RADICALLY OPEN SECURITY

Penetration Test Report

Open Tech Fund

V 1.0
Diemen, September 13th, 2018
Confidential

Document Properties

Client	Open Tech Fund
Title	Penetration Test Report
Target	github.com/measurement-kit/libndt git revision: 89193025a4c59793b2f03d590efdaaea20c8cf58
Version	1.0
Pentester	Stefan Marsiske
Authors	Stefan Marsiske, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	September 10th, 2018	Stefan Marsiske	Main report
0.2	September 13th, 2018	Marcus Bointon	Review
1.0	September 13th, 2018	Marcus Bointon	Final version

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Overdiemerweg 28 1111 PP Diemen The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between September 1, 2018 and September 10, 2018, Radically Open Security B.V. carried out a code audit for Open Tech Fund.

This report contains our findings as well as detailed explanations of exactly how ROS performed the code audit.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- github.com/measurement-kit/libndt git revision: 89193025a4c59793b2f03d590efdaaea20c8cf58

1.3 Project objectives

Audit the code of libndt for security issues and proper usage of 3rd party dependencies.

1.4 Timeline

The Security Audit took place between 1st September and 10th September, 2018.

1.5 Results In A Nutshell

The usage of 3rd party dependencies appear correct. Curl is used sparingly and when it is, it's using only the high-level abstracted interface, which is appropriate. The JSON parser seems to be rock solid after more than 10 days of fuzzing. OpenSSL is also only used minimally, again using a high-level interface appropriately.

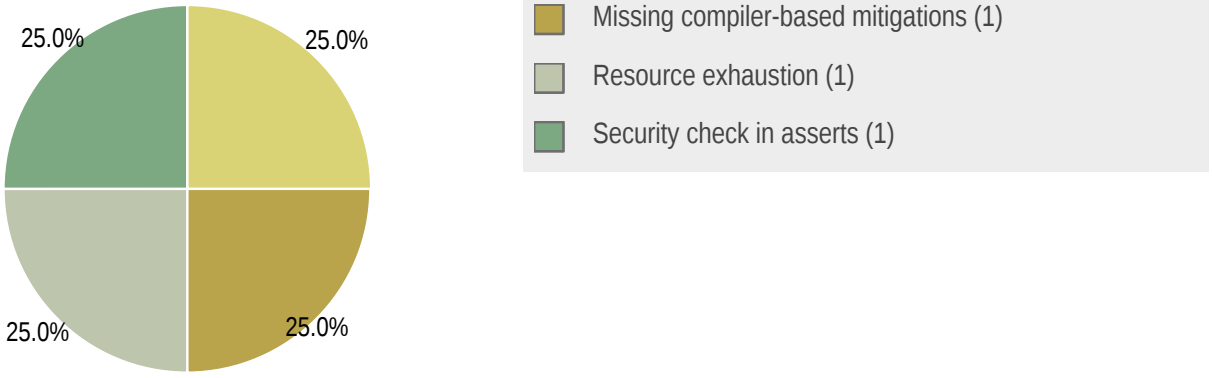
1.6 Summary of Findings

ID	Type	Description	Threat level
NDT-001	Unauthenticated Communication	Some MitM could impersonate the NDT server or inject/ modify packages in transit.	High

NDT-002	Missing Compiler-based Mitigations	Compiler hardening switches are missing from the build scripts. These switches provide protective measures that make exploitation harder.	Moderate
NDT-003	Resource Exhaustion	The function query_mlabns() uses curl to fetch a JSON array, which is completely unbounded and can lead to resource exhaustion or other issues.	Moderate
NDT-004	Security Check in Asserts	Since assert() can be disabled at compile-time by defining the NDEBUG symbol, doing so creates many security issues.	Low

1.6.1 Findings by Threat Level

1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
NDT-001	Unauthenticated Communication	Authenticate all messages at the protocol level, or enforce TLS and provide a hard-coded CA for verifying TLS certificates.
NDT-002	Missing Compiler-based Mitigations	Enable hardening switches in the build scripts.
NDT-003	Resource Exhaustion	Limit the size of the reply and abort if excessive data is received, or if the received response does not parse due to missing terminators.
NDT-004	Security Check in Asserts	Review and replace all security-relevant asserts with code that cannot be disabled.

2 Methodology

2.1 Planning

Our general approach during this code audit was as follows:

1. **Code reading**
   We read through all the code, in multiple phases we first read through sequentially to get a feel for the code, then in a second round we go through the code in the logical order the code is executed.
2. **Grepping**
   We attempted to identify areas of interest by searching for memory operations: new, delete, malloc, calloc, realloc, gralloc, free.
3. **Static checks**
   We also used two automated tools, flawfinder and cppcheck, to look for issues. Besides lots of false positives cppcheck came up with some compiler optimization recommendations.
4. **Fuzzing**
   For good measure we tried to fuzz the most exposed component - the JSON engine.

2.2 Risk Classification

Throughout the document, vulnerabilities or risks are labeled and categorized as:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.
- **High**
  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**
  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**
  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**
  Low risk of security controls being compromised with measurable negative impacts as a result.

Please note that this risk rating system was taken from the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting.

3 Reconnaissance and Fingerprinting

Through automated scans we were able to gain the following information about the software and infrastructure. Detailed scan output can be found in the sections below.

3.1 Automated Scans

As part of our active reconnaissance we used the following automated scans:

- flawfinder – https://www.dwheeler.com/flawfinder/
- cppcheck – http://cppcheck.sourceforge.net/
- clang-analyzer – https://clang-analyzer.llvm.org/

4 Pentest Technical Summary

4.1 Findings

We have identified the following issues:

4.1.1 NDT-001 — All Communication Is Unauthenticated

Vulnerability ID: NDT-001

Vulnerability type: Unauthenticated Communication

Threat level: High

Description:

Some MitM could impersonate the NDT server or inject/modify packages in transit.

Technical description:

All NDT messages are unauthenticated. The only exception if TLS is enabled, but this seems optional. For strong adversaries who are in a MitM position and also have their own CA which is included in most Operating Systems Certificate Authority stores, even this is not a big hurdle. Since NDT is about discovering and evaluating MitM adversaries this is a significant issue.

Impact:

A MitM attacker could DoS unrelated 3rd parties or to fake results with its own NDT server.

Recommendation:

Authenticate all messages at the protocol level, or enforce TLS and provide a hard-coded CA for verifying TLS certificates.

4.1.2 NDT-002 — Basic Compiler Hardening Missing

Vulnerability ID: NDT-002

Vulnerability type: Missing Compiler-based Mitigations

Threat level: Moderate

Description:

Compiler hardening switches are missing from the build scripts. These switches provide protective measures that make exploitation harder.

Technical description:

No hardening switches are enabled at all, a generic set of switches for linux-based systems might be:

```
-D_FORTIFY_SOURCE=2
-fstack-protector-strong
-Wformat
-Werror=format-security
-Wl,-z,relro,-z,now
-Wconversion -Wsign-conversion
-mmitigate-rop
-ftrapv
```

On Windows-based systems these switches might be useful:

```
/DYNAMICBASE
/SafeSEH
/GS
/NXCompat
```

For more detailed information consult https://www.owasp.org/index.php/C-Based_Toolchain_Hardening

Impact:

If vulnerabilities are found, the lack of these protections makes exploitation cheaper and easier.

Recommendation:

Enable hardening switches in the build scripts.

4.1.3 NDT-003 — Unlimited Response From Query_mlabns

Vulnerability ID: NDT-003

Vulnerability type: Resource Exhaustion

Threat level: Moderate

Description:

The function `query_mlabns()` uses curl to fetch a JSON array, which is completely unbounded and can lead to resource exhaustion or other issues.

Technical description:

Unlike the rest of the code `query_mlabns()` uses curl to download a list of servers, and unlike the rest of the code this network traffic is unbounded, an adversary could send a huge response exhausting local resources or possibly redirect traffic to unrelated 3rd parties effectively DoS-ing them.

Impact:

The client could run out of memory, or consume lots of network traffic, possibly even initiating connections to unrelated 3rd parties.

Recommendation:

Limit the size of the reply and abort if excessive data is received, or if the received response does not parse due to missing terminators.

4.1.4 NDT-004 — Asserts Used for Security Checks

Vulnerability ID: NDT-004

Vulnerability type: Security Check in Asserts

Threat level: Low

Description:

Since `assert()` can be disabled at compile-time by defining the `NDEBUG` symbol, doing so creates many security issues.

Technical description:

Some examples from the code to show the problem:

```
1239:  assert(fqdns != nullptr);
```

or

```
3391:  assert(base != nullptr && actual != nullptr);
```

Impact:

Compilation with `-DNDEBUG` can introduce security bugs.

Recommendation:

Review and replace all security-relevant asserts with code that cannot be disabled.

4.2 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

4.2.1 Static Checks

A number of static checks were performed:

- clang-analyzer: did not find anything
- flawfinder: found a few false-positives
- cppcheck: found a few possible compiler optimizations (e.g. static or const modifiers to functions or parameters)

Recommendation: subscribe to the free tier of Coverity for free software and regularly run scans with them.

4.2.2 Checking Heap Operations

A major issue with C/C++ programs is incorrect heap usage, so we explicitly check this aspect in code audits. In this case we could not find any invocations of `new()`, `delete()`, `malloc()`, `realloc()`, `calloc()` or `free()`. A few invocations of `SSL_free()` were spotted, but those seem to be used correctly.

4.2.3 Fuzzing of the JSON Engine

For good measure we fuzzed the JSON engine used in libndt. In total we ran AFL for more than 10 days on 8 dedicated CPUS, totalling in more than 66K cycles and more than 11G invocations of the JSON parser, this effort did not find any crashes.

5 Future Work

- **Fuzz the websocket implementation.**
  Since libndt contains a homebrew implementation of a websocket client it makes sense to fuzz this specifically.
- **Apply to the Free Software program of Coverity**
  Have the code regularly checked by Coverity for newly introduced bugs.
- **Consider amending the NDT protocol so that all messages are authenticated.**
  As the goal of NDT is to uncover MitM attackers interfering with network traffic the lack of authentication makes NDT an easily circumvented and abused tool for exactly the adversaries it tries to detect.

6 Conclusion

The audited code is a small, well-written library with only a small scope, written in modern C++ with good inline documentation. Very few issues were identified and are either minor and difficult to trigger or of a cosmetic nature. The only serious issue is that the NDT protocol itself is unauthenticated and allows MitM attackers to arbitrarily interact with the tests conducted. This is especially concerning as the purpose of these tests is to identify MitM interfering with the traffic.

Finally, let us emphasize that security is a process, and this audit is just a single snapshot. Security must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Do not hesitate to let us know if you have any further questions or need further clarification of anything in this report.

Appendix 1 Testing team

Stefan Marsiske	Stefan runs workshops on radare2, embedded hardware, lock-picking, soldering, gnuradio/SDR, reverse-engineering, and crypto topics. In 2015 he scored in the top 10 of the Conference on Cryptographic Hardware and Embedded Systems Challenge. He has run training courses on OPSEC for journalists and NGOs.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.