com.  SOA     ...
     dept.com.  DNAME   dept.edu.
host.dept.com.  A       1.1.1.1
```

Query: ⟨host.dept.com., A⟩

The zone file is considered invalid as it violates condition vii in Table 2. nsd-checkzone and kzonecheck preprocessors reject the zone file but named-checkzone and pdnsutil do not raise any errors or warnings and accept the zone file. When queried for the A record, POWERDNS returned this record even though it should have used the DNAME record. POWERDNS has a long-standing open issue about handling DNAME occlusion (records below a DNAME, which should be ignored), and pdnsutil generally gives a warning but did not in this specific case. We filed a bug report for this test and the developers confirmed a bug in pdnsutil when the DNAME is at the apex of the zone. This is now fixed and pdnsutil gives a warning as in other occlusion cases.

5.4 Small-scope Property Validation

Finally, we performed an experiment to validate the small-scope property that justifies our approach — many interesting behaviors can be covered with small tests. We used FERRET to generate valid tests where the length of each domain name and the number of records in the zone were limited to n, for different values of n. Table 7 shows the results. For example, when $n = 2$ there are 52 feasible paths through the model. FERRET generated the corresponding 52 tests in 10 minutes, out of which 12 had more than one group, and these 12 fell into 9 fingerprints. By inspecting those failed tests, we identified 4 unique bugs, which are a subset of the ones identified by our evaluation described in § 5.1, where $n = 4$.

Our experiment identifies two distinct forms of small-scope property. First, the DNS query resolution protocol itself, as represented by our logical model, has a small-scope property. In particular, when $n = 2$ all leaf nodes in Figure 5 are covered by at least one test, except for the R1 leaf, and all leaf nodes are covered when n is 3 or higher. Hence, although we are restricted to generating small zones, we can still cover all return points in our formal model, each of which represents a distinct RFC behavior.

Second, the DNS nameserver implementations have a small-scope property. In part the fact that we have identified dozens of subtle new errors is evidence that small tests can explore interesting behaviors. The results in Table 7 add further evidence. As we increase the size of n from 2 to 3 to 4, the number of bugs identified goes from 4 to 14 to 24. In the $n = 5$ case, FERRET generated over 646K tests and took almost 14 days to finish. The distribution of tests across model cases is similar to the $n = 4$ breakdown shown in Table 4, where the majority of tests fall into the E1, E4 and R1 cases. We randomly sampled 50K tests to run from these three cases, according to their proportions. The other cases totalled to around 1000 tests, so we ran all of them. Out of the resulting 115 fingerprints, 50 fingerprints were in common with the fingerprints of $n = 4$. We therefore decided to examine the remaining 65 fingerprints to search for new bugs. For these 65 fingerprints, the median number of tests in each fingerprint was 3, and the mode was 1. We found three bugs that we did not find with $n = 4$, but all three bugs were covered by the tests for invalid zones with $n = 4$ (§ 5.2). In other words, increasing n from 4 to 5 has so far not uncovered any new errors in the DNS nameserver implementations.

6 Discussion

Our SCALE approach worked surprisingly well at identifying subtle errors in implementations. This was not obvious from the beginning, since each implementation can have very different control logic compared to one another and compared to our formal model. And yet seemingly the tests derived from paths through our formal RFC model frequently uncover bugs in rare control paths for these implementations.

On the other hand, this approach is not a panacea. We found situations where one path in the model corresponds to multiple paths in an implementation due to the internal data structures that it uses to represent different record types, which can lead to FERRET missing some issues. This showed up, for example, with empty non-terminals (ENTs) – domain names that own no resource records but have subdomains that do. Since there is no explicit branch that differentiates empty non-terminals in the model, FERRET did not generate test cases where the zone file had both an ENT and a query targeting that ENT. However, by manually testing a few such cases, we found two more bugs in COREDNS. Going forward it may be possible to extend FERRET to find more cases like this by adding additional non-semantic branches to the model to expose behavior thought to be error-prone.

More generally, we believe our SCALE approach to RFC compliance testing and "ferreting" out bugs through (i) symbolic execution of a small formal model to jointly generate configurations together with inputs, combined with (ii) differential testing, and (iii) fingerprinting, could be useful more broadly beyond the DNS. For instance, there are many other complex and distributed protocols used at different network layers such as routing protocols like BGP and OSPF, flow control protocols like PFC, new transport layer protocols such as QUIC, and many more. It would be interesting future work to apply the SCALE methodology beyond DNS.

7 Related Work

FERRET and SCALE are related to several lines of prior work in DNS and in automated testing.

Verified DNS implementations. One approach is to build, from scratch, a nameserver implementation verified to be correct. This approach has found some success in other domains, for example, in operating system microkernels [73] using proof assistants such as Coq [79]. Ironsides [12] is an implementation of a DNS resolver and authoritative nameserver that uses SPARK [3] to prove the absence of dataflow errors such as buffer overflows. While this work is promising, it does not formalize the DNS RFC semantics and thus cannot provide any functional correctness guarantees. Moreover, open source implementations such as BIND [23] are already used pervasively in the Internet. Providing a new verified implementation does not help these existing deployments.

Models for DNS. In our prior work on the GROOT zone-file verifier [71] we provided the first formalization of DNS semantics. However, it was a paper formalism and was only used to prove the correctness of the equivalence-class generation algorithm that forms the core of GROOT's approach to verifying zone files. Indeed, GROOT assumes that DNS implementations conform to the DNS RFCs. Our work is therefore complementary, but we used GROOT's logical model as a basis for our executable Zen model. We also leveraged GROOT's equivalence-class generation algorithm to create queries for invalid zone files.

Fuzz testing. Fuzz testing with semi-random and/or grammar-based tests has seen success in recent years [1, 5, 40, 78, 101]. However, as mentioned in § 1, fuzzing cannot easily be used in our setting due to the need to navigate complex constraints and dependencies, and hence existing fuzzers for DNS [10, 89, 99] are limited to testing DNS parsers and use a fixed zone file.

Symbolic execution. Symbolic execution [36, 37], which systematically solves for inputs that take different execution paths in a program, has also been successful [9, 11]. However, as described in § 1, due to the scale and complexity of DNS nameserver implementations, symbolic execution has been used only on individual functions and has avoided the need to generate zone files [93]. Our SCALE approach uses symbolic execution to drive test generation, but it does so on an executable model of the RFC behavior, which is significantly smaller and simpler than an implementation and has carefully chosen data representations that are amenable to symbolic execution. As a result, symbolic execution on our model is tractable and allows us to jointly generate (small) zone files and DNS queries that exercise interesting behaviors.

Model- and specification-based testing. In model-based testing (MBT) [8, 88, 90, 104] a user builds an abstract model of the system to test (e.g., a finite state machine [8, 104]). A tester implementation then generates paths through this abstract model and creates concrete tests by "filling in" missing information from the abstract example. Closest to our work are model-based testers for black-box network functions (e.g., [30, 98]), which also use symbolic execution to generate tests. However, they respectively use finite-state machine models [30] and a domain-specific language for specifying network function behavior [98], while we have implemented a full functional model of DNS in a general modeling language [4]. Further, their setting does not require generating configurations, which is the key technical challenge for testing protocols like DNS.

Specification-based testing leverages a user-provided specification of the valid inputs to a function. Most commonly, tests are generated by finding inputs that satisfy a given precondition [7]. Like SCALE these approaches typically rely on a small-scope hypothesis [43] and hence focus on generating small inputs. Recent work has developed an approach to automated testing for QUIC implementations [81, 82] that leverages a formal specification, but in a very different way than in our approach. Specifically, the specification models the party that is interacting with the implementation being tested and is used to generate valid responses.

Finally, recent works automatically learn protocol models from implementations [31] or RFCs [105]. We could potentially adopt these techniques in the future to reduce the burden of producing our formal model.

8 Conclusion

Despite its importance as the "phonebook" of the Internet, DNS is fraught with implementation bugs that can impact millions of users. In this paper, we introduced FERRET, the first automatic test generator for RFC compliance of DNS nameserver implementations. The SCALE approach underlying FERRET uses symbolic execution of a formal model to jointly generate configurations together with inputs. FERRET combines this technique with differential testing and fingerprinting to identify and automatically triage implementation errors. In total FERRET identified 30 new bugs, including at least two for each of the 8 implementations that we tested. We believe that this combination of techniques can generalize to "ferret" out subtle RFC-compliance bugs in large implementation code bases for other network protocols that use configurations.

Acknowledgements

We thank our shepherd Phillipa Gill and the anonymous reviewers for their insightful comments. We also thank the DNS developers and the DNS-OARC community for their feedback on the bug reports. This work was partially supported by NSF grants CNS-1704336 and CNS-1901510.

References

[1] American Fuzzing Lop (AFL). Afl 2018. https://lcamtuf.coredump.cx/afl/.

[2] Amazon. Route 53. https://aws.amazon.com/route53/.

[3] John Barnes. *Spark: The Proven Approach to High Integrity Software*. Altran Praxis, London, GBR, 2012.

[4] Ryan Beckett and Ratul Mahajan. A general framework for compositional network modeling. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, page 8–15, New York, NY, USA, 2020. Association for Computing Machinery.

[5] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. Coverage-based greybox fuzzing as markov chain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1032–1043, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Stéphane Bortzmeyer. DNS Query Name Minimisation to Improve Privacy. RFC 7816, March 2016.

[7] Chandrasekhar Boyapati, Sarfraz Khurshid, and Darko Marinov. Korat: Automated testing based on java predicates. In *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '02, page 123–133, New York, NY, USA, 2002. Association for Computing Machinery.

[8] Josip Bozic, Lina Marsso, Radu Mateescu, and Franz Wotawa. A formal tls handshake model in lnt. In John P. Gallagher, Rob van Glabbeek, and Wendelin Serwe, editors, Proceedings Third Workshop on *Models for Formal Analysis of Real Systems* and Sixth International Workshop on *Verification and Program Transformation,* Thessaloniki, Greece, 20th April 2018, volume 268 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–40, Greece, 2018. Open Publishing Association.

[9] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, page 209–224, USA, 2008. USENIX Association.

[10] Frederic Cambus. Fuzzing dns zone parsers. https://www.cambus.net/fuzzing-dns-zone-parsers/.

[11] Marco Canini, Vojin Jovanović, Daniele Venzano, Dejan Novaković, and Dejan Kostić. Online testing of federated and heterogeneous distributed systems. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 434–435, New York, NY, USA, 2011. Association for Computing Machinery.

[12] M. Carlisle and B. Fagin. Ironsides: Dns with no single-packet denial of service or remote code execution vulnerabilities. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 839–844, Anaheim, CA, USA, 2012. IEE.

[13] Peng Chen and Hao Chen. Angora: Efficient fuzzing by principled search. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 711–725, 2018.

[14] Taolue Chen, Yan Chen, Matthew Hague, Anthony W. Lin, and Zhilin Wu. What is decidable about string constraints with the replaceall function. *Proc. ACM Program. Lang.*, 2(POPL), December 2017.

[15] Bind Community. Bind gitlab issues. https://gitlab.isc.org/isc-projects/bind9/-/issues.

[16] CoreDNS community. Coredns. https://coredns.io/. Code commit used: https://github.com/coredns/coredns/tree/6edc8fe7f6c2f57844c8ee7f7f5deef71085ebe8.

[17] Dnspython Community. Dnspython. https://dnspython.readthedocs.io/en/latest/index.html.

[18] Knot community. kzonecheck – knot dns zone file checking tool. https://www.knot-dns.cz/docs/2.5/html/man_kzonecheck.html.

[19] NSD Community. Nsd github issues. https://github.com/NLnetLabs/nsd/issues.

[20] PowerDNS community. Pdnsutil. https://doc.powerdns.com/authoritative/manpages/pdnsutil.1.html.

[21] PowerDNS Community. Powerdns. https://www.powerdns.com/. Code commit used: https://github.com/PowerDNS/pdns/tree/a03aaad7554483ee6efe72a81eda00a9d1a94fe5.

[22] PowerDNS Community. Powerdns github issues. https://github.com/PowerDNS/pdns/issues?q=is%3Aissue+is%3Aopen+label%3Aauth.

[23] Internet Systems Consortium. Bind 9. https://www.isc.org/bind/. Code commit used: https://gitlab.isc.org/isc-projects/bind9/-/tree/dbcf683c1a57f49876e329fca183cb39d20ca3a4.

[24] Internet Systems Consortium. named-checkzone(8). https://linux.die.net/man/8/named-checkzone.

[25] CZ.NIC. Knot. https://www.knot-dns.cz/. Code commit used: https://gitlab.nic.cz/knot/knot-dns/-/tree/563fcdd886b5d5c52bceeb8fda3c4bda59ece73e.

[26] National Vulnerability Database. CVE-2021-25215 Detail. https://nvd.nist.gov/vuln/detail/CVE-2021-25215.

[27] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, page 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[28] Robert Elz and Randy Bush. Clarifications to the DNS Specification. RFC 2181, July 1997.

[29] EURid.eu. Yadifa. https://www.yadifa.eu/. Code commit used: https://github.com/yadifa/yadifa/tree/dc5bed2fb8ec204af9b65eeb91934c2c85098cbb.

[30] Seyed K. Fayaz, Tianlong Yu, Yoshiaki Tobioka, Sagar Chaki, and Vyas Sekar. BUZZ: Testing Context-Dependent policies in stateful networks. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 275–289, Santa Clara, CA, March 2016. USENIX Association.

[31] Tiago Ferreira, Harrison Brewton, Loris D'Antoni, and Alexandra Silva. Prognosis: Closed-box analysis of network protocol implementations. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 762–774, New York, NY, USA, 2021. Association for Computing Machinery.

[32] Jonathan Foote. How to fuzz a server with american fuzzy lop. https://www.fastly.com/blog/how-fuzz-server-american-fuzzy-lop, 2015.

[33] Benjamin Fry and Community. Trust-dns. http://trust-dns.org/. Code commit used: https://github.com/bluejekyll/trust-dns/tree/7d9b186121fb5cb331cf2ec6baa47846b83de8fc.

[34] James Fryman. Dns outage post mortem. https://github.blog/2014-01-18-dns-outage-post-mortem/, 2014.

[35] Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin Rinard. Word equations with length constraints: What's decidable? In *Proceedings of the 8th International Conference on Hardware and Software: Verification and Testing*, HVC'12, page 209–226, Berlin, Heidelberg, 2012. Springer-Verlag.

[36] Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. Grammar-based whitebox fuzzing. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, page 206–215, New York, NY, USA, 2008. Association for Computing Machinery.

[37] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: Directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, page 213–223, New York, NY, USA, 2005. Association for Computing Machinery.

[38] Suzanne Goldlust, Michał Kępień, Peter Davies, and Everett Fulton. CVE-2021-25215: An assertion check can fail while answering queries for DNAME records that require the DNAME to be processed to resolve itself. https://kb.isc.org/v1/docs/cve-2021-25215.

[39] Google. Cloud dns. https://cloud.google.com/dns.

[40] Sam Hocevar. zzuf: multi-purpose fuzzer. http://caca.zoy.org/wiki/zzuf/., 2007.

[41] Paul E. Hoffman, Andrew Sullivan, and Kazunori Fujiwara. DNS Terminology. RFC 8499, January 2019.

[42] Dyn Inc. Dynamic dns. https://account.dyn.com/.

[43] Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, April 2002.

[44] Siva Kakarla, Mark Andrews, Michał Kępień, Peter Davies, and Michal Nowak. [CVE-2021-25215] An assertion check can fail while answering queries for DNAME records that require the DNAME to be processed to resolve itself. https://gitlab.isc.org/isc-projects/bind9/-/issues/2540.

[45] Siva Kesava R Kakarla and Mark Andrews. Glue records can be returned when the name server's name is same as the zone origin. https://gitlab.isc.org/isc-projects/bind9/-/issues/2385.

[46] Siva Kesava R Kakarla, Mark Andrews, and Michał Kępień. DNAME: synthetized CNAME might be perfect answer to CNAME query. https://gitlab.isc.org/isc-projects/bind9/-/issues/2284.

[47] Siva Kesava R Kakarla, Mark Andrews, and Michał Kępień. Sibling (In-bailiwick rule of RFC 8499) domain IP records not returned. https://gitlab.isc.org/isc-projects/bind9/-/issues/2384.

[48] Siva Kesava R Kakarla and Benjamin Fry. CNAME loops throws off the server. https://github.com/bluejekyll/trust-dns/issues/1283.

[49] Siva Kesava R Kakarla and Benjamin Fry. Wildcards match only one label. https://github.com/bluejekyll/trust-dns/issues/1342.

[50] Siva Kesava R Kakarla and Benjamin Fry. Zone cut NS RRs returned as authoritative records. https://github.com/bluejekyll/trust-dns/issues/1273.

[51] Siva Kesava R Kakarla, Benjamin Fry, and Jonas Bushart. Glue records returned as authoritative records by the server . https://github.com/bluejekyll/trust-dns/issues/1272.

[52] Siva Kesava R Kakarla and Miek Gieben. Handling wildcard CNAME loops. https://github.com/coredns/coredns/issues/4378.

[53] Siva Kesava R Kakarla and Miek Gieben. NXDOMAIN returned when the domain exists. https://github.com/coredns/coredns/issues/4374.

[54] Siva Kesava R Kakarla and Miek Gieben. Sibling (In-bailiwick rule of RFC 8499) domain IP records can also be returned along with NS records. https://github.com/coredns/coredns/issues/4377.

[55] Siva Kesava R Kakarla and Chris O'Haver. Non-existent CNAME target in the same zone should be returned with NXDOMAIN instead of NOERROR rcode. https://github.com/coredns/coredns/issues/4288.

[56] Siva Kesava R Kakarla, Chris O'Haver, and Kohei Yoshida. CNAME need not be followed after a synthesized CNAME for a CNAME query. https://github.com/coredns/coredns/issues/4398.

[57] Siva Kesava R Kakarla, Chris O'Haver, and Kohei Yoshida. Return code for synthesized CNAME records (from wildcards and DNAMEs). https://github.com/coredns/coredns/issues/4341.

[58] Siva Kesava R Kakarla, Libor Peltan, and Daniel Salzman. Record incorrectly synthesized from wildcard record. https://gitlab.nic.cz/knot/knot-dns/-/issues/715.

[59] Siva Kesava R Kakarla, Libor Peltan, and Daniel Salzman. Records below delegation are not ignored (kzonecheck also does not raise any issue). https://gitlab.nic.cz/knot/knot-dns/-/issues/713.

[60] Siva Kesava R Kakarla, Libor Peltan, Daniel Salzman, and mscbg. DNAME-DNAME loop test case is not a loop. https://gitlab.nic.cz/knot/knot-dns/-/issues/703.

[61] Siva Kesava R Kakarla, Libor Peltan, Daniel Salzman, and Vladimír Čunát. DNAME not applied more than once to resolve the query. https://gitlab.nic.cz/knot/knot-dns/-/issues/714.

[62] Siva Kesava R Kakarla and Peter van Dijk. CNAME need not be followed after a synthesized CNAME for a CNAME query. https://github.com/PowerDNS/pdns/issues/9886.

[63] Siva Kesava R Kakarla and Peter van Dijk. pdnsutil DNAME checks have issues. https://github.com/PowerDNS/pdns/issues/9734.

[64] Siva Kesava R Kakarla and Wouter Wijngaards. '*' in Rdata causes the return code to be NOERROR instead of NX. https://github.com/NLnetLabs/nsd/issues/152.

[65] Siva Kesava R Kakarla and Wouter Wijngaards. DNAME not applied more than once to resolve the query.
https://github.com/NLnetLabs/nsd/issues/151.

[66] Siva Kesava R Kakarla and Wouter Wijngaards. DNAME: synthesized CNAME might be perfect answer to CNAME query.
https://github.com/NLnetLabs/nsd/issues/140.

[67] Siva Kesava R Kakarla and Wouter Wijngaards. NS Records below delegation are not ignored (nsd-checkzone also does not raise any issue).
https://github.com/NLnetLabs/nsd/issues/174.

[68] Siva Kesava R Kakarla and yadifa. Records below delegation are not ignored.
https://github.com/yadifa/yadifa/issues/12.

[69] Siva Kesava R Kakarla, yadifa, and edfeu. Non-existent CNAME target in the same zone should be returned with NXDOMAIN instead of NOERROR.
https://github.com/yadifa/yadifa/issues/11.

[70] Siva Kesava R Kakarla, yadifa, and edfeu. Why are CNAME chains not followed?
https://github.com/yadifa/yadifa/issues/10.

[71] Siva Kesava Reddy Kakarla, Ryan Beckett, Behnaz Arzani, Todd Millstein, and George Varghese. Groot: Proactive verification of dns configurations. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 310–328, New York, NY, USA, 2020. Association for Computing Machinery.

[72] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, July 1976.

[73] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. Sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 207–220, New York, NY, USA, 2009. Association for Computing Machinery.

[74] Eduard Kovacs. Dns servers crash due to bind security flaw.
https://www.securityweek.com/dns-servers-crash-due-bind-security-flaw, 2018.

[75] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going wild: Large-scale classification of open dns resolvers. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, page 355–368, New York, NY, USA, 2015. Association for Computing Machinery.

[76] NLnet Labs. Nsd.
https://nlnetlabs.nl/projects/nsd/about/. Code commit used: https://github.com/NLnetLabs/nsd/tree/4043a5ab7be7abaec969011e48e4d0d60a0056a6.

[77] NLnet Labs. nsd-checkzone - nsd zone file syntax checker.
https://www.nlnetlabs.nl/documentation/nsd/nsd-checkzone/.

[78] Hyojeong Lee, Jeff Seibert, Dylan Fistrovic, Charles Killian, and Cristina Nita-Rotaru. Gatling: Automatic performance attack discovery in large-scale distributed systems. *ACM Trans. Inf. Syst. Secur.*, 17(4), April 2015.

[79] Pierre Letouzey. *Programmation fonctionnelle certifiée: l'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris Sud, 2004.

[80] Edward P. Lewis. The Role of Wildcards in the Domain Name System. RFC 4592, July 2006.

[81] Kenneth L. McMillan and Lenore D. Zuck. Compositional testing of internet protocols. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 161–174, 2019.

[82] Kenneth L. McMillan and Lenore D. Zuck. Formal specification and testing of quic. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 227–240, New York, NY, USA, 2019. Association for Computing Machinery.

[83] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239):2, March 2014.

[84] Microsoft. Github, inc.
https://github.com/.

[85] Microsoft. Microsoft dns.
https://en.wikipedia.org/wiki/Microsoft_DNS.

[86] P. Mockapetris. Domain names - concepts and facilities. RFC 1034, November 1987.

[87] Paul Mockapetris. Domain names - implementation and specification. RFC 1035, November 1987.

[88] B. Neelakantan and S. V. Raghavan. *Protocol Conformance Testing — A Survey*, pages 175–191. Springer US, Boston, MA, 1995.

[89] NMAP Organization. Dns-fuzz. https://nmap.org/nsedoc/scripts/dns-fuzz.html.

[90] Javier Paris and Thomas Arts. Automatic testing of tcp/ip implementations using quickcheck. In *Proceedings of the 8th ACM SIGPLAN Workshop on ERLANG*, ERLANG '09, page 83–92, New York, NY, USA, 2009. Association for Computing Machinery.

[91] Libor Peltan and Daniel Salzman. DNAME: synthesized CNAME might be perfect answer to CNAME query. https://gitlab.nic.cz/knot/knot-dns/-/merge_requests/1217.

[92] Libor Peltans. Nsd and knot discussion. https://github.com/NLnetLabs/nsd/issues/142#issuecomment-732753256.

[93] David A. Ramos and Dawson Engler. Under-constrained symbolic execution: Correctness checking for real code. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 49–64, Washington, D.C., August 2015. USENIX Association.

[94] Fahmida Y. Rashid. Isc updates critical dos bug in bind dns software. https://www.infoworld.com/article/3126472/isc-updates-critical-dos-bug-in-bind-dns-software.html, 2016.

[95] Andrew Reynolds, Jasmin Christian Blanchette, Simon Cruanes, and Cesare Tinelli. Model finding for recursive functions in smt. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning*, pages 133–151, Cham, 2016. Springer International Publishing.

[96] Scott Rose and Wouter Wijngaards. DNAME Redirection in the DNS. RFC 6672, June 2012.

[97] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K. Sitaraman. Akamai dns: Providing authoritative answers to the world's queries. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 465–478, New York, NY, USA, 2020. Association for Computing Machinery.

[98] Harsha Sharma, Wenfei Wu, and Bangwen Deng. Symbolic execution for network functions with time-driven logic. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–8, 2020.

[99] Robert Swiecki and *et al.* Honggfuzz - security oriented software fuzzer. https://github.com/google/honggfuzz/tree/master/examples/bind.

[100] Dmitriy Zaporozhets Sytse "Sid" Sijbrandij. Gitlab, inc. https://gitlab.com/.

[101] Peach Tech. Peach fuzzer platform. peach.tech/products/peach-fuzzer/peach.tech/products/peach-fuzzer/.

[102] Sam Trenholme. Maradns. https://maradns.samiam.org/. Code commit used: https://github.com/samboy/MaraDNS/tree/3ec477f227b2bf6947be8fbe8fd0ab73130227d0.

[103] Liam Tung. Azure global outage: Our dns update mangled domain records, says microsoft. https://www.zdnet.com/article/azure-global-outage-our-dns-update-mangled-domain-records-says-microsoft/, 2019.

[104] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, and Lev Nachmanson. *Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer*, pages 39–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[105] Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. Semi-automated protocol disambiguation and code generation. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 272–286, New York, NY, USA, 2021. Association for Computing Machinery.

[106] Dan York. Hbo now dnssec misconfiguration makes site unavailable from comcast networks (fixed now). https://www.internetsociety.org/blog/2015/03/hbo-now-dnssec-misconfiguration-makes-site-unavailable-from-comcast-networks-fixed-now/.