SCALE
Automatically Finding RFC Compliance Bugs in DNS Nameservers

Siva Kesava Reddy Kakarla
Ryan Beckett
Todd Millstein
George Varghese

Microsoft
University of California, Los Angeles
Intentionet
Website Domain Name → IP

DNS (Domain Name System)
Website Domain Name → IP

DNS (Domain Name System)

Siva Kakarla

SCALE: Automatically Finding RFC Compliance Bugs in DNS Nameservers
Many DNS Implementations

Open-Source
- BIND
- POWERDNS
- NSD
- CoreDNS
- KNOTDNS

Closed-Source
- Akamai
- Cloudflare
- Route53
DNS Software needs to be absolutely Correct!

- Incorrect responses from DNS servers can cause service unavailability
- Attackers can exploit security vulnerabilities (code bugs) to mount DDoS attacks
DNS Software needs to be absolutely Correct!

- Incorrect responses from DNS servers can cause service unavailability
- Attackers can exploit security vulnerabilities (code bugs) to mount DDoS attacks
- DNS outages have a “large blast radius”
DNS Software needs to be absolutely Correct!

- Incorrect responses from DNS servers can cause service unavailability
- Attackers can exploit security vulnerabilities (code bugs) to mount DDoS attacks
- DNS outages have a “large blast radius”

Bind DoS Bug

ISC updates critical DoS bug in BIND DNS software
The denial-of-service flaw in BIND can be triggered by specially crafted DNS packages and is capable of knocking critical servers offline
DNS Software needs to be absolutely Correct!

- Incorrect responses from DNS servers can cause service unavailability
- Attackers can exploit security vulnerabilities (code bugs) to mount DDoS attacks
- DNS outages have a “large blast radius”

Bind DoS Bug

**ISC updates critical DoS bug in BIND DNS software**

The denial-of-service flaw in BIND can be triggered by specially crafted DNS packages and is capable of knocking critical servers offline

Slack Outage due to Route 53 bug

**Slack is down for some people, and of course, the problem is DNS**

*If you’ve been having trouble contacting co-workers, this may be why*

By Mitchell Clark | Updated Sep 30, 2021, 4:10pm EDT
How the Domain Name System Works

Query microsoft.com
How the Domain Name System Works

Query microsoft.com

1. User
2. DNS resolver
How the Domain Name System Works

Query microsoft.com

1. User queries DNS resolver
2. DNS resolver queries DNS root nameservers
3. DNS root nameservers return zone file

microsoft.com
How the Domain Name System Works

1. Query: microsoft.com

2. DNS resolver

3. microsoft.com

DNS root nameservers

Zone file (configuration)
How the Domain Name System Works

1. Query microsoft.com
2. Go to nameserver for com.
3. microsoft.com
How the Domain Name System Works

1. Query microsoft.com
2. Go to nameserver for com.
3. microsoft.com
4. Go to nameserver for microsoft
5. microsoft.com

20.70.246.20
How the Domain Name System Works

Query: microsoft.com

1. User queries DNS resolver
2. DNS resolver goes to nameserver for com.
4. Go to nameserver for microsoft
5. Nameserver for microsoft.com
6. Resolve to 20.70.246.20
How the Domain Name System Works

1. Query microsoft.com

2. Go to nameserver for com.
   20.70.246.20

3. microsoft.com
   Go to nameserver for microsoft
   20.70.246.20

4. microsoft.com

5. microsoft.com

6. Microsoft Web Server
DNS is way more complex than people think!
DNS is way more complex than people think!

Nondeterminism in which nameserver to ask next
DNS is way more complex than people think!

Nondeterminism in which nameserver to ask next

Complex record types each with unique semantics
- DNAME records: domain (partial) rewrite
- CNAME records: alias another domain name
- Wildcard records: match anything not otherwise matched
- NS records: nameserver redirection
- 56 other records types across ~30 RFCs
DNS is way more complex than people think!

**Nondeterminism** in which nameserver to ask next

**Complex record types** each with unique semantics
- DNAME records: domain (partial) rewrite
- CNAME records: alias another domain name
- Wildcard records: match anything not otherwise matched
- NS records: nameserver redirection
- 56 other records types across ~30 RFCs

The DNS is a lot like chess; it’s a simple game in terms of the rules, but phenomenally complex in the way it can be played.

Geoff Huston (APNIC)
Our Goal

Automatically generate test cases for DNS nameserver implementations covering as many RFC (specification) behaviors as possible.
Our Goal

Automatically generate test cases for DNS nameserver implementations covering as many RFC (specification) behaviors as possible.

Challenge – Need to generate config (zone file) and input (query) jointly.
Previously Unknown BIND Crash Bug

Tool Generated Test Case

1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

⟨foo.attack.foo.attack.com,,DNAME⟩  2. Query
Previously Unknown BIND Crash Bug

**Tool Generated Test Case**

1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

(foo.attack.foo.attack.com.,DNAME)

2. Query

BIND Server

(foo.attack.foo.
attack.com.,DNAME)
Previously Unknown BIND Crash Bug

Tool Generated Test Case

**Existence of DNAME record**

1. **Zone file**

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

〈foo.attack.foo.attack.com,,DNAME〉

2. **Query**
Previously Unknown BIND Crash Bug

Tool Generated Test Case

Existence of DNAME record

1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

2. Query

 ⟨foo.attack.foo.attack.com.,DNAME⟩

Rewrite to the parent domain

SCALE: Automatically Finding RFC Compliance Bugs in DNS Nameservers | 26
Previously Unknown BIND Crash Bug

Tool Generated Test Case

1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
<tr>
<td>foo.attack.com</td>
<td>DNAME</td>
<td>com.</td>
</tr>
</tbody>
</table>

2. Query

(\texttt{foo.attack.foo.attack.com,,DNAME})

3. Rewrite to the parent domain

Existence of DNAME record

Exactly this type

Siva Kakarla
Previously Unknown BIND Crash Bug

Tool Generated Test Case

1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com</td>
<td>SOA</td>
<td>ns1.exm.</td>
</tr>
</tbody>
</table>

2. Query

(foo.attack.foo.attack.com., DNAME)

4. "foo.attack." part must repeat at least twice, and it must end with "com."

3. Exactly this type
Previously Unknown BIND Crash Bug

**Tool Generated Test Case**

### 1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com</td>
<td>SOA</td>
<td>ns1.exm.</td>
</tr>
<tr>
<td>foo.attack.com</td>
<td>DNAME</td>
<td>com.</td>
</tr>
</tbody>
</table>

**Existence of DNAME record**

1. 

**Rewrite to the parent domain**

2. 

**2. Query**

(\{foo.attack.foo.attack.com.,DNAME\})

“foo.attack.” part must repeat at least twice, and it must end with “com.”

Exactly this type

- Server crashes due to an assertion failure
- Easily-weaponizable denial-of-service vector
- Remotely Exploitable

---

Scale: Automatically Finding RFC Compliance Bugs in DNS Nameservers

Siva Kakarla
**BIND Crash Remote Exploitation**

**Scenario 1: Attack on a DNS hosting service that uses BIND**

**Attacker**
BIND Crash Remote Exploitation

Scenario 1: Attack on a DNS hosting service that uses BIND

1. Host attack.com zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>
**Scenario 1: Attack on a DNS hosting service that uses BIND**

1. Host attack.com zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

**DNS Hosting Service**

- E.g., Dyn, Infoblox

**Authoritative Server**

- Instance 1
- Instance 2
- Instance 3
Scenario 1: Attack on a DNS hosting service that uses BIND

**1. Host attack.com zone file**

**2. Query for:** \(\langle\text{foo.attack.foo.attack.com.}, \text{DNAME}\rangle\)

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

**Attacker**

**DNS Hosting Service**
- E.g., Dyn, Infoblox
- Authoritative Server Instance 1
- Authoritative Server Instance 2
- Authoritative Server Instance 3
**BIND Crash Remote Exploitation**

**Scenario 1: Attack on a DNS hosting service that uses BIND**

1. **Host attack.com zone file**
2. **Query for:** 
   ⟨foo.attack.foo.attack.com.,DNAME⟩

**Domain Name** | **Type** | **Data**
--- | --- | ---
attack.com. | SOA | ns1.exm. ...

**Attack**

Crashes and takes down other customer zone files – Remote DoS Attack

**Authoritative Server Instance 1**

**DNS Hosting Service**
E.g., Dyn, Infoblox

**Authoritative Server Instance 2**

**Authoritative Server Instance 3**
BIND Crash Remote Exploitation

Scenario 2: Attack on a public BIND DNS Resolver

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

Authoritative nameserver for attack.com. (under attacker control)
**BIND Crash Remote Exploitation**

**Scenario 2: Attack on a public BIND DNS Resolver**

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

$q_1: \langle$foo.attack.com., DNAME$\rangle$

Public BIND resolver

Attacker

Authoritative nameserver for attack.com. (under attacker control)
**Scenario 2: Attack on a public BIND DNS Resolver**

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

1. $q_1: \{\text{foo.attack.com.}, \text{DNAME}\}$

2. Authoritative nameserver for attack.com. (under attacker control)
BIND Crash Remote Exploitation

Scenario 2: Attack on a public BIND DNS Resolver

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

Public BIND resolver

$q_1$: (foo.attack.com., DNAME)

$q_1$ to DNAME record

Authoritative nameserver for attack.com. (under attacker control)
**BIND Crash Remote Exploitation**

**Scenario 2: Attack on a public BIND DNS Resolver**

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

1. **Attacker**
   - $q_1: \langle \text{foo.attack.com.}, \text{DNAME} \rangle$

2. **Public BIND resolver**
   - Query $q_1$

3. **Authoritative nameserver for attack.com. (under attacker control)**
   - DNAME record

4. **cache record**
**BIND Crash Remote Exploitation**

Scenario 2: Attack on a public BIND DNS Resolver

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

Public BIND resolver

$q_1: \langle \text{foo.attack.com.}, \text{DNAME} \rangle$

1. **Attacker** sends a DNS query $q_1$ for a DNAME record.
2. The public BIND resolver responds with a DNAME record.
3. The authoritative nameserver for attack.com. (under attacker control)
4. The cache record is updated.
5. The DNAME record is resolved.
BIND Crash Remote Exploitation

Scenario 2: Attack on a public BIND DNS Resolver

Public BIND resolver

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

$q_1: \langle \text{foo.attack.com.}, \text{DNAME} \rangle$

$q_2: \langle \text{foo.attack.foo.attack.com.}, \text{DNAME} \rangle$

$q_1$:

1. Attacker
2. DNAME record
3. Authoritative nameserver for attack.com. (under attacker control)
4. cache record
5. DNAME record
6. DNAME record

$q_2$:
### BIND Crash Remote Exploitation

#### Scenario 2: Attack on a public BIND DNS Resolver

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

- **Resolver Crashes**
  - Bind accounts for over half of all DNS resolvers in use
  - Simple DDoS against numerous ISPs and public resolvers

---

**Public BIND resolver**

1. $q_1: \{\text{foo.attack.com.}, \text{DNAME}\}$
2. $q_1$
3. DNAME record
4. cache record
5. DNAME record
6. $q_2: \{\text{foo.attack.foo.attack.com.}, \text{DNAME}\}$

---

Siva Kakarla

SCALE: Automatically Finding RFC Compliance Bugs in DNS Nameservers
BIND Crash Remote Exploitation

Scenario 1: Attack on a DNS hosting service that uses BIND

Scenario 2: Attack on a public BIND DNS Resolver
**BIND Crash Disclosure**

**Scenario 1: Attack on a DNS hosting service that uses BIND**
- Initiated a responsible disclosure with BIND

**Scenario 2: Attack on a public BIND DNS Resolver**
Scenario 1: Attack on a DNS hosting service that uses BIND

Scenario 2: Attack on a public BIND DNS Resolver

Initiated a responsible disclosure with BIND

CVE: CVE-2021-25215
Document version: 2.0
Posting date: 28 April 2021
Program impacted: BIND
Versions affected: BIND 9.0.0 -> 9.11.29
BIND Supported Preview Edition, as well
Severity: High
Exploitable: Remotely
Description:
DNAME records, described in RFC 6672.
**BIND Crash Disclosure**

**Scenario 1: Attack on a DNS hosting service that uses BIND**

Initiated a responsible disclosure with BIND

**Scenario 2: Attack on a public BIND DNS Resolver**

Affected all maintained BIND versions affecting NetApp, Ubuntu, Infoblox, and Red Hat.

---

**CVE:** [CVE-2021-25215](#)

**Document version:** 2.0

**Posting date:** 28 April 2021

**Program impacted:** BIND

**Versions affected:** BIND 9.0.0 -> 9.11.29
BIND Supported Preview Edition, as well

**Severity:** High

**Exploitable:** Remotely

**Description:**
DNAME records, described in [RFC 6672](#).
Previously Unknown BIND Crash Bug

Tool Generated Test Case

1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

(foo.attack/foo.attack.com.,DNAME)

2. Query

BIND Server

(foo.attack.foo.attack.com.,DNAME)
Previously Unknown BIND Crash Bug

Tool Generated Test Case

1. Zone file

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack.com.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
</tbody>
</table>

(foo.attack.foo.attack.com.,DNAME)

2. Query

(foo.attack.foo.
attack.com.,DNAME)

BIND Server

Joint auto generation of query and zone file is required
Standard Automated Testers are Insufficient

Fuzz testing for DNS Implementations

- Scalable to large codebases
- Can't navigate complex semantic requirements and dependencies to generate zone files
- Generates queries only to check zone file parsers
- No coverage guarantees

DNS Nameserver Implementation (BIND)
Standard Automated Testers are Insufficient

Fuzz testing for DNS Implementations
- Scalable to large codebases
- Can't navigate complex semantic requirements and dependencies to generate zone files
- Generates queries only to check zone file parsers
- No coverage guarantees

Symbolic execution for DNS Implementations
- Solves for path input conditions
- Path explosion and difficulty with complex data structures
- Explores a subset of implementation paths
- Coverage guarantees in theory
Standard Automated Testers are Insufficient

**Fuzz testing for DNS Implementations**
- Scalable to large codebases
- Can't navigate complex semantic requirements and dependencies to generate zone files
- Generates queries only to check zone file parsers
- No coverage guarantees

**Symbolic execution for DNS Implementations**
- Solves for path input conditions
- Path explosion and difficulty with complex data structures
- Explores a subset of implementation paths
- Coverage guarantees in theory

Current automated testers for DNS do not generate zone files and hence do not find RFC violations

DNS Nameserver Implementation (BIND)
Our Approach

- Small-scope
- Constraint-driven
- Automated
- Logical
- Execution
Our Approach

S - Small-scope
C - Constraint-driven
A - Automated
L - Logical
E - Execution

- Jointly generates zone files & queries
- Covers many different RFC behaviors
- Applicable to black-box implementations
Our Approach

- Small-scope
- Constraint-driven
- Automated
- Logical
- Execution

- **Jointly** generates zone files & queries
- Covers many *different* RFC behaviors
- Applicable to *black-box* implementations

**Specification of DNS RFCs**
1034, 4592, 6672, ...
Our Insight

- S: Small-scope
- C: Constraint-driven
- A: Automated
- L: Logical
- E: Execution

Use DNS formal model to guide test generation
Our Insight

S
Small-scope

C
Constraint-driven

A
Automated

L
Logical

E
Execution

Use DNS formal model to guide test generation

DNS logical model from RFCs

DNS Nameserver Implementation (BIND)

config

$q_1, q_2, q_3, \ldots$

inputs

bug!
Our Insight

S  Small-scope
C  Constraint-driven
A  Automated
L  Logical
E  Execution

Use DNS formal model to guide test generation

High RFC behavior coverage – Tests cover all return points (different RFC scenarios) in the logical model

DNS logical model from RFCs

DNS Nameserver Implementation (BIND)

Use DNS formal model to guide test generation

High RFC behavior coverage – Tests cover all return points (different RFC scenarios) in the logical model

DNS logical model from RFCs

DNS Nameserver Implementation (BIND)
Ferret: Tool based on SCALE for DNS

DNS RFCs
1034, 4592, 6672, ...

Test Generation Module

Tests
FERRET: Tool based on SCALE for DNS

Test Generation Module

DNS RFCs
1034, 4592, 6672, ...

Tests

1. BIND
2. NSD
3. KNOT
4. PDNS
5. ...

Siva Kakarla
FERRET: Tool based on SCALE for DNS

Test Generation Module

DNS RFCs 1034, 4592, 6672, ...

Tests

1. BIND
2. NSD
3. KNOT
4. PDNS
5. ...

Response Grouping

Single group

> 1 group

?
Test Generation Module

Formal Model

RFCs 1034, 4592, 6672, ...

Declarative (Mathematical) specification of the nameserver logic

† GROOT: Proactive Verification of DNS Configurations – Siva Kakarla et al., SIGCOMM 2020
Test Generation Module

Formal Model

Constraints

<table>
<thead>
<tr>
<th>Query</th>
<th>Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECTBESTRECORDS</td>
<td></td>
</tr>
</tbody>
</table>

1. SELECTBESTRECORDS
2. EXACTMATCH
3. YES
4. AUTHORITATIVE
   - ExactType
     - E1
     - CNAME
     - E2
     - E3
   - E4
3. NO
4. WILDCARDMATCH
   - ExactType
     - W1
     - CNAME
     - W2
     - W3
   - DNAME
     - D1
     - REFERRAL
     - R1
     - R2
Test Generation Module

Formal Model

Constraints

Solve (1,2,3,4) for inputs \( (z, q) \)
Test Generation Module

Formal Model

Solve (1, 2, 3, 4) for inputs \((z, q)\)
Test Generation Module

Formal Model

Executable version in Zen

```
SELECTBESTRECORDS

QUERY Zone

YES

AUTHORITATIVE

EXACTTYPE

NAME TYPE

R2

E1

W1

W2

W3

D1

R2

NO

WILD CARD MATCH

EXACTTYPE

NAME TYPE

R1

Siva Kakarla

SCALE: Automatically Finding RFC Compliance Bugs in DNS Nameservers
```
An **executable version** of formal model is implemented in **Zen**, a domain-specific modeling language embedded in **C#** with built-in support for **symbolic execution**.
An **executable version** of formal model is implemented in **Zen**, a domain-specific modeling language embedded in **C#** with built-in support for **symbolic execution**.
Challenge – Generating Valid Zones

Valid

$q = (d, t), z = [r_1, r_2, ...]$  

Invalid

Symbolic Execution Path

SELECTBESTRECORDS

EXACTMATCH

AUTHORITATIVE

EXACTTYPE

EI

Query Zone

Solve

YES
Challenge – Generating Valid Zones

- Zone must satisfy several conditions to be valid
- Example condition $C_1$: There can be only one DNAME record for a domain name
Challenge – Generating Valid Zones

- Zone must satisfy several conditions to be valid
- Example condition $C_1$ - There can be only one DNAME record for a domain name

$$r_1 = \{ \text{foo.com.}, \text{DNAME}, \text{bar.edu.} \} \implies r_n$$
$$\neq \{ \text{foo.com.}, \text{DNAME}, \text{web.in.} \}$$

$$q = (d, t), z = [r_1, r_2, \ldots]$$

Valid

Invalid

Solve

Symbolic Execution Path

SELECTBESTRECORDS

QUERY

ZONE

AUTHORITATIVE

EXACTMATCH

EXACTTYPE

EI
Challenge – Generating Valid Zones

- Zone must satisfy several conditions to be valid
- Example condition $C_1$ - There can be only one DNAME record for a domain name
- Conditions $C_1, C_2, ... \rightarrow$ Zen predicates $P_1, P_2, ...$

Let $r_1 = \{\text{foo.com., DNAME, bar.edu.}\} \implies r_n$ $\neq \{\text{foo.com., DNAME, web.in.}\}$
Challenge – Generating Valid Zones

\[ r_1 = \{ \text{foo.com, DNAME, bar.edu.} \} \Rightarrow r_n \]
\[ \neq \{ \text{foo.com, DNAME, web.in.} \} \]

- Zone must satisfy several conditions to be valid
- Example condition C₁ - There can be only one DNAME record for a domain name
- Conditions C₁, C₂, ... → Zen predicates P₁, P₂, ...
Challenge – Generating Valid Zones

Zone must satisfy several conditions to be valid

- Example condition $C_1$ - There can be only one DNAME record for a domain name
- Conditions $C_1$, $C_2$, ... → Zen predicates $P_1$, $P_2$, ...

$r_1 = \{\text{foo.com.}, \text{DNAME}, \text{bar.edu.}\} \implies r_n$
$r_1 \neq \{\text{foo.com.}, \text{DNAME}, \text{web.in.}\}$
Challenge – Generating Valid Zones

- Zone must satisfy several conditions to be valid
- Example condition $C_1$ - There can be only one DNAME record for a domain name
- Conditions $C_1, C_2, \ldots \rightarrow$ Zen predicates $P_1, P_2, \ldots$

$$r_1 = \{\text{foo.com.}, \text{DNAME}, \text{bar.edu.}\} \implies r_n$$
$$\neq \{\text{foo.com.}, \text{DNAME}, \text{web.in.}\}$$

We also generate invalid zone files using Zen predicates
Using small-scope property of DNS we limit the length of each domain name & the number of records in the zone ≤ 4
Exhaustive Model Coverage with Test Generation

All model leaves are covered

Using small-scope property of DNS we limit the length of each domain name & the number of records in the zone ≤ 4

<table>
<thead>
<tr>
<th>Model Case</th>
<th>Number of Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>3180</td>
</tr>
<tr>
<td>E2</td>
<td>12</td>
</tr>
<tr>
<td>E4</td>
<td>96</td>
</tr>
<tr>
<td>W1</td>
<td>6036</td>
</tr>
<tr>
<td>W2</td>
<td>60</td>
</tr>
<tr>
<td>W3</td>
<td>24</td>
</tr>
<tr>
<td>D1</td>
<td>18</td>
</tr>
<tr>
<td>R1</td>
<td>230</td>
</tr>
<tr>
<td>R2</td>
<td>2980</td>
</tr>
<tr>
<td>Total</td>
<td>12,673</td>
</tr>
</tbody>
</table>
DNS Differential Testing

Tests 12,673

1. BIND
2. NSD
3. KNOT
4. PDNS
5. ...

Response Grouping

Single group 4,433

>1 group

8,240
DNS Differential Testing

Tests: 12,673

1. BIND
2. NSD
3. KNOT
4. PDNS
5. ...

Response Grouping

- Single group: 4,433
- >1 group: 8,240

Too many to check manually!
DNS Differential Testing

Tests 12,673

1. BIND
2. NSD
3. KNOT
4. PDNS
5. ...

Response Grouping

- Single group: 4,433
- >1 group: 8,240
- Too many to check manually!

Hybrid Fingerprinting - 75 fingerprints

More test failures than bugs (root causes)
### Hybrid Fingerprinting

<table>
<thead>
<tr>
<th>Model Case</th>
<th>Number of Tests</th>
<th>Number of Tests Failing</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>3180</td>
<td>239</td>
</tr>
<tr>
<td>E2</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>E3</td>
<td>96</td>
<td>12</td>
</tr>
<tr>
<td>E4</td>
<td>6036</td>
<td>5312</td>
</tr>
<tr>
<td>W1</td>
<td>60</td>
<td>33</td>
</tr>
<tr>
<td>W2</td>
<td>24</td>
<td>21</td>
</tr>
<tr>
<td>W3</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>D1</td>
<td>230</td>
<td>65</td>
</tr>
<tr>
<td>R1</td>
<td>2980</td>
<td>2529</td>
</tr>
<tr>
<td>R2</td>
<td>37</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Fingerprint** failed tests
- Based on model case and the unique implementations in each group from the responses
- Example fingerprint – \(\{R1, \{\text{NsD, Knot, PowerDns, Yadifa}\}, \{\text{Bind, CoreDns}\}, \{\text{TrustDns, Maradns}\}\)\
- Unlikely for different unique bugs to have the same fingerprint
Hybrid Fingerprinting

<table>
<thead>
<tr>
<th>Model Case</th>
<th>Number of Tests</th>
<th>Number of Tests Failing</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>3180</td>
<td>239</td>
</tr>
<tr>
<td>E2</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>E3</td>
<td>96</td>
<td>12</td>
</tr>
<tr>
<td>E4</td>
<td>6036</td>
<td>5312</td>
</tr>
<tr>
<td>W1</td>
<td>60</td>
<td>33</td>
</tr>
<tr>
<td>W2</td>
<td>24</td>
<td>21</td>
</tr>
<tr>
<td>W3</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>D1</td>
<td>230</td>
<td>65</td>
</tr>
<tr>
<td>R1</td>
<td>2980</td>
<td>2529</td>
</tr>
<tr>
<td>R2</td>
<td>37</td>
<td>3</td>
</tr>
</tbody>
</table>

Hybrid Fingerprint:  
- Signature + Behavior  
- Commonly used in IDS but not for triaging bugs

Fingerprint failed tests

Structural Signature (specification-based)

Example fingerprint – \{R1, \{Nsd, Knot, PowerDns, Yadifa\}, \{Bind, Coredns\}, \{TrustDns, Maradns\}\}

Behavioral Differences (differential testing)

Unlikely for different unique bugs to have the same fingerprint
Hybrid Fingerprinting

<table>
<thead>
<tr>
<th>Model Case</th>
<th>Number of Tests</th>
<th>Number of Tests Failing</th>
<th>Number of Fingerprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>3180</td>
<td>239</td>
<td>7</td>
</tr>
<tr>
<td>E2</td>
<td>12</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>E3</td>
<td>96</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>E4</td>
<td>6036</td>
<td>5312</td>
<td>11</td>
</tr>
<tr>
<td>W1</td>
<td>60</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>W2</td>
<td>24</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>W3</td>
<td>18</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>D1</td>
<td>230</td>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>R1</td>
<td>2980</td>
<td>2529</td>
<td>27</td>
</tr>
<tr>
<td>R2</td>
<td>37</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Fingerprint** failed tests
- Based on model case and the unique implementations in each group from the responses
- Example fingerprint – \{R1, \{Nsd, Knot, PowerDns, Yadifa\}, \{Bind, Coredns\}, \{TrustDns, Maradns\}\)
- Unlikely for different unique bugs to have the same fingerprint
### Bugs Found and Confirmed in Open-source DNS Implementations

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Language</th>
<th>Description</th>
<th>Bugs found</th>
<th>Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIND</td>
<td>C</td>
<td>De facto standard</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>POWERDNS</td>
<td>C++</td>
<td>Popular in North Europe</td>
<td>2</td>
<td>--</td>
</tr>
<tr>
<td>NSD</td>
<td>C</td>
<td>Hosts several TLDs</td>
<td>4</td>
<td>--</td>
</tr>
<tr>
<td>KNOT</td>
<td>C</td>
<td>Hosts several TLDs</td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>COREDNS</td>
<td>Go</td>
<td>Used in Kubernetes</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>YADIFA</td>
<td>C</td>
<td>Created by EURid (.eu)</td>
<td>3</td>
<td>--</td>
</tr>
<tr>
<td>TRUSTDNS</td>
<td>Rust</td>
<td>Security, safety focused</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>MARADNS</td>
<td>C</td>
<td>Lightweight server</td>
<td>2</td>
<td>--</td>
</tr>
</tbody>
</table>
# Bugs Found and Confirmed in Open-source DNS Implementations

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Language</th>
<th>Description</th>
<th>Bugs found</th>
<th>Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIND</td>
<td>C</td>
<td>De facto standard</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>POWERDNS</td>
<td>C++</td>
<td>Popular in North Europe</td>
<td>2</td>
<td>--</td>
</tr>
<tr>
<td>NSD</td>
<td>C</td>
<td>Hosts several TLDs</td>
<td>4</td>
<td>--</td>
</tr>
<tr>
<td>KNOT</td>
<td>C</td>
<td>Hosts several TLDs</td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>COREDNS</td>
<td>Go</td>
<td>Used in Kubernetes</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>YADIFA</td>
<td>C</td>
<td>Created by EURid (.eu)</td>
<td>3</td>
<td>--</td>
</tr>
<tr>
<td>TRUSTDNS</td>
<td>Rust</td>
<td>Security, safety focused</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>MARADNS</td>
<td>C</td>
<td>Lightweight server</td>
<td>2</td>
<td>--</td>
</tr>
</tbody>
</table>

Tests part of CI/CD pipeline in Amazon Route 53 DNS
Example Bug – COREDNS Crash

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>example.</td>
<td>SOA</td>
<td>ns1.exm. ...</td>
</tr>
<tr>
<td>*.example.</td>
<td>CNAME</td>
<td>foo.example.</td>
</tr>
</tbody>
</table>

Query: ("baz.bar.example., A")

- Query is rewritten using CNAME to: "baz.bar.example. CNAME foo.example."
- The rewritten query will match the wildcard again!
Example Bug – COREDNS Crash

FERRET Generated Test Case

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>example.</td>
<td>SOA</td>
<td>ns1.exam. ...</td>
</tr>
<tr>
<td>*.example.</td>
<td>CNAME</td>
<td>foo.example.</td>
</tr>
</tbody>
</table>

Query: ("baz.bar.example., A")

- Query is rewritten using CNAME to: "baz.bar.example. CNAME foo.example."
- The rewritten query will match the wildcard again!

Others

1. Synthesized CNAMEs as Response

2. COREDNS

No Response!

Popular open-sourced server written in Go Recommended Server for Kubernetes
Example Bug – COREDNS Crash

**FERRET Generated Test Case**

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>example.</td>
<td>SOA</td>
<td></td>
</tr>
<tr>
<td>*.example.</td>
<td>CNAME</td>
<td>foo.example</td>
</tr>
</tbody>
</table>

Query: ⟨"baz.bar.example", A⟩

- Query is rewritten using `CNAME` to: "baz.bar.example.CNAME foo.example.".
- The rewritten query will match the wildcard again!

```
runtime: goroutine stack exceeds 1000000000-byte limit
runtime: sp=0xc03c6c0378 stack=[0xc03c6c0000, 0xc05c6c0000]
fatal error: stack overflow
```

Crashes !!
Serious Security Vulnerability

Fixed by adding a loop counter† – “For now it’s more important to protect ourselves than to give the client a valid answer”

†[https://github.com/coredns/coredns/issues/4378](https://github.com/coredns/coredns/issues/4378)

Go Recommended Server for Kubernetes

---

Siva Kakarla
Comments from DNS Community

“This is awesome, thank you for this work, and thank you for your very clear bug reports, both to us (PowerDNS) and to other projects.”

“I was not kidding about the excellent bug reports, by the way..”

— Peter Van Dijik
(Senior PowerDNS Developer)

“I was skeptical because I thought – why should I believe his tests, but he proved them by running against so many DNS servers through them”

“So, possibly new RFCs should come with their own logic diagram which can be used to generate the tests”

— Vicky Risk
(Director of Marketing, ISC Bind)
And
Pauel Hauffman
(IETF & ICANN)

Replying to @dnsoarc @SivaKesavaRK and @UCLAengineering

Incredible reception from the audience on @SivaKesavaRK presentation. The automation tool received great compliments from the DNS experts

#OARC35 #LoveDNS #DNS ^MV
8:12 AM · May 7, 2021 · TweetDeck
Summary

Technical Challenge
Must *jointly* generate structured zone files and queries in order to check RFC behavior compliance of DNS nameserver implementations.

Key Idea
Leverages the small-scope property to build an executable model of DNS resolution and symbolically execute it to generate high-coverage tests that cover all paths in the model.

Impact
Found dozens of bugs across 8 nameserver implementations, including 3 critical security vulnerabilities.

FERRET: [github.com/dns-groot/Ferret](http://github.com/dns-groot/Ferret)
Dataset: [github.com/dns-groot/FerretDataset](http://github.com/dns-groot/FerretDataset)
Generating Invalid Zone Files

- Bugs also occur when handling of ill-formed zones
- Fuzzing (previous work) → syntactically invalid inputs
- Zen → semantically invalid zone files
- GROOT → Test queries for invalid zone files

Zen validity predicates

\[ \neg P_1 \land P_2 \land \ldots \]
\[ P_1 \land \neg P_2 \land \ldots \]
\[ \vdots \]

\text{Solve}

Invalid zone files

Reject the zone file due to syntactic or semantic errors

Accept the zone file to serve - converted to a valid one

Preprocessor like kzonecheck or pdnsutil

Valid/Invalid zone file