CoreDNS

Security Assessment

March 10, 2022

Prepared for:
Judd Luckey
Infoblox

Prepared by:
Alex Useche
Shaun Mirani

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and mutually agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As such, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Table of Contents

Executive Summary

Engagement Overview

The Linux Foundation engaged Trail of Bits to review the security of its CoreDNS application. From January 10 to January 14, 2022, a team of two consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We performed automated testing and a manual review of the code, in addition to running system elements. We also conducted a lightweight threat model of the application, with the goal of providing the Linux Foundation with a high-level picture of the threats present in the design of CoreDNS.

Summary of Findings

The audit uncovered one high-severity issue (TOB-CDNS-8) concerning a bug that could lead to cache poisoning attacks. The majority of the other issues are of informational or low severity; these include several resulting from insufficient data validation, specifically from assumptions about the data processed by various functions, which we discovered by running fuzzing harnesses. Most of the findings pertain to denial-of-service vulnerabilities.

A summary of the findings is provided below; note that because of the requirements and time constraints of the audit, we could not determine each finding's severity and difficulty level.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Medium	1
Low	7
Informational	5
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Undefined Behavior	1
Error Reporting	1
Denial of Service	7
Cryptography	3
Data Validation	1
Data Exposure	2

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan.guido@trailofbits.com

Cara Pearson, Project Manager
cara.pearson@trailofbits.com

The following engineers were associated with this project:

Alex Useche, Consultant
alex.useche@trailofbits.com

Shaun Mirani, Consultant
shaun.mirani@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
January 4, 2022	Pre-project kickoff call
January 7, 2022	Status update meeting #1
January 18, 2022	Delivery of report draft and report readout meeting

Project Goals

The engagement was scoped to provide a security assessment of CoreDNS and to conduct a high-level lightweight threat model of the application's design. As part of our code review and dynamic testing of the application, we sought to answer the following non-exhaustive list of questions:

- Does the application properly handle all errors?
- Are any components of the application susceptible to log injection attacks?
- Could an attacker bypass the access controls enforced by the `acl` (access control list) plugin?
- Is the application reasonably secure by default when installed and configured as outlined in the documentation?
- Does the application implement proper data validation controls?
- Are the cryptographic controls used throughout the codebase sound?

Project Targets

The engagement involved a review and testing of the following target.

CoreDNS

Repository	https://github.com/coredns/coredns
Version	7ee128a53da7ca1ee512422b56f31d4a24ed7b8b
Type	DNS server
Platform	Go

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- **DNS server**. We reviewed the core logic supporting the DNS server, looking for issues related to data validation, the use of concurrency, the handling of errors, and the code's correctness. This review also included manual and automated static analysis of the logic supporting DNS over HTTPS, over the Google remote procedure call (gRPC), and over Transport Layer Security (TLS).

- **Plugin-loading logic**. We reviewed the core plugin-loading logic for any correctness issues.

- **Built-in plugins**. The CoreDNS repository includes 52 plugins. Given the engagement's time constraints, we focused on the plugins deemed the most critical and those identified as priorities by the CoreDNS team (i.e., those typically used by default in Kubernetes deployments). We reviewed the following plugins: `kubernetes`, `tls`, `file`, `log`, `loop`, `reload`, `etcd`, `prometheus`, `errors`, `grpc`, `pprof`, `azure`, `route53`, `health`, `loadbalance`, `ready`, `forward`, `cache`, `acl`.

- **Fuzz testing**. We used go-fuzz to perform fuzz testing of the CoreDNS components listed in Appendix C.

- **Other logic**. We sought to identify any issues in the use of cryptography and ran the following automated static analysis tools against the entire codebase: Semgrep, CodeQL, ineffassign, errcheck, gosec, and GCatch.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the plugins not listed above. Furthermore, our threat model is not exhaustive, as those other plugins may include functionalities that increase the complexity of the design described in the CoreDNS Threat Model section. Lastly, because we performed lightweight threat modeling, the threat model should be expanded and further examined as CoreDNS continues to be developed.

CoreDNS Threat Model

As part of the audit, Trail of Bits conducted a lightweight threat model, drawing from the Mozilla Rapid Risk Assessment methodology and the National Institute of Standards and Technology's (NIST) guidance on data-centric threat modeling (NIST 800-154). We began our assessment of the application's design by reviewing the documentation on the CoreDNS website and the various README files in the CoreDNS repository. We gained an initial understanding of the system through our review of the code, which we conducted in tandem with the threat model. Lastly, at a short meeting on January 14, 2022, we interviewed engineers and members of the CoreDNS red team on the design of the application, security considerations, and assumptions regarding the use of the application and its external plugins.

The threat model focuses on the use cases of the core logic as well as common data flow paths; these include the paths of the HTTP endpoints exposed by the `metrics`, `pprof`, `ready`, and `health` plugins.

Assumptions

In conducting this threat model, we did not make any specific assumptions regarding the plugins. We performed a high-level threat model of CoreDNS, considering how the use of certain types of plugins could affect its data flow, data storage, and communication with external parties such as cloud providers.

Similarly, we did not make any specific assumptions regarding the back end for which CoreDNS provides DNS querying, forwarding, and mapping capabilities. For instance, the "back-end services" referenced in the "Data Flow" diagram could comprise a traditional network with servers and services hosted on-premises, services hosted in a Kubernetes cluster, or a cloud infrastructure.

Data Types

Data	Description
DNS Queries	DNS queries originate from DNS clients such as web browsers, terminal utilities, and other applications. Queries can be sent over the User Datagram Protocol (UDP), the Transmission Control Protocol (TCP), or gRPC, or through DNS over HTTPS (DoH) or DNS over TLS (DoT).
DNS Replies	When CoreDNS receives a DNS query from a client, it uses the plugin chain to process the query and then sends an appropriate DNS reply back to the client. CoreDNS may also receive DNS replies from upstream servers such as those used in the `forward` and `grpc` plugins.
Plugin API Calls	Plugins may expose independent APIs over transports like HTTP. Clients can issue requests to these APIs and receive responses outside of the typical DNS data flows. For example, the `health` plugin exposes an HTTP endpoint that clients can query to determine the health status of the server.
Cloud API Calls	CoreDNS includes plugins for serving zones from the DNS services of cloud providers such as AWS, Azure, and the Google Cloud Platform. To authenticate to the services of those providers and interact with data, CoreDNS must make network calls to their APIs on the wider internet.
Log Data	CoreDNS provides logging capabilities through the `log`, `errors`, and `dump` plugins. Logged data can include query and reply details and error messages that reveal information about the application's internal state.
Metrics	CoreDNS is instrumented to collect basic metrics and statistical information that can be exported to a location specified by the end user via the `prometheus` plugin.
Secrets	CoreDNS handles and processes secrets including private keys used to facilitate TLS-based transport and to create credentials for the cloud services used by the plugins.

Configuration Data	CoreDNS uses configuration data including the main Corefile and zone data.

Components

Component	Description
DNS Server	This server runs CoreDNS and handles DNS queries and tasks for other services running in a network.
Plugins	CoreDNS chains plugins responsible for tasks such as connecting to cloud services, logging requests, and integrating with Kubernetes. It provides internal plugins (plugins included in the CoreDNS codebase) and enables the use of external plugins, which anyone can develop. To add an external plugin, one must rebuild CoreDNS after incorporating the external plugin source code.
Storage	The `etcd` plugin enables CoreDNS to interact with etcd, and the `trace` plugin enables it to store local log files.
HTTP Server	Plugins such as the `metrics` and `pprof` plugins expose data endpoints via HTTP endpoints.

Trust Zones

Trust zones capture logical boundaries where controls should or could be enforced by the system and allow developers to implement controls and policies between components' zones.

Zone	Description	Included Components
Internet	The wider external-facing internet zone, which typically includes users and software that interface with the service but	• CLI • Browser clients • Cloud infrastructure

	does not contain core application logic	
Local Network	The local network where CoreDNS is run	• CoreDNS server • Plugin chain
Service Network(s)	The network(s) accessed by CoreDNS	• K8s clusters • Other networks used by CoreDNS to run DNS services

Trust Zone Connections

We can draw from our understanding of what data flows between trust zones and why to enumerate attack scenarios.

Originating Zone	Destination Zone	Data	Connection Types	Authentication and Authorization Controls
Internet	Local Network	DNS queries Plugin API calls	UDP, TCP, DoH, DoT, and gRPC HTTP	`acl` plugin
Local Network	Internet	DNS replies External API calls	UDP, TCP, DoH, DoT, and gRPC HTTP	Authentication via API calls to cloud services Authorization of responses by `acl` plugin
Local Network	Local Network	DNS queries Plugin API calls	UDP, TCP, DoH, DoT, and gRPC HTTP	`acl` plugin

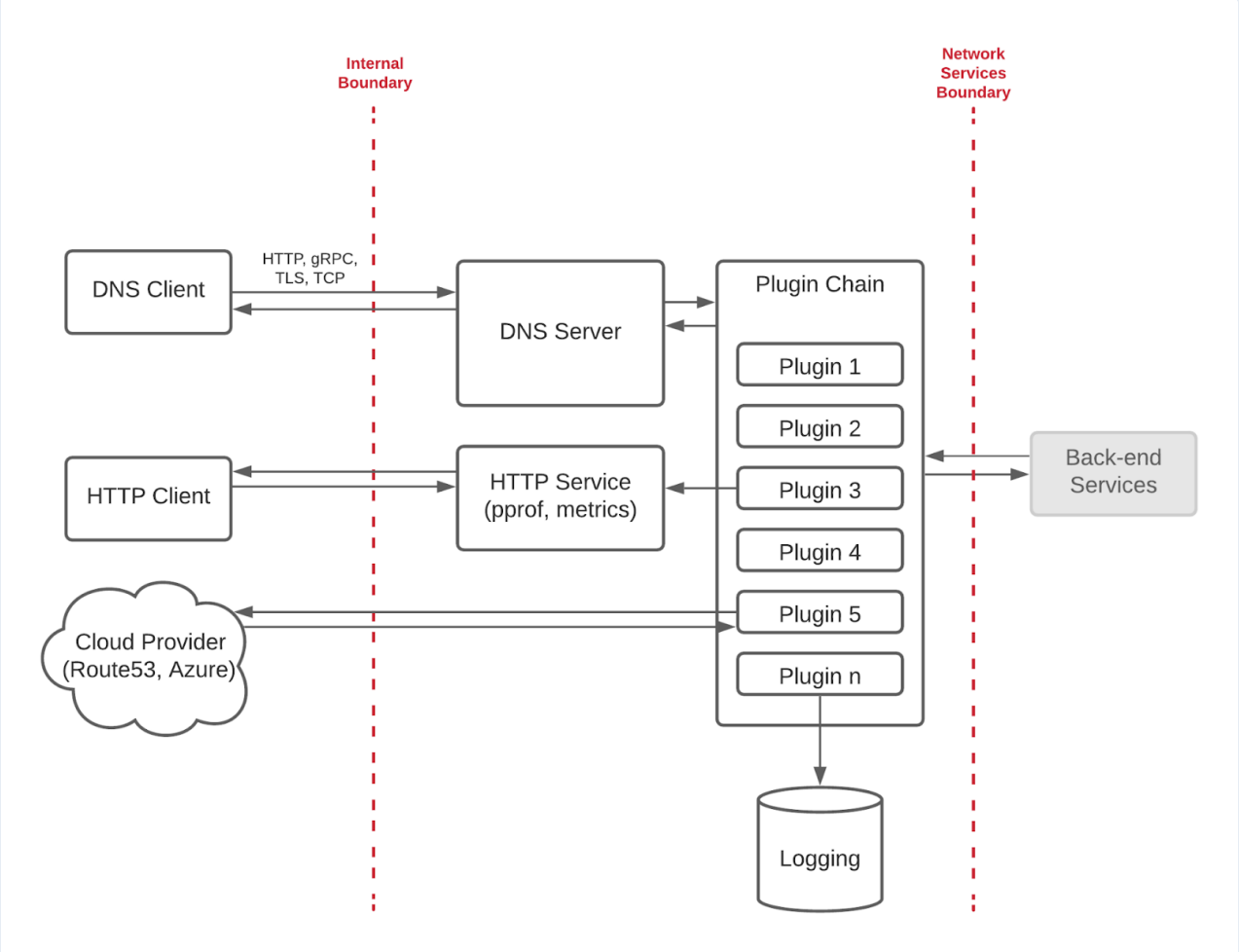
Local Network	Services Network	API calls DNS queries	UTP, TCP, and HTTP	N/A

Threat Actors

Similarly to establishing trust zones, defining malicious actors before conducting a threat model is useful in determining which protections, if any, are necessary to mitigate or remediate a vulnerability. We also define other "users" of the system who may be impacted by, or induced to undertake, an attack.

Actor	Description
Malicious Internal User	Malicious internal users often have privileged access to a wide range of resources, such as the Kubernetes cluster that hosts CoreDNS.
External Plugin Developer	A developer of an external plugin for CoreDNS could intentionally include malware in the plugin.
Internal Attacker	An internal attacker is one who has transited one or more trust boundaries, such as an attacker with access to a cluster or to the machine hosting CoreDNS.
Operator	An operator could misconfigure CoreDNS, resulting in unexpected behavior or error conditions that weaken its security controls and may not be adequately reported. To mitigate such concerns, the system should maintain an audit trail by logging all errors, and administrators should be provided extensive documentation.
End User	External users of the services supported by CoreDNS communicate with the application through DNS queries, gRPC, and HTTP. An end user may be a legitimate external actor or one with malicious intent against the CoreDNS server and its infrastructure.

Data Flow



Threat Scenarios

Threat	Description	Actor(s)	Component(s)
Compromise of secrets stored in the Corefile	The Corefile includes secrets used to establish connections with cloud providers.	Malicious local user	Configuration files for CoreDNS
Modification of secrets in the Corefile	An attacker modifies the Corefile to force CoreDNS to communicate with	Malicious local user	Configuration files for CoreDNS

	unintended cloud services.		
Faulty external plugin	An external plugin introduces issues that could crash the application, such as goroutine leaks.	External plugin developer	Plugins
Malicious plugin compiled with the CoreDNS binary	The CoreDNS binary is compiled with a malicious plugin, which is downloaded by a user and runs unauthorized actions.	External plugin developer	Plugins
Distributed denial of service (DDoS) via DNS amplification	An attacker launches a DNS amplification attack against a CoreDNS server, causing a loss of availability.	End user and internal attacker	DNS server
Access to metrics and debugging endpoints	An attacker uses the Go runtime debugging data in the `metrics` and `pprof` plugins to formulate a denial-of-service (DoS) attack.	End user and internal attacker	Plugins
Misconfiguration of a CoreDNS endpoint	Misconfiguration of a CoreDNS endpoint results in invalid DNS responses, ineffective ACL controls, or the unintended sharing of debugging data.	Operator and end user	Configuration files for CoreDNS and zone files
Cache poisoning vulnerability	A faulty assumption or data validation issue in the `cache` plugin logic results in a cache poisoning vulnerability, compromising	End user	DNS server and plugins

	the integrity of DNS replies.		

Recommendations

- Develop guidance on security best practices for CoreDNS users. This guidance can be modeled after HashiCorp Nomad's Security Model and related security recommendations. It should recommend plugins that can help harden the security of CoreDNS and implement security-in-depth controls, such as the `tls` and `acl` plugins.
- Establish requirements for the development of external plugins, such as unit testing requirements.
- Extend this threat model and continue to explore potential system flaws that could lead to vulnerabilities. To follow a Rapid Risk Assessment methodology, use the template in Appendix D.
- Extend the GitHub Action tests to include additional tools, such as those used in this assessment.
- Implement rate-limiting controls in CoreDNS. For instance, consider incorporating the `rrl` plugin, used for rate limiting, into the main set of CoreDNS plugins to help prevent DoS attacks.
- Consider establishing a process for checking that external plugins adhere to best practices. Document the process and share it with users, or run through the process before listing external plugins as "verified."

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Risk of a race condition in the secondary plugin's setup function	Undefined Behavior	Low
2	Upstream errors captured in the grpc plugin are not returned	Error Reporting	Low
3	Index-out-of-range panic in autopath plugin initialization	Denial of Service	Informational
4	Index-out-of-range panic in forward plugin initialization	Denial of Service	Informational
5	Use of deprecated PreferServerCipherSuites field	Cryptography	Informational
6	Use of the MD5 hash function to detect Corefile changes	Cryptography	Low
7	Use of default math/rand seed in grpc and forward plugins' "random" server-selection policy	Cryptography	Low
8	Cache plugin does not account for hash table collisions	Data Validation	High
9	Index-out-of-range reference in kubernetes plugin	Denial of Service	Undetermined
10	Calls to time.After() in select statements can lead to memory leaks	Denial of Service	Low
11	Incomplete list of debugging data exposed by the prometheus plugin	Data Exposure	Low

12	Cloud integrations require cleartext storage of keys in the Corefile	Data Exposure	**Medium**
13	Lack of rate-limiting controls	Denial of Service	**Low**
14	Lack of a limit on the size of response bodies	Denial of Service	**Informational**
15	Index-out-of-range panic in grpc plugin initialization	Denial of Service	**Informational**

Detailed Findings

1. Risk of a race condition in the secondary plugin's setup function

Severity: **Low**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-CDNS-1
Target: `plugin/secondary/setup.go#L19-L53`	

Description

When it fails to transfer a zone from another server, the `setup` function of the `secondary` plugin prints a message to standard output. It obtains the name of the zone, stored in the variable n, from a loop and prints the message in an anonymous inner goroutine. However, the variable is not copied before being used in the anonymous goroutine, and the value that n points to is likely to change by the time the scheduler executes the goroutine. Consequently, the value of n will be inaccurate when it is printed.

```
19    func setup(c *caddy.Controller) error {
24    // (...).
26    for _, n := range zones.Names {
27            // (...)
29            c.OnStartup(func() error {
30                    z.StartupOnce.Do(func() {
31                            go func() {
32                                    // (...)
35                                    for {
36                                            // (...)
40                                            log.Warningf("All '%s' masters failed to
transfer, retrying in %s: %s", n, dur.String(), err)
41                                            // (...)
46                                    }
47                                    z.Update()
48                            }()
49                    })
50                            return nil
51            })
52    }
53  }
```

Figure 1.1: The value of n is not copied before it is used in the anonymous goroutine and could be logged incorrectly. (`plugin/secondary/setup.go#L19-L53`)

Exploit Scenario

An operator of a CoreDNS server enables the `secondary` plugin. The operator sees an error in the standard output indicating that the zone transfer failed. However, the error points to an invalid zone, making it more difficult for the operator to troubleshoot and fix the issue.

Recommendations

Short term, create a copy of n before it is used in the anonymous goroutine. See Appendix B for a proof of concept demonstrating this issue and an example of the fix.

Long term, integrate Trail of Bits' `anonymous-race-condition` Semgrep rule into the CI/CD pipeline to catch this type of race condition.

2. Upstream errors captured in the grpc plugin are not returned

Severity: **Low**	Difficulty: **Undetermined**
Type: Error Reporting	Finding ID: TOB-CDNS-2
Target: `plugin/grpc/grpc.go#L77-L95`	

Description

In the `ServeDNS` implementation of the `grpc` plugin, upstream errors are captured in a loop. However, once an error is captured in the `upstreamErr` variable, the function exits with a `nil` error; this is because there is no `break` statement forcing the function to exit the loop and to reach a return statement, at which point it would return the error value. The `ServeDNS` function of the `forward` plugin includes a similar but correct implementation.

```
func (g *GRPC) ServeDNS(ctx context.Context, w dns.ResponseWriter, r *dns.Msg) (int, error)
{
                // (...)
                upstreamErr = err

                // Check if the reply is correct; if not return FormErr.
                if !state.Match(ret) {
                        debug.Hexdumpf(ret, "Wrong reply for id: %d, %s %d", ret.Id,
state.QName(), state.QType())

                        formerr := new(dns.Msg)
                        formerr.SetRcode(state.Req, dns.RcodeFormatError)
                        w.WriteMsg(formerr)
                        return 0, nil
                }

                w.WriteMsg(ret)
                return 0, nil
        }

        if upstreamErr != nil {
                return dns.RcodeServerFailure, upstreamErr
        }
```

Figure 2.1: `plugin/secondary/setup.go#L19-L53`

Exploit Scenario

An operator runs CoreDNS with the `grpc` plugin. Upstream errors cause the gRPC functionality to fail. However, because the errors are not logged, the operator remains unaware of their root cause and has difficulty troubleshooting and remediating the issue.

Recommendations

Short term, correct the ineffectual assignment to ensure that errors captured by the plugin are returned.

Long term, integrate `ineffassign` into the CI/CD pipeline to catch this and similar issues.

3. Index-out-of-range panic in autopath plugin initialization

Severity: **Informational**	Difficulty: **Undetermined**
Type: Denial of Service	Finding ID: TOB-CDNS-3
Target: `plugin/autopath/setup.go#L53`	

Description

The following syntax is used to configure the `autopath` plugin:

```
autopath [ZONE...] RESOLV-CONF
```

The `RESOLV-CONF` parameter can point to a `resolv.conf(5)` configuration file or to another plugin, if the string in the `resolv` variable is prefixed with an "@" symbol (e.g., "@kubernetes"). However, the `autoPathParse` function does not ensure that the length of the `RESOLV-CONF` parameter is greater than zero before dereferencing its first element and comparing it with the "@" character.

```
func autoPathParse(c *caddy.Controller) (*AutoPath, string, error) {
	ap := &AutoPath{}
	mw := ""

	for c.Next() {
		zoneAndresolv := c.RemainingArgs()
		if len(zoneAndresolv) < 1 {
			return ap, "", fmt.Errorf("no resolv-conf specified")
		}
		resolv := zoneAndresolv[len(zoneAndresolv)-1]
		if resolv[0] == '@' {
			mw = resolv[1:]
```

Figure 3.1: The length of `resolv` may be zero when the first element is checked. (`plugin/autopath/setup.go#L45-L54`)

Specifying a configuration file with a zero-length `RESOLV-CONF` parameter, as shown in figure 3.2, would cause CoreDNS to panic.

```
0
autopath ""
```

Figure 3.2: An `autopath` configuration with a zero-length `RESOLV-CONF` parameter

```
panic: runtime error: index out of range [0] with length 0

goroutine 1 [running]:
github.com/coredns/coredns/plugin/autopath.autoPathParse(0xc000518870)
       /home/ubuntu/audit-coredns/client-code/coredns/plugin/autopath/setup.go:53 +0x35c
github.com/coredns/coredns/plugin/autopath.setup(0xc000518870)
       /home/ubuntu/audit-coredns/client-code/coredns/plugin/autopath/setup.go:16 +0x33
github.com/coredns/caddy.executeDirectives(0xc00029eb00, {0x7ffdc770671b, 0x8}, {0x324cfa0,
0x31, 0x1000000004b7e06}, {0xc000543260, 0x1, 0x8}, 0x0)
       /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:661 +0x5f6
github.com/coredns/caddy.ValidateAndExecuteDirectives({0x22394b8, 0xc0003e8a00},
0xc0003e8a00, 0x0)
       /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:612 +0x3e5
github.com/coredns/caddy.startWithListenerFds({0x22394b8, 0xc0003e8a00}, 0xc00029eb00, 0x0)
       /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:515 +0x274
github.com/coredns/caddy.Start({0x22394b8, 0xc0003e8a00})
       /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:472 +0xe5
github.com/coredns/coredns/coremain.Run()
       /home/ubuntu/audit-coredns/client-code/coredns/coremain/run.go:62 +0x1cd
main.main()
       /home/ubuntu/audit-coredns/client-code/coredns/coredns.go:12 +0x17
```

Figure 3.3: CoreDNS panics when loading the `autopath` configuration.

Exploit Scenario

An operator of a CoreDNS server provides an empty `RESOLV-CONF` parameter when configuring the `autopath` plugin, causing a panic. Because CoreDNS does not provide a clear explanation of what went wrong, it is difficult for the operator to troubleshoot and fix the issue.

Recommendations

Short term, verify that the `resolv` variable is a non-empty string before indexing it.

Long term, review the codebase for instances in which data is indexed without undergoing a length check; handling untrusted data in this way may lead to a more severe denial of service (DoS).

4. Index-out-of-range panic in forward plugin initialization

Severity: **Informational**	Difficulty: **Undetermined**
Type: Denial of Service	Finding ID: TOB-CDNS-4
Target: `plugin/forward/setup.go#97`	

Description

Initializing the `forward` plugin involves parsing the relevant configuration section.

```
func parseStanza(c *caddy.Controller) (*Forward, error) {
	f := New()

	if !c.Args(&f.from) {
		return f, c.ArgErr()
	}
	origFrom := f.from
	zones := plugin.Host(f.from).NormalizeExact()
	f.from = zones[0] // there can only be one here, won't work with non-octet reverse
```

Figure 4.1: The length of the `zones` variable may be zero when the first element is checked. (`plugin/forward/setup.go#L89-L97`)

An invalid configuration file for the `forward` plugin could cause the `zones` variable to have a length of zero. A Base64-encoded example of such a configuration file is shown in figure 4.2.

```
Lgpmb3J3YXJkIE5vTWF0Pk69VL0vvVN0ZXJhbENoYXJDbGFzc0FueUNoYXJOb3ROTEEniez6bnlDaGFyQmVnaW5MaW5l
RW5kTGluZUJlZ2luVGV4dEVuZFRleHRXb3JkQm91bmRhcnlOb1dvYXRpbmcgc3lzdGVtIDogImV4dCIsICJ4ZnMiLCAi
bnRTaW50NjRLaW5kZnMiLiB5IGluZmVycmVkIHRvIGJlIGV4dCBpZiB1bnNwZWNpZmllZCBlIDogaHR0cHM6Di9rdWJl
cm5ldGVzaW9kb2NzY29uY2VwdHNzdG9yYWdldm9sdW1lcyMgIiIiIiIiIiIiIiIiJyCFmIWlsZj//4WuhZilr4WY5bCR
mPCd
```

Figure 4.2: The Base64-encoded `forward` configuration file

Specifying a configuration file like that shown above would cause CoreDNS to panic when attempting to access the first element of `zones`:

```
panic: runtime error: index out of range [0] with length 0

goroutine 1 [running]:
github.com/coredns/coredns/plugin/forward.parseStanza(0xc000440000)
```

```
        /home/ubuntu/audit-coredns/client-code/coredns/plugin/forward/setup.go:97 +0x972
github.com/coredns/coredns/plugin/forward.parseForward(0xc000440000)
        /home/ubuntu/audit-coredns/client-code/coredns/plugin/forward/setup.go:81 +0x5e
github.com/coredns/coredns/plugin/forward.setup(0xc000440000)
        /home/ubuntu/audit-coredns/client-code/coredns/plugin/forward/setup.go:22 +0x33
github.com/coredns/caddy.executeDirectives(0xc0000ea800, {0x7ffdf9f6e6ed, 0x36}, {0x324cfa0,
0x31, 0x1000000004b7e06}, {0xc00056a860, 0x1, 0x8}, 0x0)
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:661 +0x5f6
github.com/coredns/caddy.ValidateAndExecuteDirectives({0x22394b8, 0xc00024ea80},
0xc00024ea80, 0x0)
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:612 +0x3e5
github.com/coredns/caddy.startWithListenerFds({0x22394b8, 0xc00024ea80}, 0xc0000ea800, 0x0)
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:515 +0x274
github.com/coredns/caddy.Start({0x22394b8, 0xc00024ea80})
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:472 +0xe5
github.com/coredns/coredns/coremain.Run()
        /home/ubuntu/audit-coredns/client-code/coredns/coremain/run.go:62 +0x1cd
main.main()
        /home/ubuntu/audit-coredns/client-code/coredns/coredns.go:12 +0x17
```

Figure 4.3: CoreDNS panics when loading the `forward` configuration.

Exploit Scenario

An operator of a CoreDNS server misconfigures the `forward` plugin, causing a panic. Because CoreDNS does not provide a clear explanation of what went wrong, it is difficult for the operator to troubleshoot and fix the issue.

Recommendations

Short term, verify that the `zones` variable has the correct number of elements before indexing it.

Long term, review the codebase for instances in which data is indexed without undergoing a length check; handling untrusted data in this way may lead to a more severe DoS.

5. Use of deprecated PreferServerCipherSuites field

Severity: **Informational**	Difficulty: **Undetermined**
Type: Cryptography	Finding ID: TOB-CDNS-5
Target: `plugin/tls/tls.go#L36`	

Description

In the `setTLSDefaults` function of the `tls` plugin, the TLS configuration object includes a `PreferServerCipherSuites` field, which is set to `true`.

```
func setTLSDefaults(tls *ctls.Config) {
        tls.MinVersion = ctls.VersionTLS12
        tls.MaxVersion = ctls.VersionTLS13
        tls.CipherSuites = []uint16{
                ctls.TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
                ctls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
                ctls.TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,
                ctls.TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,
                ctls.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
                ctls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
                ctls.TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
                ctls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
                ctls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
        }
        tls.PreferServerCipherSuites = true
}
```

Figure 5.1: `plugin/tls/tls.go#L22-L37`

In the past, this property controlled whether a TLS connection would use the cipher suites preferred by the server or by the client. However, as of Go 1.17, this field is ignored. According to the Go documentation for `crypto/tls`, “Servers now select the best mutually supported cipher suite based on logic that takes into account inferred client hardware, server hardware, and security.”

When CoreDNS is built using a recent Go version, the use of this property is redundant and may lead to false assumptions about how cipher suites are negotiated in a connection to a CoreDNS server.

Recommendations

Short term, add this issue to the internal issue tracker. Additionally, when support for Go versions older than 1.17 is entirely phased out of CoreDNS, remove the assignment to the deprecated `PreferServerCipherSuites` field.

6. Use of the MD5 hash function to detect Corefile changes

Severity: **Low**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-CDNS-6
Target: `plugin/reload/reload.go#L81`	

Description

The `reload` plugin is designed to automatically detect changes to a Corefile and to reload it if necessary. To determine whether a file has changed, the plugin periodically compares the current MD5 hash of the file to the last hash calculated for it (`plugin/reload/reload.go#L81-L107`). If the values are different, it reloads the Corefile.

However, the MD5 hash function's vulnerability to collisions decreases the reliability of this process; if two different files produce the same hash value, the plugin will not detect the difference between them.

Exploit Scenario

An operator of a CoreDNS server modifies a Corefile, but the MD5 hash of the modified file collides with that of the old file. As a result, the `reload` plugin does not detect the change. Instead, it continues to use the outdated server configuration without alerting the operator to its use.

Recommendations

Short term, improve the robustness of the `reload` plugin by using the SHA-512 hash function instead of MD5.

7. Use of default math/rand seed in grpc and forward plugins' "random" server-selection policy

Severity: **Low**	Difficulty: **Low**
Type: Cryptography	Finding ID: TOB-CDNS-7
Target: `plugin/grpc/policy.go#L19-L37`, `plugin/forward/policy.go#L19-L37`	

Description

The `grpc` and `forward` plugins use the `random` policy for selecting upstream servers. The implementation of this policy in the two plugins is identical and uses the `math/rand` package from the Go standard library.

```
func (r *random) List(p []*Proxy) []*Proxy {
        switch len(p) {
        case 1:
                return p
        case 2:
                if rand.Int()%2 == 0 {
                        return []*Proxy{p[1], p[0]} // swap
                }
                return p
        }

        perms := rand.Perm(len(p))
        rnd := make([]*Proxy, len(p))

        for i, p1 := range perms {
                rnd[i] = p[p1]
        }
        return rnd
}
```

Figure 7.1: `plugin/grpc/policy.go#L19-L37`

As highlighted in figure 7.1, the `random` policy uses either `rand.Int` or `rand.Perm` to choose the order of the upstream servers, depending on the number of servers that have been configured.

Unless a program using the `random` policy explicitly calls `rand.Seed`, the top-level functions `rand.Int` and `rand.Perm` behave as if they were seeded with the value 1, which

is the default seed for `math/rand`. CoreDNS does not call `rand.Seed` to seed the global state of `math/rand`. Without this call, the `grpc` and `forward` plugins' "random" selection of upstream servers is likely to be trivially predictable and the same every time CoreDNS is restarted.

Exploit Scenario

An attacker targets a CoreDNS instance in which the `grpc` or `forward` plugin is enabled. The attacker exploits the deterministic selection of upstream servers to overwhelm a specific server, with the goal of causing a DoS condition or performing an attack such as a timing attack.

Recommendations

Short term, instantiate a `rand.Rand` type with a unique seed, rather than drawing random numbers from the global `math/rand` state. CoreDNS takes this approach in several other areas, such as the `loop` plugin.

8. Cache plugin does not account for hash table collisions

Severity: **High**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-CDNS-8
Target: `plugin/cache`	

Description

To cache a DNS reply, CoreDNS maps the FNV-1 hash of the query name and type to the content of the reply in a hash table entry.

```
func key(qname string, m *dns.Msg, t response.Type) (bool, uint64) {
        // We don't store truncated responses.
        if m.Truncated {
                return false, 0
        }
        // Nor errors or Meta or Update.
        if t == response.OtherError || t == response.Meta || t == response.Update {
                return false, 0
        }

        return true, hash(qname, m.Question[0].Qtype)
}

func hash(qname string, qtype uint16) uint64 {
        h := fnv.New64()
        h.Write([]byte{byte(qtype >> 8)})
        h.Write([]byte{byte(qtype)})
        h.Write([]byte(qname))
        return h.Sum64()
}
```

Figure 8.1: `plugin/cache/cache.go#L68-L87`

To check whether there is a cached reply for an incoming query, CoreDNS performs a hash table lookup for the query name and type. If it identifies a reply with a valid time to live (TTL), it returns the reply. CoreDNS assumes the stored DNS reply to be the correct one for the query, given the use of a hash table mapping.

However, this assumption is faulty, as FNV-1 is a non-cryptographic hash function that does not offer collision resistance, and there exist utilities for generating colliding inputs to

FNV-1. As a result, it is likely possible to construct a valid (`qname`, `qtype`) pair that collides with another one, in which case CoreDNS could serve the incorrect cached reply to a client.

Exploit Scenario

An attacker aiming to poison the cache of a CoreDNS server generates a valid (`qname*`, `qtype*`) pair whose FNV-1 hash collides with a commonly queried (`qname`, `qtype`) pair.

The attacker gains control of the authoritative name server for `qname*` and points its `qtype*` record to an address of his or her choosing. The attacker also configures the server to send a second record when (`qname*`, `qtype*`) is queried: a `qtype` record for `qname` that points to a malicious address.

The attacker queries the CoreDNS server for (`qname*`, `qtype*`), and the server caches the reply with the malicious address. Soon thereafter, when a legitimate user queries the server for (`qname`, `qtype`), CoreDNS serves the user the cached reply for (`qname*`, `qtype*`), since it has an identical FNV-1 hash. As a result, the legitimate user's DNS client sees the malicious address as the record for `qname`.

Recommendations

Short term, store the original name and type of a query in the value of a hash table entry. After looking up the key for an incoming request in the hash table, verify that the query name and type recorded alongside the cached reply match those of the request. If they do not, disregard the cached reply.

Short term, use the keyed hash function SipHash instead of FNV-1. SipHash was designed for speed and derives a 64-bit output value from an input value and a 128-bit secret key; this method adds pseudorandomness to a hash table key and makes it more difficult for an attacker to generate collisions offline. CoreDNS should use the `crypto/rand` package from Go's standard library to generate a cryptographically random secret key for SipHash on startup.

9. Index-out-of-range reference in kubernetes plugin

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Denial of Service	Finding ID: TOB-CDNS-9
Target: `plugin/kubernetes/parse.go#L96`	

Description

The `parseRequest` function of the `kubernetes` plugin parses a DNS request before using it to query Kubernetes. By fuzzing the function, we discovered an out-of-range issue that can cause a panic. The issue occurs when the function calls `stripUnderscore` with an empty string, as it does when it receives a request with the `qname` ".o.o.po.pod.8" and the `zone` "interwebs".

```
// stripUnderscore removes a prefixed underscore from s.
func stripUnderscore(s string) string {
        if s[0] != '_' {
                return s
        }
        return s[1:]
}
```

Figure 9.1: `plugin/kubernetes/parse.go#L97`

Because of the time constraints of the audit, we could not find a way to directly exploit this vulnerability. Although certain tools for sending DNS queries, like `dig` and `host`, verify the validity of a host before submitting a DNS query, it may be possible to exploit the vulnerability by using custom tooling or DNS over HTTPs (DoH).

Exploit Scenario

An attacker finds a way to submit a query with an invalid host (such as ".o.o.po.pod.8") to a CoreDNS server running as the DNS server for a Kubernetes endpoint. Because of the index-out-of-range bug, the `kubernetes` plugin causes CoreDNS to panic and crash, resulting in a DoS.

Recommendations

Short term, to prevent a panic, implement a check of the value of the string passed to the `stripUnderscore` function.

10. Calls to time.After() in select statements can lead to memory leaks

Severity: **Low**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-CDNS-10
Target: Various files	

Description

Calls to the `time.After` function in `select/case` statements within `for` loops can lead to memory leaks. This is because the garbage collector does not clean up the underlying `Timer` object until the timer has fired. A new timer is initialized at the start of each iteration of the `for` loop (and therefore with each `select` statement), which requires resources. As a result, if many routines originate from a `time.After` call, the system may experience memory overconsumption.

```
for {
        select {
        case <-ctx.Done():
                log.Debugf("Breaking out of CloudDNS update loop for %v: %v", h.zoneNames,
ctx.Err())
                return
        case <-time.After(1 * time.Minute):
                if err := h.updateZones(ctx); err != nil && ctx.Err() == nil /* Don't log
error if ctx expired. */ {
                        log.Errorf("Failed to update zones %v: %v", h.zoneNames, err)
                }
```

Figure 10.1: A `time.After()` routine that causes a memory leak (`plugin/clouddns/clouddns.go#L85-L93`)

The following portions of the code contain similar patterns:

- `plugin/clouddns/clouddns.go#L85-L93`
- `plugin/azure/azure.go#L87-96`
- `plugin/route53/route53.go#87-96`

Exploit Scenario

An attacker finds a way to overuse a function, which leads to overconsumption of a CoreDNS server's memory and a crash.

Recommendations

Short term, use a ticker instead of the `time.After` function in `select/case` statements included in `for` loops. This will prevent memory leaks and crashes caused by memory exhaustion.

Long term, avoid using the `time.After` method in `for-select` routines and periodically use a Semgrep query to detect similar patterns in the code.

References

- DevelopPaper post on the memory leak vulnerability in `time.After`
- "Golang <-time.After() Is Not Garbage Collected before Expiry" (Medium post)

11. Incomplete list of debugging data exposed by the prometheus plugin

Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-CDNS-11
Target: `prometheus` plugin	

Description

Enabling the `prometheus` (metrics) plugin exposes an HTTP endpoint that lists CoreDNS metrics. The documentation for the plugin indicates that it reports data such as the total number of queries and the size of responses. However, other data that is reported by the plugin (and also available through the `pprof` plugin) is not listed in the documentation. This includes Go runtime debugging information such as the number of running goroutines and the duration of Go garbage collection runs. Because this data is not listed in the `prometheus` plugin documentation, operators may initially be unaware of its exposure. Moreover, the data could be instrumental in formulating an attack.

```
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.4756e-05
go_gc_duration_seconds{quantile="0.25"} 6.0522e-05
go_gc_duration_seconds{quantile="0.5"} 7.1476e-05
go_gc_duration_seconds{quantile="0.75"} 0.000105802
go_gc_duration_seconds{quantile="1"} 0.000205775
go_gc_duration_seconds_sum 0.010425592
go_gc_duration_seconds_count 123
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 18
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
```

Figure 11.1: Examples of the data exposed by `prometheus` and omitted from the documentation

Exploit Scenario

An attacker discovers the metrics exposed by CoreDNS over port 9253. The attacker then monitors the endpoint to determine the effectiveness of various attacks in crashing the server.

Recommendations

Short term, document all data exposed by the `prometheus` plugin. Additionally, consider changing the data exposed by the `prometheus` plugin to exclude Go runtime data available through the `pprof` plugin.

12. Cloud integrations require cleartext storage of keys in the Corefile

Severity: **Medium**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-CDNS-12
Target: `route53`, `azure`, and `clouddns`	

Description
The `route53`, `azure`, and `clouddns` plugins enable CoreDNS to interact with cloud providers (AWS, Azure, and the Google Cloud Platform (GCP), respectively). To access `clouddns`, a user enters the path to the file containing his or her GCP credentials. When using `route53`, CoreDNS pulls the AWS credentials that the user has entered in the Corefile. If the AWS credentials are not included in the Corefile, CoreDNS will pull them in the same way that the AWS command-line interface (CLI) would. While operators have options for the way that they provide AWS and GCP credentials, Azure credentials must be pulled directly from the Corefile. Furthermore, the CoreDNS documentation lacks guidance on the risks of storing AWS, Azure, or GCP credentials in local configuration files.

Exploit Scenario
An attacker or malicious internal user gains access to a server running CoreDNS. The malicious actor then locates the Corefile and obtains credentials for a cloud provider, thereby gaining access to a cloud infrastructure.

Recommendations
Short term, remove support for entering cloud provider credentials in the Corefile in cleartext. Instead, load credentials for each provider in the manner recommended in that provider's documentation and implemented by its CLI utility. CoreDNS should also refuse to load credential files with overly broad permissions and warn users about the risks of such files.

13. Lack of rate-limiting controls	
Severity: **Low**	Difficulty: **Medium**
Type: Denial of Service	Finding ID: TOB-CDNS-13
Target: CoreDNS	

Description

CoreDNS does not enforce rate limiting of DNS queries, including those sent via DoH. As a result, we were able to issue the same request thousands of times in less than one minute over the HTTP endpoint `/dns-query`.

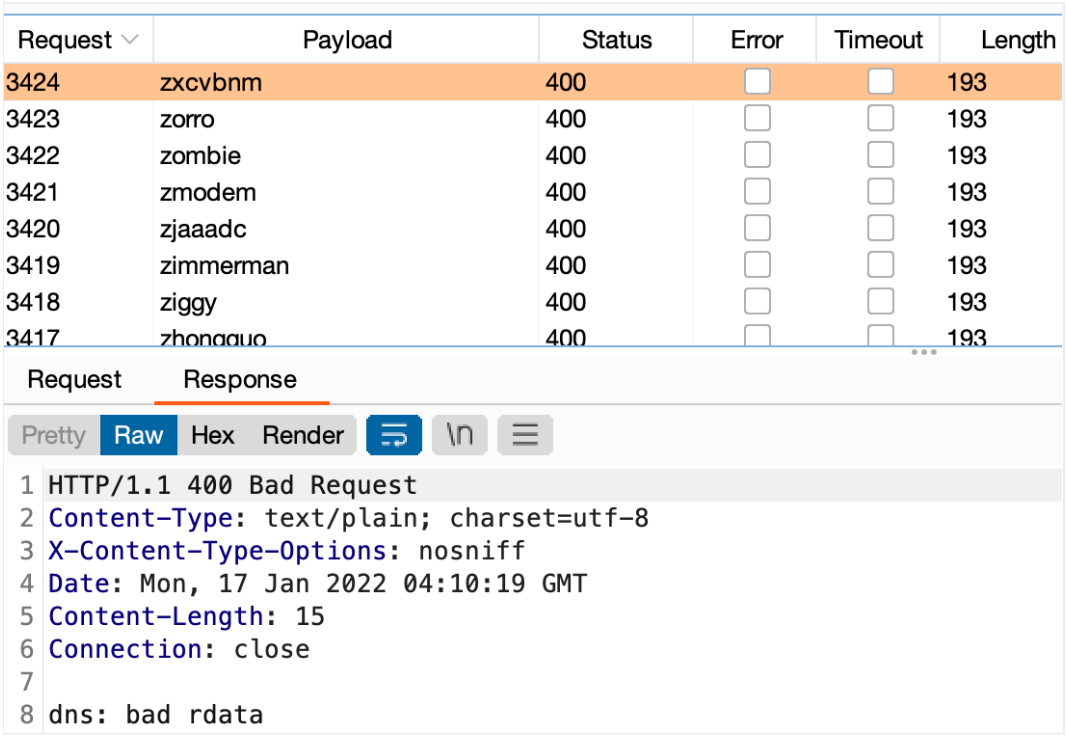


Figure 13.1: We sent 3,424 requests to CoreDNS without being rate limited.

During our tests, the lack of rate limiting did not appear to affect the application. However, processing requests sent at such a high rate can consume an inordinate amount of host resources, and a lack of rate limiting can facilitate DoS and DNS amplification attacks.

Exploit Scenario

An attacker floods a CoreDNS server with HTTP requests, leading to a DoS condition.

Recommendations

Short term, consider incorporating the `rrl` plugin, used for the rate limiting of DNS queries, into the CoreDNS codebase. Additionally, implement rate limiting on all API endpoints. An upper bound can be applied at a high level to all endpoints exposed by CoreDNS.

Long term, run stress tests to ensure that the rate limiting enforced by CoreDNS is robust.

14. Lack of a limit on the size of response bodies

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-CDNS-14
Target: `plugin/pkg/doh/doh.go#L94-L102`	

Description

The `ioutil.ReadAll` function reads from a source input until encountering an error or the end of the file, at which point it returns the data that it read. The `toMsg` function, which processes requests for the HTTP server, uses `ioutil.ReadAll` to parse requests and to read POST bodies.

However, there is no limit on the size of request bodies. Using `ioutil.ReadAll` to parse a large request that is loaded multiple times may exhaust the system's memory, causing a DoS.

```
func toMsg(r io.ReadCloser) (*dns.Msg, error) {
        buf, err := io.ReadAll(r)
        if err != nil {
                return nil, err
        }
        m := new(dns.Msg)
        err = m.Unpack(buf)
        return m, err
}
```

Figure 14.1: `plugin/pkg/doh/doh.go#L94-L102`

Exploit Scenario

An attacker generates multiple POST requests with long request bodies to `/dns-query`, leading to the exhaustion of its resources.

Recommendations

Short term, use the `io.LimitReader` function or another mechanism to limit the size of request bodies.

Long term, consider implementing application-wide limits on the size of request bodies to prevent DoS attacks.

15. Index-out-of-range panic in grpc plugin initialization

Severity: **Informational**	Difficulty: **Undetermined**
Type: Denial of Service	Finding ID: TOB-CDNS-15
Target: `plugin/grpc/setup.go#59`	

Description

Initializing the `grpc` plugin involves parsing the relevant configuration section.

```
func parseStanza(c *caddy.Controller) (*GRPC, error) {
	g := newGRPC()

	if !c.Args(&g.from) {
		return g, c.ArgErr()
	}
	g.from = plugin.Host(g.from).NormalizeExact()[0] // only the first is used.
```

Figure 15.1: `plugin/grpc/setup.go#L53-L59`

An invalid configuration file for the `grpc` plugin could cause the call to `NormalizeExtract` (highlighted in figure 15.1) to return a value with zero elements. A Base64-encoded example of such a configuration file is shown below.

```
MApncnBjIDAwMDAwMDAwMDAwhK2FhYKtMIStMITY2NnY2dnY7w==
```

Figure 15.2: The Base64-encoded `grpc` configuration file

Specifying a configuration file like that in figure 15.2 would cause CoreDNS to panic when attempting to access the first element of the return value.

```
panic: runtime error: index out of range [0] with length 0

goroutine 1 [running]:
github.com/coredns/coredns/plugin/grpc.parseStanza(0xc0002f0900)
	/home/ubuntu/audit-coredns/client-code/coredns/plugin/grpc/setup.go:59 +0x31b
github.com/coredns/coredns/plugin/grpc.parseGRPC(0xc0002f0900)
	/home/ubuntu/audit-coredns/client-code/coredns/plugin/grpc/setup.go:45 +0x5e
github.com/coredns/coredns/plugin/grpc.setup(0x1e4dcc0)
	/home/ubuntu/audit-coredns/client-code/coredns/plugin/grpc/setup.go:17 +0x30
github.com/coredns/caddy.executeDirectives(0xc0000e2900, {0x7ffc15b696e0, 0x31}, {0x324cfa0,
0x31, 0x1000000004b7e06}, {0xc000269300, 0x1, 0x8}, 0x0)
```

```
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:661 +0x5f6
github.com/coredns/caddy.ValidateAndExecuteDirectives({0x2239518, 0xc0002b2980},
0xc0002b2980, 0x0)
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:612 +0x3e5
github.com/coredns/caddy.startWithListenerFds({0x2239518, 0xc0002b2980}, 0xc0000e2900, 0x0)
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:515 +0x274
github.com/coredns/caddy.Start({0x2239518, 0xc0002b2980})
        /home/ubuntu/go/pkg/mod/github.com/coredns/caddy@v1.1.1/caddy.go:472 +0xe5
github.com/coredns/coredns/coremain.Run()
        /home/ubuntu/audit-coredns/client-code/coredns/coremain/run.go:62 +0x1cd
main.main()
        /home/ubuntu/audit-coredns/client-code/coredns/coredns.go:12 +0x17
```

Figure 15.3: CoreDNS panics when loading the `grpc` configuration.

Exploit Scenario

An operator of a CoreDNS server misconfigures the `grpc` plugin, causing a panic. Because CoreDNS does not provide a clear explanation of what went wrong, it is difficult for the operator to troubleshoot and fix the issue.

Recommendations

Short term, verify that the variable returned by `NormalizeExtract` has at least one element before indexing it.

Long term, review the codebase for instances in which data is indexed without undergoing a length check; handling untrusted data in this way may lead to a more severe DoS.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Anonymous Race Condition Proof of Concept

Figure B.1 shows a typical anonymous race condition similar to the one described in TOB-CDNS-1.

The n variable used in the `print` statement on line 15 will always point to the same memory location. However, the values held in that memory location will change. As shown in the output in figure B.2, the race condition occurs because the scheduler cannot keep up with the changes to those values.

```
 1    package main
 2
 3    import (
 4    "fmt"
 5    "time"
 6    )
 7
 8    func main() {
 9        names := []string{"one", "two", "three", "four", "five", "six", "seven", "eight",
"nine", "ten"}
10        closures := make([]func(), 10)
11
12        for i, n := range names {
13            c := func() {
14                    go func() {
15                            fmt.Printf("the name at index %v is %v\n", i, n)
16                    }()
17            }
18            closures[i] = c
19        }
20
21        for _, c := range closures {
22            c()
23        }
24
25        time.Sleep(3 * time.Second)
26    }
```

Figure B.1: A proof of concept for the anonymous race condition (https://go.dev/play/p/VeOjPkWHFHa)

```
the name at index 9 is ten
the name at index 9 is ten
the name at index 9 is ten
the name at index 9 is ten
the name at index 9 is ten
```

```
the name at index 9 is ten
the name at index 9 is ten
the name at index 9 is ten
the name at index 9 is ten
the name at index 9 is ten

Program exited.
```

Figure B.2: The output of the goroutine that experiences a race condition

To solve this issue, create a copy of the values used in the anonymous goroutine, as shown in lines 14–15 in the figure below.

```
 1    package main
 2
 3    import (
 4    "fmt"
 5    "time"
 6    )
 7
 8    func main() {
 9    names := []string{"one", "two", "three", "four", "five", "six", "seven", "eight",
"nine", "ten"}
10    closures := make([]func(), 10)
11
12    for i, n := range names {
13            // capture the values
14            i := i
15            n := n
16
17            c := func() {
18                    go func() {
19                            fmt.Printf("the name at index %v is %v\n", i, n)
20                    }()
21            }
22            closures[i] = c
23    }
24
25    for _, c := range closures {
26            c()
27    }
28
29    time.Sleep(3 * time.Second)
30    }
```

Figure B.3: A fix for the anonymous race condition

C. Fuzzing CoreDNS

During the audit, Trail of Bits used fuzzing, an automated testing technique in which code paths are executed with random data to find bugs resulting from the incorrect handling of unexpected data. We used `dvyukov/go-fuzz`, an in-process coverage-guided fuzzer for Golang, to develop fuzzing harnesses for CoreDNS functions. This utility helped us to find the issues detailed in TOB-CDNS-3, TOB-CDNS-4, TOB-CDNS-9, and TOB-CDNS-15.

We ran the harnesses for a limited period of time. We recommend running them further, such as until the fuzzer does not find input generating new coverage for a few hours or longer. In such a case, we recommend investigating the coverage of all corpus input that the fuzzer generated by creating a small program that executes the fuzzed function with a given payload and instrumenting it for code coverage. This could help to find code paths that were not executed and to manually craft or modify the corpus to achieve higher coverage and find new bugs.

Fuzzing with dvyukov/go-fuzz 101

The `dvyukov/go-fuzz` package provides an AFL-like mutational fuzzing interface, in which testing harnesses can be built entirely in Go. This framework is typically used when a library implemented in Go parses, interprets, or otherwise interacts with blobs of data. An example of such a use case can be seen in figure C.1, in which a harness for the Go standard library's image-processing library is defined.

```
package png

import (
        "bytes"
        "image/png"
)

func Fuzz(data []byte) int {
        png.Decode(bytes.NewReader(data))
        return 0
}
```

Figure C.1: An example test harness for `png.Decode()`, shown in the official `go-fuzz` README

In this example, the function `Fuzz` accepts an array of bytes for `data`, which is then converted into a `Reader` for the `png.Decode` function to read from. When `Fuzz` is compiled and invoked, it is executed repeatedly, using `data` as the input generated for each test case execution.

Optimizing `go-fuzz`'s generation of test case input requires an understanding of return values. Typically, a panic indicates a crash with a given test case input. However, when no crash occurs, but no errors are raised, or errors are raised gracefully, return values can be used to help guide `go-fuzz` to mutate input appropriately.

- A return value of 1 indicates the input generator should increase the priority of a given input during subsequent fuzzing.
- A return value of -1 indicates the input generator should never be added to the corpus, despite the added coverage.
- In all other cases, the function should return `0`.

To build and run this example, you must have Go and the `go-fuzz` package installed. You can then navigate to the directory in which the test harness in figure C.1 is stored and execute `go-fuzz-build` (figure C.2). Assuming the harness builds correctly, it will produce a ZIP file for use with the `go-fuzz` executor. To start the fuzzing harness, you can execute `go-fuzz` in the same directory as the ZIP file produced by `go-fuzz-build` (figure C.3). This will create three directories, if they do not already exist.

```
user@host:~/Desktop/png_fuzz$ ls
png_harness.go
user@host:~/Desktop/png_fuzz$ go-fuzz-build
user@host:~/Desktop/png_fuzz$ ls
png-fuzz.zip  png_harness.go
```

Figure C.2: The generated `png-fuzz.zip` package used by `go-fuzz`

```
user@host:~/Desktop/png_fuzz$ go-fuzz
2019/09/14 16:00:37 workers: 2, corpus: 30 (0s ago), crashers: 0, restarts: 1/0, execs: 0
(0/sec), cover: 0, uptime: 3s
2019/09/14 16:00:40 workers: 2, corpus: 31 (2s ago), crashers: 0, restarts: 1/0, execs: 0
(0/sec), cover: 205, uptime: 6s
2019/09/14 16:00:43 workers: 2, corpus: 31 (5s ago), crashers: 0, restarts: 1/6092, execs:
48742 (5415/sec), cover: 205, uptime: 9s
2019/09/14 16:00:46 workers: 2, corpus: 31 (8s ago), crashers: 0, restarts: 1/7829, execs:
101779 (8481/sec), cover: 205, uptime: 12s
2019/09/14 16:00:49 workers: 2, corpus: 31 (11s ago), crashers: 0, restarts: 1/8147, execs:
146656 (9777/sec), cover: 205, uptime: 15s
2019/09/14 16:00:52 workers: 2, corpus: 31 (14s ago), crashers: 0, restarts: 1/8851, execs:
203582 (11310/sec), cover: 205, uptime: 18s
2019/09/14 16:00:55 workers: 2, corpus: 31 (17s ago), crashers: 0, restarts: 1/8950, execs:
259563 (12360/sec), cover: 205, uptime: 21s
^C2019/09/14 16:00:56 shutting down...
```

Figure C.3: The CLI output of running `go-fuzz` with the `png-fuzz.zip` package

The created directories contain suppressions, crashers, and a corpus, respectively (figure C.4). The suppressions are used to prevent the same message values from being collected every time the fuzzer runs, polluting your crasher samples. The crashers are the program's crashdumps—the STDOUT and STDERR of the program when the test case input causes an error. Finally, the corpus directory stores the test case inputs used throughout the test harness's execution. This directory will collect mutated versions of each input as necessary.

```
user@host:~/Desktop/png_fuzz$ ls -R
.:
corpus  crashers  png-fuzz.zip  png_harness.go  suppressions

./corpus:
21339f0e4b8b5a8e0cb5471f1f91907d1917be50-6
215d99d0c7acdec5ad4c5aa8bec96a171b9ffae0-8
22f545ac6b50163ce39bac49094c3f64e0858403-11
// (...)

./crashers:

./suppressions:
```

Figure C.4: The directory and file output produced by `go-fuzz`

While running the harness on a single machine typically produces good results, `go-fuzz` also supports a clustered mode, allowing test harness execution to scale horizontally across an arbitrary number of worker nodes. More information on this functionality can be found within the repository's README.

Fuzzing CoreDNS Functions

The harnesses we developed required us to use the data slice to generate input for the various fuzzed functions.

We discovered the issue detailed in TOB-CDNS-9 by using the following harness. A unit test that replicates the panic described in that finding is included in figure C.6.

```
package kubernetes

import "github.com/trailofbits/go-fuzz-utils"

func Fuzz_parseRequest(data []byte) int {
        type FuzzStructure struct {
                name string
                zone string
        }

        var testStructure FuzzStructure

        tp, err := go_fuzz_utils.NewTypeProvider(data)
        if err != nil {
                return 0
        }

        err = tp.Fill(&testStructure)
        if err != nil {
                return 0
        }

        parseRequest(testStructure.name, testStructure.zone)

        return 0
}
```

Figure C.5: A fuzzing harness targeting the `parseRequest` function of the `kubernetes` plugin

```
func TestCrash(t *testing.T) {
	k := New([]string{"interwebs.test."})
	k.APIConn = &APIConnServiceTest{}

	type svcTest struct {
		qname  string
		qtype  uint16
	}
	test := svcTest {
		// Cluster IP Service
		qname: ".o.o.po.pod.8", qtype: dns.TypeA,
	}

	state := request.Request{
		Req:  &dns.Msg{Question: []dns.Question{{Name: test.qname, Qtype:
test.qtype}}},
		Zone: "interwebs", // must match from k.Zones[0]
	}
	_, _ = k.Services(context.TODO(), state, false, plugin.Options{})
}
```

Figure C.6: A unit test that replicates the panic described in TOB-CDNS-9

D. Rapid Risk Assessment Template

This appendix provides a template for a Rapid Risk Assessment meant to complement and extend the threat model presented in this report. Refer to the Mozilla documentation for additional guidance on conducting this type of assessment.

Overview

- Component:
- Owner(s):
- SIG/WG(s) at Meeting:
- Service Data Classification:
- Highest Risk Impact:

Service Notes

This section should walk through the components, describe the connections between the components and their relevant controls, and explain how the components fulfill their roles. This section can also include questions about the components. For example, with a component that accepts an HTTP connection, there may be questions about channel security (TLS and cryptography), authentication, authorization, non-repudiation/auditing, and logging. These questions are meant to guide discussions and to keep meetings and calls on track but should not be the only drivers of discussions. Examples are provided below.

- How does the service work?
- Are there any subcomponents or shared boundaries?
- What communication protocols does it use?
- Where does it store data?
- What is the most sensitive data it stores?
- How is that data stored?

Data Dictionary

Data Dictionary		
Category	**Description**	**Comments**
Data		

Control Families

These areas of control are chosen by the audit working group.

In this context, a “control” is a logical section of an application or system that handles a security requirement. According to CNSSI,

> The management, operational, and technical controls (i.e., safeguards or countermeasures) prescribed for an information system to protect the confidentiality, integrity, and availability of the system and its information.

For example, a system may have authorization requirements such as the following:

- Users must be registered with a central authority.
- Every request must be verified to be owned by the user who issued it.
- Each user account must have unique attributes that can identify the user.

In this assessment, we are looking at five basic control families:

- Networking
- Cryptography
- Secret Management
- Authentication
- Authorization (Access Controls)

Obviously, we can consider a control family to be “not applicable” if a component does not require it. For example, a component with the sole purpose of interacting with the local file system may not have a meaningful networking component. If a control family is simply not applicable, it is not considered a weakness.

For each control family, we want to ask the following:

- What does the component do for this control?
- What sorts of data pass through that control?
    - For example, a component may have sensitive data (secret management) that never leaves the component's storage via networking.
- What can an attacker do with access to this component?
- What is the simplest way to attack it?
- Are there any safeguards that we recommend (e.g., “Always use an interstitial firewall”)?

- What happens if the component stops working (because of a DoS or another condition)?
- Have there been similar attacks in the past? What were the mitigations?

Threat Scenarios

Identify potential attack scenarios, considering the current state of the system being tested and the responses to the previous sections of this template.

- An external attacker without access to the client application
    - Attack scenario #1
    - Attack scenario #2