Code Audit Report

Least Authority TFA GmbH

V 1.0
Amsterdam, September 7th, 2022
Public

Document Properties

Client	Least Authority TFA GmbH
Title	Code Audit Report
Targets	Client Library: https://github.com/LeastAuthority/wormhole-william Native extension library: https://github.com/LeastAuthority/dart_wormhole_william Flutter app/frontend: https://github.com/LeastAuthority/flutter_wormhole_gui Mailbox (rendezvous) server: https://github.com/magic-wormhole/magic-wormhole-mailbox-server Transit relay: https://github.com/magic-wormhole/magic-wormhole-transit-relay
Version	1.0
Pentester	Jonathan Levin
Authors	Jonathan Levin, Peter Mosmans
Reviewed by	Peter Mosmans
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	May 26th, 2022	Jonathan Levin	Initial draft
0.2	June 29th, 2022	Peter Mosmans	Minor textual edits
0.3	June 30th, 2022	Jonathan Levin, Peter Mosmans	Minor textual and style edits
0.4	August 23rd, 2022	Jonathan Levin	Retest and Verification
0.5	August 29th, 2022	Peter Mosmans	Minor textual and style edits, show only open recommendations
1.0	September 7th, 2022	Peter Mosmans	Minor grammar corrections, declassify report

Contact

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

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255

Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between May 17, 2022 and June 29, 2022, Radically Open Security B.V. carried out a penetration test for Least Authority TFA GmbH.

On July 30, 2022, Least Authority TFA GmbH provided a response to the initial findings along with proposed fixes. Between August 22 and August 25, 2022, Radically Open Security B.V. verified the proposed fixes and retested the findings.

This report contains our findings, retest and verification results, as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the code audit was limited to the following targets:

- Client Library: https://github.com/LeastAuthority/wormhole-william
- Native extension library: https://github.com/LeastAuthority/dart_wormhole_william
- Flutter app/frontend: https://github.com/LeastAuthority/flutter_wormhole_gui
- Mailbox (rendezvous) server: https://github.com/magic-wormhole/magic-wormhole-mailbox-server
- Transit relay: https://github.com/magic-wormhole/magic-wormhole-transit-relay

The target repository sources and specific commit hashes for the initial review:

- Client Library: LeastAuthority/wormhole-william - `9087ef25a20fff714a9ad07ca3a208a44468bede`
- Magic Wormhole Transit Relay - `80e02d4a77be5abfbca966d609408f6927cb3843`
- Magic Wormhole Rendezvous Server - `de2f1b4dd4902f2a8459e6b0778e5665caef8a9d`
- Native extension library - `07fc72a34e9dcb20533ed2967f58b32ad0103ce0`
- Flutter GUI - `94ac6fb9dd7a723a1c6f4ff05394c4b21e161285`

- Audit of wormhole-william: 4 days
- Audit of magic-wormhole-rendezvous-server: 2 days
- Audit of magic-wormhole-transit-relay: 1 days
- Audit of dart-wormhole-william: 1 days
- Audit of flutter-wormhole-gui: 2 days
- Reporting: 2 days
- Retest and fix verification: 1.5 days

- **Total effort: 13.5 days**

1.3 Project objectives

ROS will perform a code audit of the target libraries in order to assess their security under realistic conditions. To do so ROS will analyze the repositories and guide Least Authority in attempting to find vulnerabilities, exploiting those found to evaluate their impact, and recommend suitable solutions or mitigations.

1.4 Timeline

The initial Security Audit took place between May 17, 2022 and June 29, 2022.

Retesting and verification of the client's proposed mitigations took place between August 22, 2022 and August 25, 2022.

1.5 Results In A Nutshell

During this code audit I found 1 Elevated, 2 Moderate, and 3 Low-severity issues.

The six findings represent a diverse set of security issues. One finding concerns the improper retention of information on the rendezvous server. Another relates to vulnerability to denial of service. Two findings represent errors during the receipt of a file directory, and two relate to improper error checks and input validation.

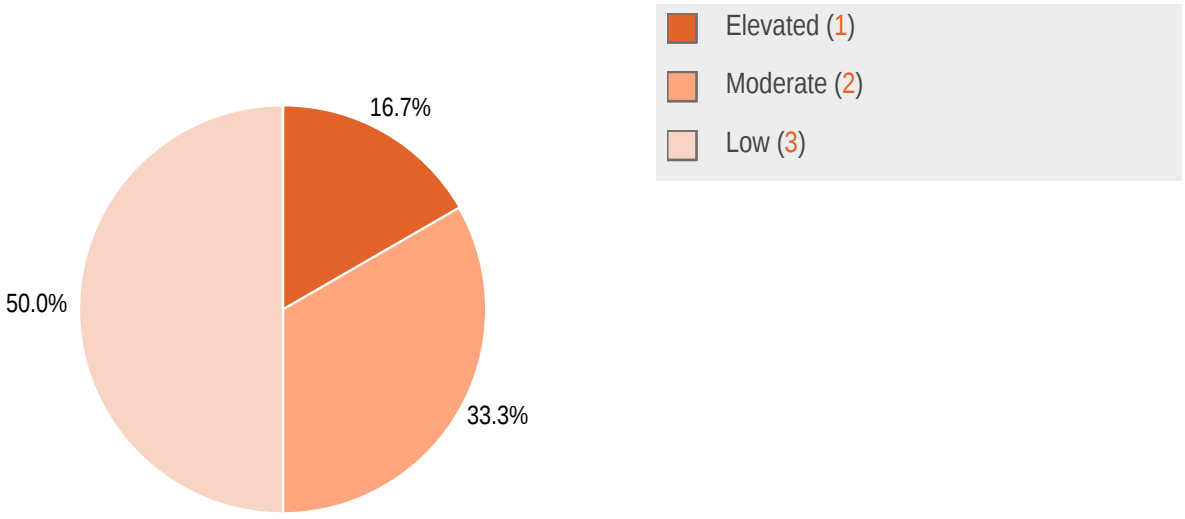
By exploiting these issues, an attacker may be able to cause a recipient to accept unexpected files, recover sensitive information, and/or interrupt service.

1.6 Summary of Findings

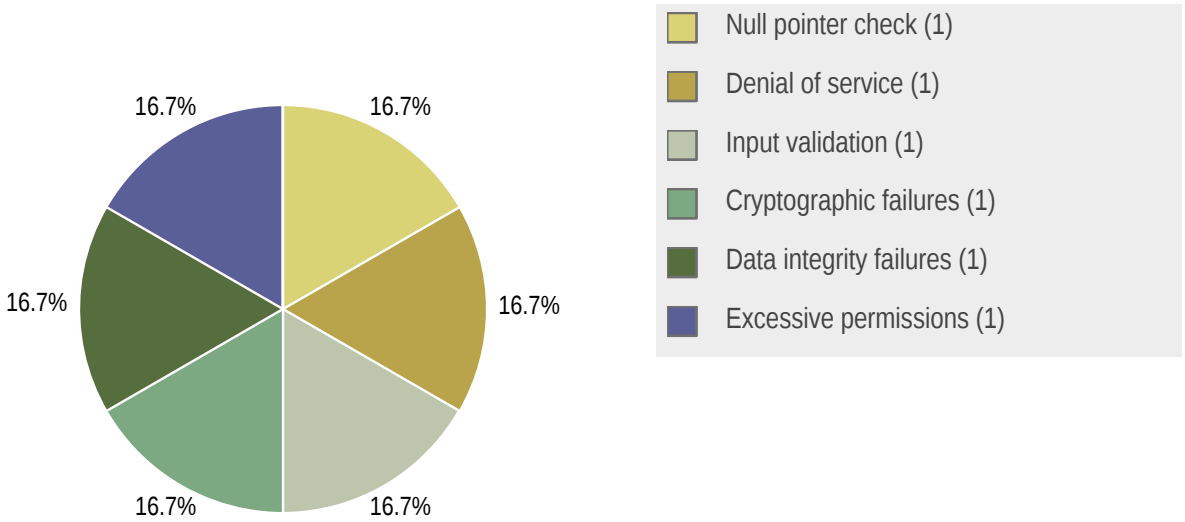
ID	Type	Description	Threat level
LMW-007	Data Integrity Failures	When receiving a file directory, receiver unzips the received archive without checking if it matches what was offered.	Elevated
LMW-009	Input validation	Both magic-wormhole-mailbox-server and magic-wormhole-transit-relay rely almost exclusively on assert statements for type checking and input validation.	Moderate
LMW-001	Excessive Permissions	Receiver sets default directory/file permissions to 0777/0666, respectively	Moderate
LMW-011	NULL Pointer Check	Function char *copy_str(const char *str) in dart-wormhole-william uses a malloc -allocated buffer without checking if it is NULL.	Low

LMW-010	Denial of Service	A network attacker can prevent mailboxes from closing and its messages from being deleted.	Low
LMW-008	Cryptographic Failures	By default Magic Wormhole uses plaintext websockets, which allows a man-on-the-side attacker to read protocol messages between client and rendezvous server and insert packets into the TCP stream.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of New and Open Recommendations

ID	Type	Recommendation
LMW-010	Denial of Service	The scenario given in the impact section above assumes a high-resource attack and is also very targeted. Given that being able to tolerate disconnecting and reconnecting clients is a feature of the protocol, deleting messages immediately after sending them may be an overcompensation for the given risk. Another solution would be to change the pruning routine to delete messages based on time since creation rather than time of last connection. This means that even if a client disconnects for 10 minutes their message will be there when they reconnect, but importantly, even if they never disconnect, the message is guaranteed not to be there after 11 minutes.
LMW-009	Input validation	Replace `assert` with `try` / `except` clauses and/or explicitly cast untrusted input values to the required type, where appropriate.
LMW-008	Cryptographic Failures	Use websockets over TLS by default.

2 Methodology

2.1 Planning

During the code audit we verify if the proper security controls are present, work as intended and are implemented correctly. If vulnerabilities are found, we determine the threat level by assessing the likelihood of exploitation of this vulnerability and the impact on the Confidentiality, Integrity and Availability (CIA) of the system. We will describe how an attacker would exploit the vulnerability and suggest ways of fixing it.

This requires an extensive knowledge of the platform the application is running on, as well as the extensive knowledge of the language the application is written in and patterns that have been used. Therefore a code audit is done by highly-trained specialists with a strong background in programming.

During this code audit, we take the following approach:

1. **Thorough comprehension of functionality**
   We try to get a thorough comprehension of how the application works and how it interacts with the user and other systems. Having detailed documentation at this stage is very helpful, as it aids the understanding of the application.
2. **Comprehensive code reading**
   Goals of the comprehensive code reading are:
   - to get an understanding of the whole code
   - identify adversary controlled inputs and trace their paths
   - identify issues

3. **Static analysis**
   Static analysis is a useful tool to automate the detection of well-known security issues that may be present in the code. During this code audit, static analysis is used to complement manual code inspection.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **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.

3 Automated Testing

A portion of this code audit was based on static analysis tools. This section will give a brief overview of the technologies used. All potential findings were then verified manually.

- Bandit – A tool for finding common security vulnerabilities in Python
- gosec - Golang Security Checker – Inspects source code for security problems by scanning the Go AST.

4 Findings

We have identified the following issues:

4.1 LMW-011 — Missing NULL pointer check after malloc

Vulnerability ID: LMW-011 **Status:** Resolved

Vulnerability type: NULL Pointer Check

Threat level: Low

Description:

Function `char *copy_str(const char *str)` in `dart-wormhole-william` uses a `malloc` -allocated buffer without checking if it is NULL.

Technical description:

The following function in `async_callback.c` does not check if the pointer `*copy` is `NULL` before passing it to `strcpy` .

```
char *copy_str(const char *str) {
  if (str == NULL)
    return NULL;
  char *copy = malloc(strlen(str) + 1);
  strcpy(copy, str);
  return copy;
}
```

Passing a NULL pointer to strcpy causes undefined behavior.

Additionally, `strcpy` is a notorious source of buffer overflows, although its use in this function does not appear to lead to a buffer overflow, assuming copy is non-NULL.

Impact:

If memory is limited this could lead to undefined behavior.

Recommendation:

Add a check for NULL after the call to `malloc` . Additionally, for good hygiene and to avoid possible buffer overflows in the future, replace `strcpy` with `strncpy` .

Update :

Commits `efba20802a97e8c9ff6dbd88ded1c1be8315cf7e` and `b12f3fc73bca093a46fc9ea3f8d611131caf420d` implement NULL pointer checks and error handling for a failed call to Malloc. Also the highly error-prone call to `strcpy` has been replaced correctly by the less error-prone `strncpy`.

Client Response

We have implemented the proposed recommendation. PR:

https://github.com/LeastAuthority/wormhole-william/pull/69

https://github.com/LeastAuthority/dart_wormhole_william/pull/33

4.2 LMW-010 — Attacker can keep messages on rendezvous server indefinitely

Vulnerability ID: LMW-010	**Status:** Unresolved
Vulnerability type: Denial of Service	
Threat level: Low	

Description:

A network attacker can prevent mailboxes from closing and its messages from being deleted.

Technical description:

In `magic-wormhole/magic-wormhole-mailbox-server` , mailbox messages are only deleted after either 1) the mailbox server has received a "close" message from both listeners to a given mailbox, or 2) There have not been any active websocket connections to a given mailbox for more than 11 minutes. After this time a periodic pruning routine will delete old mailboxes and their contents.

A network attacker who can block a "close" wormhole message to the server can also block websocket/TCP FIN messages and send TCP keepalive messages to the server. This is sufficient to keep the number of listeners on an open mailbox greater than zero, preventing the mailbox from being closed by either an active "close" message or the pruning routine.

This vulnerability in the mailbox server exists even if the server is using secure websockets, as the size and ordering of wormhole messages is predictable, and TCP keepalive messages are zero-length TCP segments and thus not encrypted with TLS.

Impact:

Mailbox messages can contain PAKE-encrypted text or metadata about file transfers, but in the case of file/directory transfers they do not contain the file contents themselves.

Additionally, ensuring the persistent availability of mailbox messages is a feature of the protocol that allows clients who become disconnected to reconnect and recover their state.

However, under especially hostile circumstances this could potentially lead to compromising the confidentiality of messages. In a scenario, for instance, where sensitive messages are being communicated via a wormhole (perhaps a journalist communicating with a high-risk source, for instance), an attacker can keep messages stored on the rendezvous server indefinitely and then later attempt to exfiltrate them from (or just steal) the server. In a forward secrecy model where the shared PAKE secret is later somehow recovered or leaked (the journalist is arrested, and the PAKE key is recovered from their device's swap partition, for example), any encrypted communication would then be recovered.

However, this example falls well outside the intended use case of the service, and so the overall the impact of this vulnerability is low.

Recommendation:

The scenario given in the impact section above assumes a high-resource attack and is also very targeted. Given that being able to tolerate disconnecting and reconnecting clients is a feature of the protocol, deleting messages immediately after sending them may be an overcompensation for the given risk.

Another solution would be to change the pruning routine to delete messages based on time since creation rather than time of last connection. This means that even if a client disconnects for 10 minutes their message will be there when they reconnect, but importantly, even if they never disconnect, the message is guaranteed not to be there after 11 minutes.

Update :

Least Authority investigated this finding and accepts the risk, as they have deemed it is very low for their current application and confidential information is not stored in the mailbox server (with the exception of users setting a sensitive filename in the offer message). Even if persistent messages are stolen from the mailbox server, an attacker still must recover the PAKE secret to decrypt them. Further, other protocols such as Dilation, (not in scope of this audit) may depend on messages remaining in the mailbox across re-connects over an indefinite period of time. That is, the current message-pruning behavior is a feature, and a hard time limit for pruning may degrade service.

Client Response

We have investigated this finding and accept the risk as it is very low for our current applications and we currently do not store sensitive information in the mailbox server (except if users set a sensitive filename) as only the offer message is sent via mailbox. Even if messages are stolen from the mailbox server, an attacker still must steal the PAKE secret, which requires access to the actual device, at which point all files are accessible.

If/when we do put sensitive messages on the mailbox server, we do not want to pursue trusting the mailbox server to delete the messages. A better mitigation would be a cryptographic mitigation which would require changing the mailbox protocol and updating all implementations to support it.

Other protocols (including Dilation) may depend on messages remaining in the mailbox across re-connects (for example). That is, the current message-pruning behavior is a feature used by some clients and clients need to remain in control. Any such upgrade to the mailbox protocol should leave clients in charge of when messages become unreadable.

4.3 LMW-009 — Widespread use of Python assert to validate input

Vulnerability ID: LMW-009 **Status:** Unresolved

Vulnerability type: Input validation

Threat level: Moderate

Description:

Both magic-wormhole-mailbox-server and magic-wormhole-transit-relay rely almost exclusively on assert statements for type checking and input validation.

Technical description:

These libraries rely on Python's `assert` to validate input instead of try/except clauses. In optimized code assert statements are removed entirely.

Impact:

If the plugin is optimized by Twisted or by another framework used in the future, then there is practically no input validation happening in these libraries when deployed. Even if the assert statements are not optimized out there is no graceful handling of errors.

Recommendation:

Replace `assert` with `try` / `except` clauses and/or explicitly cast untrusted input values to the required type, where appropriate.

Update :

The client considers this finding mitigated in their deployment and therefore chose to leave the `assert` statements in place.

Client Response

We have evaluated this in detail and consider it is mitigated on our deployment. Assert validation is enabled on our servers and we are in control of the environment. If the user wants to disable assert validation, they have to explicitly request it.

However, there are possibilities for improvement in terms of verifying that incoming messages match a schema, but that is not relevant to the assert statements and is something we can pursue in the future.

4.4 LMW-008 — Unencrypted websockets used by default

Vulnerability ID: LMW-008	**Status:** Unresolved
Vulnerability type: Cryptographic Failures	
Threat level: Low	

Description:

By default Magic Wormhole uses plaintext websockets, which allows a man-on-the-side attacker to read protocol messages between client and rendezvous server and insert packets into the TCP stream.

Technical description:

The magic wormhole protocol involves a predictable exchange of websocket messages between each client and the rendezvous server. By default these messages are unencrypted. A man-on-the-side attacker can read session-specific information like sideID and insert spoofed packets into the stream. The client reads messages of each message type in a FIFO queue, so a fake wormhole message that arrives to the client before a legitimate one does will be read, and the legitimate message will be ignored.

Impact:

A MotS attacker can easily disrupt the client's connection by sending a phony "claimed" message before the legitimate one arrives, causing the client to try to open an incorrect mailbox, for example.

This attack has a more limited impact than simply DoS-ing the rendezvous server as described in https://magic-wormhole.readthedocs.io/en/latest/attacks.html#dos-attack-on-the-rendezvous-server , and requires a stronger attacker model as well. However, if the phony mailbox exists on the server (because it was opened by the attacker on the rendezvous server, for example), the victim will not have any indication that something is wrong, other than not receiving any messages.

Recommendation:

Use websockets over TLS by default.

Update :

Wormhole-william does not enforce the use of secure websockets when connecting to a rendezvous server, but the rendezvous server deployed by Least Authority and which is used in the flutter application uses TLS. As a result, the client considered this finding mitigated in their deployment.

Client Response

We have reviewed and discussed this issue and we consider it is mitigated in our application as we do not allow users to change server url and use a non-TLS connection. Wormhole-william CLI is the only way that non-TLS would be used. Our fork of wormhole-william that is used does not include downloadable binaries for end users.

However, we agree that in the future, if we allow users to change the server address in the GUI application, additional checks should be implemented to verify TLS connection and warn about or reject non-TLS.

4.5 LMW-007 — Receiver will automatically unzip malicious zip files

Vulnerability ID: LMW-007	**Status:** Resolved
Vulnerability type: Data Integrity Failures	
Threat level: Elevated	

Description:

When receiving a file directory, receiver unzips the received archive without checking if it matches what was offered.

Technical description:

In `func recvAction(cmd *cobra.Command, args []string)` from `wormhole-william/cmd/recv.go`, a receiver who expects a file directory from the initial offer message automatically unzips whatever zip file it receives.

However, a malicious sender or MiTM can send a fake offer message for a file directory and instead send a malicious zip file unrelated to the offer message.

For example, by modifying `SendFile` in `wormhole-william/wormhole/send.go` as follows, we can send a specifically crafted zip file as a directory.

```
// SendFile sends a single file via the wormhole protocol. It returns a nameplate+passhrase code to
 give to the
// receiver, a result channel that will be written to after the receiver attempts to read (either
 successfully or not)
// and an error if one occurred.
func (c *Client) SendFile(ctx context.Context, fileName string, r io.ReadSeeker, disableListener
 bool, opts ...TransferOption) (string, chan SendResult, error) {
    size, err := readSeekerSize(r)
    if err != nil {
        return "", nil, err
    }

    var offer *offerMsg

    if fileName[len(fileName)-3:] == "zip" {
        offer = &offerMsg{
            Directory: &offerDirectory{
                Dirname:  "NotScaryAtALL",
                Mode:     "zipfile/deflated",
                NumBytes: 10000000,
                NumFiles: 10,
                ZipSize:  16800098,
            },
        }
    } else {
        offer = &offerMsg{
            File: &offerFile{
                FileName: fileName,
                FileSize: size,
            },
        }

    }
    return c.sendFileDirectory(ctx, offer, r, disableListener, opts...)
}
```

As the sender we can then send our malicious zip file:

```
$ go run main.go send bomb.zip
On the other computer, please run: wormhole receive (or wormhole-william recv)
Wormhole code is: 2-resistor-tactics
16.02 MiB / 16.02 MiB [-------------------------------------------------------------------]
 100.00% 47.10 MiB p/s 1s
file sent
```

The receiver, however, sees the following:

```
$ go run main.go receive
Enter receive wormhole code: 2-resistor-tactics
Receiving directory (16.8 MB) into: NotScaryAtALL
10 files, 10.0 MB (uncompressed)
ok? (y/N):y
16.02 MiB / 16.02 MiB [-----------------------------------------------------------------------]
 100.00% 1.37 MiB p/s 12s
$ du -h -s NotScaryAtALL
825M    NotScaryAtALL
```

Impact:

Users can be tricked into blindly unzipping potentially malicious zip archives. We were able to trick the receive function to unzip a zip bomb, despite "offering" a directory with completely unrelated metadata.

The security of the specific zip implementation used here is out of scope, but there have been many instances of bugs in various zip implementations allowing for remote code execution.

As a mitigating factor, it is good to note also that the function does correctly check for path traversal, so overwriting system files is not possible.

Recommendation:

The receiver should always inspect the zip archive to see if its content matches the metadata described in the offer message. Any differences between the archive metadata and the offer message should result in a cancelled wormhole session and the deletion of the archive.

Update :

Commit `7bd6bf8b104a4e1276089db74ddb44d8a7c1a9a5` adds a check that the uncompressed archive contains the same number and size of files that are advertised in the offer message.

Client Response

Per the recommendation, Least Authority has added a validation mechanism to check its content matches the metadata described in the offer message prior to exporting. PR: https://github.com/LeastAuthority/wormhole-william/pull/68

4.6 LMW-001 — Excessive directory permissions for received directories

Vulnerability ID: LMW-001	**Status:** Resolved
Vulnerability type: Excessive Permissions	
Threat level: Moderate	

Description:

Receiver sets default directory/file permissions to 0777/0666, respectively

Technical description:

In `func recvAction(cmd *cobra.Command, args []string)` in `wormhole-william/cmd/recv.go` new directories are created on the receiver's machine in lines 175 and 218 with default permission bits set to 0777.

Line 175:

```
err = os.Mkdir(msg.Name, 0777)
```

Line 218

```
err = os.MkdirAll(dir, 0777)
```

Files from the received archive are then created with a call to `os.Create()` , beginning on line 223, which by default sets permission bits to 0666.

```
f, err := os.Create(p)
                    if err != nil {
                        bail("Failed to open %s: %s", p, err)
                    }

                    _, err = io.Copy(f, rc)
                    if err != nil {
                        bail("Failed to write to %s: %s", p, err)
                    }
```

By default these directories and files can thus be read and written do by all users on the system, potentially violating the principle of least privilege.

Impact:

If the receiver does not have a umask set then the received files will be globally readable/writable.

Recommendation:

Set permission bits to 0700/0600 for directories/files respectively, or at least something without global write permissions like 755/644.

Unfortunately the zip format does not preserve Unix file permissions, so there is no way to restore the original file permissions of the transmitted directory without some extra bookkeeping.

Update :

Commit `75a194a5d4271c57d84884edc5940f84730989de` sets directory permissions to 0700 when unzipping a directory archive. The client also submitted an upstream pull request to add this change to the parent repository.

Client Response

We have evaluated this in detail and consider it is mitigated on our deployment, as we do not support directory transfer in Destiny (GUI) applications at the moment. However we have implemented a recommendation to set strict permission 700. PR: https://github.com/LeastAuthority/wormhole-william/pull/70

5 Non-Findings

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

5.1 NF-002 — Bias toward slowest connection when multiple relay-v1 endpoints are offered

The function `func (t *fileTransport) connectViaRelay(otherTransit *transitMsg)` in `wormhole-william/wormhole/file_transport.go` always returns the last successfully connected relay endpoint, biasing the slowest connection.

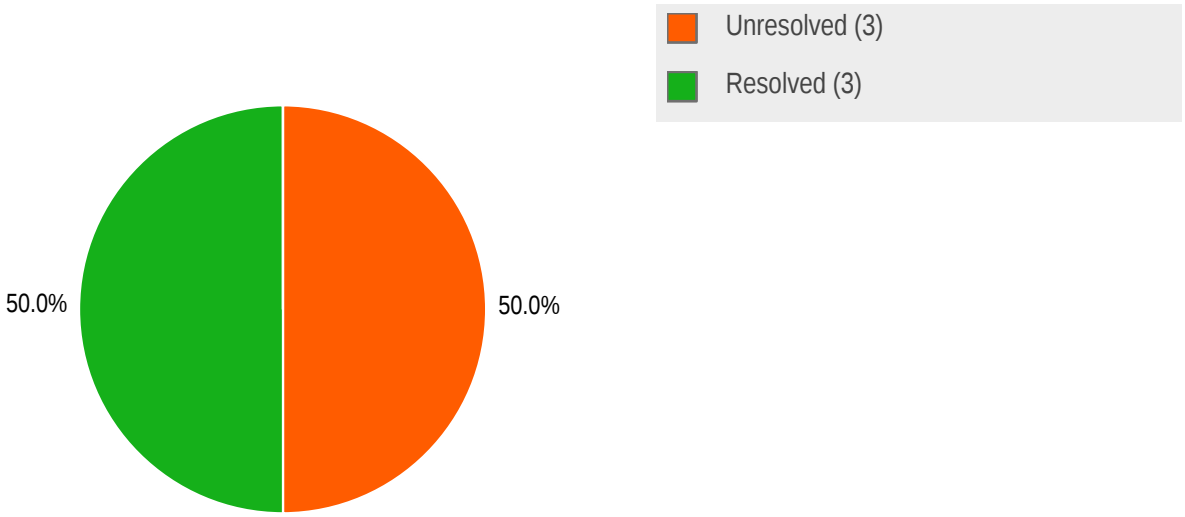
6 Retest

On August 23, 2022 Radically Open Security B.V. carried out a retest for Least Authority TFA GmbH to assess the status of the previously discovered findings.

Summary of the finding status with original issue severity ratings:

- **Resolved**: 1 Low, 1 Elevated and 1 Moderate
- **Unresolved**: 2 Low, 1 Moderate
- **Not retested**:

6.1 Findings by Retest Status



6.2 Retest Conclusion

Of the six findings identified in the report, three (LMW-001, LMW-007, and LMW-011) have been completely resolved.

LMW-008, which related to the use of secure websockets, is mitigated by Least Authority's deployed mailbox server and GUI application, but was left unresolved in Wormhole-William to maintain backward compatibility with other Magic-Wormhole implementations. Their deployed mailbox server uses secure websockets and the GUI client is hard-wired to use this server.

LMW-010, a finding relating to the ability of an attacker to prevent protocol messages from being deleted from the mailbox server, was left unresolved due to the users' occasional need to be able to re-access mailbox messages multiple times in case of connectivity issues, and so enforcing message deletion would interfere with usability.

Finally, LMW-009, which involved the widespread use of Python `assert` statements for server input validation, was considered an accepted risk and left as-is.

7 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

8 Conclusion

We discovered 3 Low, 2 Moderate and 1 Elevated -severity issues during this penetration test.

Overall, the quality of the code was very good. Each target performs just a few, well-documented functions, and generally does them well. However, the security of the protocol relies heavily on the security of the PAKE, and surprisingly does not offer some low-hanging defense-in-depth fruit like TLS by default to the rendezvous server. This means the unencrypted/unauthenticated messages of the core protocol are vulnerable to modification by a network attacker. The ability for an attacker to modify or block some of these messages was an important part for two of the findings.

Other defense-in-depth measures, such as comparing the contents of offer message to the contents of a received zip archive, and setting conservative permissions for the unzipped files, were also absent.

The majority of the code was written in Go, Python, and Dart, with a small amount written in C, each language having its own advantages and pitfalls from a security perspective. Two of the findings stem directly from unsafe uses of the language itself: one possibly leading to undefined behavior, and the other resulting in improper input validation.

The inspection of the Dart code was mainly geared toward finding logic errors and improper handling of untrusted input. However, no findings stemmed from the Dart code in the Dart plugin or the Flutter GUI.

The client was regularly informed of questions and potential findings throughout the audit and responded with excellent feedback and questions, which was greatly appreciated.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture 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 that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Jonathan Levin	Jonathan is a cryptographic researcher specializing in network protocols and post-quantum cryptography. He has performed cryptographic software audits on various projects including attribute-based credential systems, virtual private network protocols, and PKI implementations.
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.