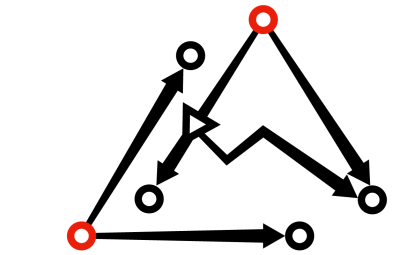

Least Authority

PRIVACY MATTERS

Signing and Permissioning Toolkit

Security Audit Report

Capsule

Combined Updated Final Audit Report: 13 February 2024

Table of Contents

Overview

Background

Capsule has requested that Least Authority perform a security audit of their Capsule Signing and Permissioning Toolkit. Our team completed two reviews on the same components of the Capsule system, as outlined in the Target Code and Revision section. In the second review, our team focused only on the incremental changes implemented after the first review. As a result, this Final Audit Report combines two previously completed Initial Audit Reports, as noted below, to provide a more comprehensive audit of the system, and all details covered in those previous reports were part of the verification review and included in this report.

Project Dates

Initial Review

- **September 4, 2023 - October 4, 2023:** Initial Code Review *(Completed)*
- **October 6, 2023:** Delivery of Initial Audit Report *(Completed)*
- **18 December, 2023 - 22 December, 2023:** Verification Review *(Completed)*
- **22 December, 2023:** Delivery of Final Audit Report *(Completed)*

Second Review

- **18 December, 2023 - 22 December, 2023:** Second Code Review *(Completed)*
- **22 December, 2023:** Delivery of Second Audit Report *(Completed)*
- **3 January, 2024:** Verification Review *(Completed)*
- **3 January, 2024:** Delivery of Combined Final Audit Report - Version 1 *(Completed)*
- **8 January, 2024:** Delivery of Updated Combined Final Audit Report - Version 2 *(Completed)*
- **22 January, 2024:** Delivery of Updated Combined Final Audit Report - Version 3 *(Completed)*
- **13 February, 2024:** Delivery of Updated Combined Final Audit Report - Version 4 *(Completed)*

Review Team

- Mehmet Gönen, Cryptography Researcher and Engineer
- Nikos Iliakis, Security Researcher and Engineer
- Anna Kaplan, Cryptography Researcher and Engineer
- Ann-Christine Kycler, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer
- Florian Sesser, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Capsule Signing and Permissioning Toolkit followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

Infrastructure

- https://github.com/capsule-org/infrastructure/tree/main/modules/ecs

- https://github.com/capsule-org/infrastructure/tree/main/modules/encryption
- https://github.com/capsule-org/infrastructure/tree/main/modules/security
- https://github.com/capsule-org/infrastructure/tree/main/modules/network

Mobile SDK

- https://github.com/capsule-org/react-native-sdk/blob/main/capsule/src/KeyContainer.ts
- https://github.com/capsule-org/react-native-sdk/blob/main/capsule/src/transmissionUtils.ts
- https://github.com/capsule-org/react-native-sdk/blob/main/capsule/src/CapsuleSigner.ts
- https://github.com/capsule-org/react-native-sdk/blob/main/capsule/android/src/main/java/com/capsule/reactnativewallet/CapsuleSignerModule.java
- https://github.com/capsule-org/react-native-sdk/blob/main/capsule/ios/CapsuleSignerModule.m

Web SDK

- https://github.com/capsule-org/web-sdk/blob/main/src/cryptography
- https://github.com/capsule-org/web-sdk/blob/main/src/shares/shareDistribution.ts
- https://github.com/capsule-org/web-sdk/blob/main/src/wallet/keygen.ts

Authentication

- https://github.com/capsule-org/user-management/blob/main/src/app.ts
- https://github.com/capsule-org/user-management/tree/main/src/middleware
- https://github.com/capsule-org/user-management/blob/main/src/services/biometricService.ts

Key Storage

- https://github.com/capsule-org/key-store/blob/main/services/mpcService
- https://github.com/capsule-org/key-store/blob/main/external/mpcNetwork/mpcNetwork.go
- https://github.com/capsule-org/key-store/blob/main/models/capsuleSignerModel.go
- https://github.com/capsule-org/key-store/blob/main/external/aws/kms
- https://github.com/capsule-org/key-store/blob/main/main.go

MPC - Current

- https://github.com/taurusgroup/multi-party-sig
    - Limited to the Kudelski Security Audit Report

MPC <> EOA

- https://github.com/geometryresearch/multi-party-sig
    - Limited to the 3 most recent commits
- https://github.com/geometryresearch/capsule-dkg

Specifically, we examined the Git revisions for our review:

- Infrastructure:
    - Initial Review: `52df9fcfd365baac436d155ebda99c3e7833e8cb`
    - Second Review: `ba66396b6f94e0e5d54b9aacfba45c25cc93cbe1`
- Mobile SDK:
    - Initial Review: `763fa4b6125eca9d085642c3979ea05d16ccd632`
    - Second Review: `614fcc9ccf9e74626e421749257fb38455914a4f`
- Web SDK:
    - Initial Review: `1526445dc4d848c6c09b744e78dde228d7f61cf0`
    - Second Review: `8c022201e3164fb0b40f35822c5c8ccdf5f9e69c`
- Authentication:
    - Initial Review: `6cb54700ba74b903ce571b7caeb533fd90f27843`
    - Second Review: `e8c6fcaf245529b9995fd34c18edc0250f62c76e`
- Key Storage:
    - Initial Review: `a10ef324b0848621485d33b9f11d70110bca308b`

    - Second Review: `4455fde432d74a6eab16c406cb4a8e3bcbf02117`
- MPC <> EOA
    - Initial Review: `c627b0ad65412d37eefc9234824c9321c8fe8596`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Infrastructure: https://github.com/LeastAuthority/capsule-infrastructure
- Mobile SDK: https://github.com/LeastAuthority/react-native-sdk
- Web SDK: https://github.com/LeastAuthority/Capsule-web-sdk
- Authentication: https://github.com/LeastAuthority/user-management
- Key Storage: https://github.com/LeastAuthority/capsule-key-storage
- MPC - Current: https://github.com/LeastAuthority/taurus-group-multi-party-sig
- MPC <> EOA:
    - https://github.com/LeastAuthority/geometryresearch-multi-party-sig
    - https://github.com/LeastAuthority/geometryresearch-capsule-dkg

For the verification, we examined the Git revisions:

- Infrastructure: ba66396b6f94e0e5d54b9aacfba45c25cc93cbe1
- Mobile SDK: 614fcc9ccf9e74626e421749257fb38455914a4f
- Web SDK: 3638ea7de0d719a3161304a36587fcd827b7dfec
- Authentication: e8c6fcaf245529b9995fd34c18edc0250f62c76e
- Key Storage: 0072d52527465e40f68898f74f2ecef8254774b5

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Docs:
  https://docs.usecapsule.com
- DKG with pre-chosen shares:
  https://hackmd.io/Hlfwj7FzReyBtwqfY8tL3A
- Capsule:
  https://www.figma.com/proto/27ltoRejjtLKkY9BeqhepE/%5BExternal%5D-Capsule-Sales-Deck?node-id=1-51&scaling=contain&page-id=0%3A1
- `Kudelski Security Audit of taurusgroup/multi-party-sig library.pdf` *(shared with Least Authority via Slack on 8 September 2023)*

In addition, this audit report references the following documents and links:

- R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts." *IACR Cryptology ePrint Archive*, 2021, [CGG+21]
- J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ECDSA from ECDSA Assumptions: The Multiparty Case." *IACR Cryptology ePrint Archive*, 2019, [DKL+19]
- R. Gennaro and S. Goldfeder, "Fast Multiparty Threshold ECDSA with Fast Trustless Setup." *IACR Cryptology ePrint Archive*, 2019, [GG18]
- R. Gennaro and S. Goldfeder, "One Round Threshold ECDSA with Identifiable Abort." *IACR Cryptology ePrint Archive*, 2020, [GG20]
- Session Management - OWASP Cheat Sheet Series:
  https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#session-id-properties

- HTTP Parameter Pollution Attack: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/04-Testing_for_HTTP_Parameter_Pollution
- HPP package: https://www.npmjs.com/package/hpp
- `Helmet.js`: https://helmetjs.github.io
- Cookie Encryption: https://saturncloud.io/blog/what-encryption-algorithm-is-best-for-encrypting-cookies
- `Gosec`: https://github.com/securego/gosec
- `Nancy`: https://github.com/sonatype-nexus-community/nancy
- elliptic - npm: https://www.npmjs.com/package/elliptic
- Hashicorp - AWS documentation https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc_security_group_ingress_rule

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are correctly passed to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed two security reviews of the Capsule Signing and Permissioning Toolkit. The Capsule is designed as a support system, providing several software development kits (SDKs) as well as infrastructure that can be used by developers and partner projects.

In our first review, our team audited the `web-sdk` and the `react-native-sdk`, which can be used for browser and mobile applications respectively. Additionally, we reviewed the `user-management`, which is a server application providing an API to be used by client applications, and the `key-store`, which provides functionality to perform and manage the Capsule server's side of multi-party-computations.

On request of the Capsule team, we also reviewed the Kudelski Audit of the `taurusgroup/multi-party-sig` library. The Capsule team is planning to switch to the Geometry Research implementation for their multi-party computation, which itself is a fork of the Taurus Group

library and adds minor changes to the Taurus Group library. In our initial review, we focused on the last three commits that were added in the Geometry Research fork. We additionally reviewed some AWS configuration files used for setting up the Capsule backend infrastructure.

In our second review, our team focused on the incremental changes implemented to the aforementioned repositories after the initial review. The most significant changes made by the Capsule team include the addition of code to support the DKL+19 2-out-of-2 threshold signature protocol, as described in [DKL+19], in addition to the CGG+21 threshold signature protocol (which was implemented previously), as described in [CGG+21].

Overall, we identified several Issues and areas of improvement, which relate to both the system design and the coded implementation, as detailed below and outlined in the Issues and Suggestions of this report.

System Design

The Capsule system makes use of multi-party-computation to split externally-owned account (EOA) keys into shares that are held by the Capsule infrastructure and the client applications. Therefore, transaction signatures require performing the multi-party-signature protocol, as described in [CGG+21], in which all parties use the locally held secrets to generate a shared signature.

Handling of Client Secrets

Our team found that in trying to avoid user-memorized secrets like mnemonics or passphrases, the Capsule team uses a shares-based method and tries to avoid requiring locally stored secrets. This, however, does not fully succeed since the encryption of the share and the storage on the network only moves the key management problem to the encryption key, which is currently handled insufficiently. Our team identified Issues with the encryption of the share in which the secret used to derive an encryption key is sent over the network (Issue A).

API

Our team found that an unauthenticated API request that only requires the user's email address would be sufficient to cancel a user's recovery process (Issue B), which attackers could leverage to lock users out of their funds. In addition, several API methods provide privileged functionalities yet only require basic authentication (Issue K).

Overall, our team found that the API methods can be improved by implementing a consistent design and adhering to best practices (Suggestion 4).

Infrastructure

As part of our review, our team audited parts of the network infrastructure configuration files for their AWS instances, provided as infrastructure-as-code in the form of terraform files. These files describe the architecture and properties of the internal network, in addition to providing access control and network segregation specifications. Our team identified several Issues and suggestions regarding network configuration, including the use of insecure communication protocols (Issue E) and the improper configuration of security groups (Issue D).

MPC and DKL+19 Integration

We investigated the way the forked implementation of DKL+19 2-out-of-2 threshold signature protocol, as described in [DKL+19], is used in the rest of the system and did not identify any issues relating to the integration of the protocol.

The protocol code is in a fork of Taurus Group's `multi-party-sig` library, and the code specific to the protocol has been unchanged. However, this fork does make changes to other parts of the library. These changes have been made to address issues in the original version that were identified by Kudelski Security in a previous audit of the Taurus Group's `multi-party-sig` library.

This forked version of the Taurus Group's `multi-party-sig` library is consumed in the Capsule system's `go-sdk`. In that module, we reviewed the `internal/core`, `signer`, and `WebAssembly` (`Wasm`) packages. The code in `internal/core` uses `multi-party-sig`, and the `signer` package uses `internal/core`. The `Wasm` package consumes functions in the `signer` package and provides an API that, by means of WebAssembly, can be used in JavaScript. The `Wasm` package is consumed in the worker code in the `web-sdk` repository. This repository contains duplicate functionality in order to accommodate different JavaScript runtimes. As a result, the `web-sdk` repository contains the two directories `web` and `server`, each containing a worker. We reviewed how these workers call the WebAssembly code, as well as the interaction between the worker and the main runtime. We did not identify issues in any of these directories.

Our team reviewed the entire code path to investigate how DKLS is defined and used in `web-sdk`, `multi-party-sig`, and `go-sdk` and did not identify any issues.

We additionally reviewed how the keys are used in the `key-store` repository. We found that in most instances, the integration of DKL+19 was done correctly. However, in the case of key refresh, we found that the code will attempt to run the CGG+21 protocol, even when the keys in question are DKL+19 keys. Since DKL+S19 does not support key refresh, our team recommends aborting early and returning an error instead (Issue L).

Code Quality

We performed a manual review of the repositories in scope and found the MPC codebase to be generally in adherence with coding best practices. However, we found that the organization could be improved in other areas by implementing more modules and avoiding repetitions between the client applications (`utils.ts`, etc.). Our team also found instances of misleading function names and ambiguous naming conventions. These coding practices inhibit the understanding of the intended functionality of the code and increase the likelihood of implementation errors going unnoticed (Suggestion 1, Suggestion 3).
In addition, the codebase contains unresolved TODOs, which can lead to lack of clarity and cause confusion about the completeness of the implementation. We recommend that all TODOs be resolved and removed from the codebase (Suggestion 5). Moreover, we found that in some locations in web-sdk, errors were insufficiently handled (Suggestion 6).

Tests

Our team found the test coverage of the repositories in scope to be insufficient. Although there are some tests implemented for basic functionality, they only cover a small fraction of the codebase. A robust test suite includes a minimum of unit tests and integration tests that test for success and failure cases to help identify errors and bugs and protect against potential edge cases, which may lead to security-critical vulnerabilities or exploits. We recommend that test coverage be improved (Suggestion 14).

Documentation & Code Comments

The project documentation provided for this security review was insufficient. While some high level documentation is available on the Capsule website, it targets the users of the SDK and does not offer a detailed description of the general architecture of the system, each of the components, and how those components interact with each other.
In addition, there are almost no code comments in the codebase. The current code comments consist mainly of notes exchanged between the developers and do not describe security-critical components and

functions in the codebase. We recommend that the project documentation and code comments be improved (Suggestion 2).

Scope

Our team found the scope of the project to be complex and required an in-depth study of a considerable amount of out-of-scope areas of concern (e.g., `Go-sdk`, `capsule-org/multi-party-sig`, `usermanagement-client`, and the `mpc-network` endpoint) in order to reach a comprehensive understanding of the Capsule Signing and Permissioning Toolkit. We recommend performing a comprehensive security audit that includes all the aforementioned out-of-scope areas to identify the security implications of those areas on the system at large in order to efficiently reason about the overall security of the Capsule Signing and Permissioning Toolkit (Suggestion 7).

Dependencies

Our team examined all the dependencies implemented in the codebase and found that an unmaintained cryptographic library was used to derive an encryption key for the user shares held in the client applications (Issue H). Additionally, we found dependencies that have known vulnerabilities (Issue J).

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Secret Used to Encrypt Shares Sent Over Network	Resolved
Issue B: A Malicious Actor Can Terminate the Recovery Process	Unresolved
Issue C: Non-Uniform Private Key Generation	Resolved
Issue D: Security Groups and Security Group Rules Are Both Used for Traffic Access Control	Unresolved
Issue E: HTTP Traffic to Services Is Allowed	Resolved
Issue F: Sensitive Flag Is Not Set To True for Secret Variables	Resolved
Issue G: Egress Traffic Is Allowed to Any Instance	Resolved
Issue H: Usage of Unmaintained Cryptographic Library	Resolved
Issue I: Missing Domain Separation for Session Identifier	Resolved
Issue J: Update and Replace Vulnerable Dependencies	Resolved
Issue K: Sensitive API Calls Are Not Restricted	Unresolved
Issue L: DKL+19 Does Not Support Key Refresh (Second Review)	Resolved

Suggestion 1: Use Terminology Consistently and Correctly	Planned
Suggestion 2: Improve Project Documentation and Code Comments	Planned
Suggestion 3: Create a Style Guide	Planned
Suggestion 4: Adhere to Node.js Best Practices for Security	Planned
Suggestion 5: Resolve TODOs in Codebase	Planned
Suggestion 6: Improve Error Handling	Planned
Suggestion 7: Perform a Follow-up Audit on Out-Of-Scope Elements	Planned
Suggestion 8: Allow Type 2 Transactions	Resolved
Suggestion 9: Integrate AWS Cloudtrail With Cloudfront for API Usage and Account Activity Monitoring	Unresolved
Suggestion 10: Use Rotational Keys for IAM Users	Planned
Suggestion 11: Implement Access Control Lists for the Network and S3 Buckets (Partially Out of Scope)	Unresolved
Suggestion 12: Reconsider IP Subnetting To Allow For Easier Maintenance	Unresolved
Suggestion 13: Use Firewalls to Improve Perimeter Security	Unresolved
Suggestion 14: Improve Test Coverage	Planned
Suggestion 15: Update Vulnerable Dependencies (Second Review)	Resolved

Issue A: Secret Used to Encrypt Shares Sent Over Network

Location

`src/cryptography/utils.ts#L72-L79`

`src/app.ts#L395`

Synopsis

The Capsule team uses a 16-byte random `userHandle` to derive a keypair, which is then used in the portal code (out of scope) to encrypt shares that are sent to the server. The `userHandle` is stored as a username in the `navigator.credentials` API user agent solution and is used to generate a `PublicKeyCredential`. The response of `navigator.credentials.get` contains the username. This response object is sent to the server to the `biometrics/verify` API endpoint.

Impact

Since the secret that was used to encrypt the shares is sent to the server, which also receives the encrypted share, the server has the opportunity to have the user's part of the secret. Any attacker that was

able to observe the network traffic will know the secret as well. Consequently, if an attacker has the opportunity to obtain the user's share, this could result in the loss of funds.

Preconditions

An attacker would need to have access to the encrypted share and be able to listen in on the network traffic, transmitting the credentials to the server.

Technical Details

By encrypting the share with a key derived from a 16-byte random value called `userHandle`, the `userHandle` becomes a secret that needs to be protected with the same effort as the share itself. The goal of sending the share to the Capsule server is to provide a recovery method for the user. However, in such a situation, the `userHandle` is needed for the recovery. As a result, the key management problem of the share is not solved but shifted to the `userHandle`, thus introducing new attack vectors.

Remediation

We recommend refraining from sending user shares over the network.

Status

The Capsule team has made changes, such that the `userHandle` field is not sent to the server.

Verification

Resolved.

Issue B: A Malicious Actor Can Terminate the Recovery Process

Location

`src/app.ts#L910`

Synopsis

The `users/cancel-recovery` endpoint is used to terminate an active recovery process. The only parameter that is accepted by the endpoint is the email of the user, whose process is to be terminated. The email is publicly known information, so a malicious actor could use this endpoint to terminate a user's recovery.

Impact

This could be used by a malicious user to interrupt processes and spam the network. Moreover, it could even prevent a user from performing a successful recovery and lock them out of their funds.

Preconditions

A malicious actor would need to know the user's email, in addition to knowing that they have started a recovery process.

Mitigation

We recommend implementing an authentication mechanism to verify that a legitimate user is trying to end the process.

Status

The Capsule team has acknowledged the finding but stated that this is an intentional product decision. The team noted that adding authentication at this layer risks a user being locked out of having access to the recovery flow with no recourse, whereas rate-limiting, support intervention, and other measures can be

employed to mitigate an attacker's attempt to perform a Denial of Service (DOS) attack on an active recovery process.

Verification

Unresolved.

Issue C: Non-Uniform Private Key Generation

Location

`capsule/src/transmissionUtils.ts#L18`

`src/transmission/transmissionUtils.ts#L9-L11`

Synopsis

The `uploadKeyshare` function calls `keyFromPrivate` instead of `genKeypair` from the elliptic `npm` package without preventing non-uniform key sampling.

Impact

A non-uniform distribution of private keys will result in some private keys being more likely, thus reducing the security of the cryptographic operations and giving attackers leverage when attacking encrypted secrets.

Technical Details

When a private key is derived in elliptic curve cryptography, it should be a random number smaller than the group order n and larger than zero. In this case, a random 32-byte number is generated, and a private elliptic curve key is then generated from it. This results in a modulo bias where some private key values are more likely than others.
In the case of the elliptic `npm` package, the `genKeyPair` function performs the correct checks before any randomness is chosen as private keys.

Remediation

We recommend switching to a maintained cryptographic library as described in Issue H. When implementing this switch, we recommend adhering to the recommended use of the dependency, thereby avoiding any biases in key sampling.

Status

The Capsule team stated that this key is only used to encrypt and shorten the temporary passkey link for the user and that it is a one-time link, which is not used to manage funds. The Capsule team additionally updated the code to remove the bias by using the `privateToPublic` function from `@ethereumjs/util`.

We examined the changes implemented in the codebase and verified that the code segment at the time of review is used for encrypting links only.

Verification

Resolved.

Issue D: Security Groups and Security Group Rules Are Both Used for Traffic Access Control

Location

`security/main.tf`

Synopsis

Both security groups and security group rules are being used for resource access control purposes.

Impact

Using both solutions in a Virtual Private Cloud (VPC) could lead to undefined behavior. Both of these resources were added before AWS assigned a security group rule unique ID, and they do not work efficiently in all scenarios using the description and tag attributes, which rely on the unique ID. The `aws_vpc_security_group_ingress_rule` resource has been added to address these limitations and should be used for all new security group rules.

Preconditions

If there is conflict between a group and a rule, there is no way to guarantee the order in which they will be enforced.

Feasibility

A misconfiguration is likely to occur. For example, the `allow_ssh` group and the `ecs_cluster_ssh_security_group_rule` group rule have conflicting CIDR blocks specified.

Remediation

We recommend refraining from using both solutions simultaneously and using, instead, `aws_vpc_security_group_ingress_rule` and `aws_vpc_security_group_egress_rule` in a VPC.

Status

The Capsule team stated that while both security groups and security group rules are used, security group rules for egress rules are not associated with security groups that have egress rules defined inline, and security group rules for ingress rules are not associated with security groups that have ingress rules defined inline. The Capsule team is therefore not concerned with overlap issues. As a result, the team does not consider this Issue to be urgent and plans to make updates to use `aws_vpc_security_group_ingress_rule/aws_vpc_security_group_egress_rule` in the future.

Verification

Unresolved.

Issue E: HTTP Traffic to Services Is Allowed

Location

Examples (non-exhaustive):

`security/main.tf#L200`

`security/main.tf#L481`

security/main.tf#L257

security/main.tf#L313

Synopsis

HTTP traffic is allowed through the load balancers responsible for traffic to services such as the MPC computation.

Impact

Allowing HTTP traffic to services leaves them vulnerable to a plethora of attack vectors related to unencrypted traffic, such as MPC protocol messages being intercepted by unintended participants.

Preconditions

The traffic would need to be intercepted by a malicious party.

Remediation

We recommend only allowing HTTPS traffic or redirecting HTTP to HTTPS.

Status

The Capsule team has resolved this Issue by disabling the HTTP traffic.

Verification

Resolved.

Issue F: Sensitive Flag Is Not Set To True for Secret Variables

Location

ecs/variables.tf

Synopsis

Password variables in the `ecs` module do not have the sensitive flag set to `true`.

Impact

The variables will be echoed to the user upon applying terraform after the planning step, which should be avoided for secrets, such as passwords.

Remediation

We recommend setting the sensitive flag to `true` for secret variables to prevent them from being echoed.

Status

The Capsule team has updated the flag as recommended.

Verification

Resolved.

Issue G: Egress Traffic Is Allowed to Any Instance

Location

security/main.tf#L142

Synopsis

Currently, there is a security group to allow all egress traffic to any instance it is attached to.

Impact

Security groups are stateful, so this practice could lead to compromisation of the infrastructure.

Remediation

We recommend refraining from using this group and, instead, fine-graining the egress rules to each instance, as needed.

Status

The Capsule team has removed the security group as recommended.

Verification

Resolved.

Issue H: Usage of Unmaintained Cryptographic Library

Location

`package.json#L27` (e.g. here: `src/transmission/transmissionUtils.ts#L2C24-L2C24`)

`main/capsule/package.json#L111`

`capsule/src/transmissionUtils.ts`

Synopsis

For elliptic curve operations, the elliptic npm library is used. This library is unmaintained, as highlighted in the Github Issues 308 and 251. It was also not audited by an independent third party.

Impact

An unmaintained cryptographic library dependency performing fundamental elliptic curve operations poses a risk in case of attacks on the elliptic curves used.

Remediation

We recommend updating the elliptic curve dependency to a maintained and audited cryptographic library.

Status

The Capsule team has switched to the ECIES hybrid encryption library by Celo.

Verification

Resolved.

Issue I: Missing Domain Separation for Session Identifier

Location

`taurus-group-multi-party-sig`

`internal/core/core.go#L43-L61`

Synopsis

This Issue is based on issues KS-SBCF-F-03 and KS-SBCF-F-07 from the Kudelski Security Audit Report for the `multi-party-sig` library, which the current implementation is also based on. In the `multi-party-sig` library, the session identifier remains constant for different executions of the protocol, and since the issue found by Kudelski Security was not remediated for the library, the application layer needs to set the session identifier to be unique. However, this does not happen in the application layer for Capsule.

Impact

A non-unique session identifier allows for a malicious party to replay messages from other executions of the protocol or carry out replay attacks of the zero-knowledge proofs.

Remediation

We recommend setting the session identifier to be unique for this application in the application layer.

Status

The Capsule team is using the `protocol ID` as a session identifier, which is uniquely generated before the MPC protocol interaction starts.

Verification

Resolved.

Issue J: Update and Replace Vulnerable Dependencies

Location

`go.mod`

`go.sum`

Synopsis

Our team identified issues with the dependencies implemented in the codebase.

Technical Details

We found the following dependency vulnerabilities in the Capsule codebase:

- CVE-2023-3978 Golang HTML: Golang HTML package XSS vulnerability due to improper input validation
- CVE-2023-40591 go-ethereum: Possible research exhaustion on memory

Remediation

We recommend following a process that emphasizes secure dependency usage to avoid introducing vulnerabilities to the Capsule codebase, which includes:

- Manually reviewing and assessing currently used dependencies;
- Upgrading dependencies with known vulnerabilities to patched versions with fixes;
- Replacing unmaintained dependencies with secure and battle-tested alternatives, if possible;
- Pinning dependencies to specific versions, including pinning build-level dependencies in the respective file to a specific version;
- Only upgrading dependencies upon careful internal review for potential backward compatibility issues and vulnerabilities; and

- Incorporating an automated dependency security check into the CI workflow, such as gosec or nancy.

Status
The Capsule team has updated the `go-ethereum` dependency and is no longer using the HTML package.

Verification
Resolved.

Issue K: Sensitive API Calls Are Not Restricted

Location
src/app.ts#L1355

src/app.ts#L1545-L1597

Synopsis
Specific sensitive API methods related to user management are not restricted to the internal network.

Impact
Calls to API methods can be made by unauthorized users.

Remediation
We recommend using IAM tags to restrict the invocation of methods to authorized users.

Status
The Capsule team stated that they currently use basic authentication on all API calls intended to be called from other internal services. Additionally, the team noted that the addition or switch over to using IAM groups instead would not offer any additional protections beyond this and would potentially add complexity or make it more difficult to spot issues. As a result, the team has decided not to implement the recommended remediation.

Verification
Unresolved.

Issue L: DKL+19 Does Not Support Key Refresh (Second Review)

Location
key-storage/blob/main/main.go#L191

key-storage/blob/main/main.go#L199

mpcService/mpcService.go#L158

mpcNetwork/mpcNetwork.go#L146

signer/signer.go#L172

internal/core/core.go#L63

Synopsis

The handler for the API endpoint for key refreshing does not check whether the key to refresh is a CGG+21 key or a DKL+19 key. Instead, it assumes the key is a CGG+21 key and attempts to run the key refresh protocol for that scheme. However, DKL+19 does not support key refreshing.

Impact

This Issue will result in the key refresh failing, potentially affecting keys stored on the server.

Preconditions

The user would need to start a key refresh.

Feasibility

Straightforward.

Remediation

We recommend aborting with an error if the key is a DKL+19 key.

Status

The Capsule team has updated the functions to call an update to the DKL+19 protocol with a key refresh functionality, implemented by Taurus Group in PR68. However, at the time of verification, our team noted that this update had not been documented. We recommend documenting this DKL+19 edit.

Verification

Resolved.

Suggestions

Suggestion 1: Use Terminology Consistently and Correctly

Location

`src/cryptography/utils.ts#L72`

`src/entities/biometricEntity.ts#L70-L79`

Synopsis

The inconsistent use of terminology found in the documentation and codebase may lead to confusion, increased difficulty in understanding how the systems work, and misinterpretations. For example:

- A method that derives an asymmetric key pair from a random value is called `getPublicKeyFromSignature`. A signature in the context of cryptography has a well-defined meaning, which is to use a private key to sign a message that can then be verified using the public key. A public key cannot be derived from a signature. Therefore, this naming is confusing and misleading;
- The terms `share` and `signer` are used interchangeably in the codebase. Being consistent with the name used will reduce confusion, as the name should also ideally and accurately describe the specific role(s) of that function parameter / return value / variable; and

- The term 'biometrics' is used to describe public keys for clients. However, 'biometrics' generally refers to an individual's physical characteristics. This is misleading to developers using the SDK and should be avoided.

Mitigation

We recommend using correct technical terms consistently throughout the documentation and codebase. This applies not only to the specifically listed examples in this suggestion, but needs to be applied across all parts of the system.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to make these changes in the future.

Verification

Planned.

Suggestion 2: Improve Project Documentation and Code Comments

Synopsis

The general documentation provided by the Capsule team was minimal. Robust and comprehensive documentation allows a security team to assess the in-scope components and understand the expected behavior of the system being audited. In addition, clear and concise user documentation provides users with a guide to utilize the SDK correctly and securely, in accordance with security best practices.

Additionally, the codebase lacks explanation in some areas. This reduces the readability of the code and, as a result, makes reasoning about the security of the system more difficult. Comprehensive in-line documentation explaining, for example, expected function behavior and usage, input arguments, variables, and code branches can greatly benefit the readability, maintainability, and auditability of the codebase. This allows both maintainers and reviewers of the codebase to comprehensively understand the intended functionality of the implementation and system design, which increases the likelihood for identifying potential errors that may lead to security vulnerabilities.

Mitigation

We recommend that the Capsule team improve the project's general documentation by creating a high-level description of the system, each of the components, and the interactions between those components. This can include developer documentation and architectural diagrams. We additionally recommend expanding and improving the code comments within the components to facilitate reasoning about the security properties of the system, as well as the secure usage of functions. This is especially important since this implementation is an SDK intended to be used by other projects.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to make these changes in the future.

Verification

Planned.

Suggestion 3: Create a Style Guide

Synopsis

Much of the implementation's source code is inconsistent in its layout and naming conventions, which may potentially confuse users and contributors. Consistent naming is very helpful in conveying structure and intended functionality. Additionally, having conventions in place demonstrating the standard code style and best practices helps improve the readability and maintainability of the code.

Mitigation

We recommend developing a guide for a consistent Capsule style and applying it to existing and future Capsule programs.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to create a style guide in the future.

Verification

Planned.

Suggestion 4: Adhere to Node.js Best Practices for Security

Synopsis

Endpoints depend on input provided by the client (e.g. here). However, currently, there is no proper sanitization of this input. Consequently, the application could be susceptible to the HTTP Parameter Pollution attack, which could make the server fail (DOS attack).

Mitigation

We recommend using the HPP package, which handles this case. Similar attacks can be avoided by using `Helmet.js`, which secures the application by setting HTTP response headers.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to implement this change in the future.

Verification

Planned.

Suggestion 5: Resolve TODOs in Codebase

Location

`src/middleware/authMiddleware.ts#L158`

`src/middleware/corsMiddleware.ts#L38`

`capsule/src/helpers.ts#L35`

Synopsis

Our team identified several instances of unresolved TODOs. Unresolved TODOs decrease code readability and may create confusion about the completeness of the protocol and the intended functionality of each

of the system components. This can hinder the ability for security researchers to identify implementation errors.

Mitigation

We recommend identifying and resolving all pending TODOs in the codebase.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to make these changes in the future.

Verification

Planned.

Suggestion 6: Improve Error Handling

Location

`Capsule-web-sdk/src/CoreCapsule.ts`

`portal/src/AuthLogin.tsx#L64`

Synopsis

In the aforementioned locations, the `touchSession` function returns `Promise<any>`, and after the result is stored in the `res` variable, the data property (`res.data.partherId`) is accessed without any check that verifies if it exists and is an actual property. This could lead to unhandled runtime errors, which could make the application crash. Error handling can be improved in order to provide user-friendly feedback, further aid developers in debugging possible issues, and enhance code maintainability and quality.

Mitigation

We recommend implementing sufficient error handling, such that errors are handled consistently and useful information is provided to help users resolve errors.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to make these changes in the future.

Verification

Planned.

Suggestion 7: Perform a Follow-up Audit on Out-Of-Scope Elements

Synopsis

Several components of the system, such as `capsule-org/multi-party-sig`, `go-sdk`, `usermanagement-client`, and `mpc-network`, which are critical dependencies of the system, were out of scope for this review. Our team had to operate under the assumption that the out-of-scope components of the implementation are safe and function as intended. In order to efficiently reason about the overall security of the system, these areas warrant further investigation to identify the security implications of those areas on the system at large.

Hence, a comprehensive audit of the system by a third party is needed to check for potential

vulnerabilities, ensure that the interaction of the components functions as intended, and to review the overall security of the system.

Mitigation

We strongly recommend performing a comprehensive, follow-up security audit of the changes introduced to `capsule-org/multi-party-sig` (unless replaced by another audited dependency, as was indicated to us by the Capsule team). Once the Issues and suggestions in this report have been adequately addressed, we also recommend auditing `go-sdk`, which consists of the WebAssembly (Wasm) and the `key-store-used` methods that then call `multi-party-sig` and have important functions. Additionally, there are other components that were out of scope for this audit, such as `usermanagement-client` and `mpc-network`, which play a role in the usage of other functionalities and should therefore be considered in future audits as well. We also recommend auditing the monitoring system and the rate-limiting performed.

Furthermore, we recommend having a third party perform penetration tests on duplicated instances of the deployed infrastructure elements to ensure sufficient securing of internal components and prevent against various attacks, including, but not limited to, denial of service attacks.

Status

The Capsule team is currently planning a follow-up audit on the aforementioned components.

Verification

Planned.

Suggestion 8: Allow Type 2 Transactions

Location

`capsule/src/CapsuleSigner.ts#L241`

Synopsis

The code expects `gasPrice` to be part of the transaction object. This would put all the transactions in type 1 (before EIP1559 has taken place) and possibly result in higher gas prices for the users.

Mitigation

We recommend updating the application, such that it also allows type 2 transactions, without the `gasPrice` field present but with `max_prioritity_fee_per_gas` and `max_fee_per_gas`.

Status

The Capsule team has stated that the code referenced is only relevant to the old logic and is not being used anymore.

Verification

Resolved.

Suggestion 9: Integrate AWS Cloudtrail With Cloudfront for API Usage and Account Activity Monitoring

Synopsis

Clients are expected to interact with the AWS Virtual Private Cloud (VPC) via API calls, but these calls are not monitored.

Mitigation

We recommend integrating Cloudtrail with the existing Cloudfront implementation to monitor account activity, API usage, as well as resources.

Status

This suggestion remains unresolved at the time the verification review was performed.

Verification

Unresolved.

Suggestion 10: Use Rotational Keys for IAM Users

Location

`security/main.tf#L734`

Synopsis

The KMS signer key provided is immutable.

Mitigation

We recommend using rotational keys to improve the security of the estate.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to make these changes in the future.

Verification

Planned.

Suggestion 11: Implement Access Control Lists for the Network and S3 Buckets (Partially Out of Scope)

Location

`s3/main.tf#L117`

`s3/main.tf#L48`

`s3/main.tf#L266`

`security/main.tf`

Synopsis

Network access is currently restricted via the use of security groups. While security groups are used as a solution implemented at the client level, access control lists sit on the network level and are thus stateless. The S3 buckets access is currently unrestricted.

Mitigation

We recommend implementing access control lists additionally to the security groups to restrict network access. The same solution should be applied to restrict resource access on the S3 buckets.

Status

This suggestion remains unresolved at the time the verification review was performed.

Verification

Unresolved.

Suggestion 12: Reconsider IP Subnetting To Allow For Easier Maintenance

Location

`network/main.tf#L51-L152`

Synopsis

The subnet ranges specified are sequential, mixing private, isolated, and public subnets. While there are subnets reserved for later use, maintenance may prove to be difficult with further expansion.

Remediation

We recommend reconsidering the currently implemented IP subnetting ranges and utilizing, instead, the full range of the VPC for easier maintenance. For example, the range `10.0.x.x.` could be reserved for public networks and `10.1.x.x.` could be reserved for private/isolated networks. The last two bytes could be used to specify resources, such as using one range for servers and another for load balancers. Such a notation would allow for easier management.

Status

This suggestion remains unresolved at the time the verification review was performed.

Verification

Unresolved.

Suggestion 13: Use Firewalls to Improve Perimeter Security

Location

`security/main.tf`

Synopsis

Currently, security groups are used to restrict traffic. However, security groups are not sufficient solutions when it comes to perimeter security.

Remediation

We recommend strengthening the security of the VPC by installing network firewalls on the perimeter. While security groups sit on top of the instances, and access control lists restrict access to resources, the VPC itself should have an extra layer of protection with a network firewall, and the load balancers should

be after a web application firewall to restrict malicious traffic to the load balancers and only allow it when it comes from legitimate sources.

Status

This suggestion remains unresolved at the time the verification review was performed.

Verification

Unresolved.

Suggestion 14: Improve Test Coverage

Location

`capsule/src/test`

`tree/main/src`

`main/tests/unit`

Synopsis

There is insufficient test coverage implemented to test the correctness of the implementation and that the system behaves as expected. Sufficient test coverage should include tests for success and failure cases (all possible branches), which helps identify potential edge cases, and protect against errors and bugs that may lead to vulnerabilities. A test suite that includes sufficient coverage of unit tests and integration tests adheres to development best practices. In addition, end-to-end testing is also recommended to assess if the implementation behaves as intended.

Mitigation

We recommend that comprehensive unit test coverage be implemented in order to identify any implementation errors and to verify that the implementation behaves as expected.

Status

The Capsule team has acknowledged this suggestion and stated that they plan to make these changes in the future.

Verification

Planned.

Suggestion 15: Update Vulnerable Dependencies (Second Review)

Location

`Capsule-web-sdk`

`react-native-sdk`

Synopsis

Many vulnerabilities were reported (critical, high, and moderate) in the dependencies of the packages. Vulnerabilities in dependencies can lead to significant issues and should be addressed to avoid undesirable outcomes.

Remediation

We recommend resolving all critical and high vulnerabilities before going to production.

Status

The Capsule team has updated the dependencies, as recommended.

One vulnerable dependency (`nth-check`) remains; however, it is not used in production (since it is a development dependency) and, thus, does not affect the application.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.