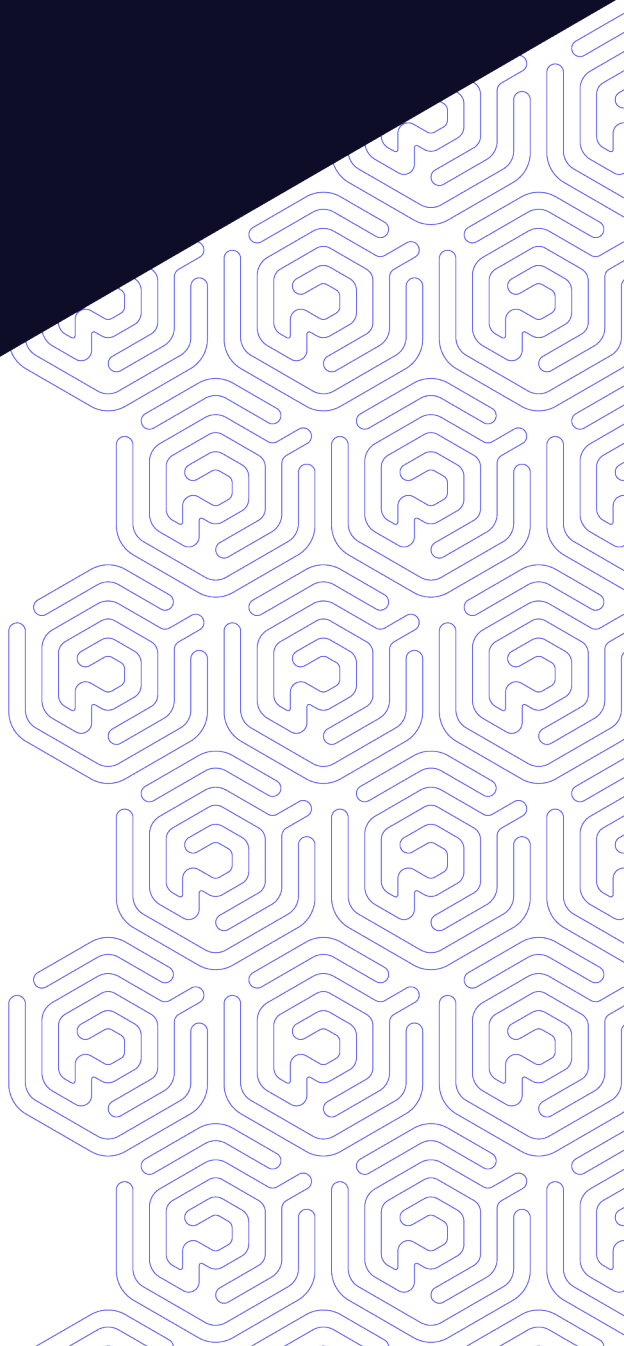
ExpressVPN

December 20, 2024

ExpressVPN Lightway Security Assessment



praetorian

praetorian

Table of Contents

Contributors

Elizabeth Buckley
Forrest Carver
Saullo Branco

Praetorian

6001 W Parmer Ln
Ste 370
PMB 2923
Austin, TX 78727

Tel +1.512.410.0350
Fax +1.512.410.0356

praetorian.com

Retest Addendum

This report contains Praetorian's findings from a security assessment of Lightway carried out between September 16, 2024 and October 10, 2024.

ExpressVPN implemented remediations following the security assessment and Praetorian retested each finding between December 16, 2024 and December 17, 2024. During the retest, Praetorian determined whether ExpressVPN had fixed each finding and implemented corresponding mitigating controls.

Praetorian has updated each finding in this report to indicate whether the finding was fixed, partially fixed, or not fixed.

The table below provides a summary of the retest outcomes by severity level.

	Original Findings	Fixed	Not Fixed	Partially Fixed	Accepted Risk	Unverified
Critical	0	0	0	0	0	0
High	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Low	2	2	0	0	0	0
Info	0	0	0	0	0	0

For details of ExpressVPN's remediations and retest procedures for each finding, please see the Summary of Weaknesses:

- Summary of Weaknesses: Risk-Informed Security Assessment

Executive Summary

ExpressVPN engaged Praetorian to perform a risk-informed security assessment to determine the security posture of the Lightway protocol and its components, which have been rewritten in Rust. The engagement was scoped in Statement of Work 2024-08-2601, and Praetorian completed the work between September 16, 2024 and October 10, 2024. Our risk-informed assessment focused on people, processes, and technology as we pursued ExpressVPN's primary goal of identifying security weaknesses in the cryptographic primitives and opportunities to corrupt the components' memory and undermine confidentiality, integrity, and authenticity.

Praetorian uncovered two (2) low-risk findings related to security configuration and insufficient validation issues as follows:

- Insufficient File Permissions Validation (Low).
- Sensitive Data Exposed on the Command Line (Low).

During the retest, Praetorian confirmed that ExpressVPN fixed each finding and implemented corresponding mitigating controls.

BUSINESS IMPACT

Praetorian identified attack vectors that allowed adversaries with low privileges on the machine running the Lightway components to access sensitive data, including credentials, server certificates, and keys, if users configured Lightway components insecurely. Their probability of exploitation was likely low since they required previous compromise of the machine and insecure configuration of the components.

If users configured the Lightway components with world-readable and/or world-writable configuration files, the components would accept them without warnings. Attackers with access to the machine could then obtain the configuration files, set up a malicious server, or overwrite the client configuration to induce a connection to an attacker-controller server to intercept the client traffic.

Additionally, if users passed their credentials via command-line arguments to launch the Lightway components, local attackers could obtain them through process listing, connect to the Lightway server and access its private network from another machine.

ENGAGEMENT SCOPE

For further details on the scope of the engagement, please see Appendix A: Engagement Scope.

EFFECTIVE CONTROLS

Effective controls are components of ExpressVPN's people, processes, or technology that mitigate security risks. Praetorian noted that the following effective controls were particularly beneficial and warrant special recognition.

Secure Usage of Rust Unsafe Blocks: Praetorian observed that the `unsafe` blocks had `SAFETY` comments and were executed in secure conditions. The blocks contained function calls that depended on internally defined variables or respected safe invariants. The secure usage of `unsafe` blocks helps to prevent out-of-bounds reads and writes, and memory corruption issues.

Strong Cryptographic Primitives: ExpressVPN built the Lightway protocol on WolfSSL using strong cryptographic primitives. The Lightway components supported different protocols depending on their connection mode. The Lightway client supported TLSv1.3 in TCP mode and DTLSv1.3 in UDP mode, while the Lightway server supported TLSv1.3 in TCP mode and DTLS versions 1.2 and 1.3 in UDP mode. The Lightway components used only AEAD ciphers based on AES-256 and Chacha20. Those primitives effectively protected the encrypted traffic against replay, injection, tampering, and cache-timing attacks.

Responsive Security Team: The ExpressVPN security team was responsive to questions, roadblocks, and findings raised by Praetorian throughout the engagement. This responsiveness showed desire and commitment to improving ExpressVPN's security posture.

NOTABLE OBSERVATIONS

Notable observations represent a high-level overview of issues identified within the environment that warrant expedited attention. Praetorian identified one (1) notable observation, and we recommend that ExpressVPN prioritize its remediation efforts accordingly.

Insufficient Validation and Misconfiguration in the Lightway Components: Praetorian observed that Lightway did not enforce strict file permissions and accepted credentials as command-line arguments. The overly permissive settings on configuration files, combined with the use of command-line arguments for credentials, could expose sensitive information to adversaries on the machine running the components if the users configured Lightway components insecurely.

OTHER OBSERVATIONS

Other observations represent areas with no security impact where ExpressVPN could make minor improvements to improve the overall quality of Lightway. Praetorian shared five (5) observations with ExpressVPN throughout the security assessment.

Insufficient Validation of Credentials: Praetorian observed that the Lightway components did not properly validate if the user provided credentials. The component launched without credentials used empty strings as `user` and `password` for authentication. That behavior didn't pose a security risk because Lightway provided an authentication framework. The implementer must define the authentication details leveraging it.

Insufficient Validation of the `iouring_entry_count` argument: Praetorian observed that the Lightway components needed to validate the `iouring_entry_count` argument properly. The components panicked when the argument was set to 0 and couldn't communicate when it was set to 1. That behavior didn't pose a security risk because it only blocked communication as any component misconfiguration would.

Broken Paths in the Test Configuration Files: Praetorian observed that the Lightway test configuration files had broken paths. This was merely an inaccurate documentation and not a code quality issue. If one didn't modify the `tests/client/client_config.yaml` and `tests/server/server_config.yaml` files, the broken paths would cause an error while launching the Lightway components with the commands in the Dev-Testing section of the Lightway repository README file. ExpressVPN addressed this promptly to improve the quality of the Lightway documentation.

Missing Explicit IP Forward Settings in Test Script: Praetorian observed the `tests/setup.sh` script in the Lightway repository missed `net.ipv4.ip_forward=1` setting commands. Not explicitly setting the key on the `lightway-middle` and `lightway-server` components could prevent the testing environment from working. ExpressVPN addressed this promptly to ensure that a usable testing and development setup could be easily configured.

Redundant Commands in Instructions to Decrypt TLS Traffic: Praetorian observed that the instructions for decrypting TLS traffic with Wireshark in the test environment had redundant and conflicting commands. The first command launched the `lightway-client` without the `--keylog` flag, while the second one launched the component after it had been built. ExpressVPN addressed this promptly to ensure that a usable testing and development setup could be easily configured.

OVERALL RECOMMENDATIONS

Praetorian recommends that ExpressVPN consider undertaking certain initiatives that will both directly improve their security posture and keep their users safer. While these often require extensive planning and execution, some initiatives are not complex and can reduce risk quickly, improve security policies, and support defense-in-depth procedures.

Praetorian recommends undertaking the following initiative within the technology category. Their relative urgency is noted in accordance with Table 1.

Risk Category	Timeline for Implementation
High Risk (H)	One week
Medium Risk (M)	One month
Low Risk (L)	Three months

Table 1: Priority of recommendations.

Technology

L **Enforce strong controls in the Lightway components:** Praetorian observed insufficient validation and security misconfigurations in the Lightway components. Those security gaps could expose sensitive data to malicious actors if users used the Lightway components insecurely. Praetorian recommends that ExpressVPN validate the permissions of all configuration files, read credentials from secure sources, and warn users about any insecure usage.

CONCLUSION

In light of the above findings, their potential business impact and the retest results, Praetorian recommends that ExpressVPN continue enforcing strong controls in the Lightway components to prevent unauthorized access to their sensitive data.

Praetorian appreciates the opportunity to conduct a risk-informed security assessment on behalf of ExpressVPN. We stand ready to answer any questions about this report and act as a strategic partner in facilitating ExpressVPN's movement toward a stronger, more adaptable security posture.

Threat Model and Test Cases

Praetorian's threat modeling methodology begins by identifying the Lightway attack surface and critical assets. Praetorian then enumerates potential attack paths that lead from these attack surfaces to the critical assets.

For each attack path, Praetorian tests a defined set of test cases. In this manner, Praetorian uses the threat model to guide testing activities.

SYSTEM COMPONENTS AND TRUST BOUNDARIES

Praetorian decomposed the system under test into various components and trust boundaries. See Figure 1 below.

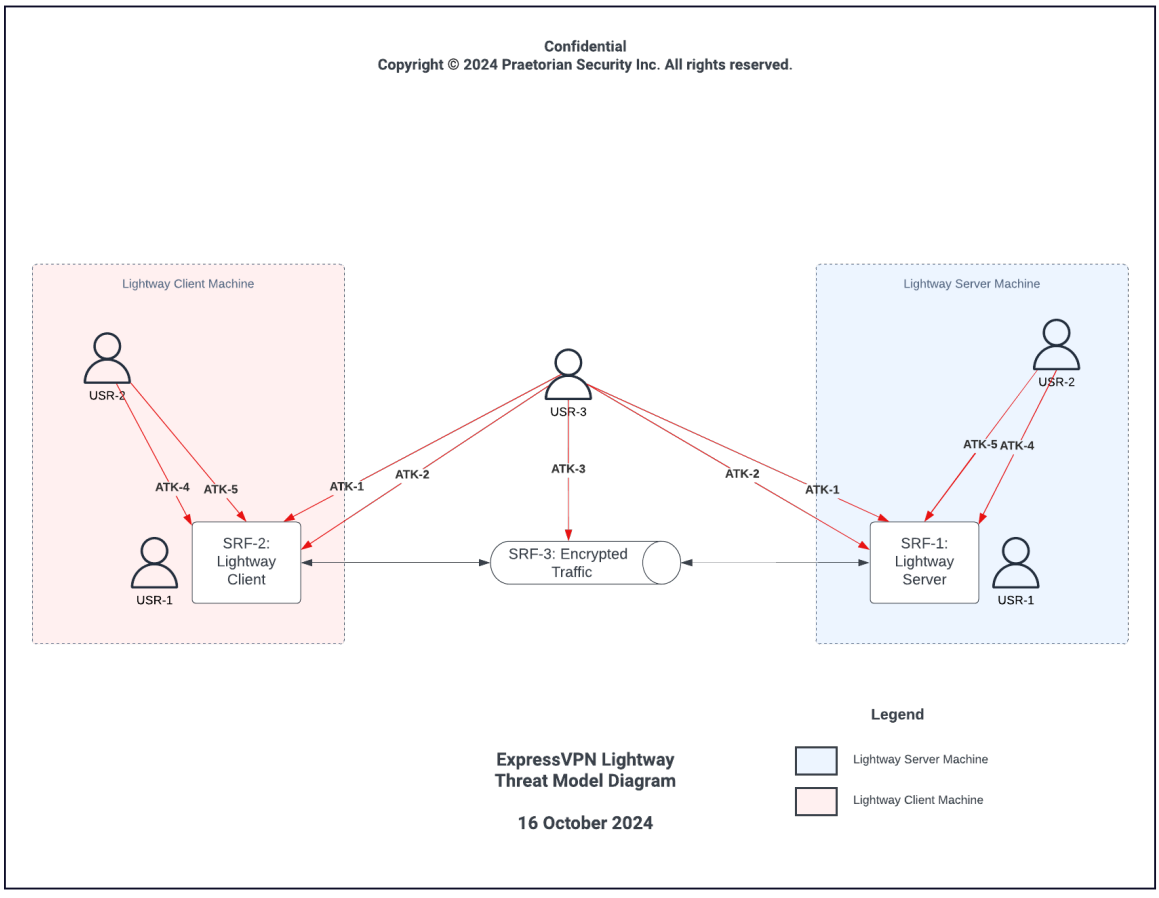


Figure 1: A diagram of major system components and trust boundaries for Lightway.

USER ROLES

The Lightway application incorporated the following security-relevant user roles evaluated by Praetorian:

User Role	Description
USR-1: Lightway User	Users of the Lightway components. They are responsible for setting up the Lightway Rust server and/or connecting to it using the Lightway Rust client.
USR-2: Peer User	User with permission to execute arbitrary code on a low-privileged process on the machine running the Lightway components.
USR-3: Person-in-the-Middle User	User positioned between the server and the client to sniff and modify encrypted traffic.

CRITICAL ASSETS

Praetorian pursued a compromise of the following critical assets within the system under test.

Critical Asset	Description
CRT-1: Lightway Rust client	The client application implements the Lightway protocol, connects, and communicates with the Lightway server.
CRT-2: Lightway Rust server	The server application implements the Lightway protocol and facilitates communication with the Lightway client.
CRT-3: Lightway protocol	It is the public name for ExpressVPN's VPN protocol. Designed to be light on its feet, Lightway runs faster, uses less battery, and is easier to audit and maintain.

ATTACKER TERMINAL GOALS

Praetorian considered threats to the confidentiality, integrity, and availability of the system's critical assets. These threats might be exploited by an attacker seeking to achieve the following terminal goals:

Terminal Goal	Description
TGO-1: Compromise or weaken the Lightway protocol	Attack the cryptographic functions and primitives to sniff and/or modify the encrypted traffic between the Lightway components.
TGO-2: Modify the behavior of the components	Perform unintended writes that lead to changes in the component's state.
TGO-3: Unauthorized access to component's data	Perform unintended reads of or parts of the component's state.

ATTACK SURFACE

Praetorian considered attack paths originating from the following exposed attack surfaces.

Attack Surface	Description
SRF-1: Lightway Rust client	The client application implements the Lightway protocol, connects, and communicates with the Lightway server.
SRF-2: Lightway Rust server	The server application implements the Lightway protocol and facilitates communication with the Lightway client.
SRF-3: Encrypted traffic	The encrypted traffic flows through the tunnel between the server and the client.

ATTACK PATHS

Praetorian attempted to exploit each of the attack paths shown below. Please see Test Cases for specific mechanisms of attack.

Attack Path	Description
ATK-1: PitM user compromises and/or weakens the Lightway protocol.	The PitM user exploits security flaws or weaknesses to attack the cryptographic functions and primitives and access the secured communication.
ATK-2: PitM user performs unintended writes to the components' memory.	The PitM user sends crafted messages to the Lightway Rust component to corrupt the process memory.
ATK-3: PitM user performs unintended reads from the components' memory.	The PitM user sends crafted messages to the Lightway Rust component, leading to data leakage.
ATK-4: Peer user modifies the behavior of the Lightway components.	The peer user abuses weaknesses in the components, performs unintended writing, and interferes with their behavior.
ATK-5: Peer user accesses sensitive data from the Lightway components.	The peer user abuses weaknesses in the components to obtain sensitive data.

TEST CASES

Praetorian executed the test cases shown below to determine the exploitability of the attack paths listed in Attack Paths.

ATK-1: PitM user compromises and/or weakens the Lightway protocol.
TC-1.1: Attempt to decrypt the encrypted traffic flowing through the tunnel as a PitM user.
TC-1.2: Attempt to perform packet injection as a PitM user.
TC-1.3: Attempt to perform network replay attacks as a PitM user.
TC-14: Attempt to inject anomalies into the communication as a PitM user.
TC-15: Attempt to perform analysis of traffic patterns as a PitM user.
TC-16: Attempt to compromise or weaken the cryptographic functions as a PitM user.

ATK-2: PitM user performs unintended writes to the components' memory.
TC-2.1: Attempt to perform arbitrary write operations as a PitM user.
TC-2.2: Attempt to trigger a Remote Code Execution (RCE) as a PitM user.
TC-2.3: Attempt to perform a Denial of Service (DoS) as a PitM user.

ATK-3: PitM user performs unintended reads from the components' memory.
TC-3.1: Attempt to perform arbitrary read operations as a PitM user.
TC-3.2: Attempt to violate the Lightway privacy policy as a PitM user.

ATK-4: Peer user modifies the behavior of the Lightway components.
TC-4.1: Attempt to perform arbitrary write operations as a peer user.
TC-4.2: Attempt to trigger a Remote Code Execution (RCE) as a peer user.
TC-4.3: Attempt to perform a Denial of Service (DoS) as a peer user.

ATK-5: Peer user accesses sensitive data from the Lightway components.
TC-5.1: Attempt to perform arbitrary read operations as a peer user.
TC-5.2: Attempt to violate the Lightway privacy policy as a peer user.

Client/Server Application Assessment

During the desktop application security assessment, Praetorian identified vulnerabilities within the application. Praetorian examined all identified vulnerabilities to determine whether they can be exploited by an attacker to compromise targeted systems and/or used to gain access to sensitive information.

SUMMARY OF WEAKNESSES

The detailed findings section describes potential vulnerabilities, the likelihood or difficulty of exploitation, the relative level of impact on ExpressVPN's business, and Praetorian's recommendations. Vulnerabilities are arranged in order of business impact, with the critical-impact issues appearing first. The following findings have the potential to impact the confidentiality, integrity, and/or availability of ExpressVPN assets.

Critical Risk Findings

- None

High Risk Findings

- None

Medium Risk Findings

- None

Low Risk Findings

- Insufficient File Permissions Validation - Fixed
- Sensitive Data Exposed On Command Line - Fixed

Informational Risk Findings

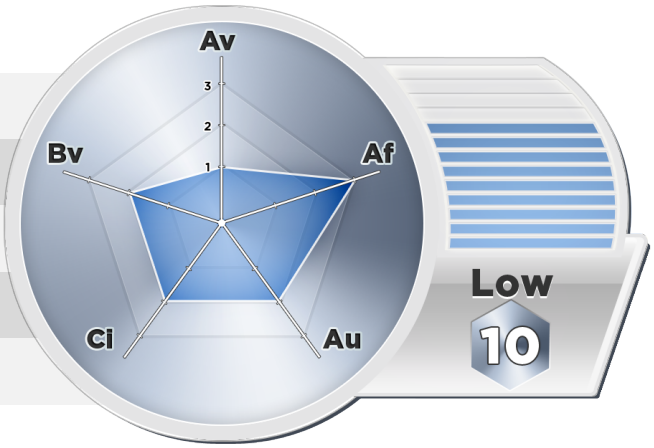
- None

Low Risk Findings

FINDING

Insufficient File Permissions Validation

Access Vector	(Av)	(1) Local
Attack Feasibility	(Af)	(3) Demonstrated
Authentication	(Au)	(2) User
Compromise Impact	(Ci)	(2) Partial
Business Value	(Bv)	(2) System



CVSS v3.1 Risk Rating: 6.3 (Medium)

CVSS Vector String: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:L

Metric	Rating
Attack Vector (AV)	Local (AV:L)
Attack Complexity (AC)	Low (AC:L)
Privileges Required (PR)	Low (PR:L)
User Interaction (UI)	None (UI:N)
Scope (S)	Changed (S:C)
Confidentiality Impact (C)	Low (C:L)
Integrity Impact (I)	Low (I:L)
Availability Impact (A)	Low (A:L)

Status

Fixed

ExpressVPN integrated the `fs_mistrust` Rust library to validate the permissions of sensitive files. `lightway-client` refused to use configuration files that were world-accessible. `lightway-server` applied the same validation to its sensitive files: the server private key and the user database files. Therefore, the Lightway components only used sensitive files that were not world-readable or world-writable.

Vulnerability Description

The Lightway client and server used configuration files such as certificates and private keys to set up their operation variables. After cloning the Lightway repository, those files were world-readable. Praetorian modified them to add the world-writable attribute and observed that those components didn't perform sufficient validation of the files' permissions. The lax permissions allowed on these files exposed sensitive information to any user on the machine.

Impact

Insufficient permission validations allow other users to access data or perform actions they should not be allowed to access or execute. Since those files were world-readable after cloning the `expressvpn/lightway` repository, they would be exposed to unauthorized access without warnings from the Lightway components if the user used them with their original permissions.

In this scenario, attackers on the same machine could access the system credentials, connect to the Lightway server, and access its private network, if one existed, from another machine. Those attackers could also access the server certificate and private key, set up a malicious server masquerading as the legitimate server, and intercept client traffic.

Additionally, local attackers could overwrite the client configuration files to induce a connection to an attacker-controlled server, allowing access to the client's communication if the user mistakenly made the configuration files world-writable.

Components Impacted

- Lightway Client
- Lightway Server

Verification and Attack Information

Praetorian identified this risk by making the configuration files world-writable and world-readable with the command `chmod 777 <configuration-file>` and launching the Lightway components with no warnings.

The Lightway server used the `server_config.yaml`, `server.crt`, and `server.key` files to set up its operation variables. Figure 2 shows that those files had world-readable and world-writable permissions, and figure 3 shows that the Lightway server operated with no complaints about their excessive permissions.

```
root@lw-client:~/targets/lightway# ls -lha tests/server/server_config.yaml
-rwxrwxrwx 1 me me 549 Sep 24 10:38 tests/server/server_config.yaml
root@lw-client:~/targets/lightway# ls -lha tests/certs/
total 24K
drwxrwxr-x 2 me me 4.0K Sep 17 11:54 .
drwxrwxr-x 7 me me 4.0K Sep 18 22:54 ..
-rwxrwxrwx 1 me me 1.3K Sep 17 11:54 ca.crt
-rw-rw-r-- 1 me me  163 Sep 17 11:54 Earthfile
-rwxrwxrwx 1 me me 1.3K Sep 17 11:54 server.crt
-rwxrwxrwx 1 me me 1.7K Sep 17 11:54 server.key
```

Figure 2: The Lightway server configuration files had excessive permissions.

```
2024-09-24T13:59:24.717303Z  INFO lightway_server: Server started with inside ip: 10.125.0.1
2024-09-24T13:59:24.788702Z  INFO lightway_server::io::outside::tcp: Accepting traffic on 0.0.0.0:27690
2024-09-24T13:59:24.789828Z  INFO lightway_server::statistics: Session Statistics total=0 current=0 standby.five_minutes=0 standby.fifteen_minute
five_minutes=0 active.fifteen_minutes=0 active.sixty_minutes=0 pending_session_id_rotations=0
2024-09-24T13:59:24.789896Z  INFO lightway_server::statistics: IP Statistics current=0
2024-09-24T13:59:52.731460Z  INFO lightway_core::connection: state=LinkUp
2024-09-24T13:59:52.731779Z  INFO lightway_server::ip_manager: Alloc ip=10.125.119.38
2024-09-24T13:59:52.732201Z  INFO lightway_core::connection: state=Online
2024-09-24T13:59:52.732372Z  INFO lightway_server::connection_manager: State changed for 192.168.0.2:56130 session=00ed478948fcecc2 state=Online
2024-09-24T13:59:52.732441Z  INFO lightway_server::connection_manager: State changed for 192.168.0.2:56130 session=00ed478948fcecc2 state=LinkUp
2024-09-24T13:59:54.789916Z  INFO lightway_server::statistics: Session Statistics total=1 current=1 standby.five_minutes=0 standby.fifteen_minute
five_minutes=1 active.fifteen_minutes=1 active.sixty_minutes=1 pending_session_id_rotations=0
2024-09-24T13:59:54.790045Z  INFO lightway_server::statistics: IP Statistics current=1
```

Figure 3: The Lightway server didn't complain about its configuration files' excessive permissions.

Similarly, the Lightway client used the `client_config.yaml` and `ca.crt` to set up its operation variables. Figure 4 shows that those files had world-readable and world-writable permissions, and figure 5 shows that the Lightway client operated with no complaints about their excessive permissions.

```
root@lw-client:~/targets/lightway# ls -lha tests/client/client_config.yaml
-rwxrwxrwx 1 me me 477 Sep 24 10:39 tests/client/client_config.yaml
root@lw-client:~/targets/lightway# ls -lha tests/certs/ca.crt
-rwxrwxrwx 1 me me 1.3K Sep 17 11:54 tests/certs/ca.crt
```

Figure 4: The Lightway client configuration files had excessive permissions.

```
2024-09-24T14:03:39.740402Z  INFO lightway_core::connection::builders: New Connection inside_mtu=1350
2024-09-24T14:03:39.742682Z  INFO lightway_core::connection: state=LinkUp
2024-09-24T14:03:39.742807Z  INFO lightway_core::connection: state=Authenticating
2024-09-24T14:03:39.784397Z  INFO lightway_core::connection: Authentication succeeded config=AuthSuccessWithConfigV4 { local_ip: "169.254.10.2",
69.254.10.5", mtu: "1350", session: 12a4fa301d420aca }
2024-09-24T14:03:39.784463Z  INFO lightway_core::connection: state=Online
```

Figure 5: The Lightway client didn't complain about its configuration files' excessive permissions.

Note: Praetorian observed that the test configuration files were world-readable after cloning the Lightway repository. Praetorian made them world-writable as well to assess the Lightway components' validations.

```
me@lw-client:~/targets$ git clone git@github.com:expressvpn/lightway.git
Cloning into 'lightway'...
remote: Enumerating objects: 2000, done.
remote: Counting objects: 100% (860/860), done.
remote: Compressing objects: 100% (281/281), done.
remote: Total 2000 (delta 681), reused 697 (delta 576), pack-reused 1140 (from 1)
Receiving objects: 100% (2000/2000), 809.10 KiB | 1.52 MiB/s, done.
Resolving deltas: 100% (1124/1124), done.
me@lw-client:~/targets$ ls -lha lightway/tests/client/
total 24K
drwxrwxr-x 2 me me 4.0K Oct 28 17:50 .
drwxrwxr-x 7 me me 4.0K Oct 28 17:50 ..
-rw-rw-r-- 1 me me  472 Oct 28 17:50 client_config.yaml
-rwxrwxr-x 1 me me 1.8K Oct 28 17:50 docker-entrypoint.sh
-rw-rw-r-- 1 me me  641 Oct 28 17:50 Earthfile
-rwxrwxr-x 1 me me 2.6K Oct 28 17:50 run-test-inside.sh
me@lw-client:~/targets$ ls -lha lightway/tests/server/
total 32K
drwxrwxr-x 2 me me 4.0K Oct 28 17:50 .
drwxrwxr-x 7 me me 4.0K Oct 28 17:50 ..
-rwxrwxr-x 1 me me 1.6K Oct 28 17:50 docker-entrypoint.sh
-rw-rw-r-- 1 me me  740 Oct 28 17:50 Earthfile
-rw-rw-r-- 1 me me   66 Oct 28 17:50 lwpasswd
-rw-rw-r-- 1 me me  549 Oct 28 17:50 server_config.yaml
-rw-rw-r-- 1 me me 1.7K Oct 28 17:50 token.priv
-rw-rw-r-- 1 me me  451 Oct 28 17:50 token.pub
```

Figure 6: The test configuration files were world-readable after cloning the Lightway repository.

Recommendation

Praetorian recommends validating the permissions of all configuration files to ensure they are accessible to the expected users only.

Files containing sensitive information should be made accessible only to the user or, at most, to the user and their group. Praetorian also recommends that the user running the Lightway component own all configuration files.

References

OWASP: DA5 - Improper Authorization

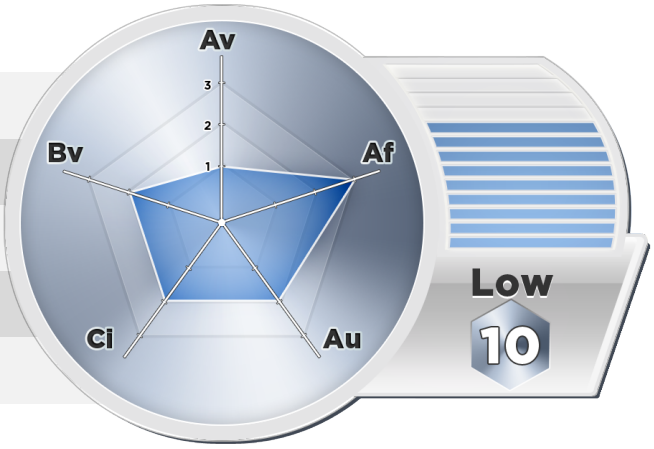
OWASP: File System - Insecure Permissions

DigitalOcean: An Introduction to Linux Permissions

FINDING

Sensitive Data Exposed on Command Line

Access Vector	(Av)	(1) Local
Attack Feasibility	(Af)	(3) Demonstrated
Authentication	(Au)	(2) User
Compromise Impact	(Ci)	(2) Partial
Business Value	(Bv)	(2) System



CVSS v3.1 Risk Rating: 3.8 (Low)

CVSS Vector String: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:L/I:N/A:N

Metric	Rating
Attack Vector (AV)	Local (AV:L)
Attack Complexity (AC)	Low (AC:L)
Privileges Required (PR)	Low (PR:L)
User Interaction (UI)	None (UI:N)
Scope (S)	Changed (S:C)
Confidentiality Impact (C)	Low (C:L)
Integrity Impact (I)	None (I:N)
Availability Impact (A)	None (A:N)

Status

Fixed

ExpressVPN hide the sensitive arguments from the help message and added warnings to the repository `README.md` file to inform users of their usage risks. `lightway-client` didn't show the username, password or token arguments in its help message. At the same time, `lightway-server` obtained the credentials from a user database and didn't use sensitive arguments. Additionally, ExpressVPN suggests using configuration files as secure default and kept the command line argument option to maintain ease of use for users that accept its risk.

Vulnerability Description

The Lightway client and server accepted the username and password as command line arguments.

Impact

Other users or applications on the system can see these arguments through process listings if the Lightway users launched the components with credentials on the command line. In this instance, other users could see the credentials for the Lightway server and use this information to access its private network.

Note: Praetorian observed that although Lightway supported passing credentials on the command line, ExpressVPN didn't guide users to use this method. Therefore, this finding would affect only users configuring the components non-standardly.

Components Impacted

- Lightway Server
- Lightway Client

Verification and Attack Information

Praetorian identified this vulnerability through a code review of the application. Praetorian found that the components' command accepted the user and password arguments to override the values in the configuration file.

The snippet below shows the affected source code from the `lightway-server/src/args.rs` and `lightway-client/src/args.rs` files.

```
pub struct Config {
  ...
  /// Username for auth
  #[clap(long, default_value_t)]
  pub user: String,

  /// Password for auth
  #[clap(long, default_value_t)]
  pub password: String,
  ...
}
```

Praetorian confirmed that the Lightway client accepted the `--password` argument while launching it with `root` permissions. Praetorian then could access the sensitive command argument with the `me` (regular user) permissions. The Lightway server behaved similarly.

```
root@lw-client:~/targets/lightway# ./target/debug/lightway-client --config-file tests/client/client_config.yaml --server
server:27690 --keylog "/tmp/client.log" | grep connection
2024-09-24T18:23:32.199313Z  INFO lightway_core::connection::builders: New Connection inside_mtu=1350
2024-09-24T18:23:32.201784Z  INFO lightway_core::connection: state=LinkUp
2024-09-24T18:23:32.201835Z  INFO lightway_core::connection: state=Authenticating
Error: Outside IO loop exited: Ok(Err(Unauthorized))
root@lw-client:~/targets/lightway# ./target/debug/lightway-client --config-file tests/client/client_config.yaml --server
server:27690 --keylog "/tmp/client.log" --password pass | grep connection

2024-09-24T18:23:49.484973Z  INFO lightway_core::connection::builders: New Connection inside_mtu=1350
2024-09-24T18:23:49.486786Z  INFO lightway_core::connection: state=LinkUp
2024-09-24T18:23:49.486857Z  INFO lightway_core::connection: state=Authenticating
2024-09-24T18:23:49.529082Z  INFO lightway_core::connection: Authentication succeeded config=AuthSuccessWithConfigV4 { lo
cal_ip: "169.254.10.2", peer_ip: "169.254.10.1", dns_ip: "169.254.10.5", mtu: "1350", session: a695af8d4363524f }
2024-09-24T18:23:49.529145Z  INFO lightway_core::connection: state=Online
```

Figure 7: The Lightway client could connect after using the `--password` command argument.

```
me@lw-client:~/targets/lightway$ ps aux | grep lightway | grep -v sudo
root       41611  0.3  0.0 354592 12196 pts/5    Sl+  15:45   0:00 ./target/debug/lightway-server --config-file tests/ser
ver/server_config.yaml --password praetorian
root       41618  0.0  0.0 419420 10368 pts/7    Sl+  15:45   0:00 ./target/debug/lightway-client --config-file tests/cli
ent/client_config.yaml --server server:27690 --keylog /tmp/client.log --password praetorian
me         41627  0.0  0.0  17812  2304 pts/11   S+   15:45   0:00 grep --color=auto lightway
```

Figure 8: The regular user `me` accessed the `--password` command arguments through process listing.

Recommendation

Praetorian recommends reading passwords or other sensitive information from a file, an environment variable, or through interactive user input. If using a file, then the file should have the appropriate user permissions.

References

CWE-214: Invocation of Process Using Visible Sensitive Information

OWASP: DA3 - Sensitive Data Exposure

Category	CWE	OWASP Top 10
Configuration	CWE-214	OWASP-DA3

SUMMARY OF EFFECTIVE CONTROLS

While undertaking the test cases enumerated in the threat model, Praetorian determined that there was minimal risk of an attacker exploiting some of the attack paths due to the presence of one or more effective controls. This section lists particularly notable effective controls along with tags that correlate them to the test cases identified in the threat model.

- **STRONG CRYPTOGRAPHIC PRIMITIVES**
- **MEMORY-SAFE USAGE OF UNSAFE BLOCKS**

Effective Controls

EFFECTIVE CONTROL

Strong Cryptographic Primitives

Related Attack Paths

ATK-1: PitM user compromises and/or weakens the Lightway protocol

ATK-5: Peer user accesses sensitive data from the Lightway components

Control Description

Praetorian observed that the Lightway protocol was built on WolfSSL and supported two modes, UDP and TCP. The Lightway components supported different protocols depending on the mode. The Lightway client supported TLSv1.3 in TCP mode and DTLSv1.3 in UDP mode, while the Lightway server supported TLSv1.3 in TCP mode and DTLS versions 1.2 and 1.3 in UDP mode. The Lighway server supported DTLSv1.2 for backward compatibility. Praetorian also observed that the Lightway components supported two AEAD ciphers, one based on AES-256 and the other based on Chacha20.

Praetorian attempted to perform replay, injection, and tampering attacks to interfere with the Lightway components' behavior as a Machine-in-the-Middle (MitM) user and observed that the strong cryptographic primitives protected the encrypted communication from them. Finally, Praetorian confirmed that the Lightway protocol leveraged the WolfSSL's bitslicing implementation to protect the AES-256 keys against cache-timing attacks of a malicious peer user.

Components Impacted

- Lightway Protocol

Evidence

While dynamically testing the Lightway protocol, Praetorian used a script as a proxy to intercept the Lightway messages. The script was positioned in the middle of communication, playing the client and server roles. While the script forwarded the messages between the components, it also collected them for replay, tampering and injection attacks.

Praetorian observed that the Lightway components running in TCP mode raised errors, sent Alert messages back, and closed the connection whenever Praetorian replayed TLS messages. They raised errors with the "AES-GCM Authentication check fail" messages when using the `AES256` cipher and the "verify mac problem" message when using the `Chacha20` cipher.

Figure 9 shows a replay attack attempt against the Lightway server, whose IP address was 172.16.0.1, and the respective Alert message as a response. Figures 10 and 11 show the raised errors when the Lightway server ran in TCP mode with the `AES256` and `Chacha20` ciphers, respectively.

```
74154.841728… 172.16.0.2  172.16.0.1  TLSv1.3    175 Application Data
74154.842676… 172.16.0.1  172.16.0.2  TLSv1.3    175 Application Data
74154.842683… 172.16.0.2  172.16.0.1  TCP         66 54170 → 27690 [ACK] Seq=544 Ack=1741 Win=64128 Len=0 TSval=3
74161.256925… 172.16.0.2  172.16.0.1  TLSv1.3    175 Application Data
74161.257416… 172.16.0.1  172.16.0.2  TLSv1.3     90 Alert (Level: Fatal, Description: Bad Record MAC)
74161.257433… 172.16.0.2  172.16.0.1  TCP         66 54170 → 27690 [ACK] Seq=653 Ack=1765 Win=64128 Len=0 TSval=3
74161.258121… 172.16.0.1  172.16.0.2  TLSv1.3     89 Application Data
74161.258148… 172.16.0.2  172.16.0.1  TCP         66 54170 → 27690 [ACK] Seq=653 Ack=1788 Win=64128 Len=0 TSval=3
74161.258321… 172.16.0.1  172.16.0.2  TCP         66 27690 → 54170 [FIN, ACK] Seq=1788 Ack=653 Win=64896 Len=0 TS
74161.298677… 172.16.0.2  172.16.0.1  TCP         66 54170 → 27690 [ACK] Seq=653 Ack=1789 Win=64128 Len=0 TSval=3
74162.258692… 172.16.0.2  172.16.0.1  TLSv1.3    175 Application Data
```

Figure 9: The Lightway server responded a replay attack with a Alert message.

```
2024-09-27T19:55:55.730854Z  INFO lightway_server::ip_manager: Free ip=10.125.50.46
2024-09-27T19:55:55.730999Z  INFO lightway_server::io::outside::tcp: Connection closed: Outside data fatal error

Caused by:
    0: WolfSSL Error: Fatal: code: -180, what: AES-GCM Authentication check fail
    1: Fatal: code: -180, what: AES-GCM Authentication check fail
2024-09-27T19:55:55.731201Z  INFO lightway_server::connection_manager: Connection has gone away, stopping
```

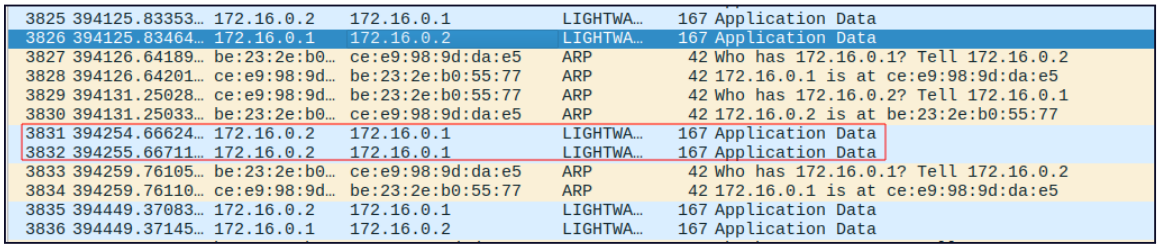
Figure 10: The Lightway server running in TCP mode with the `AES256` cipher raised an error.

```
2024-09-27T19:20:11.555299Z  INFO lightway_server::ip_manager: Free ip=10.125.50.46
2024-09-27T19:20:11.555369Z  INFO lightway_server::io::outside::tcp: Connection closed: Outside data fatal error

Caused by:
    0: WolfSSL Error: Fatal: code: -305, what: verify mac problem
    1: Fatal: code: -305, what: verify mac problem
2024-09-27T19:20:11.555637Z  INFO lightway_server::connection_manager: Connection has gone away, stopping
```

Figure 11: The Lightway server running in TCP mode with the `Chacha20` cipher raised an error.

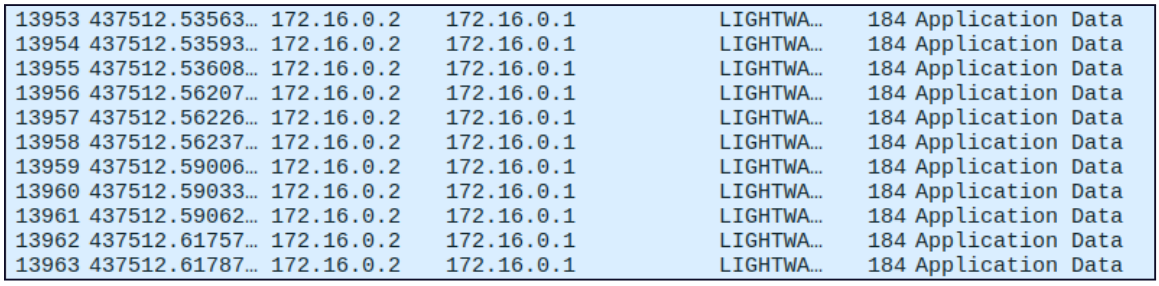
Praetorian observed that the Lightway components running in UDP mode ignored replayed messages. Figure 12 shows two replayed messages to the Lightway server that didn't get any responses.



3825	394125.83353…	172.16.0.2	172.16.0.1	LIGHTWA…	167	Application Data
3826	394125.83464…	172.16.0.1	172.16.0.2	LIGHTWA…	167	Application Data
3827	394126.64189…	be:23:2e:b0…	ce:e9:98:9d:da:e5	ARP	42	Who has 172.16.0.1? Tell 172.16.0.2
3828	394126.64201…	ce:e9:98:9d…	be:23:2e:b0:55:77	ARP	42	172.16.0.1 is at ce:e9:98:9d:da:e5
3829	394131.25028…	ce:e9:98:9d…	be:23:2e:b0:55:77	ARP	42	Who has 172.16.0.2? Tell 172.16.0.1
3830	394131.25033…	be:23:2e:b0…	ce:e9:98:9d:da:e5	ARP	42	172.16.0.2 is at be:23:2e:b0:55:77
3831	394254.66624…	172.16.0.2	172.16.0.1	LIGHTWA…	167	Application Data
3832	394255.66711…	172.16.0.2	172.16.0.1	LIGHTWA…	167	Application Data
3833	394259.76105…	be:23:2e:b0…	ce:e9:98:9d:da:e5	ARP	42	Who has 172.16.0.1? Tell 172.16.0.2
3834	394259.76110…	ce:e9:98:9d…	be:23:2e:b0:55:77	ARP	42	172.16.0.1 is at ce:e9:98:9d:da:e5
3835	394449.37083…	172.16.0.2	172.16.0.1	LIGHTWA…	167	Application Data
3836	394449.37145…	172.16.0.1	172.16.0.2	LIGHTWA…	167	Application Data

Figure 12: The Lightway server running in UDP mode ignored the replayed messages.

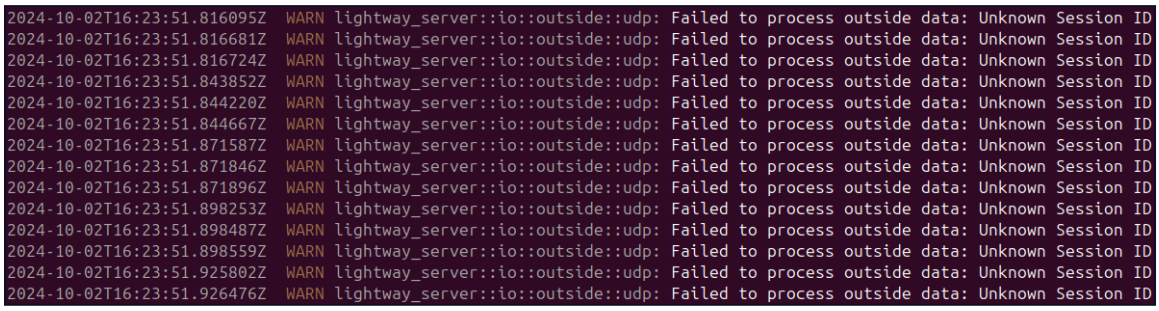
Praetorian observed that the Lightway components behaved similarly with replay, injection, and tampering attack attempts. They raised errors when running in TCP mode and ignored the messages when running in UDP mode. Additionally, Praetorian observed that the component running in UDP mode whose message was ignored kept re-sending it.



13953	437512.53563…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13954	437512.53593…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13955	437512.53608…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13956	437512.56207…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13957	437512.56226…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13958	437512.56237…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13959	437512.59006…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13960	437512.59033…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13961	437512.59062…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13962	437512.61757…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data
13963	437512.61787…	172.16.0.2	172.16.0.1	LIGHTWA…	184	Application Data

Figure 13: The Lightway client running in UDP mode kept sending messages while the server ignored the tampered ones.

Praetorian attempted to inject DTLS messages into an existing session by tampering with the `session_id` of the Lightway protocol header and observed that the Lightway server ignored them and raised warnings with the “Failed to process outside data: Unknown Session ID”.



```
2024-10-02T16:23:51.816095Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.816681Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.816724Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.843852Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.844220Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.844667Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.871587Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.871846Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.871896Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.898253Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.898487Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.898559Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.925802Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
2024-10-02T16:23:51.926476Z  WARN lightway_server::io::outside::udp: Failed to process outside data: Unknown Session ID
```

Figure 14: The Lightway server raised warnings for tampered messages with an existing `session_id`.

Praetorian observed that the Lightway server didn't accept certificates with RSA encryption keys shorter than 2048-bit equivalents. When Praetorian attempted to use certificates with shorter RSA keys, the server raised an error.

```
    },
    lightway_server_ip: 169.254.10.1,
    lightway_client_ip: 169.254.10.2,
    lightway_dns_ip: 169.254.10.5,
    enable_pqc: false,
    enable_tun_iouring: true,
    iouring_entry_count: 16,
    key_update_interval: 900s,
    inside_plugins: 0 plugins,
    outside_plugins: 0 plugins,
    bind_address: 0.0.0.0:27690,
}
2024-10-10T20:23:23.054288Z  INFO lightway_server: Server started with inside ip: 10.125.0.1
2024-10-10T20:23:23.116613Z  INFO lightway_app_utils::iouring: uring main task ring_size=16 nr_tx_rx_slots=8
2024-10-10T20:23:23.116695Z  INFO lightway_app_utils::iouring: Entering i/o uring loop
Error: WolfSSL Error: Fatal: code: 0, what: unknown error number

Caused by:
    Fatal: code: 0, what: unknown error number
```

Figure 15: The Lightway server didn't accept certificates with RSA keys shorted than 2048-bit equivalents.

Finally, Praetorian observed that the Lightway protocol enabled the WolfSSL's bitslicing implementation, added in version 5.6.6, and protected the AES-256 keys the components might use while communicating against malicious peer users performing cache-timing attacks.

```
89              // Disable dynamic library
90              .disable_shared()
91              // Disable sys ca certificate store
92              .disable("sys-ca-certs", None)
93              // Enable AES bitsliced implementation (cache attack safe)
94              .enable("aes-bitsliced", None)
95              // Enable Curve25519
96              .enable("curve25519", None)
```

Figure 16: The Lightway Rust interface for WolfSSL enabled its AES bitsliced implementation.

References

IETF: The Transport Layer Security (TLS) Protocol Version 1.3

IETF: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3

IETF: AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol

IETF: Chacha20 and Poly1305 for IETF Protocol

WolfSSL: TLS 1.3 Protocol Support

Cache-timing attacks on AES

WolfSSL: Release 5.6.6

EFFECTIVE CONTROL

Memory-Safe Usage of Unsafe Blocks

Related Attack Paths

ATK-2: PitM user performs unintended writes to the components' memory

ATK-3: PitM user performs unintended reads from the components' memory

ATK-4: Peer user modifies the behavior of the Lightway components

Control Description

Praetorian observed that the Lightway codebase had a limited number of Rust `unsafe` blocks and ExpressVPN added `SAFETY` comments explaining why they are safe and which invariants were used and must be respected. Praetorian also observed that the `unsafe` blocks contained function calls with internally defined variables in most cases. In other cases, the blocks contained function calls with variables whose values depended on the `iouring_entry_count` command argument but were in bounds.

Consequently, Praetorian could not perform unintended reads from or writes to the components' memory or trigger any memory corruption issues by passing a crafted input to them.

Components Impacted

- Lightway client
- Lightway server

Evidence

While reviewing the source code of the `expressvpn/lightway` repository, Praetorian observed that the Lightway codebase had 28 `unsafe` blocks split between the `lightway-app-utils` and `lightway-server` folders. The `unsafe` blocks in the `lightway-app-utils` folder were part of the code implementing the IO-uring interface for the tunnel and setting socket options, while the ones in the `lightway-server` folder were part of the code implementing the read and write operations of ancillary messages.

Praetorian observed that the `unsafe` blocks in the `lightway-app-utils` folder contained function calls depending only on internally defined variables in most cases.

Figures 17 and 18 show that the Lightway server internally defined all variables used to enable the `IP_PKTINFO` socket option.

```rust
/// Enable IP_PKTINFO sockopt.
pub fn socket_enable_pktinfo(sock: &impl AsRawFd) -> std::io::Result<()> {
    // SAFETY: `setsockopt` requires a valid fd and a valid buffer of `c_int` size
    let res: i32 = unsafe {
        libc::setsockopt(
            socket: sock.as_raw_fd(),
            level: libc::SOL_IP,
            name: libc::IP_PKTINFO,
            value: &1 as *const libc::c_int as *const libc::c_void,
            option_len: std::mem::size_of::<libc::c_int>() as libc::socklen_t,
        )
    };

    if res == -1 {
        Err(std::io::Error::last_os_error())
    } else {
        Ok(())
    }
}
```

Figure 17: The `libc::setsockopt` function depended on the value of `sock`.

```rust
impl UdpServer {
    pub(crate) async fn new(
        conn_manager: Arc<ConnectionManager>,
        bind_address: SocketAddr,
    ) -> Result<UdpServer> {
        let sock: Arc<UdpSocket> = Arc::new(data: tokio::net::UdpSocket::bind(addr: bind_address).
        await?);

        let bind_mode: BindMode = if bind_address.ip().is_unspecified() {
            BindMode::UnspecifiedAddress {
                local_port: bind_address.port(),
            }
        } else {
            BindMode::SpecificAddress {
                local_addr: bind_address,
            }
        };

        let socket: SockRef<'_> = socket2::SockRef::from(&sock);
        socket.set_send_buffer_size(SOCKET_BUFFER_SIZE)?;
        socket.set_recv_buffer_size(SOCKET_BUFFER_SIZE)?;

        if bind_mode.needs_pktinfo() {
            socket_enable_pktinfo(&sock)?;
        }
```

Figure 18: The `sock` variable was set to a bound UDP socket.

Figures 19, 20, 21, and 22 show that the Lightway client internally defined all variables used to get the `IP_MTU_DISCOVER` socket option.

```rust
/// Get IP_MTU_DISCOVER sockopt
pub fn get_ip_mtu_discover(sock: &impl AsRawFd) -> std::io::Result<IpPmtudisc> {
    let mut value: MaybeUninit<libc::c_int> = MaybeUninit::uninit();
    let mut len: u32 = std::mem::size_of::<libc::c_int>() as libc::socklen_t;

    let level: i32;
    let optname: i32;

    #[cfg(target_os = "macos")]
    {
        level = libc::IPPROTO_IP;
        optname = libc::IP_DONTFRAG;
    }

    #[cfg(not(target_os = "macos"))]
    {
        level = libc::SOL_IP;
        optname = libc::IP_MTU_DISCOVER;
    }

    // SAFETY: `getsockopt` requires an fd and a valid buffer of `c_int` size
    let res: i32 = unsafe {
        libc::getsockopt(
            sockfd: sock.as_raw_fd(),
            level,
            optname,
            optval: value.as_mut_ptr().cast(),
            optlen: &mut len,
        )
    };
```

Figure 19: The function `libc::getsockopt` depended on the value of `sock`.

```rust
impl Udp {
    pub async fn new(remote_addr: &str, sock: Option<UdpSocket>) -> Result<Arc<Self>> {
        let sock: UdpSocket = match sock {
            Some(s: UdpSocket) => s,
            None => tokio::net::UdpSocket::bind(addr: "0.0.0.0:0").await?,
        };
        let default_ip_pmtudisc: IpPmtudisc = sockopt::get_ip_mtu_discover(&sock)?;
```

Figure 20: The variable `sock` was set to a bound UDP if it was `None`.

```rust
pub async fn client<A: 'static + Send + EventCallback>(
    mut config: ClientConfig<'_, A>,
) -> Result<ClientResult> {
    println!("Client starting with config:\n{:#?}", &config);

    validate_client_config(&config)?;

    let mut join_set: JoinSet<()> = JoinSet::new();

    let (connection_type: ConnectionType, outside_io: Arc<dyn OutsideIO>): (ConnectionType, Arc<dyn
    io::outside::OutsideIO>) =
        match config.mode {
            ClientConnectionType::Datagram(maybe_sock: Option<UdpSocket>) => {
                let sock: Arc<Udp> = io::outside::Udp::new(remote_addr: &config.server, maybe_sock)
                impl Future<Output = …
                    .await Result<Arc<Udp>, Error>
                    .context("Outside IO UDP")?;
```

Figure 21: The value of the variable `sock` depended on the value of `maybe_sock`.

```
53      let mode: ClientConnectionType = match config.mode {
54          ConnectionType::Tcp => ClientConnectionType::Stream(None),
55          ConnectionType::Udp => ClientConnectionType::Datagram(None),
56      };
57
58      let root_ca_cert: RootCertificate<'_> = RootCertificate::PemFileOrDirectory(&config.ca_cert);
59
60      let mut tun_config: Configuration = TunConfig::default();
61      tun_config.tun_name(config.tun_name);
62      if let Some(inside_mtu: &u16) = &config.inside_mtu {
63          tun_config.mtu(*inside_mtu);
64      }
65
66      let (ctrlc_tx: Sender<()>, ctrlc_rx: Receiver<()>) = tokio::sync::oneshot::channel();
67      let mut ctrlc_tx: Option<Sender<()>> = Some(ctrlc_tx);
68      ctrlc::set_handler(move || {
69          if let Some(Err(err: ())) = ctrlc_tx.take().map(|tx: Sender<()>| tx.send(())) {
70              tracing::warn!("Failed to send Ctrl-C signal: {err:?}");
71          }
72      })?;
73
74      let config: ClientConfig<'_, EventHandler> = ClientConfig {
75          mode,
```

Figure 22: The variable `maybe_sock` was set to None.

In the remaining cases, the `unsafe` blocks in the `lightway-app-utils` folder called functions with variables dependent on the `iouring_entry_count` command argument, but they were safe.

Figures 23 and 24 show two `unsafe` blocks containing function calls with slot indexes, and Figure 25 shows the snippet defining the Rx and Tx slot indexes and ensuring they were always in bounds.

```
// SAFETY: By construction instances of SlotIdx are always in bounds.
#[allow(unsafe_code)]
let RxState {
    sender: maybe_sender: &mut Option<OwnedPermit<BytesMut>>,
    buf: &mut BytesMut,
} = unsafe { rx_state.get_unchecked_mut(index: slot.idx()) };
```

Figure 23: An `unsafe` block called function with a slot index.

```
// SAFETY: By construction instances of SlotIdx are always in bounds.
#[allow(unsafe_code)]
let state: &mut RxState = unsafe { state.get_unchecked_mut(index: slot.idx()) };
```

Figure 24: Another `unsafe` block called function with a slot index.

```
// Using half of total io-uring size for rx and half for tx
let nr_tx_rx_slots: isize = (ring_size / 2) as isize;

let mut rx_slots: Vec<_> = (0..nr_tx_rx_slots).map(SlotIdx::Rx).collect();
let mut rx_state: Vec<_> = rx_slots Vec<SlotIdx>
    .iter() Iter<'_, SlotIdx>
    .map(|_| RxState {
        sender: None,
        buf: BytesMut::with_capacity(mtu),
    }) impl Iterator<Item = RxState>
    .collect();
for state: &mut RxState in rx_state.iter_mut() {
    state.sender = Some(rx_queue.clone().reserve_owned().await?)
}

let mut tx_slots: Vec<_> = (0..nr_tx_rx_slots).map(SlotIdx::Tx).collect();
let mut tx_bufs: Vec<Option<Bytes>> = vec![None; tx_slots.len()];
```

Figure 25: The snippet code defining the Rx and Tx slot indexes.

Praetorian observed that the `unsafe` blocks in the `lightway-server` folder contained function calls that depended only on internally defined variables and added validations to ensure their safe use.

Figures 26 and 27 show two `unsafe` blocks calling functions with internally defined variables, and Figure 28 shows a validation preventing an out-of-bounds write.

```
#[allow(unsafe_code)]
let mut peer_sock_addr: SockAddr = {
    // SAFETY: sockaddr_storage is defined
    // (<https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/sys_socket.h.
    html>)
    // as being a suitable size and alignment for
    // "all supported protocol-specific address
    // structures" in the underlying OS APIs.
    //
    // All zeros is a valid representation,
    // corresponding to the `ss_family` having a
    // value of `AF_UNSPEC`.
    let addr_storage: libc::sockaddr_storage = unsafe { std::mem::zeroed() };
    let len: u32 = std::mem::size_of_val(&addr_storage) as libc::socklen_t;
    // SAFETY: We initialized above as `AF_UNSPEC`
    // so the storage is correct from that
    // angle. The `recvmsg` call will change this
    // which should be ok since `sockaddr_storage`
    // is big enough.
    unsafe { SockAddr::new(addr_storage, len) }
};
```

Figure 26: An `unsafe` block called function with internally defined variables.

```rust
/// # Safety
///
/// `control_len` must have been set to the number of bytes of the
/// buffer which have been initialized.
pub(crate) unsafe fn iter(&self, control_len: usize) -> Iter<N> {
    // Build a `msghdr` so we can use the `CMSG_*` functionality in
    // libc. We will only use the `CMSG_*` macros which only use
    // the `msg_control*` fields.
    let msghdr: msghdr = libc::msghdr {
        msg_name: std::ptr::null_mut(),
        msg_namelen: 0,
        msg_iov: std::ptr::null_mut(),
        msg_iovlen: 0,
        msg_control: self.0.as_ptr() as *mut _,
        msg_controllen: control_len,
        msg_flags: 0,
    };
    // SAFETY: We constructed a sufficiently valid `msghdr` above.
    // `msg_control[..msg_controllen]` are valid initialized bytes
    // per the safety requirements for calling this method.
    let cursor: *mut cmsghdr = unsafe { libc::CMSG_FIRSTHDR(mhdr: &msghdr) };
    Iter {
        msghdr,
        cursor,
        _phantom: std::marker::PhantomData,
    }
}
```

Figure 27: Another `unsafe` block called function with internally defined variables.

```
// SAFETY: `self.cmsghdr` is a valid `cmsghdr` from a prior
// call to `CMSG_FIRSTHDR` or `CMSG_NXTHDR`, see full argument
// above.
let cmsg_data: *mut u8 = unsafe { libc::CMSG_DATA(cmsg: self.cmsghdr) };

// Check that we have sufficient space remaining. `CMSG_DATA`
// does not do this.
let max: usize = self.msghdr.msg_control as usize + self.msghdr.msg_controllen;
let end: usize = cmsg_data as usize + data_size;

if end > max {
    return Err(std::io::Error::other(
        error: "cmsg buffer: insufficient space for data",
    ));
}

let cmsg_data: *mut T = cmsg_data as *mut T;
// SAFETY:
//
// `CMSG_DATA` always returns a valid pointer given a valid
// `cmsghdr`, which we gave it.
//
// We validated there was enough room for a `T` above.
//
// `CMSG_DATA` returns a pointer validly aligned for a
// `cmsghdr`. We asserted above that `T` does not have a
// stricter alignment requirement.
unsafe { cmsg_data.write(val: data) };

// SAFETY: `self.cmsghdr` is a valid `cmsghdr` from a prior
// call to `CMSG_FIRSTHDR` or `CMSG_NXTHDR`. If the result is
// NULL this will be checked on the next call to `fill_next`.
self.cmsghdr = unsafe { libc::CMSG_NXTHDR(mhdr: &self.msghdr, cmsg: self.cmsghdr) };
```

Figure 28: This validation prevented out-of-bounds writes.

Further Work/Caveats

Praetorian validated that ExpressVPN used the `unsafe` blocks safely. However, Praetorian observed that the Lightway components did not properly validate the `iouring_entry_count` command argument. The components' main thread panicked when the argument was set to 0, while the components didn't communicate after connection when it was set to 1 due to the `nr_tx_rx_slots` being 0.

Figure 29: The Lightway client main threat panicked when the variable `iouring_entry_count` was set to 0.

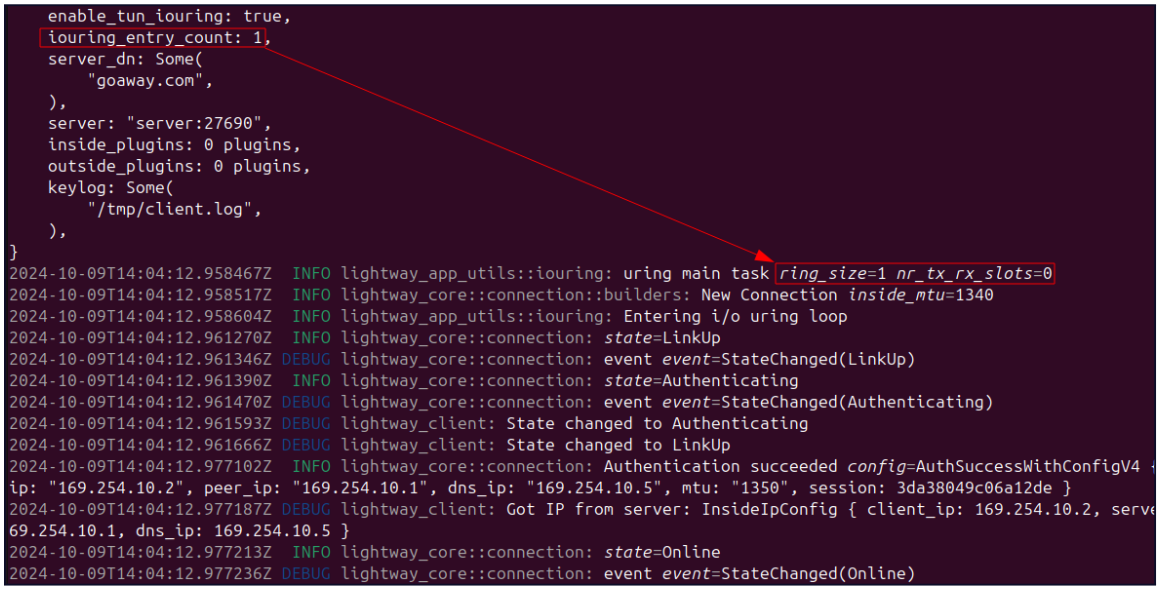


Figure 30: The Lightway client connected to the server but couldn't communicate when the variable `iouring_entry_count` was set to 1.

References

The Rust Reference: Unsafe blocks

Standard library developers Guide: Safety comments policy

Unsafe Rust in the Wild: Notes on the Current State of Unsafe Rust

Appendices

APPENDIX A: ENGAGEMENT SCOPE

The table below lists the components that were in-scope for Statement of Work 2024-08-2601.

Component Name	Identifier	Description
Lightway	`expressvpn/lightway`	ExpressVPN's VPN protocol re-written in Rust.
Rust WolfSSL bindings	`expressvpn/wolfssl-rs`	The Rust interface for the WolfSSL library.

In accordance with the Statement of Work, all other components were explicitly out-of-scope.

APPENDIX B: SEVERITY RATINGS

Praetorian's risk rating scale is based on Common Vulnerability Scoring System (CVSS) 3.1 [1] and Microsoft DREAD [2], two widely used scales for rating security vulnerabilities. However, some aspects of those scales are not ideal for Praetorian's consulting needs. As the CVSS specification notes, rating systems must "accurately [reflect] the risk posed by the vulnerability to the user's environment," and DREAD points out that it "can [be extended] to meet your needs." Therefore, Praetorian has combined and tailored these scales into a new system that meets the needs of Praetorian and our clients as a whole. The Risk Rating Scale 2.0 assesses vulnerabilities and associated risk against five independent risk factors:

Access Vector (Av):
The network location from which the attack originates

Attack Feasibility (Af):
The extent to which the attack has been demonstrated or publicized

Authentication Requirements (Au):
What credentials, if any, are necessary for the attack to succeed

Compromise Impact (Ci):
The extent to which the attacker can control the system in a successful attack

Business Value of Data (Bv):
The type of data put at risk from the vulnerability

Total of Risk Factors	Risk Rating	Recommended Time Frame for Action Plan Development
15	Critical	24 hours
14	High	1 week
13	Medium	1 month
12 ... 10	Low	3 months
9 ... 5	Informational	Addressed at client's discretion

[1] https://www.first.org/cvss/user-guide
[2] https://msdn.microsoft.com/en-us/library/ff648644.aspx

Each risk factor is assessed at one of three levels. If measurement is not possible, the values are estimated according to expert opinions applied to real-world scenarios. The assigned values for each of the five risk factors are then added with equal weight to form the Risk Rating 2.0.

Risk Factor	Value	Definition
Access Vector (Av)	3	**External:** The attack can be executed from anywhere across the Internet.
	2	**Internal/Adjacent:** The attack can take place only from within or adjacent to the target network.
	1	**Local:** The attack requires read/write/execute capabilities on the target system.
Attack Feasibility (Af)	3	**Demonstrated:** The attack has been demonstrated and/or published in proof-of-concept form.
	2	**Not Demonstrated:** The attack has not been carried out, but a demonstration would be possible given particular resources.
	1	**Theoretical:** The attack may be difficult or impossible to demonstrate.
Authentication Requirements (Au)	3	**None:** No prior authentication knowledge is required to execute the attack.
	2	**User:** Credentials from a low-privilege user account are sufficient to execute the attack.
	1	**Privileged:** Credentials from a high-privilege account are required to execute the attack.
Compromise Impact (Ci)	3	**Complete:** There is total information disclosure, full loss of system.
	2	**Partial:** There is a partial compromise of the system's confidentiality, integrity, and/or availability.
	1	**Trivial:** There is at most a trivial impact on the system's confidentiality, integrity, or availability.
Business Value (Bv)	3	**Crucial:** Breach of sensitive customer, business, or financial data.
	2	**System:** Material breach of system or application data.
	1	**Trivial:** No important data is breached from the vulnerability.

APPENDIX C: ENGAGEMENT TEAM

Praetorian Team	Role
Elizabeth Buckley	Account Manager
Forrest Carver	Project Manager
Saullo Branco	Technical Execution

APPENDIX D: CONTACT INFORMATION

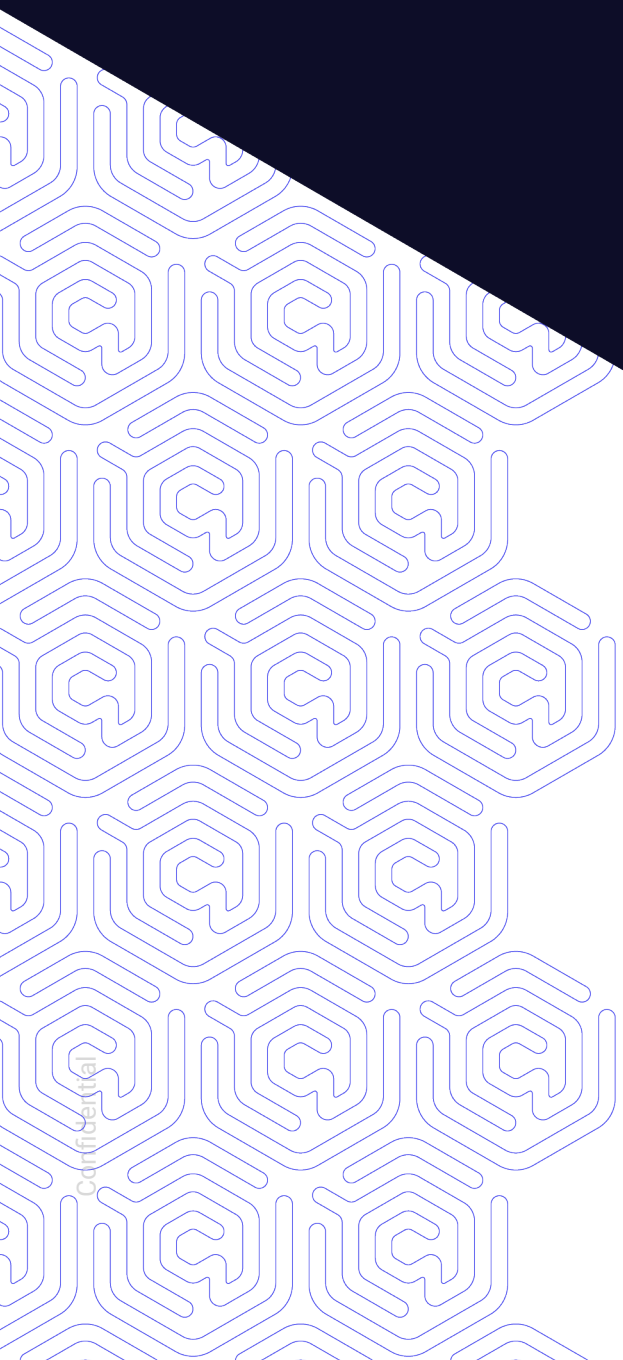
Company:	Praetorian Security, Inc.
Address:	6001 W Parmer Ln Ste 370 PMB 2923 Austin, TX 78727
Contact:	Elizabeth Buckley
Title:	Enterprise Account Manager
Phone:	(504) 722-8634
Fax:	(512) 410-0356
Email:	elizabeth.buckley@praetorian.com

APPENDIX E: ABOUT PRAETORIAN

Praetorian empowers enterprises to thrive in a digital world without compromising cybersecurity. Enterprise security requires the ability to proactively detect vulnerabilities; yet with attack surfaces increasing in complexity, separating signals from noise becomes more difficult.

Our offensive security services pair adversarial experts with our award-winning continuous offensive security platform to detect vulnerabilities across complex digital environments. Whether from supporting ad-hoc projects to managing continuous improvement programs, we maintain radical customer focus as we inform strategic decision making through technical analysis across the full range of enterprise security needs.

With Praetorian as the vanguard, enterprises stay ahead of attackers.



praetorian