SafeStake

Operator Networking

Security Assessment Report

Version: 2.1

September, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of selected SafeStake components. The review focused solely on the security aspects of the Rust implementation, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the SafeStake components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the SafeStake components included in the scope of this review.

Overview

SafeStake is a trust minimized middle layer for decentralized ETH staking. It implements Distributed Validator Technology (DVT) by splitting the validator key into shares, allowing an operator to continue operations if a single node goes down.

This review focuses on the networking implementation of the peer-to-peer discovery component, which heavily uses the Lighthouse and discv5 dependencies, maintained by Sigma Prime.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the SafeStake repository.

The scope of this time-boxed review was strictly limited to files at commit f24463e.

Two files are listed as in scope for this review.

- `src/bin/dvf_root_node.rs`
- `src/node/discovery.rs`

The fixes of the identified issues were assessed at commits 34d03e5 then 4aca529.

Note: third party crates and dependencies were excluded from the scope of this assessment.

Approach

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the components in scope. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Rust language and Ethereum protocol.

To support this review, the testing team also utilised the following testing tools:

- Clippy linting: `https://doc.rust-lang.org/stable/clippy/index.html`

Output for these automated tools is available upon request.

Additionally, the testing team leveraged fuzz testing techniques (i.e. fuzzing), which is a process allowing the identification of bugs by providing randomised and unexpected data inputs to software with the purpose of causing crashes (in Rust, *panics*) and other unexpected behaviours (e.g. broken invariants,).

Sigma Prime produced fuzzing targets targeting the components using cargo-fuzz, a fuzz-testing framework for Rust software. This framework focuses on:

- **In-process fuzzing**: the fuzzing engine executes the target many times with multiple data inputs in the same process. It must tolerate any kind of input (empty, huge, malformed, etc.);
- **White-box fuzzing**: `cargo-fuzz` leverages compiler instrumentation and requires access to the source code;
- **Coverage-guided fuzzing**: for every input/test case, `cargo-fuzz` tracks code paths (sections of the code reached), and produces variants of each test case to generate additional input data to increase code coverage.

The testing team created a set of fuzzing targets to complement the manual review.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 11 issues during this assessment. Categorised by their severity:

- Medium: 6 issues.
- Low: 2 issues.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the SafeStake components in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
SSO-01	Calls To `unwrap()` & `panic()` In `discovery.rs`	Medium	**Resolved**
SSO-02	Calls To `unwrap()` & `panic()` In `dvf_root_node.rs`	Medium	**Resolved**
SSO-03	Negative Overflows In `discovery.rs`	Medium	**Resolved**
SSO-04	Negative Overflows In `dvf_root_node.rs`	Medium	**Resolved**
SSO-05	Unprotected Secrets File With Hard-Coded Path	Medium	**Resolved**
SSO-06	Use Of Vulnerable & Outdated Components	Medium	**Closed**
SSO-07	Unnecessary Reliance On External, Centrally Controlled Web Services	Low	**Resolved**
SSO-08	Non-Zeroed Private Key Leaks Secret Information	Low	**Closed**
SSO-09	Implicit Dependency On `curl` Executable	Informational	**Resolved**
SSO-10	Overflow In `discovery.rs`	Informational	**Closed**
SSO-11	Miscellaneous General Comments	Informational	**Resolved**

SSO-01	Calls To `unwrap()` & `panic()` In `discovery.rs`		
Asset	`src/node/discovery.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

There are numerous calls to `unwrap()` in `discovery.rs`, which may be reached under certain conditions, causing the client to crash.

The following code snippets show where a panic may occur.

```
let _ = discv5.start().await;
let mut event_stream = discv5.event_stream().await.unwrap();
```

If the node sets a port that the operating system does not have permission to open, this will error and cause a panic when opening the `event_stream`.

```
let enr_key = CombinedKey::secp256k1_from_bytes(&mut secret_key[..]).unwrap();
```

May panic if provided with an invalid SECP256k1 secret key by the node operator.

```
let store = Store::new(&store_path.to_str().unwrap()).unwrap();
```

May panic if `rocksdb` cannot initialize the database.

```
store.write(enr.public_key().encode(), bincode::serialize(&SocketAddr::new(IpAddr::V4(ip), discv_port -
      ↪ DISCOVERY_PORT_OFFSET)).unwrap()).await;
```

If `bincode::serialize()` errors, then `unwrap()` will panic. However, the likelihood of this issue is rated as low as it is not expected these panics are reachable under standard conditions. The impact is rated as high as these occur in the main event loop and would cause the client to crash.

Recommendations

Avoid using `unwrap()` and handle all errors gracefully. For cases where the error case is deemed unreachable, `expect()` should be used with a description provided explaining why it is not reachable.

Furthermore, include relevant logging for each error.

If a fatal error occurs, exit the program gracefully. If the error occurs in the main event loop and is not fatal, then the loop should continue to the next iteration.

Resolution

This has been acknowledged by the development team and no code changes will be made. The provided explanation describes adequate mitigations for identified security risks, and included the following points:

- The code is intended to be run only in a controlled environment (the docker container defined in this repository), where configured ports can be reasonably expected to be available.
- The `Discovery` struct implementation is intended for use only within context of the existing node codebase, where `Discovery::spawn()` is called once during startup. As such, the first three panics would occur during the node initialisation phase, where there is no need to cleanly handle the errors or recover.
- The first three panics would be caused by node operator misconfiguration or unrecoverable issues with the environment that require manual intervention (like a full disk).
- The serialisation in the fourth snippet will not fail with the current implementation and bincode configuration. Where the untrustworthy external input is already verified to be a valid IP and port.

In general, it is deemed safe to crash in the currently feasible failure scenarios. The testing team notes some residual risk, under scenarios where changes to the third party bincode or serde dependencies introduce other possible reasons for `bincode::serialize()` to return an error.

SSO-02	Calls To `unwrap()` & `panic()` In `dvf_root_node.rs`		
Asset	`src/bin/dvf_root_node.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

There are numerous calls to `unwrap()` which would cause the program to panic on an error. This is considered undesirable and suboptimal programming practice.

One example is shown below, related to the RocksDB instantiation.

```
let store_dir = base_dir.join(DEFAULT_STORE_DIR);
let store = Store::new(store_dir.to_str().unwrap()).unwrap();
```

If the `DEFAULT_STORE_DIR` is is not accessible, this may panic. It will also panic if `rocksdb` fails instantiation for any reason.

Furthermore, a number of `unwrap()` statements are present in the main event loop. If `bincode::serialize()` returns an error they will raise a panic. This would cause the client to crash, which is considered high impact. However, the likelihood is rated as low since it is unlikely for this function to error.

Recommendations

Avoid using `unwrap()` and handle all errors gracefully. For cases where the error case is deemed unreachable, `expect()` should be used with a description provided explaining why it is not reachable.

Furthermore, include relevant logging for each error.

If a fatal error occurs, exit the program gracefully. If the error occurs in the main event loop and is not fatal, then the loop should continue to the next iteration.

Resolution

This has been acknowledged by the development team and no code changes will be made. The justification includes:

- This code is to be run as a P2P boot node, under a highly controlled environment.
- It is safe to crash during initialisation due to an invalid configuration or problems with the environment.
- The `bincode::serialize()` function calls will not fail in the current implementation.

More explanation is provided in SSO-01 Resolution

In general, it is deemed safe to crash in the currently feasible failure scenarios. The testing team notes some residual risk, under scenarios where changes to the third party bincode or serde dependencies introduce other possible reasons for `bincode::serialize()` to return an error.

SSO-03	Negative Overflows In `discovery.rs`		
Asset	`src/node/discovery.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

There are six possibilities for a negative integer overflow in `discovery.rs`. These occur when a UDP discovery port is provided that is less than the offset.

The overflows occur on the following lines.

- line [88]
- line [138]
- line [154]
- line [164]
- line [173]
- line [196]

The following code snippet highlights the potential overflow on line [154]. The overflow will occur if an ENR contains a `discv_port` value that is less than `DISCOVERY_PORT_OFFSET`.

```
Some(event) = event_stream.recv() => {
    match event {
        Event::Discovered(enr) => {
            if let Some(ip) = enr.ip4() {
                // update public key ip
                if let Some(discv_port) = enr.udp4() {
                    // update public key socket address
                    store.write(enr.public_key().encode(), bincode::serialize(&SocketAddr::new(IpAddr::V4(ip), discv_port -
                          ↪  DISCOVERY_PORT_OFFSET)).unwrap()).await; // @audit overflow
                    set_metrics(&store, enr.public_key().encode()).await;
                }
            };
        },
```

Some of the negative overflow may be caused by an external actor and therefore are assigned a high likelihood. The impact is rated low as the release build will cause the overflow to wrap. However, in debug mode or compiled with `overflow-checks=true`, these overflows would panic causing the client to crash.

Recommendations

It is recommended to use checked operations rather than the native operators for all mathematical operations in the codebase. For example, use `checked_sub()` rather than `-`.

Resolution

The recommended checked arithmetic has been added in commits 34d03e5 and 9dc4b83. Added `unwrap()` calls occur in acceptable initialisation scenarios, with the same justification as for SSO-01.

SSO-04	Negative Overflows In `dvf_root_node.rs`		
Asset	`src/bin/dvf_root_node.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

There are two possibilities of a negative integer overflow in the `async main()` function. This will panic when compiling with the `debug` flag and wrap around if compiled with the `release` flag.

The following code snippet shows two potential negative overflows.

```
loop {
        tokio::select! {
            Some(event) = event_stream.recv() => {
                match event {
                    Event::Discovered(enr) => {
                        if let Some(enr_ip) =  enr.ip4() {
                            if let Some(discv_port) = enr.udp4() {
                                store.write(enr.public_key().encode(), bincode::serialize(&SocketAddr::new(IpAddr::V4(enr_ip),
                                      ↪  discv_port - DISCOVERY_PORT_OFFSET)).unwrap()).await; //@audit overflow
                            }
                        }
                    },
                    Event::SessionEstablished(enr,  _addr) => {

                        if let Some(enr_ip) =  enr.ip4() {
                            if let Some(discv_port) = enr.udp4() {
                                let socketaddr = SocketAddr::new(IpAddr::V4(enr_ip), discv_port - DISCOVERY_PORT_OFFSET); //@audit
                                      ↪  overflow
                                info!("A peer has established session: public key: {}, base addr: {:?}",
                                base64::encode(enr.public_key().encode()), socketaddr);
                                store.write(enr.public_key().encode(), bincode::serialize(&socketaddr).unwrap()).await;
                            }
...
```

These negative overflows may be caused by an external actor and therefore are assigned a high likelihood. The impact is rated low as the release build will cause the overflow to wrap. However, in debug mode or compiled with `overflow-checks=true` these overflows would panic causing the client to crash.

Recommendations

Handle overflows gracefully. The port can be set to a default value in case `discv_port < DISCOVERY_PORT_OFFSET`.

Furthermore, all integer mathematical operations throughout the codebase should use checked operations rather than the native operators. For example, use `checked_sub()` rather than `-`.

Resolution

The recommended checked arithmetic has been added in commit 9dc4b83.

SSO-05	Unprotected Secrets File With Hard-Coded Path		
Asset	`src/bin/dvf_root_node.rs` & `hotstuff/config/src/lib.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The `main()` function in `src/bin/dvf_root_node.rs` utilises a file with unprotected permission and a hard-coded path to store and read the local node `Secret` structure.

The stored information is the public key and secret key for the node. The following code segment shows a constant being used to set the file path.

```
let secret_dir = base_dir.join(DEFAULT_SECRET_DIR);
```

Similarly, the following code segments show the call to and the `read()` function, which extracts the `Secret` structure from `DEFAULT_SECRET_DIR`.

```
let secret = if secret_dir.exists() {
        info!("Loading secret file: {}", &secret_dir.to_str().unwrap());
        Secret::read(secret_dir.to_str().unwrap()).unwrap()
}
```

This issue was given a medium impact as the local node Public/Private Key combination used in `discv5` could be compromised in case of unauthorised access to the host. The issue has been given a medium probability as this configuration will always occur, however full exploitation would require compromising the node.

Recommendations

Restrictive permissions should be enforced on this file. In the Unix operating environment the exclusive read-write is represented as `600` file permission.

If relying on `docker-compose` a good practice would be to involve Docker secrets.

Furthermore, hard-coding the path also prevents a configurable setup, enabling better at-rest data protection, for example, providing the file via a temporary file system that is not on-disk.

Resolution

This issue has been resolved for the most part in commit 9dc4b83.

The default implementation of `Export::write()` (at `hotstuff/config/src/lib.rs:39`) has been modified such that creating a new secret file sets appropriate file permissions. These permissions, however, are not enforced when reading the file.[1]

[1] For example, the OpenSSH client exits with an "Bad owner or permissions" error if the secret file it reads has lax permissions.

SSO-06	Use Of Vulnerable & Outdated Components		
Asset	`Cargo.toml`, `Cargo.lock`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The SafeStake Operator codebase depends on Rust packages with publicly known vulnerabilities.

No significant exploits for the reported vulnerabilities were identified in the current version of the software. The assessed security risk is primarily associated with flaws in the project's dependency management and monitoring processes.

The following dependent packages were identified to contain known vulnerabilities:

- atty:
  - Potential unaligned read.
    * `https://rustsec.org/advisories/RUSTSEC-`2021-0145
  - Furthermore, it is unmaintained.
  - It is only used by `env_logger`
  - **Recommendation:** Upgrade `env_logger` to `>=0.10.0`
- bytes:
  - Version `1.6.0` has been yanked from `crates.io`, signalling that it should not be used.
  - This appears to be due to a bug with `Bytes::is_unique()` resolved in tokio-rs/bytes#718.
  - **Recommendation:** Upgrade to `>=1.6.1`
- curve25519-dalek and ed25519-dalek:
  - Timing variability in `curve25519-dalek`'s `Scalar29::sub` / `Scalar52::sub`
    * `https://rustsec.org/advisories/RUSTSEC-`2024-0344
  - Insecure API allows double public key signing function oracle Attack on `ed25519-dalek`
    * `https://rustsec.org/advisories/RUSTSEC-`2022-0093
  - These are problematic vulnerabilities but curve25519 does not appear to be used in SafeStake Operator
  - **Recommendation:** upgrade to `>=4.1.3` or remove the dependency
- openssl:
  - `MemBio::get_buf` has undefined behaviour with empty buffers.
    * `https://rustsec.org/advisories/RUSTSEC-`2024-0357
  - **Recommendation:** upgrade to `>=0.10.66`
- serde_yaml

- This package is no longer maintained. This is noted in the README, the most recent release and the repository is archived.
- **Recommendation:** Evaluate alternatives, though the package currently appears to be relatively stable. Consider the security trade-offs involved with trusting a different maintainer.
  * As of the time of this report, serde_yaml_ng appears the most promising fork.
  * The testing team cautions **against** the serde_yml fork, which appears to exhibit several security concerns.

- yaml-rust
  - This package is no longer maintained and missing several important fixes.
    * `https://rustsec.org/advisories/RUSTSEC-2024-0320`
  - **Recommendation:** This is currently only a transitive dependency and can be removed by upgrading to `serde_yaml` `>=0.9` (note above, that `serde_yaml` is also unmaintained).
    * Alternatively, there are currently maintained forks yaml-rust2 or saphyr.
- zerovec and zerovec-derive:
  - Incorrect usage of `#[repr(packed)]`.
    * `https://rustsec.org/advisories/RUSTSEC-2024-0346`
    * `https://rustsec.org/advisories/RUSTSEC-2024-0347`
  - Leads to memory corruption when compiled with rust `>=1.80.0-beta`.
  - **Recommendation:** upgrade `zerovec` to `>=0.10.4`

Many critical dependent packages were found to be severely outdated. Some include:

- libsecp256k1 `0.7.0` released in September 2021 - `0.7.1` was released in July 2022
- rocksdb `0.19.0` released in August 2022 - `0.22.0` was released in February 2024
  - This is required in `hotstuff/store/Cargo.toml`
  - Note that the current version fails to compile on systems with gcc `>=13`, explained in `https://github.com/facebook/rocksdb/pull/11118`
- ring `0.16.19` released in December 2020 - `0.17.8` was released in February 2024
  - Only the latest release is publicly supported.
- secp256k1 `0.24.0` released in July 2022 - `0.29.0` was released in April 2024
- sha2 `0.9.2` released in November 2020 - `0.10.8` was released in September 2023
- sha256 `1.0.3` released in November 2021 - `1.5.0` released in January 2024

Recommendations

Upgrade the outdated dependencies as suggested above.

In addition, the testing team strongly recommends revisiting dependency management and alerting procedures. Ensure dependencies are regularly updated, and security advisories in dependencies raise visible alerts that are promptly acted upon.

Revisit `Cargo.toml` and remove direct dependencies that are not currently used by SafeStake Operator.

Solutions could involve dependabot, CI with cargo audit and cargo outdated.

Resolution

This has been partially resolved in commits 34d03e5 and 9dc4b83.

- Vulnerable dependencies identified by `cargo audit` have been updated.
- Some explicitly identified, outdated dependencies have been updated but not all.

Note that the following action items have not been implemented.

- No unused, direct dependencies have been removed.
- No evidence was provided of changes to dependency management or vulnerability alerting procedures.

SSO-07	Unnecessary Reliance On External, Centrally Controlled Web Services		
Asset	`src/utils/ip_util.rs, src/bin/dvf_root_node.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The utility functions `get_public_ip()` and `get_local_ip()` rely on external, centrally controlled web services.

`get_public_ip()` queries the site `http://ifconfig.me` to get an external IP address. This is used by the SafeStake bootnode executable to obtain its default IP used in constructing its ENR.

If the site is offline or returns a malformed response, the bootnode panics during startup at `dvf_root_node.rs` line [**84**]. It is also possible for the site to maliciously return an incorrect response, which could be used to advertise the IP of a malicious boot node.

The testing team notes that `get_local_ip()` is unused outside of test code, hence the low severity of this issue.

Recommendations

Consider comparing results from several services or supplying the IP from configuration (obtained from the ISP). If running the bootnode requires a static IP, this can be obtained during configuration.

Alternatively, consider querying several services, allowing for failure and malformed responses. Return the most common result.

Always prioritise HTTPS connections, to protect against a Man-in-the-Middle (MITM) attack modifying the result.

Resolution

This has been resolved in commit 9dc4b83, which removes the use of `get_public_ip()` and requires the public IP to be passed to the bootnode executable as a CLI argument.

SSO-08	Non-Zeroed Private Key Leaks Secret Information		
Asset	`src/bin/dvf_root_node.rs, src/node/discovery.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The private key is copied multiple times without clearing the underlying data from memory.

In general, copying any kind of sensitive data in Rust is not recommended. If the variable is not properly `zeroized`, the sensitive data could remain in memory indefinitely, leading to potential unnecessary exposure, since Rust does not guarantee memory is cleared after it is released. Future operations on the machine may read the uncleared memory to extract the secret information.

The following code snippet shows the line where the secret key is copied.

```
let mut secret_key = secret.secret.0[..].to_vec();
```

A similar issue exists on line [**76**] of `src/node/discovery.rs`.

The issue is given a medium impact as extraction of the secret key may lead to a compromised node. The likelihood is rated as low as an attacker must be able to read specific memory to extract the key, which requires advanced compromise and/or exploitation.

Recommendations

The `Zeroize` crate should be used with the `Secret` struct, to zero it out of memory when it gets dropped.

Since only `secret.name` is needed to print a log, the log should first be printed, then secret key should be moved into the `ENR`. All other references to the `secret` should be dropped and `zeroed` from memory.

Resolution

This has been partially resolved in commit 34d03e5.

The `Zeroize` crate has been used to zero memory of the `Secret` struct when dropped. However, unnecessary references to secret are retained for the life of the bootnode and the `Discovery` struct, and the data is copied into other unprotected vectors via direct access to the underlying array. There are no protections against the value being copied by the compiler during moves.

SSO-09	Implicit Dependency On `curl` Executable
Asset	`src/utils/ip_util.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

`get_public_ip()` runs the `curl` executable as a child process. This is not documented, and may cause issues if SafeStake Operator code is run in a limited environment.

Recommendations

Consider instead using an in-process library like reqwest, which is already a dependency. This also avoids performance overheads of spawning a child process. Alternatively, ensure the dependency on `curl` is well documented.

Resolution

This has been resolved in commit 9dc4b83, which removes `get_public_ip()`.

SSO-10	Overflow In `discovery.rs`
Asset	`src/node/discovery.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

There is a possible overflow when calculating `new_seq` on line [65].

```
let new_seq = u64::from_le_bytes(value.try_into().unwrap()) + 1;
```

`ENR seq` starts with 2 and is incremented one by one. The `ENR seq` is incremented every time discovery is started, not checking if the record needs to change.

Recommendations

Consider implementing a check to determine if an increment is needed, similar to Lighthouse. Use checked arithmetic and generate a new P2P keypair if possible sequence numbers have been exhausted.

Resolution

This has been acknowledged by the development team and no code changes will be made. It is improbable that incrementing the sequence number will overflow `u64` during the lifetime of any node.

SSO-11	Miscellaneous General Comments
Asset	/*
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Use Of `unwrap()` On `Result` After Checking If It Is An `Error` In `discovery.rs`**

   Related Asset(s): ***`src/node/discovery.rs`***

   `Result` is checked for error before on line [**257**] of `discovery.rs`, after which an `unwrap()` is called on the same result as demonstrated by the following code snippet.

   ```
    let curve_pk = secp256k1::PublicKey::from_slice(pk);
           if curve_pk.is_err() {
               error!("Failed to construct secp256k1 public key from the slice");
               return None;
           };
   let curve_pk = curve_pk.unwrap();
   ```

   This is not a security issue, rather a stylistic suggestion for code readability.

   A `match` statement would be better suited in this instance.

   ```
   let curve_pk =  match secp256k1::PublicKey::from_slice(pk) {
               Ok(pk) => pk,
               Err(err) => {
                   error!("Failed to construct secp256k1 public key from the slice: {}", err);
                   return None;
               }
   };
   ```

2. **Divide By Zero in `Discovery::spawn()`**

   Related Asset(s): ***`src/node/discovery.rs`***

   Division by zero may occur in the `Discovery::update_addr()` function on line [**224**], as seen below.

   ```
   let boot_idx = rand::random::() % self.boot_enrs.len();
   ```

   This issue has been labelled as informational as `boot_enrs` is set by the user in the configuration.

   Return error if `self.boot_enrs.len() == 0`.

3. **Usage Of Unused `CombinedKey` Structure**

   Related Asset(s): ***`src/bin/dvf_root_node.rs`*** **&** ***`src/node/discovery.rs`***

   The `CombinedKey` enum is not required as the ED25519 scheme is not used.

   The `CombinedKey` supports both SECP256k1 and ED25519. However, in practice, only SECP256k1 is used.

   There is no security impact of supporting both. It may improve compile times and performances, and reduce the attack surface, to only use `secp256k1` keys and remove support for `ed25519` keys.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

These comments have been acknowledged by the development team and actioned where appropriate.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

	High	Medium	High	Critical
Impact	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References