Google

libzip – 2023 Source Code Review & Fuzzing

FINAL REPORT

limitless innovation. no compromise.

Prepared for: Dave Tamasi
Security TPM - Engineering

OSTIF

Thomas Klausner
LibZip package maintainer

October 23, 2023

PO#: 4100346637
Testing Dates: 08/23/23 - 10/19/23
Actual Person Days: 9.5 days / 1FTE

Version:	Final
Prepared for:	Google LLC
Date:	October 23, 2023

Table of Contents

Executive Summary

Google LLC engaged Leviathan Security Group to perform a time-bound security assessment of the `libzip` library. We performed this assessment from August 23, 2023 through October 19, 2023.

Our objectives were to review the `libzip` source code and increase the fuzzing coverage for the project in `OSS-Fuzz`. The source code review was performed both manually and using automated source code scanning methods. The work was informed and guided through threat modeling documentation.

Our review uncovered no vulnerability findings.

Observations

The time-bound source code evaluation focused on discovering vulnerabilities in the `libzip` (https://github.com/nih-at/libzip) project using manual review and static application security testing tools (SAST). Manual code review was focused on important security controls such as general input validation, cryptographic protocols, and proper memory management. The SAST analysis was centered around a holistic examination of the library.

The library uses up-to-date third-party dependencies. The compression and decompression mechanisms of zip archives were found to work correctly and mitigate many attack vectors.

Overall, the library follows best security practices. It properly validates input data, such as filesystem path or passwords for encrypted files. The implementation is not prone to memory leaks and corruptions. Although the insecure memory filling function `memset()` was noted in several instances, it does not pose any direct threats. No buffer overflow primitives were observed during the evaluation.

The main areas of focus were:

- Memory management
- Crypto primitives implementation
- Compression and decompression design
- Files handling flow
- Error handling flow

The reviewed functionality includes:

- zip archives handling (`lib/zip_open.c`; `lib/zip_source_file_win32.c`)
- compression and decompression of zip archives (`lib/zip_algorithm_bzip2.c`; `lib/zip_algorithm_deflate.c`; `lib/zip_algorithm_xz.c`; `lib/zip_algorithm_zstd.c`)
- files & buffers handling (`lib/zip_file_add.c`; `lib/zip_file_replace.c`; `lib/zip_source_buffer.c`; `lib/zip_source_open.c`; `lib/zip_buffer.c`)
- implementation of cryptographic functions (`zip_source_pkware_decode.c`; `zip_source_pkware_encode.c`; `lib/zip_source_winzip_aes_decode.c`; `lib/zip_source_winzip_aes_encode.c`; `lib/zip_crypto_commoncrypto.c`; `lib/zip_crypto_win.c`; `lib/zip_get_encryption_implementation.c`)

- usage of potentially unsafe functions, insecure RNGs, deprecated cryptography APIs (`lib/*`)

As part of the assessment, we also enhanced fuzzing coverage. Fuzzing was dedicated to finding bugs, memory leaks, and security vulnerabilities in the `libzip` library. While developing the new fuzz targets, we focused on library encryption and decryption algorithms (AES and PKWARE) as well as archive handling functions (`zip_open()`; `zip_open_from_source()`). Each fuzz target covers a specific functionality and contains comments on it. 4 new fuzz targets and 1 new corpus were developed as a result of expanding the fuzz testing coverage.

The fuzzing function coverage increased from 56.9% to 84.33% after running 21 days on the `OSS-Fuzz` infrastructure. The expanded fuzzing coverage has yielded several crashes that have been communicated to us by the repository maintainers:

- A memory leak in error handling (`lib/zip_source_zip_new.c`) detected by ASAN (AddressSanitizer) has been confirmed as an issue and fixed by the `libzip` developers (commit 479c7afa6318e5d6b16915381b176863a906b4d0); The origin of the issue is a function `zip_source_zip_file_create()` in the `lib/zip_source_zip_new.c` file, which posed a lack of freeing zip data source after setting an error;
- 2 out-of-memory crashes in the `xz` library used by `libzip` as a (de)compression back end. After review, these crashes were determined to be false positives. The reason for these 2 crashes is that ASAN has a default memory limit of 2560 MB, while the `lzma` function (from `xz`) attempts to allocate more than 4GB of memory.

Fuzzing Process and Challenges Overview

We submitted several pull requests (PRs) to improve fuzzing coverage and reached out to the repository maintainers for feedback. There was a week-long wait before we received information on the changes required for our PR to be accepted. After making the adjustments promptly, our PR was approved.

However, a few hours after approval, the maintainers made commits that altered the `OSS-Fuzz` setup and fuzzer targets. Unfortunately, on the third day, the project was removed from the `OSS-Fuzz` platform, halting the fuzzing process. This removal was due to the new fuzz targets being broken by the recent commits. In response, we identified and communicated the issue to the developers, providing suggestions for resolution. The maintainers acted on this feedback, fixing the problem within 2 days.

Following this, we encountered another challenge related to coverage report generation on the `OSS-Fuzz` side. We reported this issue in the `OSS-Fuzz` repository. The Google team acknowledged this was not a false alarm, but a genuine infrastructure problem, which was resolved in 3 days without any action on our part.

After the infrastructure issue was resolved, we observed that the maintainers had made further changes to the setup, disrupting the configuration as indicated by the `OSS-Fuzz` logs. This required us to conduct an additional investigation and make the necessary adjustments to the setup. Once these corrections were made, the fuzzer targets could operate, and coverage reports were successfully generated. This issue was resolved within a day after discussions with the maintainers.

Overall Feedback:

The process involved a 2-week wait due to issues originating from external parties. In addition, as we do not have access to the `OSS-Fuzz` platform reports, it creates additional barrier in tracking the results of fuzzing coverage expansion. According to provided crashes, the `libzip` developers received and successfully addressed memory leaks in areas covered by the new fuzzers (commit 479c7afa6318e5d6b16915381b176863a906b4d0). This suggests that the expanded fuzzing coverage aided in identifying new bugs.

Recommendations

Analyze all crashes reported by `OSS-Fuzz` and fix all sequentially identified bugs and vulnerabilities.

Vulnerability Classification

Impact

When we find a vulnerability, we assign it one of five categories of severity, describing the potential impact if an attacker were to exploit it:

Informational – Does not present a current threat but could pose one in the future if certain changes are made. To protect against future vulnerabilities, fixing the condition is advisable.

Low – May allow an attacker to gain information that could be combined with other vulnerabilities to carry out further attacks. May allow an attacker to bypass auditing or minimally disrupt availability, resulting in minor damage to reputation or financial loss.

Medium – May allow an attacker inappropriate access to business assets such as systems or servers. There may be impact to the confidentiality or integrity of data, or limited disruption of availability, resulting in moderate damage to reputation or financial loss.

High – May allow an attacker inappropriate access to business assets such as systems or servers. There may be substantial or widespread impact to the confidentiality or integrity of particularly sensitive data, or disruption of availability, resulting in significant damage to reputation or financial loss.

Critical – May allow an attacker to gain persistence, or imminently disrupt functionality or disclose data, resulting in severe reputational damage or financial loss.

Skill Level to Exploit

When we find a vulnerability, we assess how skilled an attacker must be to exploit it:

Simple – Requires minimal understanding of the underlying technology. Tools/ techniques for exploiting the vulnerability can be easily found on the internet.

Moderate – Requires significant expertise, possibly in proprietary information, or access to tools that are not readily available to individuals. The unwitting cooperation of a victim or target may also be required.

Advanced – Requires insider access or access to tools that are not publicly available. Successful exploitation of another vulnerability may be required. Direct interaction with the victim or target may also be required.

Impact Rating (Weight)	Skill Level to Exploit Rating (Weight)		
	Advanced (1)	Moderate (2)	Simple (3)
Critical (4)	4	8	12
High (3)	3	6	9
Medium (2)	2	4	6
Low (1)	1	2	3

Severity	
Critical	10-12
High	7-9
Medium	4-6
Low	1-3

Vulnerability Index

This section represents a quick view into the vulnerabilities discovered in this assessment.

ID	SEVERITY	TITLE	COMPONENT
2112430	Info	Insecure "memset" function in use	Source code

Activity Index

This section represents a quick view into the activities performed in this assessment.

COMPONENT	TITLE	STATUS
Source code	Coding best practice	Complete
Source code	Insecure functions	Complete

Observations & Analysis

For the purposes of evaluation, we grouped all aspects of this assessment into a single component, based on design documentation and discussions with the product team.

Source code

`libzip` is an open-source C language library for reading, creating, and modifying zip and zip64 archives. Files on a stage of compressing can be added from data buffers, files, or compressed data copied directly from other zip archives. Decryption and encryption of files encrypted with Winzip AES and legacy PKware is supported.

Threat Analysis

`libzip` could be vulnerable to common types of attacks such as buffer overflow, memory corruption, or failure of cryptographic primitives. Exploitation of the library could lead to information disclosure or denial of service (DoS). If exploited, these vulnerabilities could allow attackers to compromise the confidentiality and availability of client applications that use the library.

Observations

`libzip` follows security best practices and properly validates user input and handles errors. The library is protected from memory corruption.

However, we detected the use of an insecure function to fill memory chunks, which could lead to information disclosure, though this security issue does not pose any direct threats and is not exploitable.

Activities Performed

CODING BEST PRACTICE

Scope

Verify that coding best practices are being followed.

Methodology

Ensure that every return value is validated and memory is properly cleared when necessary. Verify that file descriptors, and the like, are used rather than file-name operations. Verify that preprocessor macros, and the like, are used in a safe manner. Buffer overflow primitives and format string vulnerabilities should not be present. Ensure that all pointer variables are properly initialized to NULL or a valid memory address before use to prevent undefined behavior.

Observations

This activity took 5 hours to complete. We observed that all data is properly sanitized and memory is properly cleared after memory allocation and usage. We did not detect any buffer overflow primitives.

Related Findings

No findings are associated with this activity.

INSECURE FUNCTIONS

Scope

Review code to see if insecure functions are being used.

Methodology

Review manually and through static application security testing with the semgrep tool. Review the code for potentially insecure functions, such as `strcpy`, `malloc`, and `printf`, and determine whether they are used in an insecure manner.

Our work included the following checks:

- Compiler safe memset in use
- Width of memory is defined for file reading

Observations

This activity took 6 hours to complete. We observed that the application relies on security best practices regarding the use of insecure functions.

However, we found that the application uses the insecure function `memset()` to fill memory chunks instead of the secure version `memset_s()`.

Related Findings

2112430: Insecure "memset" function in use

Vulnerabilities

INSECURE "MEMSET" FUNCTION IN USE

ID	2112430
Component	Source code
Severity	**Info**
Impact / Skill Level	Informational/Advanced
Reference	https://cwe.mitre.org/data/definitions/244.html https://stackoverflow.com/questions/246127/why-is-volatile-needed-in-c
Location	lib/
CVSS Score	0 (CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:N/E:X/RL:X/RC:X/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X)
CWE Category	CWE-14: Compiler Removal of Code to Clear Buffers

Observation

Optimizing compilers may optimize away code that writes to memory that is not subsequently read by the program. This can cause vulnerabilities in applications that try to erase sensitive data (e.g., passwords or private keys) by overwriting it.

While performing static application security testing (semgrep version 1.21.0 was used), we checked for common insecure functions in the library. We observed that the **`libzip`** library relies on an insecure function to fill memory chunks. While the **`memset()`** function is often used to clear data from buffers prior to deletion or reuse, compiler optimization or other factors may cause it to leave sensitive information intact.

`libzip` uses the **`memset`** function to initialize structures but does not use it to overwrite structures with null values. Consequently, this issue does not pose a direct threat and is not exploitable.

Source code files containing insecure functions:

- lib/zip_algorithm_xz.c
- lib/zip_crypto_openssl.c
- lib/zip_crypto_win.c
- lib/zip_dirent.c
- lib/zip_winzip_aes.c

Impact Rationale:
An attacker could exploit improper handling of sensitive information in the buffer to gain unauthorized access to critical data.

Difficulty Rationale:
An attacker would need to exploit an insecure function.

INSECURE "MEMSET" FUNCTION IN USE

Recommendation

Ensure that memory buffers containing sensitive information such as passwords and encryption keys are zeroed prior to freeing the buffer. To ensure that sensitive data is securely overwritten, it is best to use system-provided secure-erase functions such as memset_s. If such functions are not available, then the data to be erased must be marked “volatile” when it is declared.

Appendix A – Technical Services

Leviathan's Technical Services group brings deep technical knowledge to your security needs. Our portfolio of services includes software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. Our goal is to provide your organization with the security expertise necessary to realize your goals.

SOFTWARE EVALUATION We provide assessments of application, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of software. Our work includes design and architecture reviews, data flow and threat modeling, and code analysis using targeted fuzzing to find exploitable issues.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, embedded systems, mobile devices, and consumer-facing end products, to core networking equipment that powers internet backbones.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated attackers. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team an understanding of the overall security posture of your organization as well as the details of discovered vulnerabilities.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology, allowing us to find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This methodology gives your team a stronger assurance that the most significant security-impacting flaws have been found, allowing your team to address them.

INCIDENT RESPONSE & FORENSICS We respond to our customers' security incidents by providing forensics, malware analysis, root cause analysis, and recommendations for how to prevent similar incidents in the future.

REVERSE ENGINEERING We assist clients with reverse engineering efforts. We provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

Appendix B – Risk and Advisory Services

Leviathan's Retained Services group is a supplement to an organization's security and risk management capability. We offer a pragmatic information security approach that respects our clients' appetites for security process and program work. We provide access to industry leading experts with a broad set of security and risk management skills, which gives our clients the ability to have deep technical knowledge, security leadership, and incident response capabilities when they are needed.

INFORMATION SECURITY STRATEGY DEVELOPMENT We partner with boards, directors, and senior executives to shape your enterprise's overall approach to meeting information security requirements consistently across an entire organization.

ENTERPRISE RISK ASSESSMENT We develop an information asset-centric view of an organization's risk that provides insight to your organization's Enterprise Risk Management capability. This service can be leveraged with annual updates, to account for your organization's changing operations, needs, and priorities.

PRIVACY & SECURITY PROGRAM EVALUATION We evaluate your organization's existing security program to give you information on compliance with external standards, such as ISO 27000 series, NIST CSF, HIPAA, or PCI-DSS. This is often most useful before a compliance event or audit and helps to drive the next phase of growth for your Security and Risk Management programs.

VENDOR RISK ASSESSMENT We assess the risk that prospective vendors bring to your organization. Our assessment framework is compatible with legislative, regulatory, and industry requirements, and helps you to make informed decisions about which vendors to hire, and when to reassess them to ensure your ongoing supply chain security.

NATIONAL & INTERNATIONAL SECURITY POLICY In 2014, we launched a public policy research and analysis service that examines the business implications of privacy and security laws and regulations worldwide. We provide an independent view of macro-scale issues related to the impact of globalization on information assets.

M&A/INVESTMENT SECURITY DUE DILIGENCE We evaluate the cybersecurity risk associated with a prospective investment or acquisition and find critical security issues before they derail a deal.

LAW FIRM SECURITY SERVICES We work with law firms as advisors, to address security incidents and proactively work to protect client confidences, defend privileged information, and ensure that conflicts do not compromise client positions. We also work in partnership with law firms to respond to their clients' security needs, including in the role of office and testifying expert witnesses.

SAAS AND CLOUD INITIATIVE EVALUATION We give objective reviews of the realistic threats your organization faces both by moving to cloud solutions and by using non-cloud infrastructure. Our employees have written or contributed to many of the major industry standards around cloud security, which allows their expertise to inform your decision-making processes.