Least Authority
PRIVACY MATTERS

SMPC Protocol (Second Review)

Security Audit Report

Worldcoin

Final Audit Report: 22 November 2024

Table of Contents

Overview

Background

Tools for Humanity Corporation has requested that Least Authority perform a second security audit of their SMPC Protocol.

Project Dates

- **July 31, 2024 - September 10, 2024:** Initial Code Review *(Completed)*
- **September 12, 2024**: Delivery of Initial Audit Report *(Completed)*
- **November 18, 2024 - November 21, 2024:** Verification Review *(Completed)*
- **November 22, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- George Gkitsas, Security / Cryptography Researcher and Engineer
- Sven M. Hallberg, Security Researcher and Engineer
- Anna Kaplan, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and a second review of the SMPC Protocol 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:

- SMPC Protocol Repository:
  https://github.com/worldcoin/gpu-iris-mpc

Specifically, we examined the Git revision for our initial review:

- `2ab7ae6f5d14771a632aede800aa32b20b35070c`

For the verification, we examined the Git revision:

- `13afd83b7aa5fd2214f68ea4b898ae12aca5eb99`

For the review, this repository was cloned for use during the audit and for reference in this report:

- SMPC Protocol Repository:
  https://github.com/LeastAuthority/worldcoin-gpu-iris-mpc-2nd-review

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:

- Website:
  https://worldcoin.org
- Large-Scale MPC: Scaling Private Iris Code Uniqueness Checks to Millions of Users:
  [BKS+24]

In addition, this audit report references the following documents:

- P. Mohassel and P. Rindal, "ABY^3: A Mixed Protocol Framework for Machine Learning." *IACR Cryptology ePrint Archive*, 2018, [MR18]
- Previous Audit Report Delivered by Least Authority:
  https://leastauthority.com/blog/audits/audit-of-worldcoins-mpc-protocol
- CUDA Driver API :: CUDA Toolkit Documentation:
  https://docs.nvidia.com/cuda/cuda-driver-api/index.html
- cudarc - Rust:
  https://docs.rs/cudarc/latest/cudarc
- NVIDIA Collective Communication Library (NCCL) Documentation:
  https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html
- RFC 5869 - HMAC-based Extract-and-Expand Key Derivation Function (HKDF):
  https://datatracker.ietf.org/doc/html/rfc5869
- RFC 7539 - ChaCha20 and Poly1305 for IETF Protocols:
  https://datatracker.ietf.org/doc/html/rfc7539
- AWS Secrets Manager:
  https://aws.amazon.com/secrets-manager
- Sealed boxes:
  https://libsodium.gitbook.io/doc/public-key_cryptography/sealed_boxes

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 passed correctly 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 a security audit of Worldcoin's secure multi-party computation protocol (SMPC Protocol), which is used to match a given iris against a database of iris shares. Our team previously reviewed this implementation and delivered a Final Audit Report on April 4, 2024. In this review, we examined the second version of the implementation, which increases the nodes from two to three and uses a different sharing scheme, such that the iris masks are protected and not assumed to be known in plain as in the previous version. The second version also improves performance and storage requirements and utilizes GPU clusters through CUDA, featuring direct encrypted communication to speed up performance. Finally, this version provides a secure upgrade mechanism from the former version of the protocol to the current one.

The computation is split among three nodes, each maintaining a database with one share per registered iris. The protocol is set in a two-out-of-three majority setting in the semi-honest security model. Separating the computation into independent nodes improves the privacy of iris data. As a result, neither the nodes nor a central authority has access to the full iris code.

Our team additionally examined the correspondence between the protocols as well as the underlying algorithms, such as the lifting and comparison algorithms described in the papers ([BKS+24] and [MR18]), and the implementation. We assessed whether the algorithms were implemented correctly and did not find any issues. Additionally, we did not find any common implementation bugs.

We also examined the use of randomness and its implementation within the project. Each MPC node derives two seeds for each peer, which are used to generate shared secrets through a Diffie-Hellman key exchange. The resulting shared secrets are used to instantiate RNGs, for example, for masking results and GPU encryption. We could not identify severe issues with this approach but recommend updating the cryptographic wording (Suggestion 2) and strengthening the HKDF salt (Suggestion 1). We also examined the correctness of the ChaCha implementation and suggest correcting the ChaCha counter calculation (Suggestion 5).

We found that some parts of the implementation are used as helpers for testing (such as Shamir sharing and the populating of databases with dummy data). Our team noted that these components were not assessed during this review.

To better understand the server logic flow, we assessed the logic of state synchronization between GPU devices, the databases, and between the MPC nodes. We did not find any issues. We also assessed the query processing flow and did not find any issues.

We additionally examined the upgrade of the protocol from two nodes to three nodes, which was added to the project through PR#15. We could not identify any issues with this approach. We recommend that the internals of the upgrade protocol be documented (Suggestion 2) and that the communication for the upgrade run over an encrypted and authenticated channel during the deployment for the upgrade, as was initially proposed by the Worldcoin team (Suggestion 7).

System Design

Our team reviewed the system's design and implementation and found that the Worldcoin team approached this project holistically and that security has been taken into consideration, as demonstrated by the use of Rust, advanced cryptography, and encryption of communication.

The implemented MPC protocol adheres to the [BKS+24] paper but includes some modifications. However, there is no exact specification of the protocol.

Additionally, we found that the MPC nodes have adequate security in their inter-communication. GPUs are directly communicating via the NCCL point-to-point communication. The Worldcoin team has added communication encryption, which is enabled by the Rust feature flag `otp_encypt`. Our team also noted that the `otp_encrypt` feature is not enabled by default and recommends doing so in order to have the most secure configuration (Suggestion 2).

In our review of the communication security, our team found that two communications take place. The first one is between the MPC nodes, and the other is between the clients and the MPC nodes. We analyzed the communication between the GPUs (via NCCL) and the encryption scheme. Specifically, we verified whether the relevant keys are properly managed and if RNG seeds are generated according to best practices. We identified one area of improvement (Suggestion 1). Note that the communication between the clients and the MPC nodes was not fully in scope for this review. This communication is encrypted using public key cryptography. However, only the key management required for this communication was included in the scope of this audit. Although we did not identify any issues, our team noted that there is room for improvement of the node key management (Suggestion 3).

Our team noted that the choice of a semi-honest MPC operating in a two-out-three majority setting poses some risks. The semi-honest model assumes that all nodes will execute the protocol faithfully, and assures that no information can be leaked about the secret data. This covers cases where the node operators are fully trusted to behave faithfully. In reality, malicious behavior does exist and is common. Even when an operator is trusted or legally restricted, there is still a chance that a node will turn malicious due to an external compromise, at a later point. Given that one malicious node is enough to compromise the entire protocol, this poses a realistic single point of failure. It effectively shifts the problem of the security of the MPC protocol to the security controls employed by the weakest server. Our team identified two options to counter the above. The first is to use a protocol secure under the malicious model, which ensures protocol correctness and privacy protection even with malicious nodes participating. The second is to enforce the assumption of the semi-honest setting. Correct behavior could be enforced at the protocol level (e.g., including proofs of correct execution) or through other technical means (e.g., TEE - Trusted Execution Environment). During our review, the Worldcoin team reported their consideration of the enforcement through TEE. However, our team concluded that the protocol does not enforce the correct behavior of nodes. Additionally, since infrastructure and deployment were out of the scope of this audit, we did not verify the use of any other technical measures.

From our discussions with the Worldcoin team, we understand that two nodes are operated by Worldcoin and Tools for Humanity. The third node has not been defined yet but is intended to be an independent and legally bound operator. Given that the protocol requires two-out-of-three nodes and that two operators are closely connected entities, our team recommends increasing node independence by decreasing the number of related nodes or including more independent parties to the multi-party computation.

Additionally, the system relies on a single point of trust due to the choice of deploying on only one cloud provider (AWS). Diversifying the platforms on which the nodes are deployed will benefit node independence and make the attack surface less uniform.

Our team therefore recommends thoroughly documenting the threat model and its assumptions and providing a mapping between these assumptions and the controls that are in place for their enforcement (Suggestion 2).

Code Quality

We found the code to be well-organized and of high quality, in that it adheres closely to development best practices. However, we also found that it has some unused parts and obsolete functionality, which we recommend be removed to make it cleaner. The codebase would also benefit from further refactoring and abstractions to remove code repetition and improve variable naming (Suggestion 4).

Tests

The repositories in scope include tests that cover the basic functionality of the protocol.

Documentation and Code Comments

The project documentation provided by the Worldcoin team was generally sufficient in describing the intended functionality of the system. However, our team found that the security-related documentation could be improved (Suggestion 2). Additionally, while the codebase includes some code comments, the number of comments could be increased to assist with code readability and comprehension.

Scope

The scope of this review was sufficient, as our team reviewed the MPC node implementation, which includes the protocol implementation, state synchronization, and inter-node communication. Additionally, our team reviewed the upgrading process of the iris shares from the older protocol to the current one. Note that our team did not review the infrastructure and deployment parts of the project nor the communication between the client and the server.

Additionally, although the deployment and infrastructure details were out of scope for this audit, we noted some concerns. First, the MPC nodes are hosted by the same hosting service (AWS). Second, the main paper this work is based on indicates that to reach practical performance levels, the MPC nodes will have to be colocated. Such deployment decisions can hinder the independence of nodes. We therefore recommend diversifying the supported cloud providers and conducting a follow-up security audit focusing on the infrastructure and deployment configuration.

Dependencies

We examined all the dependencies utilized in the codebase and found two vulnerabilities. We recommend improving dependency management (Issue A).

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: Usage of Vulnerable Dependencies	Partially Resolved
Suggestion 1: Strengthen HKDF Salt	Resolved
Suggestion 2: Improve Documentation on Security	Partially Resolved
Suggestion 3: Improve Seal Box Key Management	Partially Resolved
Suggestion 4: Improve Code Quality	Partially Resolved
Suggestion 5: Correct ChaCha Counter Calculation	Resolved
Suggestion 6: Respect Maximum Number of CUDA Blocks	Resolved
Suggestion 7: Review Communication for Upgrade Protocol	Resolved

Issue A: Usage of Vulnerable Dependencies

Location

`worldcoin-gpu-iris-mpc-2nd-review/Cargo.lock`

`worldcoin-gpu-iris-mpc-2nd-review/Cargo.toml`

Synopsis

Analyzing the project's dependencies using Rust's `cargo audit` revealed three issues, two of which were flagged as vulnerabilities. While the extent of their exploitability remains unclear, it is good practice to keep dependencies up to date in order to avoid importing vulnerable code.

Feasibility

Although we could not find a direct way to exploit these issues, using vulnerable dependencies can open an attack vector.

Technical Details

The following dependency vulnerabilities were identified:

- RUSTSEC-2023-0071: potential key recovery through timing side channels independency `rsa`.
- RUSTSEC-2024-0363: Binary Protocol Misinterpretation caused by Truncating or Overflowing Casts in dependency `sqlx`

The following dependency warning was identified:

- RUSTSEC-2024-0320: `yaml-rust` is unmaintained.

Mitigation

We recommend updating or replacing the dependencies `rsa`, `sqlx`, and `yaml-rust` in order to eliminate the aforementioned vulnerabilities and warnings.

Remediation

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

- Manually reviewing and assessing currently used dependencies;
- Upgrading dependencies with known vulnerabilities to patched versions with fixes;
- Replacing unmaintained dependencies with more secure alternatives, if possible (more specifically, alternatives that are used, tested, and audited and that work as intended);
- 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 `cargo_audit`.

Status

The status of dependencies has stayed the same. The dependency `sqlx` has been fixed to version 0.8.2 which resolves this issue in PR #493. The dependency `yaml-rust` was updated to `yaml-rust`2.
While running cargo audit again, we identified more unmaintained dependencies: `derivative` and

`instant`, and one deprecated dependency: `sodiumoxide`. We have seen the conversations by the Worldcoin team to switch eventually to other options here.

Verification

Partially Resolved.

Suggestions

Suggestion 1: Strengthen HKDF Salt

Location

`src/server/actor.rs#L169`

`src/server/actor.rs#L753`

Synopsis

The usage of salt can make a significant contribution to the security of the output keying material in HKDF. The best option is to generate a random or pseudorandom salt value with its length equal to the digest length of the hash algorithm used.

Currently, the salt value is a fixed value. Its length is 160-bits, which is less than the ideal 256-bits that is the digest length of SHA256.

Mitigation

We recommend increasing the salt size to 256-bits to match the SHA256 digest length. We also recommend using a random or pseudorandom salt value.

Status

The Worldcoin team resolved this issue in PR #481.

Verification

Resolved.

Suggestion 2: Improve Documentation on Security

Location

`worldcoin-gpu-iris-mpc-2nd-review/README.md`

Synopsis

Some security and protocol design choices are not mentioned or elaborated on further in the project's documentation, such as:

- Deviations from the paper and their justification (e.g., `OrTree` is not used);
- Description of the internals of the upgrade protocol, including the communication channels and diagram, motivation, and threat model;
- Description of the threat model and all its security assumptions; and
    - The list of the employed security controls and the assurances they provide (such as TEE, key management services, etc).
    - The correspondence between assumptions and security controls;

- Documentation on the security feature `otp_encrypt`.

Our team additionally noted that there were some outdated architectural diagrams in the `README` file, which the Worldcoin team had also initially acknowledged.

Mitigation

We recommend documenting security-related decisions and updating the diagrams.

Status

The Worldcoin team has removed the `otp_encrypt` flag (here) and is planning to update the documentation.

Verification

Partially Resolved.

Suggestion 3: Improve Seal Box Key Management

Location

`src/bin/key_manager.rs#L74`

`src/bin/key_manager.rs#L230-L231`

Synopsis

`libsodium` seal boxes are used to anonymously communicate iris shares from the client to each MPC node. Each node has its own public key pair, with the public part being stored in a publicly accessible AWS storage (S3 bucket) and the private part being stored in the AWS Secrets Manager. The key management of these key pairs introduces unnecessary biasing and leakage risks.

Firstly, the storage of the private keys is not adequate since it can be read by anyone with access to the node's server just by requesting it from Secrets Manager.

Secondly, the implemented key rotation mechanism for these key pairs introduces unnecessary biasing and leakage risks. Specifically, `key manager`, which is a custom utility that is used for key rotation, allows its user to specify the rotation seed through the command line. Providing the seed through the command line leads to two issues. First, the seed value is not guaranteed to be random, as the key manager user can choose any value, which can lead to biases (intended or otherwise). Second, the seed value could be logged into the command line history. In that case, access to this history fully leaks the key pair.

In addition to the above, generating keys in host memory and host hardware increases leakage risks through side-channel attacks.

The assets that these keys protect are the communicated iris shares. Given that anyone with enough access to read a node's private key will also have enough access to view the node's shares, our team did not identify an immediate risk. Nevertheless, proactively strengthening key management is recommended.

Mitigation

We recommend utilizing a key management solution that keeps private keys securely stored to prevent the possibility of a malicious attacker gaining access to them. The solution should include periodic automatic key rotation.

Status

The Worldcoin team updated the code and removed the option to pass a seed (PR #369) but has not resolved the storage of private keys.

Verification

Partially Resolved.

Suggestion 4: Improve Code Quality

Location

In various locations throughout the codebase.

Synopsis

During our extensive review of the codebase, our team identified practices that impact the quality, readability, and maintainability of the codebase. To illustrate, the following is a non-exhaustive list of examples:

1. Some constants are not centrally managed:
    a. The iris code size is defined twice, here and here.
    b. The constant K is defined here and here.

2. Variables do not have informative names, which makes it difficult to understand their purpose:
    a. These buffers are used throughout the codebase, serving as both inputs and outputs of different functions.
    b. Throughout the Rust code, different CUDA functions are called via variables that are simply named `kernel` or `kernels`.

3. Significant parts of the code are duplicated with very small changes:
    a. The file `src/rng/chacha.cu` contains two functions, `chacha12` and `chacha12_xor` that differ in only a single line at the end; an assignment or XOR, respectively.
    b. The files `src/rng/chacha.rs` and `src/rng/chacha_corr.rs` share a significant amount of code. The latter is essentially a variant or extension of the former.

4. Some cryptographic terms are used in ways that are unclear or misleading:
    a. The code refers to "one-time pad" (OTP) encryption. However, ChaCha, as used for this purpose, is most accurately described simply as a *stream cipher*.
    b. The term "seed" is used in many places where more descriptive names could be used: *shared secrets* derived through Diffie-Hellman key exchange, and *keys* used with the ChaCha cipher.
    c. The pseudo-random sequence generated by ChaCha is commonly referred to as its (resulting) *keystream*.

Mitigation

We recommend improving code quality by addressing the items listed above.

Status

The Worldcoin team did a general code clean-up and removed duplicated code in PR #394. The references to "otp" and "seed" were kept (4.a. and 4.b.).

Verification

Partially Resolved.

Suggestion 5: Correct ChaCha Counter Calculation

Location

`src/rng/chacha.cu#L140`

`src/rng/chacha.cu#L219`

Synopsis

The CUDA implementation of the ChaCha stream cipher has been adapted from third-party open source code. One adaptation was the extension of the counter field to 64-bits (two 32-bit state elements) at the expense of 32-bits of nonce. This adaptation is valid with respect to the specification of the algorithm.

However, the counter value is used in two places, but the adaptation was only applied to the first occurance. Concretely, the variable `idx` is combined with only a single element, `state[12]`, when the original state matrix is added to the intermediate state at the end of the algorithm. This results in each output block of the cipher differing slightly from what it should be according to the specification.

Mitigation

We recommend correcting the algorithm to comply with the specification.

Status

The Worldcoin team resolved this suggestion in PR #367. While investigating this issue, the Wordcoin team identified another issue where the wrong order of casting data types in CUDA code led to the counter always being 0. This was fixed, too.

Verification

Resolved.

Suggestion 6: Respect Maximum Number of CUDA Blocks

Location

Examples (non-exhaustive):

`src/dot/share_db.rs#L646`

Synopsis

Several CUDA functions (e.g., `single_xor_assign_u8`) are written such that each CUDA thread processes exactly one input element. Thus, it is implicitly assumed that there can never be more input elements than the number of available CUDA threads. The maximum number of CUDA threads is considerably large but not infinite. In particular, it depends on the maximum number of thread blocks which is $2^{31}-1$. In the case of `single_xor_assign_u8`, each input element is a single byte, and applying this function to an input array of 512 GB ($256 \cdot 2^{31}$ bytes, with 256 threads per block) is not unthinkable.

Mitigation

We recommend using loops with all CUDA functions that process variable-length input to make them independent of the maximum number of blocks supported by CUDA.

Status
The Worldcoin team resolved this suggestion in PR #394.

Verification
Resolved.

Suggestion 7: Review Communication for Upgrade Protocol

Location
Deployment of upgrade protocol.

Synopsis
In order to upgrade from the former version to this version, the Worldcoin team designed an upgrade protocol, which was added to the project in PR#15. The upgrade protocol does not have any custom cryptographic measures to ensure privacy within the communication of parties or authentication of parties. The Worldcoin team noted that the privacy and authentication will be ensured during the deployment of the protocol. (Note that the deployment of the upgrade protocol was out-of-scope for this review).

Mitigation
We encourage the Worldcoin team to use a secure, private, and authenticated channel for communication during the upgrade.

Status
The Wordcoin team updated the upgrade clients to use a TLS stream client and the containers where the clients are running are using a private CA certificate (PRs #453 and #351).

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.