Fine penetration tests for fine websites

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

Pentest-Report ExpressVPN Lightway 10.-11.2022

Cure53, Dr.-Ing. M. Heiderich, MSc. H. Moesl, Dipl.-Ing. D. Gstir, R. Weinberger, P. Einkemmer

Index

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

Introduction

"Lightway is ExpressVPN's pioneering new VPN protocol, built for an always-on world. It makes your VPN experience speedier, more secure, and more reliable than ever. Designed to be light on its feet, Lightway runs faster, uses less battery, and is easier to audit and maintain."

From https://www.expressvpn.com/lightway

This report describes the results of a security assessment of the ExpressVPN complex, specifically targeting the Lightway software components. Even more precisely, the list of the assessment's targets included the Lightway server and client, the shared libraries, as well as the Lightway High-Performance TUN driver. Carried out by Cure53 in November 2022, the project included a penetration test and a dedicated audit of the source code.

Registered as *EXP-13,* the project was requested by ExpressVPN in October 2022 and then scheduled for the upcoming weeks. As for the precise timeline and specific resources allocated to *EXP-13*, Cure53 completed the examination in October and November 2022 as scheduled, with focused tests taking place from CW43 to CW45. Further of note are the facts that a total of thirty days were invested to reach the coverage expected for this assignment, whereas a team of five senior testers has been composed and tasked with this project's preparation, execution and finalization.

For optimal structuring and tracking of tasks, the work was split into four separate work packages (WPs):

- **WP1**: Source-code assisted penetration tests against Lightway server component
- **WP2**: Source-code assisted penetration tests against Lightway client component
- **WP3**: Source-code assisted penetration tests against Lightway shared libraries
- **WP4**: Source-code assisted penetration tests against Lightway High-Performance TUN driver

The white-box methodology was used in this project. Cure53 was provided with sources, libraries, documentation, as well as all other means of access required to complete the tests.

Commenting on the framework of the project, it cannot be disregarded that it belongs to a long-term and wide-scoped cooperation between Cure53 and the teams responsible for the ExpressVPN's security. Hence, Cure53 has looked at the ExpressVPN complex multiple times in the past. In fact, the Lightway client and other components were already included in a few previous audits as a secondary target (see *EXP-08, EXP-09* and *EXP-11* as well as *EXP-04* for Lightway).

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

The project progressed effectively on the whole. All preparations were done in CW42 to foster a smooth transition into the testing phase. Over the course of the engagement, the communications were done using a private, dedicated and shared Slack channel set up to connect the Cure53 and ExpressVPN personnel relevant to the test. The discussions throughout the test were very good and productive and not many questions had to be asked. Ongoing interactions positively contributed to the overall outcomes of this project. The scope was well-prepared and clear, greatly contributing to the fact that no noteworthy roadblocks were encountered during the implementation of this project.

Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was not explicitly requested at the beginning, but an executive decision was made about half-way into the project about sharing the details of the tickets on Slack. This made it possible for the responsible ExpressVPN team to have an early insight into the problems emerging and being filed as the result of the Cure53's efforts.

The good and productive flow of the test allowed for good coverage and depth levels across all four WPs. This contributed to the identification of nine issues. Among them, three problems shall be seen as security vulnerabilities of varying severity levels and six represent general weaknesses, typically characterized by lower exploitation potential. Quite clearly, the overall number of findings is moderate and can be interpreted as a good sign for the security of the inspected Lightway components. It is furthermore good to note that the majority of the discovered issues were rated to be general weaknesses, most of which should be easy to resolve and mitigate.

In the following sections, the report will first shed light on the scope and key test parameters. A dedicated chapter on test methodology and coverage then clarifies what the Cure53 team did in terms of attack-attempts and coverage. This chapter is included to demonstrate to the client which areas of the Lightway components in scope have been covered and what tests have been executed despite only a few findings having been spotted.

Next, all findings will be discussed in a chronological order alongside technical descriptions, as well as PoC and mitigation advice when applicable. Finally, the report will close with broader conclusions about this autumn 2022 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations for the ExpressVPN Lightway components, specifically the Lightway server and client components, the shared libraries, as well as the Lightway High-Performance TUN driver, are also incorporated into the final section.

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

Scope

- **Code audits & Security assessments against ExpressVPN's Lightway components**
  - **Key areas in focus:**
    - As part of this security assessment the following Lightway components were assessed and audited for security vulnerabilities:
    - Lightway server component
    - Lightway client component
    - Shared Lightway library used by the client and server components
    - Lightway client-side libraries for Lightway client.
    - Lightway TUN driver, a.k.a High-Performance TUN (HPT) driver
  - **WP1**: Source-code assisted penetration tests against Lightway server component
    - Access to the source code has been provided to Cure53
    - **Primary scope item:**
      - *xv_helium_server*
    - **Other items:**
      - Server components; focus on attacks that may lead to memory corruption, privilege escalations, etc.,
  - **WP2**: Source-code assisted penetration tests against Lightway client component
    - Access to the source code has been provided to Cure53
    - Client binaries and configuration files have been shared with the Cure53 testing team
    - **Primary scope items:**
      - *xv_helium_cli*
      - *xv_libballoon*
    - **Other items:**
      - Both the client components and client libraries for Windows, MacOS and Linux
  - **WP3**: Source-code assisted penetration tests against Lightway shared libraries
    - Access to the source code has been provided to Cure53
    - **Primary item:**
      - *lightway-core*
      - This included the core library for the Threat Manager function, core library for the Lightway Parallel Connect function and relevant helper functions
    - **Secondary items:**
      - *libdnet*
      - *libuv*
    - **Other aspects:**
      - Shared libraries
      - Privilege escalation
      - Denial-of-Service
      - Memory corruption

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

- **WP4**: Source-code assisted penetration tests against Lightway High-Performance TUN driver
  - Access to the source code has been provided to Cure53
  - **Primary item:**
    - *xv_helium_tun*
  - **Other items:**
    - The kernel module serving as alternative to the Linux TUN driver
    - Privilege escalation
    - Memory corruption
- **Test-supporting material was shared with Cure53**
- **All relevant sources were shared with Cure53**

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

Severity Glossary

This section details the varying severity levels assigned to the issues discovered in this report.

Critical represents the highest possible severity level. It is applied to issues that allow attackers to achieve extensive access to sensitive areas, such as critical systems, applications, data or other pertinent components in scope.

High characterizes issues that allow attackers to achieve limited access to sensitive areas in scope. This also includes issues with limited exploitability that can facilitate a significant impact upon the target in scope.

Medium is applied to issues that do not cause major impact on the areas in scope. Additionally, issues requiring a more limited exploitation are graded as *Medium*.

Low is assigned to issues that have minor and greatly limited implications for the areas in scope. Mostly, this scope does not depend on the level of exploitation but rather on the minor significance of the obtainable information or lower grade of damage targeting the areas in scope.

Info is added to issues included merely for information purposes. They are mostly considered as hardening recommendations or improvements that can generally enhance the security posture of the areas in scope.

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

Table of Findings

Identified Vulnerabilities

ID	Title	Severity
EXP-13-005	WP1: Potential delay of libuv I/O processing by TUN read loop	*Low*
EXP-13-008	WP4: Integer underflow in ring buffer count function	*Low*
EXP-13-009	WP3: Use of weak cipher-suites with DTLS	*Low*

Miscellaneous Issues

ID	Title	Severity
EXP-13-001	WP3: OOB reads in various server handlers	*Low*
EXP-13-002	WP4: Potential memory leak of pinned pages in TUN driver	*Low*
EXP-13-003	WP2: Default allow approach in he_domain_filter_handler	*Info*
EXP-13-004	WP1: Potential integer overflow in *he_strndup()*	*Info*
EXP-13-006	WP1: No lower bounds check for message payload handlers	*Info*
EXP-13-007	WP4: Marking unpinned user pages as dirty	*Info*

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

Test Methodology & Coverage

This section zooms in on the metrics and methodologies used to evaluate the security characteristics of the Lightway project and codebase. In addition, it includes results pertinent to individual areas of the project's security properties that were either selected by Cure53 or singled out by ExpressVPN as calling for a closer inspection.

In this assessment, the repositories listed below were especially vital:

- *xv_helium_cli: Roughly 8200 LoC*
- *xv_helmium_server: Roughly 2600 LoC*
- *xv_helium_tun: Roughly 800 LoC*
- *lightway-core: Roughly 3200 LoC*
- *xv_libballoon: Roughly 2000 LoC*

With the lesser priority, the following items were declared to be in-scope as well:

- *libdnet: Roughly 1200 LoC*
- *libuv: Roughly 1400 LoC*

For the forked *libdnet* and *libuv* repositories used by the Lightway client, only the differences in the code between the non-forked version and the forked version were examined as part of the scope.

To support an efficient assessment and dynamic testing, ExpressVPN provided testing Lightway servers with a reference Lightway implementation, which included the Lightway High Performance Tunnel driver and a Lightway client to connect to the server.

While such large-scale audits are always limited by the budget and require strong selectivity and isolated focal points, with a particular focus for the more sensitive parts of the code, Cure53's goal was to reach good coverage across the scope. With this in mind, Cure53 conducted an extensive source code analysis across the different components of the Lightway software stack.

Responding to the above, ExpressVPN provided a well-defined attacker model containing risk-assessments and valid threat actors, which effectively helped Cure53 in outlining a clear strategy. The testers identified main issues to look out for. The provisioned thorough documentation also underlines the well-thought-out development process and self-reflection of the ExpressVPN developers, who have a specific way of framing the attackers' perspective and capabilities.

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

Since the entire codebase is written in C, a thorough assessment of memory corruption and similar common faults was performed. In addition, the entire codebase was inspected for privilege escalation issues, race condition, logic bugs and correct usage of system interfaces.

The excellent preparation by ExpressVPN helped a lot and allowed Cure53 to quickly start with targeted audits of the sensitive parts of the system. The following chapters comment on each WP and provide additional details with regard to Fuzz Testing and Static Source Code Analysis.

As communicated by ExpressVPN upfront, the following components were considered out-of-scope:

- Any build dependencies, development tools, build scripts
- Code and dependency used for tests (i.e., mocks, end-to-end/e2e tests)
- The test servers on which the Lightway server was installed
- Lightway server component runs on ExpressVPN's TrustedServer, which are out of scope of this assessment as the TrustedServer has already been assessed as part of *EXP-06*
- Third-party dependencies included within the Lightway components
- Testing of the API servers, including fuzzing
- The ExpressVPN client application and its components, which have been addressed in earlier audits
- The WolfSSL library
- Denial-of-Service vectors in the Lightway client caused by out-of-memory errors
- *libxenon* which contains ExpressVPN’s proprietary obfuscation code
- The non-forked public repositories for *libdnet* and *libuv*

Coverage for WP1 - Lightway Server

The Lightway server component was provided within a dedicated testing environment and represented a real-world production setup for Lightway. It generally acts as a connection endpoint for the Lightway client to set up a secure tunnel to the Lightway server.

The following provides an overview of the executed tests and gathered insights into the code quality of the Lightway server component:

- The protocol and initial handshake between the Lightway client and server has been evaluated for logic bugs and nothing was spotted here.
- The C codebase of the Lightway server has been audited with a specific focus on memory corruption vulnerabilities, which typically either manifest within OOB

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

read or write vulnerabilities (also known as buffer overflows), Integer overflows, race conditions, Use-after-Frees or Double Frees. It was positively noticed that the ExpressVPN development team does an excellent job in protecting against classical memory corruption vulnerabilities.

- The codebase was also evaluated in great depth for privilege escalation vectors.
- It was observed that the session ID is transmitted as part of the unencrypted message. However, after consultation with ExpressVPN, it was confirmed that the session ID is only used to look up a session if the source IP and port do not match.
  - This is necessary when the client changes networks or a NAT timeout occurs. If a session is found, then the server will attempt to decrypt the payload. Only if the payload is valid, will the client's destination address be updated.
  - WolfSSL has reply protection and will reject packets it has already seen. Therefore, a MitM attacker would not be able to impersonate the server due to possessing neither a valid certificate (signed by the ExpressVPN CA) nor the correct CN (so a certificate stolen somehow from elsewhere would not be effective).
- The Lightway server has been audited with regard to race conditions. The Lightway main message processing is single-threaded and it uses an event loop which is single threaded. Therefore, there is no risk of data races.
- Due to the single-threaded nature, special attention was paid to the use of asynchronous I/O processing handlers and their runtime behavior. Specifically, if there are measures taken that prevent unbounded runtime of handlers which will block other I/O operations.

Coverage for WP2 - Lightway Client

The Lightway Client is used to communicate with the Lightway server when establishing a secure tunnel. It does this by setting up necessary interfaces that will be used for the tunnel connections between the client and the server. It should be noted that the Lightway client runs with *root* privileges in all desktop client application deployments, i.e., on Windows, MacOS and Linux clients.

The following provides an overview of the executed tests and gathered insights into the code quality of the Lightway client component:

- The communication between the Lightway client and server is encrypted using TLS version 1.3 or DTLS version 1.2. The used cipher-suites selected by the client are state-of-the-art:
  - *TLS_AES_256_GCM_SHA384* (TLS1.3)
  - *TLS_CHACHA20_POLY1305_SHA256* (TLS1.3)
  - *TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384* (DTLS1.2)

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

- *TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256* (DTLS1.2)
- With all the encryption and decryption happening inside the WolfSSL library, no further reviews and tests have been performed with regards to that.
- As the client is running with *root* privileges, the codebase was comprehensively evaluated for privilege escalation vectors and memory corruption issues.

Coverage for WP3 - Lightway Shared Libraries

The Lightway shared library provides a functional API for both the client and server-side components of Lightway. At its core, the library offers an interface for the WolfSSL library and is used for creating, tearing down, and managing connections using the standard configurations. It must be noted that Lightway itself does not perform any cryptographic functions, since all of them are provided by the WolfSSL library.

Below is an overview of the executed tests and gathered insights pertinent to the code quality of the Lightway shared libraries component:

- A substantial portion of the Lightway clients and servers utilize *lightway-core*.
  - For instance, the main message handling routines are part of lightway-core and all message handler routines have been manually audited, focusing on memory corruption vulnerabilities, which typically either manifest within OOB read or write vulnerabilities (also known as buffer overflows), Integer overflows, race conditions, Use-after-Frees or Double Frees.
  - One minor OOB read was noted but may not even result in a SIGSEGV since the amount of memory to be read out-of-bounds is capped at 5 bytes. Thus it is very unlikely to read unmapped memory.
- It was positively noticed that the ExpressVPN development team does an excellent job in protecting against classical memory corruption vulnerabilities.
- The codebase was also checked in-depth for privilege escalation vectors.
- The TLS/DTLS connection setup for client and server configurations is implemented in *lightway-core.* This has been thoroughly inspected for configuration issues and logic flaws. For servers, the cipher-suite list is not explicitly configured and, instead, the default configuration from WolfSSL is used. This default set includes some known weaker and not recommended encryption cipher-modes like *AES-CBC,* alongside authentication algorithms like *SHA1.*

Coverage for WP4 - Lightway High-Performance TUN driver

The Lightway High-Performance TUN driver kernel module is an alternative to the Linux TUNTAP driver, mapping custom ring-buffers into the kernel and userspace at the same time. This mechanism is used for passing packets received between userspace and kernelspace instead of using syscalls.

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

Furthermore, this package also includes a library named *libhpt*, which simplifies driving of this interface and removes the need to *open()* or *ioctl()* the driver directly.

The following provides an overview of the executed tests and gathered insights into the code quality of the Lightway High-Performance TUN driver component:

- User- and kernel space exchange packets using two ring buffers. These ring buffers rely on user-pages and operate in a lock-free manner using acquire-release semantics. Special attention was paid to the implementation and use of these lock-free ring buffer operations.
- The TUN driver was also audited for LPE vulnerabilities, with the focus on memory corruption vulnerabilities (heap and stack), Integer arithmetic issues, kernel race conditions and logic bugs. No major vulnerabilities were spotted here.
- The kernel code was also inspected for correct use of internal APIs provided by various kernel subsystems, as well as general robustness of the implementation. Overall, the HPT TUN is-well written and follows Linux kernel's development style

Fuzz Testing

While tests are essential for any project, their importance grows with the scale of the endeavor. Especially for large-scale compounds, testing ensures that functionality is not broken by code changes. Furthermore, it generally facilitates the premise where features operate in the ways they are supposed to. Regression tests also help guarantee that previously disclosed vulnerabilities do not get reintroduced into the codebase. Testing is therefore essential for the overall security of the project.

Lightway does not incorporate fuzz testing - to the best of Cure53's knowledge - in any of the modules in-scope, including *xv_helium_cli*, *xv_libballoon*, *xv_helium_server*, *lightway-core* and *xv_helium_tun*. In that sense, it does not offer a comprehensive fuzz test coverage across the complete codebase.

In the light of this, development in this realm should be seen as essential, particularly considering the predominance of memory corruption issues in software written in C and C++. For the purpose of this test, code coverage driven fuzzing using AFL++ - in combination with address space sanitizers such as ASAN - was performed.

In particular, the message handlers being part of the *lighway-core* library which are used within both the server and client have been fuzzed using AFL++ and active ASAN. Furthermore, the domain and DNS filter contained in the client have also been fuzzed as such parsers can be a potential source of error in software projects. Nevertheless, no issues could be identified.

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

Moving forward it is recommended to equip the modules in-scope with libFuzzer and Google's sanitizers as part of the OSS-Fuzz[1] initiative, which would continuously fuzz-tests the public codebase of Lightway.

Static Source Code Analysis

Static source code analysis is a powerful technique which facilitates identifying vulnerabilities as early as possible within the software development process. The Cure53 testing team mostly performed manual source code audits in the form of deep-dives, but also wanted to point out powerful tools that may be incorporated into the Lightway software development pipelines.

The well-known static analysis utility CodeQL[2] has been used to search for specific bug patterns within the source code repositories in-scope. Using CodeQL brings a lot of benefits, as it operates on the Abstract Syntax Tree (AST) of the program to be analyzed and, therefore, allows for deep inspection of the code with taint tracking / analysis and other variations of the analysis techniques. The standard ruleset[3] as well as hand-crafted rules were used in the hunt for vulnerabilities here, but nothing was identified using CodeQL.

[1] https://github.com/google/oss-fuzz
[2] https://codeql.github.com/
[3] https://codeql.github.com/codeql-query-help/cpp/

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 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. *EXP-13-001*) for the purpose of facilitating any future follow-up correspondence.

EXP-13-005 WP1: Potential delay of *libuv* I/O processing by TUN read loop (*Low*)

Fix Note: *The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

At its core, Lightway server uses the asynchronous I/O library *libuv*[4] to handle network traffic responsible for forwarding and various associated tasks like timers. During a source code audit of the standard TUN adapter code of the Lightway server, it was discovered that the iterations of its read loop - and thus the runtime - could be unbounded. As *libuv* is single-threaded and the affected code is called from within the *libuv* main loop, all other *libuv* operations will be delayed until this loop returns.

Testing showed that due to the way inbound client traffic was handled in Lightway server, this has consequences for a client. Specifically, it saturates its connection to cause a slight delay in *libuv's* I/O processing. It will, however, not allow a trivial remote Denial-of-service attack, since a client cannot directly cause this read loop to run endlessly.

Currently, the TCP and UDP adapter code in Lightway server prevents this by the use of *libuv*'s TCP and UDP infrastructure. Specifically, *libuv* internally limits the amount of data processed within one run of the TCP/UDP read handlers. Any changes to this logic can still cause this code to become reachable by clients, rendering it exposed to DoS attacks.

Another case where this can be a problem concerns other processes on the Lightway server host, e.g., another VPN protocol gateway, as long as they are able to send traffic into a Lightway tunnel. Since such a sender can issue an endless stream of data to the TUN interface, Lightway server will become "trapped" within its TUN read loop and all other *libuv* I/O processing in the Lightway server will be starved. This also poses a potential risk of DoS attacks vectors for malicious actors who have gained low-level execution privileges on the host.

[4] http://libuv.org

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

Affected file:
xv_helium_server/src/inside_adapters/tun_adapter.c

Affected code:

```
void on_tun_event(uv_poll_t *handle, int status, int events) {
  // Get Helium state
  he_server_t *state = (he_server_t *)handle->data;
  HE_CHECK_WITH_MSG(state, "Helium server state not found on tunnel event");

  // What event did we get? We only care about it becoming readable...
  if((events & UV_READABLE) == UV_READABLE) {
    // Loop over all available packets whilst we're here...
    while(true) {
      // Create sizeof(HE_MAX_MTU)
      // Read in IP packet

      // This needs to be set to a well understood and used variable - no magic
numbers...
      uint8_t msg_content[HE_MAX_MTU] = {0};

      // Read a packet
      int length = read_from_tun(handle->io_watcher.fd, msg_content,
       HE_MAX_MTU);

      // Would have blocked, so all packets are read - we can stop reading now
      if(length == -1) {
        return;
      }

      he_inside_process_packet(state, msg_content, length);
    }
  }
}
```

As can be seen from the code above, the read loop of the TUN interface will only abort when *read_from_tun()* would block or return an error.

It is recommended to introduce an upper limit of packets/bytes handled within this loop and then return control to *libuv*'s main loop. *Libuv* will then ensure that other I/O events are handled and then return to *on_tun_event()* should there be more packets to handle.

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

*This issue was assigned a **5.5** CVSS rating[5], as stipulated in the breakdown of each scoring component offered below:*

https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H

CWE: https://cwe.mitre.org/data/definitions/835.html

EXP-13-008 WP4: Integer underflow in ring buffer count function (*Low*)

***Fix Note:** The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Both kernel and userspace share two ring buffers to exchange network packets, while these ring buffers are implemented using read and write pointers. Since the data structure is shared between kernel and userspace, the kernel must trust these pointers.

While it does proper sanity checks in most areas of the TUN driver, it misses the case that userspace can set both pointers to values, which causes the subtraction to overflow. Resultantly, *hpt_rb_count()* returns a number much larger than the size of the ring buffer. As a consequence, userspace could force extremely long looping in the kernel function *hpt_net_rx()*.

Affected file:
xv_helium_tun/lib/hpt/hpt_common.h

Affected code:
```
static inline uint64_t hpt_rb_count(struct hpt_ring_buffer *ring,
                                    size_t ring_buffer_items)
{
        return ACQUIRE(&ring->write) - ACQUIRE(&ring->read);
}
```

Affected file:
xv_helium_tun/kernel/linux/hpt/hpt_net.c

Affected code:
```
size_t hpt_net_rx(struct hpt_dev *hpt)
{
[..]
        size_t num, i, len;
        u8 ip_version;
```

[5] https://www.first.org/cvss/calculator/3.1

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

```
        ring =  hpt->rx_ring;

        num = hpt_rb_count(ring, hpt->ring_buffer_items);

        for (i = 0; i < num; i++) {
              elem = hpt_rb_rx(ring, hpt->ring_buffer_items,
                                    hpt->rx_start);

              hpt->hd_rx_called++;
[...]
}
```

It is recommended to limit the return value of *hpt_rb_count()* to the maximal possible number of elements in the ring buffer. That way a hostile userspace process can still produce wrong counting but it will remain within bounds.

*This issue was assigned a **3.3** CVSS rating, as stipulated in the breakdown of each scoring component offered below:*

https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:L

CWE: https://cwe.mitre.org/data/definitions/191.html

EXP-13-009 WP3: Use of weak cipher-suites with DTLS (*Low*)

***Fix Note:** The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

The Lightway protocol utilizes the TLS and DTLS protocols for traffic security in the streaming (TCP) and datagram (UDP) modes, respectively. For TCP mode, it enforces TLS version 1.3 and for UDP mode it enforces the slightly older DTLS version 1.2.

During a closer inspection of the server-side TLS/DTLS setup code, it was discovered that the Lightway server does not limit the cipher-suite, but uses the default cipher-suite provided by WolfSSL. This is fine for TLS 1.3 as old cipher-suites have been removed, but for DTLS 1.2 it means that also weak ciphers like AES-CBC or SHA1 get enabled.

More precisely the following list shows the cipher suites currently supported and ordered by priority (high to low), as reported by *wolfSSL_get_cipher_list()*. Note that weak cipher-suites are highlighted:

- *TLS13-AES128-GCM-SHA256*
- *TLS13-AES256-GCM-SHA384*

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

- *TLS13-CHACHA20-POLY1305-SHA256*
- ***ECDHE-RSA-AES128-SHA***
- ***ECDHE-RSA-AES256-SHA***
- ***ECDHE-ECDSA-AES128-SHA***
- ***ECDHE-ECDSA-AES256-SHA***
- *ECDHE-RSA-AES128-GCM-SHA256*
- *ECDHE-RSA-AES256-GCM-SHA384*
- *ECDHE-ECDSA-AES128-GCM-SHA256*
- *ECDHE-ECDSA-AES256-GCM-SHA384*
- ***ECDHE-RSA-AES128-SHA256***
- ***ECDHE-ECDSA-AES128-SHA256***
- ***ECDHE-RSA-AES256-SHA384***
- *ECDHE-ECDSA-AES256-SHA384*
- *ECDHE-RSA-CHACHA20-POLY1305*
- *ECDHE-ECDSA-CHACHA20-POLY1305*
- *ECDHE-RSA-CHACHA20-POLY1305-OLD*
- *ECDHE-ECDSA-CHACHA20-POLY1305-OLD*

This issue seems to be known to the Lightway developers, as the code that limits the DTLS cipher-suites to secure versions exists, but is just currently commented out (see a listing below).

Nonetheless, SHA1 should be assumed deprecated and was shown to be broken in practice due a collision attack discovered in 2017[6]. AES-CBC is considered weak due to vulnerability to timing attacks in various implementations known as Lucky 13[7].

The WolfSSL library version used by Lightway already contains the respective fixes and is currently assumed to be secure. More generally, it is recommended to instead use AEAD ciphers like AES-GCM.

On the one hand, the impact of this is somewhat mitigated on the client-side, as the official Lightway client will enforce the cipher-suites *TLS_ECDHE_RSA_WITH_-AES_256_GCM_SHA384* or *TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA-256* for DTLS. On the other hand, should there be other clients which do not enforce this, these might be tricked into using one of the provided weaker ciphers. However, it should be noted that the ExpressVPN team confirmed that no other clients exist which can connect to the Lightway server.

[6] https://shattered.io
[7] http://www.isg.rhul.ac.uk/tls/Lucky13.html

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

Affected file:
lightway-core/src/he/ssl_ctx.c

Affected code:
```
he_return_code_t he_ssl_ctx_start_server(he_ssl_ctx_t *ctx) {
  // Return if ctx is null
  if(!ctx) {
    return HE_ERR_NULL_POINTER;
  }

  int res = he_ssl_ctx_is_valid_server(ctx);
  if(res != HE_SUCCESS) {
    return res;
  }

  // First we do the ctx->wolf_ctx setup
  if(ctx->connection_type == HE_CONNECTION_TYPE_STREAM) {
    // Create Wolf context using the TLS protocol v1.3
    ctx->wolf_ctx = wolfSSL_CTX_new(wolfTLSv1_3_server_method());
  } else if(ctx->connection_type == HE_CONNECTION_TYPE_DATAGRAM) {
    // Create Wolf context using the D/TLS protocol v1.2
    ctx->wolf_ctx = wolfSSL_CTX_new(wolfDTLSv1_2_server_method());
  }  // No need for an else clause, we will fail on the next line.

  if(ctx->wolf_ctx == NULL) {
    return HE_ERR_INIT_FAILED;
  }

  // Load server certs into ctx
  if(wolfSSL_CTX_use_certificate_file(ctx->wolf_ctx, ctx->server_cert,
SSL_FILETYPE_PEM) !=
     SSL_SUCCESS) {
    return HE_ERR_INIT_FAILED;
  }

  // Load server key into ctx
  if(wolfSSL_CTX_use_PrivateKey_file(ctx->wolf_ctx, ctx->server_key,
SSL_FILETYPE_PEM) !=
     SSL_SUCCESS) {
    return HE_ERR_INIT_FAILED;
  }

  // Initialise Wolf's RNG
  if(wc_InitRng(&ctx->wolf_rng) != 0) {
    return HE_ERR_INIT_FAILED;
  }

  /* // 2020-03-15 Setting this currently causes chacha20 clients to misbehave,
       commenting out
```

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

```
    * // while the team investigates
    *
    *
  // Explicitly set the cipher list
  if(ctx->connection_type == HE_CONNECTION_TYPE_STREAM) {
    res = wolfSSL_CTX_set_cipher_list(ctx->wolf_ctx,
        "TLS13-AES256-GCM-SHA384:TLS13-CHACHA20-POLY1305-SHA256");
  } else {
    res = wolfSSL_CTX_set_cipher_list(ctx->wolf_ctx,
        "ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-"
        "RSA-CHACHA20-POLY1305:DHE-RSA-CHACHA20-POLY1305");
  }

  // Fail if the ciphers can't be set
  if(res != SSL_SUCCESS) {
    return HE_ERR_INIT_FAILED;
  }
  */

  return he_ssl_ctx_start_common(ctx);
}
```

As can be seen from the code listing above, the code to limit the cipher-suites provided by the server is already in place, but is currently placed behind a code comment. Cure53 recommends uncommenting the code and fixing the underlying issue from 2020-03-15 with ChaCha20, as stated in the comment. Additionally, it should be considered to switch to DTLS1.3 (RFC9147)[8] published in April 2022.

*This issue was assigned a **4.2** CVSS rating, as stipulated in the breakdown of each scoring component offered below:*

https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N

CWE:

- *https://cwe.mitre.org/data/definitions/326.html*
- *https://cwe.mitre.org/data/definitions/328.html*

[8] https://datatracker.ietf.org/doc/html/rfc9147

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.

EXP-13-001 WP3: OOB reads in various server handlers *(Low)*

Fix Note: *The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

During a source code audit of the message handler routines, an OOB read vulnerability was identified. This problem can be triggered by a malicious / bogus client and the vulnerability exists within the two message handler functions *he_handle_msg_data* and *he_handle_msg_deprecated_13*.

In fact, both routines end up calling *internal_handle_data* with a pointer to the “inside packet”, without ensuring that the extracted packet length is smaller than the length argument subtracted by the offset that is used to get the pointer to the “inside packet”. The following code snippets highlight the insufficient length checks.

Affected file:
lightway-core/src/he/msg_handlers.c

Affected code:

```
he_return_code_t he_handle_msg_data(he_conn_t *conn, uint8_t *packet, int
length) {
	[...]
	uint16_t pkt_length;
	if([...]) {
		pkt_length = pkt->length;
	} else {
		pkt_length = ntohs(pkt->length);
	}

	// Check the packet length is sufficient
	if(pkt_length > length) {
		return HE_ERR_PACKET_TOO_SMALL;
	}

	uint8_t *inside_packet = packet + sizeof(he_msg_data_t);
	return internal_handle_data(conn, inside_packet, pkt_length);
}

he_return_code_t he_handle_msg_deprecated_13(he_conn_t *conn, uint8_t *packet,
int length) {
```

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

```
        [...]
        // We use this a lot so convert it just the once
        uint16_t pkt_length = ntohs(pkt->length);

        // Check the packet length is sufficient
        if(pkt_length > length) {
                return HE_ERR_PACKET_TOO_SMALL;
        }

        uint8_t *inside_packet = packet + sizeof(he_deprecated_msg_13_t);
        return internal_handle_data(conn, inside_packet, pkt_length);
}
```

The testing team wants to emphasize that the number of bytes that can be read out-of-bounds is limited to five (*sizeof(he_deprecated_msg_13_t)*) or three (*sizeof(he_msg_data)*). The application may not crash as the adjacent memory that is overread is mapped into the address space of the *lightway-server* process, therefore not resulting in a SIGSEGV. These constraints in terms of exploitability had an impact on the actual rating of this issue, as also reflected by the severity score of *Low*.

Cure53 recommends adapting the length check to ensure that the out-of-bounds read vulnerability is not possible, for example for the check within *he_handle_msg_deprecated_13*, it should be adjusted as follows:

```
he_return_code_t he_handle_msg_deprecated_13(he_conn_t *conn, uint8_t *packet,
int length) {
        [...]
        if(pkt_length > length - sizeof(he_deprecated_msg_13_t)) {
                return HE_ERR_PACKET_TOO_SMALL;
        }
        [...]
}
```

Note also that the issue can be fixed in a similar way for the function *he_handle_msg_data*, too.

*This issue was assigned a **4.3** CVSS rating, as stipulated in the breakdown of each scoring component offered below:*

https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N

CWE: https://cwe.mitre.org/data/definitions/125.html

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

EXP-13-002 WP4: Potential memory leak of pinned pages in TUN driver (*Low*)

Fix Note: *The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

While reviewing the Lightway High-Performance TUN driver (HPT), it was noticed that the error handling of *pin_user_pages* will not unpin the already pinned pages and might, thus, create a memory leak.

The Linux kernel function *pin_user_pages* with return type *long* will return the number of pinned user pages or a negative value which represents the error code. Should *pin_user_pages* not be able to pin all requested pages, it will still return a positive number, indicating the count of pages it was able to pin. The caller then has to either keep calling *pin_user_pages* until all pages were pinned or unpin the already pinned pages and abort. The HPT driver does the latter, but fails to unpin the pages. This can be seen in the code snippet.

Affected file:
xv_helium_tun/kernel/linux/hpt/hpt_core.c

Affected code:

```
down_read(&current->mm->mmap_lock);
	retval = pin_user_pages(buffer_start, npages, FOLL_WRITE, pages, NULL);
	up_read(&current->mm->mmap_lock);

	if (retval != npages) {
		pr_err("could not map all user pages");
		retval = -1;
		goto cleanup;
	}

	pinned = 1;

	nid = page_to_nid(pages[0]); // Remap on the same NUMA node.

	hpt->ring_memory = vm_map_ram(pages, npages, nid);

	if (hpt->ring_memory == NULL) {
		pr_err("cannot vm_map_ram");
		retval = -1;
		goto cleanup;
	}
	hpt->mapped_pages = pages;
	hpt->num_mapped_pages = npages;
	return 0;

cleanup:
```

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

```
if (pinned) {
        unpin_user_pages(pages, npages);
}

if (pages) {
        vfree(pages);
}

return retval;
```

In case *pin_user_pages* does not pin the required number of pages, the code will never set *pinned = 1* and will never call *unpin_user_pages*.

It is recommended to check if the return value of *pin_user_pages* is a positive number, and, if it is, call *unpin_user_pages* before returning output from the function.

*This issue was assigned a **1.9** CVSS rating, as stipulated in the breakdown of each scoring component offered below:*

https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L

CWE: https://cwe.mitre.org/data/definitions/401.html

EXP-13-003 WP2: Default allow approach in *he_domain_filter_handler* (*Info*)

***Note from ExpressVPN:** The ThreatManager feature is designed to allow users to continue browsing as normal, while preventing all apps and websites on your device from communicating with a set of third parties known to track activity or engage in malicious behavior. A default deny approach would require users to specifically add all the websites they will ever visit, which will significantly reduce the user experience without adding significant privacy or security advantages.*

During a source code review of the *xv_libballoon* repository it was noticed that the default approach within the main DNS processing routine *he_domain_filter_handler* is to allow the packet whenever no blocklist has been configured.

The testing team is aware that strictly enforcing VPN users to configure a block /allow-list is an administrative overhead. However, it would offer an additional layer of security and may prevent unintended harm.

Affected file:
xv_libballoon/libballoon-filter/src/packet_filter.c

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

Affected code:

```
static he_packet_filter_decision_t he_domain_filter_handler()
{
        [...]
        he_packet_filter_decision_t decision = HE_PACKET_FILTER_DECISION_PASS;
        [...]
        if(he_domain_cache_find_domain(ctx->whitelist, domain)) {
                return HE_PACKET_FILTER_DECISION_PASS;
        }
        if(he_domain_cache_find_domain(ctx->blacklist, domain)) {
                [...]
        }
        return decision;
}
```

As a defense in-depth approach, the use of a default deny approach could be offered to users.

*This issue was assigned a **0.0** CVSS rating, as stipulated in the breakdown of each scoring component offered below:*

https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

CWE: https://cwe.mitre.org/data/definitions/276.html

EXP-13-004 WP2: Potential integer overflow in *he_strndup()* (*Info*)

***Fix Note:** The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

The function *he_strndup()* contains an integer overflow that might lead to less memory being allocated than what is requested by the caller. The overflow happens when an attacker controls the "*n*" argument of the function *he_strndup()*, e.g. provides a very large value of *MAX_UINT32*. The addition of *MAX_UINT32 + 1* results in *0.* Some allocators do return a valid chunk of memory of, for example, size 8 or 16 bytes, even when calling *malloc(0)*. This could result in memory corruption errors in the given scenario, potentially resulting in further, unspecified harm.

The current code only calls this function with static parameters, thus making an attack infeasible, as also reflected by the severity rating set to *Info*. At the same time, this situation might change in the future. Furthermore, it must be noted that the potential integer overflow only applies to the Windows platform.

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

Affected file:
xv_libballoon/libballoon-util/src/he_malloc.c

Affected code:

```
#ifdef _WIN32
char *he_strndup(const char *s, size_t n) {
       char *m;
       size_t len = strlen(s);
       if(n < len) len = n;
       m = he_malloc(len + 1);
       if(m == NULL) return NULL;
       m[len] = '\0';
       return memcpy(m, s, len);
}
#else
       [...]
}
#endif
```

It is recommended to test if the calculation overflows, for example by using: *__builtin_add_overflow()*[9].

This issue was assigned a 0.0 CVSS rating, as stipulated in the breakdown of each scoring component offered below:

CVSS: -

CWE: https://cwe.mitre.org/data/definitions/190.html

EXP-13-006 WP1: No lower bounds check for message payload handlers (*Info*)

***Fix Note:** The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Inspecting the write callback logic in *lightway-core*, it was noticed that message write callbacks *inside_write_cb()* implemented by Lightway server assume a minimum packet length of 20 bytes when rewriting IPv4 packet headers. There is, however, no check that ensures that the Lightway payload is large enough to contain a full IPv4 header.

As the message processing code in *lightway-core* operates with a fix-sized receiving buffer, this cannot yield an out-of-bounds write, but future changes to the code might change this behavior. As such, an additional check would provide an additional safety net.

[9] https://gcc.gnu.org/onlinedocs/gcc/Integer-Overflow-Builtins.html

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

Affected file:
lightway-core/src/he/msg_handlers.c

Affected code:
```
static he_return_code_t internal_handle_data(he_conn_t *conn, uint8_t
*inside_packet,
                                             uint16_t pkt_length) {
[...]

  // Sanity-check length -- contract says that it can't be longer than
pkt_length but just in case
  if(post_plugin_length > pkt_length) {
    return HE_ERR_FAILED;
  }

  // Validate packet
  if(!he_internal_is_ipv4_packet_valid(inside_packet, post_plugin_length)) {
    // Invalid packet
    return HE_ERR_BAD_PACKET;
  }

  // Packet seems to be fine, hand it over
  if(conn->inside_write_cb) {
    conn->inside_write_cb(conn, inside_packet, post_plugin_length, conn->data);
  }

  return HE_SUCCESS;
}

he_return_code_t he_handle_msg_data(he_conn_t *conn, uint8_t *packet, int
length) {
[...]

  // We use this a lot so convert it just the once
  // Prior to May 2021, a bug here passed the "length" in host-order instead of
network order
  // We have fixed this as of protocol version 1.1 but still support the bug for
older clients
  uint16_t pkt_length;
  if(conn->protocol_version.major_version == 1 && conn-
>protocol_version.minor_version == 0) {
    pkt_length = pkt->length;
  } else {
    pkt_length = ntohs(pkt->length);
  }

  // Check the packet length is sufficient
  if(pkt_length > length) {
    return HE_ERR_PACKET_TOO_SMALL;
```

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

```
  }

  uint8_t *inside_packet = packet + sizeof(he_msg_data_t);

  return internal_handle_data(conn, inside_packet, pkt_length);
}
```

As can be seen from the code listings above, *he_handle_msg_data()* reads the payload length *pkt_length* from the Lightway message and calls *internal_handle_data()* without any lower bounds checks.

In *internal_handle_data(),* the payload itself is checked to have an IPv4 packet header, before the write callback *inside_write_cb()* is called. Note that *he_internal_is_ipv4_packet_valid()* will not check the length of the packet.

Lightway server implements two handlers for *inside_write_cb()*: *hpt_inside_write_cb()* and *tun_inside_write_cb()*. Both of them call *he_rewrite_ip_from_client_to_tun_ipv4()* to modify the IPv4 payload header. This function does not have any size checks and assumes that the length is at least that of the minimal IPv4 header.

Affected file:
xv_helium_server/src/inside_adapters/ip_rewrite.c

Affected code:

```
void he_rewrite_ip_from_client_to_tun_ipv4(he_server_connection_t *conn, uint8_t
*packet,
                                           size_t length) {
  ipv4_header_t *ipv4_hdr = (ipv4_header_t *)packet;
  ipv4_hdr->src_addr = conn->inside_ip;
  he_adjust_packet_checksums(packet, length, conn->state->client_ip_u32, conn-
>inside_ip);
}
```

As a defensive measure, Cure53 suggests including an additional check on *pkt_length* in *he_handle_msg_data()* or *internal_handle_data()* to ensure that the payload has the minimum IPv4 header size (20 bytes).

*This issue was assigned **a 0.0** CVSS rating, as stipulated in the breakdown of each scoring component offered below:*

CVSS: -

CWE: -

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

EXP-13-007 WP4: Marking unpinned user pages as dirty (*Info*)

Fix Note: *The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

A review of the kernel side of the Lightway High-Performance TUN driver in *xv_helium_tun* showed that the driver pins the shared memory supplied by the user-space process with *pin_user_pages()*. On failure or release of the TUN interface, these pinned pages are unpinned using *unpin_user_pages()* in *hpt_unmap_pages()*. As these pages are pinned with the *FOLL_WRITE* flag, they must be marked as *dirty* when they are unpinned[10].

While it is unlikely that this has any security impact in the current codebase with a recent Linux kernel, it can cause unwanted, hard to debug side-effects and potential data loss. Additionally, any future changes to the Linux kernel or the Lightway programs might trigger issues around it.

Affected file:
src/xv_helium_tun/kernel/linux/hpt/hpt_core.c

Affected code:

```
static void hpt_unmap_pages(struct hpt_dev *hpt)
{
	if (hpt->ring_memory) {
		pr_info("Freeing ring memory\n");

		// First unmap the memory from the virtual table
		vm_unmap_ram(hpt->ring_memory, hpt->num_mapped_pages);

		pr_info("Unpinning user pages");

		// Then unpin all the pages we made sure don't get swapped
		unpin_user_pages(hpt->mapped_pages, hpt->num_mapped_pages);

		pr_info("Freeing page list");

		// Next free the kernel mem we used to store the page list
		vfree(hpt->mapped_pages);
		pr_info("Freed mapped pages");

		// Now null everything
		hpt->mapped_pages = NULL;
		hpt->num_mapped_pages = 0;
		hpt->tx_start = NULL;
		hpt->rx_start = NULL;
```

[10] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/mm/gup.c?h=v6.0#n1122

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

```
                hpt->tx_ring = NULL;
                hpt->rx_ring = NULL;
                hpt->ring_memory = NULL;
                hpt->num_ring_memory = 0;

                pr_info("Free'd ring memory\n");
        }
}
```

Cure53 recommends to instead use *unpin_user_pages_dirty_lock()* (with *make_dirty=true)* which achieves the same effect, but additionally ensures that the relevant pages are marked as *dirty* when needed.

*This issue was assigned a **0.0** CVSS rating, as stipulated in the breakdown of each scoring component offered below:*

CVSS: -

CWE: -

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

Conclusions

Drawing on the combination of factors, namely the comprehensive coverage, low number of findings, and an absence of high-impact problems, it can be concluded that this Cure53 assessment of the ExpressVPN Lightway components concludes with a positive result. The testing team, which dedicated thirty days to examining the components in scope in November 2022, confirms that the inspected Lightway components are already in a very good state of security. Minor vulnerabilities and recommendations should be addressed or considered to further promote a stable security premise.

It should be clarified that the assessment featured five repositories, namely *lightway-core, xv_helium_cli, xv_helium_server, xv_helium_tun* and *xv_libballoon*. For these repositories, four work packages were defined. WP1 comprised a source code-assisted penetration test against the Lightway server component, WP2 entailed a review of the Lightway client, WP3 related to the Lightway shared library and, finally, WP4 focused on the Lightway High-Performance TUN driver. Throughout the project, Cure53 was in constant communication with the customer through a dedicated Slack channel. Help was provided whenever requested and the communication was excellent.

All source code packages consist mostly of C code. In terms of complexity, the *xv_helium_cli* folder was by far the largest component to be audited, followed by the *lightway-core* folder, which gets used by the Lightway client and server. The Lightway High-Performance TUN driver is comparably smaller in terms of size and complexity. Generally speaking, the testers got the impression that the repositories and the source code are clearly organized, and that security was integral to the development and design of all Lightway components in-scope.

The assessment included deep-dive code reviews of all provided repositories, but also a dynamic test of the Lightway client and server. Adhering to best practices for examining compiled languages such as C and C++, particular attention was paid toward unearthing memory-corruption flaws, such as integer overflows, buffer overflows, and out-of-bound reads. Similarly, logic bugs and issues that manifest in Denial-of-Service situations were subject to rigorous scrutiny.

Overall, the Lightway project uses a fitting defensive coding style and contains multiple inputs checks, which is also reflected by the lack of typical buffer overflow (OOB write) vulnerabilities. Also, the components in-scope have been audited with regards to race conditions. An initial idea was to cause a race condition within the Lightway main message processing functions. However, after consulting with ExpressVPN, it turned out that this part of Lightway is single-threaded and it uses an event loop instead. Therefore, there is no risk of data races which may be exploited by an attacker.

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

Another relevant observation was that the Lightway codebase often compares different integers, which may have unintended effects. Potentially, it can translate to security vulnerabilities, as the C programming language may implicitly downcast (and thus truncates) integer values. To the testers' surprise, no actual vulnerability related to that problem could be identified.

In summary, Cure53 identified three vulnerabilities of *Low* severity and six miscellaneous issues. Among them EXP-13-005 affects the Lightway server and can lead to a complete stall of I/O processing in case an attacker gains the capability to flood the server's (regular) TUN interface with packets. EXP-13-009 identifies a weak configuration of the DTLS cipher-suites of the Lightway server. As the most common way to tunnel packets is via UDP, this might affect a lot of server instances. On the positive side, official Lightway clients mitigate this by limiting their list of supported DTLS cipher-suites to versions that are known to be secure.

A deep dive code review of the Lightway High-Performance TUN Linux Kernel Driver supports the good impression gained for all other components. As one exception, EXP-13-008 describes a potential integer underflow in the HPT TUN driver. As userspace and kernel share a memory region, special care must be taken in the driver as the user process can modify this memory region at any time. Overall, the code does a very good job with this and contains multiple checks for such issues.

As noted in the *Methodology* section, fuzz testing harnesses have been developed. While testing is essential for any project, its importance grows with the scale of the endeavor. Since the Lightway project is implemented in languages such as C and C++, which are often prone to memory corruption vulnerabilities, continuous and multifaceted evaluations are warranted.

For the purpose of this test, code coverage-driven fuzzing using AFL++ in combination with address space sanitizers (such as ASAN) of selected message handler routines and other selected functions have been performed. Even though no new issues were identified using the developed harnesses, future work should consider fuzz testing as a fixed part of the Lightway software project, either using AFL++ or libFuzzer. It is highly encouraged to incorporate fuzz testing against a build while having features like address space sanitization (ASAN) enabled. This would help to spot memory corruption vulnerabilities which might be missed during a manual code review.

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

Similarly, the well-known static analysis utility CodeQL has been used to search for specific bug patterns within the source code repositories in-scope. Cure53 wishes to point out that using CodeQL can bring a lot of benefits, as it operates on the AST of the program to be analyzed. In this way, it permits deep inspection of the code using taint tracking / analysis and variant analysis.

Finally, it should be underscored that the Lightway solution uses third-party libraries for specific operations. For instance, it leverages WolfSSL for encryption and decryption of the secure channel. As a consequence, the security of Lightway also depends on the safety guarantees and robustness of all third-party libraries in use. Cure53 recommends keeping the dependencies - especially the WolfSSL library - under a strict update regiment to plug any security issues in a timely fashion.

Moving forward, the Lightway codebase could profit from recurring security audits as the complexity of all components is challenging to handle from a security perspective. It must be acknowledged - in the frame of this November 2022 Cure53 project and beyond - that changes within one part of the ExpressVPN system may have unintentional security impact on other parts.

Cure53 would like to thank Brian Schirmacher, Harsh S, Andre Lo, Raihaan Shouhell, and Pete Membrey from the ExpressVPN team for their excellent project coordination, support and assistance, both before and during this assignment.