Apache Milagro MPC Cryptographic Assessment

Qredo

July 16, 2020 – Version 1.3

Prepared for

Samuele Andreoli
Kealan McCusker
Brian Spector

Prepared by

Mason Hemmel
Aleksandar Kircanski

Executive Summary

Synopsis

During the spring of 2020, Qredo engaged NCC Group to conduct a security assessment of the Apache Milagro MPC library. This library implements the primitives necessary to instantiate the multi-party ECDSA signature scheme provided in Gennaro and Goldfeder's "Fast Multiparty Threshold ECDSA with Fast Trustless Setup"[1] (GG). This assessment occurred over the course of two calendar weeks and was delivered by three consultants over ten person-days. NCC Group assessed commit c5f0733 of the public `incubator-milagro-MPC` repository.

Scope

NCC Group evaluated only the code in the `incubator-milagro-MPC` repository, and did not stray into the original cryptographic libraries used in this project. As a result, the engagement team did not review the implementation of many of the low-level primitives used in this project. This includes the following:

- Hash function implementations
- Elliptic curve group and finite field operations (other than special-case modular multiplication and modular reductions)
- Bit-and-byte-level operations
- Bignum operations
- Acquisition and manipulation of randomness
- Paillier encryption implementation

The engagement proceeded with the assumption that all of the above functionality is implemented correctly and securely. For example, the team assumed that all serialization into the struct for an elliptic curve point performs full validation of that point, and that all randomness is sourced from a cryptographically secure pseudo-random number generator.

Key Findings

The assessment uncovered a set of common cryptographic flaws. The most notable involved the replayability of some of the zero-knowledge proofs generated in the protocol.

NCC Group found that a lack of liveness guarantors in zero-knowledge proofs render them replayable. This means an attacker could potentially "forge" proofs in a given round of multi-party signing by replaying proofs generated in an earlier round. The implemented protocol can reveal sensitive information in the case that malicious participants are involved in a full run of the protocol, meaning that this can have a practical impact.

These issues have been fixed prior to the publication of this report.

Strategic Recommendations

- **Encapsulate Higher-Level Primitives into Functions** While Milagro MPC offers all the functionality needed to spin up an implementation of the GG scheme, many of the larger steps are not implemented as single functions. This leaves users of the library to implement these larger steps themselves, which has historically resulted in insecure implementations. To avoid this, the library should at least implement all phases of key and signature generation as single functions.
- **Consider Vetting Novel Sections of `milagro-crypto`** The `milagro-crypto` library supplies the low-level cryptographic implementations of the core primitives listed in the Scope section. While the library's authors note it has largely been imported from a well-used and well-vetted cryptographic library, there is still a certain amount of "glue" code and some novel functionality. Implementing low-level cryptographic primitives, or even code used to stitch it together, tends to be extremely challenging. To ensure the highest level of assurance, the Milagro team should consider highlighting and further vetting any novel portions of this library.
- **Separate Functions Requiring Random Generation** Many of the Milagro MPC functions include a pattern in which an RNG struct is optionally passed into a function that alters its functionality depending on whether or not it is passed. This leads to bad functionality in cases where the RNG is meant to be passed in and has somehow acquired the `NULL` value, or vice versa. As a result, these functions should be separated into two.
- **Use Naming Conventions Consistently** The Milagro MPC library tends to use GG's naming conventions for its variables. However, this is not applied across the board, leading to potential confusion for users attempting to cross-reference functionality.
- **Warn Users About Repeated Schnorr Proofs with Unchanged Randomness** The functions used to generate non-interactive zero knowledge (NIZK) challenges support user-input randomness. If a Schnorr NIZK proof is repeated for the same randomness but different challenge values, an adversary that

[1] The paper in which this scheme was presented is available at https://eprint.iacr.org/2019/114.pdf.

observes both proof transcripts can directly calculate the prover's secret key.[2] Users are not likely to be aware of this, and may misuse the API such that this vulnerability is triggered. Alternatively, the team may refactor the functions such that randomness is guaranteed to be generated by a strong RNG every time.

[2] Assume a prover with secret a uses randomness v to commit to $V = g^v \mod p$ twice and either the verifier replies with two separate challenges or the hash function in the NIZK variant has two different counter values. The first challenge is r and the second is r', in response to which the prover computes $c = v - a * c$ and $c' = v - a * c'$. An adversary observing both proofs can calculate $\frac{r-r'}{c'-c} = \frac{v-a*c-v+a*c'}{c'-c} = a$.

Dashboard

Target Metadata

Name	Apache Milagro MPC
Type	Cryptographic Library
Platforms	C
Environment	Local Instance

Engagement Data

Type	Cryptographic Security Assessment
Method	Code-assisted
Dates	2020-04-21 to 2020-05-08
Consultants	3
Level of Effort	10 person-days

Finding Breakdown

Critical issues	0
High issues	0
Medium issues	0
Low issues	2
Informational issues	1
Total issues	**3**

Category Breakdown

Cryptography	3

Key

Critical High Medium Low Informational

When examining the Apache Milagro Multiparty Computation (MPC) project, NCC Group focused on ensuring that the project faithfully implemented its source protocol, avoided common programming issues, and was formulated such that end users could safely take advantage of the library. This library implements Gennaro and Goldfeder's "Fast Multiparty Threshold ECDSA with Fast Trustless Setup" paper[3] (GG) through a combination of additively homomorphic encryption, zero knowledge proofs (ZKP), Shamir's secret sharing, non-malleable equivocable commitments, Feldman's verifiable secret sharing protocol,[4] and ECDSA itself. For this project, zero knowledge proofs, commitments, ECDSA, and the overall structure were in scope. Each of these structures and the methodology used to test it are described in more detail below.

Homomorphic Encryption

Homomorphic encryption is a special form of encryption that allows for limited computation over encrypted values without possession of the secret key. GG use an instantiation of homomorphic encryption known as Paillier encryption, in which an entity possessing ciphertexts $c_1 = E_k(m_1), c_2 = E_k(m_2)$, and the public key k under which they were encrypted can compute $c_3 = E_k(m_1 + m_2)$. In order to combine secret shares without revealing them, GG use Paillier's instantiation of additively homomorphic encryption. The implementation of the Paillier cryptosystem was not in scope for this engagement, so NCC Group did not review it.

Zero Knowledge Proofs

At a high level, zero knowledge proofs are a cryptographic technique allowing one entity (the prover) to prove to another (the verifier) that they know a particular statement is true without revealing anything else. GG leverages six variants of this technique: range proofs, a proof of modular polynomial relations, Schnorr's protocol, a proof of knowledge of integer factorization, and two novel zero knowledge proof protocols devised by GG for use in this threshold ECDSA instantiation. In this paper's security model, these proofs work in concert to prevent malicious protocol participants from causing a failure mode that leaks information about the DSA key as well as the secret nonce value $\mathbf{k}$. Extraordinarily small leaks of these values have proven practically exploitable.[5] All of these proofs were in scope, and NCC Group examined each of them separately. They are discussed separately below.

Range Proofs

One part of the threshold ECDSA algorithm consists of "share conversion", in which each pair of signing parties performs a protocol that alters the mathematical structure of the secret shares they have to make it possible to construct the signature. This pairwise protocol requires each party to send range proofs to show that the secret share they are communicating to their partner (remember, this protocol is performed pairwise) is small enough that a modular reduction will not be necessary to compute over it. This avoids a potential information leak in the final step of signature verification, where a malicious participant can engineer a verification failure that could leak information about its peers' secret shares. (NCC Group notes that they cannot present an attack exploiting this leak, but also cannot make a proof it does not exist.) The construction used for the range proofs is given in appendix A.1 of GG.

The zero knowledge proof involves random generation of values in a variety of fields as well as comparison, modular reduction, exponentiation, addition, and inversion of these values in their respective fields. In addition, the values manipulated as part of this proof are sensitive, and so they need to be securely zeroized. This engagement's scope did not include the implementation of any of this field arithmetic or the secure zeroization with the exception of special cases of big-integer multiplication and modular reduction. NCC Group did ensure that the range proof was implemented as per its specification, and that a zeroization function was called on the sensitive data structures. The big-integer multiplication and modular reduction algorithms were implemented in terms of lower-level algorithms that were not in scope, but the team ensured that the high-level algorithmic structure suggested by the function calls were valid. Additionally, the team ensured that the implied non-interactive zero knowledge (NIZK) conversion in the

[3] This paper describes a system which allows for distributed signing among n players such that any subgroup of size $t+1$ can sign while subgroups of size equal to or less than t cannot for some $t < n$. Available at https://eprint.iacr.org/2019/114.pdf.
[4] The paper presenting this scheme is available at https://www.cs.umd.edu/~gasarch/TOPICS/secretsharing/feldmanVSS.pdf.
[5] As an example, see https://eprint.iacr.org/2020/615.pdf, which presents an attack recovering an ECDSA key from an information leak so small that it does not reliably leak even one bit.

challenge function via the Fiat-Shamir transformation used the strong version[6] and was valid.

Proof of Modular Polynomial Relations

The share conversion protocol features a number of proofs that achieve zero knowledge by mixing large random numbers with the special values to which the proof commits. To ensure that the security reduces to a well-known hard problem (a typical cryptographic goal), GG has these large numbers be the factorization of an RSA modulus. This allows for a proof sketch in which an attacker must break the Strong RSA Challenge[7] in order to violate the soundness guarantees of the proof. This means that the protocol must also ensure that the participants generate and use this auxiliary RSA modulus and its components in the proof. GG use a proof of modular polynomial relations[8] to allow both sides of the pairwise protocol to prove to one another that they have indeed generated RSA moduli and are using them as part of the protocol as per the proof.

This proof is structured around a combination of a protocol and a special bit commitment algorithm. The bit commitment algorithm requires an initial setup. This proceeds by generating two safe primes `P` and `Q`, and picking generators of the primes' associated multiplicative subgroups, a generator of the multiplicative semigroup associated with the primes' product, and two random members of this semigroup. This requires safe prime generation, the Chinese Remainder Theorem, and reliable algorithms for finding group membership. These are all implemented in Apache Milagro, although they are implemented in terms of lower-level functions whose implementations were not in scope. NCC Group ensured that the high-level algorithms were structured correctly assuming correct priors and performed dynamic tests showing that they functioned as expected by dynamically testing against known-good implementations of the same algorithms from Sage math.

The bit commitment algorithm itself proceeds by modular exponentiation of the various generators created in the setup by the desired bits and a random value. While the underlying mathematical implementations were not in scope, NCC Group ensured that the algorithm was correct assuming equally correct lower-level implementations.

The zero knowledge proof leveraging the prior bit commitments shows that the prover knows a legitimate auxiliary RSA modulus is, as in other cases, implemented in terms of lower-level functions whose implementations were not in scope. Nevertheless, NCC Group ensured that the algorithm described matched the description, and that it was correct as long as the lower-level algorithms were as well.

Schnorr's Protocol

Schnorr's protocol is a well-known zero knowledge proof that one party knows the value of a particular discrete logarithm. GG use both the classical version as well as a slight variant that proves the knowledge of two different discrete logarithms simultaneously. Apache Milagro includes tools that support the interactive and non-interactive versions of both protocols. The prover proceeds in the interactive and non-interactive cases by supplying a group generator raised to the power of the discrete logarithm, then solves a randomly generated challenge (sent by the verifier) that requires knowledge of the discrete logarithm to which they committed in the first step. The non-interactive variants replace the verifier's challenge generation with a cryptographic hash function; in the random oracle model,[9] a challenge generated by the hash of the information that they would have sent the verifier is equivalent to a random value generated by a counterparty. This technique is known as the Fiat-Shamir transformation.

NCC Group checked that the implementation as a whole was implemented correctly at a high level, since the lower-level calculations were not in scope, and also verified that the Fiat-Shamir transformation was applied correctly and its soundness property was maintained. Additionally, NCC Group ensured that the proofs themselves had liveness

[6] The "weak" and "strong" forms of the Fiat-Shamir transformation differ in that the weak form hashes only the commitment to form the challenge, which does not bind it to a particular proof statement. This is discussed in more detail in https://eprint.iacr.org/2016/771.pdf.

[7] This challenge is that, given an RSA modulus `N` and a ciphertext `c`, the adversary must find a message and public exponent `e` such that the message encrypted under the public exponent with respect to the modulus returns the chosen ciphertext. This is believed to be infeasible.

[8] The full scheme is presented in "Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations", available at https://www.researchgate.net/publication/221355462_Statistical_Zero_Knowledge_Protocols_to_Prove_Modular_Polynomial_Relations.

[9] See the series beginning at https://blog.cryptographyengineering.com/2011/09/29/what-is-random-oracle-model-and-why-3/ for more information on the random oracle model.

guarantees[10] and could not be arbitrarily replayed.

Proof of Integer Factorization

During key generation, each participant in the protocol creates an RSA modulus and proves that it is legitimately theirs via a proof showing that they know the factorization of their modulus. This Milagro implementation chose the proof method created by Poupard and Stern in "Short proofs of knowledge for factoring"[11] (PS).

The PS method uses the Fiat-Shamir transformation of an interactive statistical zero knowledge proof protocol. This protocol is structured as a series of rounds of the following sub-protocol:

1. Prover commits to several randomly chosen numbers between zero and the number whose factorization they claim to know
2. Verifier replies with a random challenge
3. Prover proves the knowledge of a small discrete log for the randomly chosen numbers in the context of the challenge.

A complete proof includes many runs of this round. PS optimize the sub-protocol via hashing the commitments into a single value and render it non-interactive by replacing the verifier's challenge with a hash of both the proof statement and the commitments. The proof is parametrized by values A and B, which largely exist to provide an upper limit on the size of the proofs to ease the proof that the scheme is in fact zero knowledge.[12]

NCC Group checked the high-level implementation of this scheme for fidelity to the original algorithm description. The team also ensured that the relevant checks were made on all inputs, and that the proof was not replayable. As before, the implementation of the underlying mathematical operations was not in scope.

MtA Respondent Proofs

The "share conversion" protocol discussed above, is called the Multiplicative-to-Additive (or MtA) protocol equivalently in GG and all references above to share conversion refer to this protocol. There are two possible methodologies offered in GG for a run of this protocol: one that is "checked" and one that is not. Apache Milagro implements both the checked and unchecked version, which are discussed separately below.

Unchecked

In the unchecked version, the respondent in the share conversion protocol must prove to the initiator that they are in possession of correctly-formed secret shares and have correctly calculated the response to the initiator. NCC Group checked that the proof was implemented as specified in appendix A.3 of GG, and additionally ensured that the implicit conversion to the NIZK version via the Fiat-Shamir transformation included the proof statement as well as all commitments and public strings.

Checked

The checked version adds an additional component to the proof showing that the respondent's share is small enough to avoid modular reduction and the attendant information leak as described above. The engagement team checked that the proof was implemented as specified in appendix A.2 of GG, and additionally ensured that the implicit conversion to the NIZK version via the Fiat-Shamir transformation included the proof statement as well as all commitments and public strings.

Shamir's Secret Sharing

The implementation of Shamir's secret sharing was not in scope for this engagement, so NCC Group did not review this component.

[10] For more detail on this, see the suggestions presented by NCC Group to finding NCC-QRED001-002 on page 10 and finding NCC-QRED001-003 on page 12.

[11] Available at https://www.di.ens.fr/~stern/data/St84.pdf.

[12] The original paper proves the zero knowledge property in a proof by contradiction. The contradiction arises from the runtime of an algorithm to factor the number in question, which is given in terms of A and B. As a result, if these values are allowed to be arbitrarily large the attacker's algorithm could conceivably be non-polynomial, meaning that the contradiction is no longer found.

Non-Malleable Equivocable Commitments (NMEC)

In general, cryptographic commitment schemes allow an entity (the sender) to commit to a particular value at a point in time (without revealing that value) such that they can "open" the commitment later and reveal the value to which they committed. An NMEC scheme is also "equivocable", meaning that an entity in possession of a particular secret can open the commitment to *any* value instead of only the one to which the sender originally committed. Additionally, the scheme is non-malleable, meaning that an adversary that observes an opening from a commitment C to value m cannot construct a commitment C' that opens to a value m' bearing a known relationship to m.

GG suggests that this can be easily implemented in the random oracle model by using a hash function. In particular, one can make an NMEC to a value x as $Com = Hash(x||r)$ where r is a fixed-length random bitstring. The Milagro MPC project uses this formulation. NCC Group ensured that the high-level construction of this level matched this algorithm. The hash function implementation, random number generation logic, and bit comparison logic used to interact with NMECs were out of scope.

Feldman's VSS

The implementation of VSS was not in scope for this implementation, so the engagement team did not review this component.

ECDSA

After the secret has been recovered, it is necessary to finally sign the desired message. In support of this Apache Milagro includes an ECDSA implementation. Again, the underlying mathematical implementations were not in scope, so did not see review. NCC Group did review the high-level algorithm, ensuring that all aspects of the algorithm appeared to be securely implemented. Additionally, the team ensured that the verification algorithm was designed for "full verification" as per NIST Special Publication 800-56A revision 3.[13]

[13] See section 5.6.2.3.3.

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 14.

Title	Status	ID	Risk
Schnorr Proofs Are Replayable	Fixed	002	Low
Proofs of Knowledge of Integer Factorization Can Be Replayed	Fixed	003	Low
Integer Factorization Proof Components Are Not Bounds-Checked	Fixed	004	Informational

Finding	**Schnorr Proofs Are Replayable**
Risk	**Low** Impact: High, Exploitability: Low
Identifier	NCC-QRED001-002
Status	Fixed
Category	Cryptography
Location	src/schnorr.c: 78 @ commit `c5f0733`
Impact	An attacker can forge a proof of knowledge of a discrete logarithm by replaying a prior valid one, potentially leading to a leak of the threshold ECDSA secret.
Description	Schnorr proofs do not include their own source of cryptographic "liveness", which means that they can be replayed unless they are implemented with safeguards. As Apache Milagro uses Schnorr proofs without any liveness safeguards, attackers can replay any Schnorr proof without detection. Recall that Schnorr proofs use a cryptographic hash to simulate the random challenge that the verifier would typically issue in an interactive proof of knowledge.[14] This hash function acts as a random oracle to both deliver an unpredictable challenge and bind the prover to the hashed values. Note, however, that the resulting proof tuple is permanently valid as a proof of knowledge for these bound values. Since Apache Milagro does not include anything other than the base relative to which the prover knows the discrete logarithm, and the prover's public key and a random value, anyone observing the original proof could re-submit it to effectively spoof their knowledge of the same value as the original prover. The MPC protocol implemented in Milagro uses these proofs to ensure all protocol participants were issued key shares. This is important, as this protocol leaks information when an adversary is able to create a full run of the protocol, which they may be able to do in the case that they can forge this proof. Since this proof is used to prove the correct formation of the range proof, it's possible that an attacker that can forge this proof would be able to convince a counterparty to perform the MtA protocol[15] with a value large enough that it would require modular reduction. The presence or absence of a modular reduction in the case of a crafted large value would leak information about both the threshold ECDSA secret and the secret nonce, potentially allowing the attacker to recover them and forge signatures.
Recommendation	The MPC protocol should hand out unique identifiers to each potential signatory while handing out key shares. Each Schnorr proof should then include this unique identifier as well as an ongoing proof of liveness in each proof. Liveness can be proven in many ways, but each requires some shared state between all participants in the protocol. Note that these techniques also serve to protect against replays across runs of the same protocol. Some exemplar liveness mechanisms are as follows: • Each user maintains a strictly increasing counter. This counter's value is included in every Schnorr proof and is incremented every time the user proves something. • Each round of proofs is securely associated with a particular nonce. This nonce is included in every Schnorr proof. • Each participant of the protocol issues their own nonce each round; every participant cre-

[14] This technique is commonly known as the Fiat-Shamir heuristic, from their introduction of the concept in How to Prove Yourself. Prominent cryptographers have mentioned that this technique was introduced by Blum in an earlier paper but this is disputed.

[15] This is a protocol used to convert the multiplicative Schnorr shares into additive shares that can be combined into the threshold ECDSA key and nonce. See Testing Methodology on page 5 or GG for more information on this protocol.

ates a Schnorr proof per-user that includes that user's nonce.

Retest Results Fixed as per NCC Group recommendations in commit 9fbae7b.

Finding	**Proofs of Knowledge of Integer Factorization Can Be Replayed**
Risk	**Low** Impact: High, Exploitability: Low
Identifier	NCC-QRED001-003
Status	Fixed
Category	Cryptography
Location	src/factoring_zk.c: 96 @ commit `c5f0733`
Impact	An attacker can forge a proof of knowledge of an integer's factoring by replaying a prior valid one, potentially leading to attacker recovery of a threshold signature participant's private key.
Description	As in finding NCC-QRED001-002 on page 10, the proof of knowledge of integer factorization used in Milagro[16] uses the Fiat-Shamir heuristic to convert a Sigma protocol to a non-interactive zero knowledge proof (NIZK). Recall that the proof is only bound to the values used in the hash to simulate the random challenge. In this case, the proof is only bound to the number whose factorization the prover knows, random numbers, and the commitments to those random numbers. As before, this means that anyone who observes this proof can replay it to falsely prove their own knowledge of the same integer factorization. The Milagro protocol uses these proofs of knowledge of integer factorization to prove that participants in the protocol are aware of the RSA modulus used to instantiate the Paillier cryptosystem. As before, since the protocol leaks information in the case that an adversary is able to cause a full run of the protocol, the attacker's ability to successfully forge such a proof may result in leaking of information, including the nonce used to sign the threshold nonce. Leaks of the nonce equivalent to a byte or less of information per run of protocol have historically proven sufficient to reveal the ECDSA secret.
Recommendation	As before, the MPC protocol should hand out unique identifiers to each potential signatory while handing out key shares. Each Schnorr proof should then include this unique identifier as well as an ongoing proof of liveness in each proof. Liveness can be proven in many ways, but each requires some shared state between all participants in the protocol. Some exemplar liveness mechanisms are as follows: • Each user maintains a strictly increasing counter. This counter's value is included in every Schnorr proof and is incremented every time the user proves something. • Each round of proofs is securely associated with a particular nonce. This nonce is included in every Schnorr proof. • Each participant of the protocol issues their own nonce each round; every participant creates a Schnorr proof per-user that includes that user's nonce.
Retest Results	Fixed as per NCC Group suggestion in commit 9fbae7b.

[16] The paper introducing this proof can be found at https://www.researchgate.net/publication/2450244_Short_Proofs_of_Knowledge_for_Factoring

Finding	**Integer Factorization Proof Components Are Not Bounds-Checked**
Risk	**Informational** Impact: High, Exploitability: Undetermined
Identifier	NCC-QRED001-004
Status	Fixed
Category	Cryptography
Location	src/factoring_zk.c: 225 @ commit `c5f0733`
Impact	An attacker that can supply an extremely large proof could be able to commit a forgery with it.
Description	The NIZK proof of knowledge of factorization used by Apache Milagro[17] requires that $0 \leq y < A$ and $0 \leq e < B$ where e is the verifier's challenge (either the output of the hash function in the NIZK context or the verifier's challenge in the Sigma protocol), y is the zero knowledge proof itself, and A and B are security parameters of the protocol. However, the Milagro library does not perform either of these checks. The NIZK proof uses these assumptions to argue for its soundness. Particularly, binding the size of e and y allows for an argument that an adversary using them to forge a proof can also directly find the factoring of the number under study in the proof protocol. As a result, not performing these checks allows for the possibility of un-sound execution. However, the likelihood of a practical attack arising from this is low, particularly because the size of these values is bound by the size of the data types used to transmit them. Nevertheless, it is best practice to ensure that the implementation matches the overall protocol. As in finding NCC-QRED001-003 on the previous page, this proof is a crucial piece of the range proof, and so successful exploitation of this issue to forge an integer factorization proof could result in a range proof forgery, leading to potential information leak and compromise of the threshold ECDSA secret.
Recommendation	Ensure that the protocol includes the relevant checks on the value of e and y.
Retest Results	Range checks added as per NCC's suggested remediation in commit c6d3d5a.

[17] The paper introducing this proof can be found at https://www.researchgate.net/publication/2450244_Short_Proofs_of_Knowledge_for_Factoring.

Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical Implies an immediate, easily accessible threat of total compromise.

High Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

Medium A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

Low Implies a relatively minor threat to the application.

Informational No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

Medium Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

Low Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Medium Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.

Low Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.