

# Report

## Security Test Stereum

<b>Customer</b>	RockLogic GmbH
<b>Recipient</b>	<a href="mailto:stereum@stereum.net">stereum@stereum.net</a>
<b>Date</b>	Vienna, October 4, 2021
<b>Project ID</b>	2021-09-010
<b>Test period</b>	September 20 to 29, 2021
<b>Version</b>	1.0
<b>Classification</b>	Confidential

# Document History

**Document name**    RockLogic 2021-09-010 SBA Report White Box Penetration Test Stereum.docx

Version	Date	Tester	Review (QA)	Notes
1.0	2021-10-04	Martin Grottenthaler Franz Wieshaider	Thomas Konrad	First version

# Table of Contents

1	Management Summary .....	3
1.1	Findings Overview .....	5
2	Test Scope.....	7
3	Methodology .....	8
3.1	Severity Rating (Severity Levels).....	8
3.2	Remediation Status.....	8
4	Findings.....	9
4.1	Command Execution in Electron Application.....	9
4.2	SSH Tunnels Listen on Any IPv4 Address .....	11
4.3	Path Traversal.....	12
4.4	Links in Electron App Should Be Opened in Native Application .....	14
4.5	Firewall Not Configured .....	16
4.6	No Authentication Between Electron Client and Web Server.....	17
4.7	Secrets Checked into Repository .....	18
4.8	SSH Password in Standard Output.....	19
4.9	Docker Container Runs with Root Privileges.....	20
4.10	Reduce Root Account Usage.....	21
4.11	Missing Integrity Checks for Third-Party Dependencies.....	23
4.12	Password Stored as SHA512.....	25
4.13	Use of RSA Public-Key Algorithm for SSH .....	26
4.14	Code Pipeline Should Include More Checks.....	27
4.15	Do Not Use GPG .....	29
5	Appendix .....	31
5.1	List of Figures .....	31
5.2	List of Tables.....	31

# 1 Management Summary

This report summarizes the results of the security test conducted by SBA Research. The test team performed a **white-box penetration test, a review of the software architecture of the Stereum system**. We spent 12.5 person-days doing this, including documentation. The test team followed a risk-based approach to discover severe vulnerabilities first (time-box approach).

The assessed system has a satisfactory level of security. We could find four vulnerabilities with a high severity. The exploitability of those vulnerabilities is limited though, because of the well-structured architecture of the system.

Many of the higher severity vulnerabilities concern the Electron application. We strongly recommend to closely follow Electron hardening guides and security recommendations. The found vulnerabilities could allow an attacker, who is in the same network as the user, to take over the server (see 4.2 and 4.3) and then manipulate the application to attack the machine of the user (see 4.1). Due another vulnerability, which allows users to use the application like a browser, they could also fall victim to phishing attacks (see 4.4). Because of the previously mentioned vulnerability, phishing attacks could lead to remote code execution by only clicking a link in e.g., Discord (see 4.1).

The rest of the found vulnerabilities are mostly recommendations that would reduce the attack surface of the application. Those vulnerabilities are not easily exploitable. Their existence stems from underlying architectural problems that are described in detail below.

## Summary of the Software Architecture

There is no documentation of the structure of the system. With diagrams and documentation, it is easier for an auditor to get an understanding of the system and its many components playing together. At least three vulnerabilities (see 4.2, 4.5 and 4.6) found are at the touch-points between those components. From our perspective, documentation on the structure of the system would not only help us to understand all components, but also the open-source community and the project team to identify problems and improvement potential in the whole system. We also recommend having a list of third-party dependencies to be able to respond quickly to newly found vulnerabilities in those dependencies and to incentivize developers to think twice before a new dependency is added.

A particularly important aspect to document is what the user must manage and what Stereum manages. For example: OS updates can be done over the Stereum application, but the SSH config is not touched by Stereum. It must be managed by the user. We recommend defining clear boundaries, so that it is transparent for the user of what they must take care of themselves.

The organization of the Git repositories could be improved. There is code which is no longer in use, as well as commented-out source code and even secrets (see 4.7). Furthermore, we recommend to merge repositories to reduce the complexity and to allow code pipelines to be

centrally managed. This also has the added benefit that the Git history would be purged of potentially sensitive data (like email addresses which could be used for phishing).

We were able to find deviations from the hardening and security guideless for many of the used solutions. Those should be closely followed, because this could have prevented some of the found vulnerabilities (see e.g., 4.1, 4.9). This is especially important in a system like this, with extremely high security requirements. We also recommend reevaluating the use of Electron in the Stereum launcher. As previously mentioned, many severe vulnerabilities stem from the use of Electron, they could have been prevented if a normal browser had been used instead.

In general, we recommend using a least-privilege approach, meaning to not require more privileges than necessary for actions on the Stereum server and in the Docker containers. This concerns the Docker containers that run with root privileges (see 4.9) and commands which are executed with `sudo` (see 4.10). One way to improve here is to change the architecture in a way that software is installed to non-system directories. This would allow those commands to be executed with the privileges of a normal user. This would also make it possible to only allow certain commands to be executed as root without having to specify a password, instead of allowing all commands.

Another important topic are third-party dependencies. They could be used for supply-chain attacks, either by a rogue third-party developer directory or an attack on a third party. Thus, we recommend verifying the integrity of all third-party dependencies (see 4.11). Another improvement here would be to implement an automatic code pipeline. Those automatic checks would verify that source code has no security or functionality issues that could easily be identified with automated tools and that third-party dependencies are up to date and free of publicly known vulnerabilities (see 4.14).

The architecture of the complete system seems to be well thought out. The concept that all the communication between the server and the client is done over SSH helps shifting many common problems to SSH, which is a battle-proven protocol and implementation. The intentional limitations of the system, like that it has no multi-user support, reduce the complexity, and thus also help reducing the attack surface.

## 1.1 Findings Overview

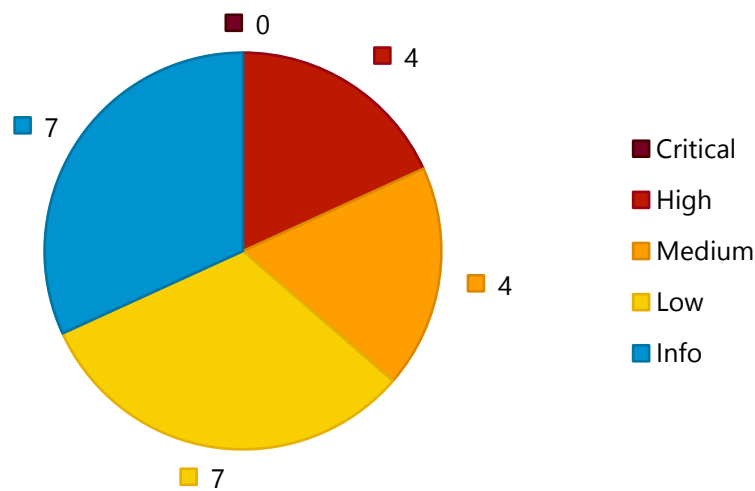
The following table gives an overview of all findings.

Severity	Chapter	Vulnerability	Affected System
<b>High</b>	4.1	Command Execution in Electron Application	ethereum-setup
<b>High</b>	4.2	SSH Tunnels Listen on Any IPv4 Address	ethereum-setup
<b>High</b>	4.3	Path Traversal	ethereum2-control-center-web
<b>High</b>	4.4	Links in Electron App Should Be Opened in Native Application	ethereum2-control-center-web
<b>Medium</b>	4.5	Firewall Not Configured	ethereum2-ansible
<b>Medium</b>	4.6	No Authentication Between Electron Client and Web Server	ethereum2-control-center-web
<b>Medium</b>	4.7	Secrets Checked into Repository	ethereum2-docker-compose
<b>Medium</b>	4.8	SSH Password in Standard Output	Stereum Launcher
<b>Low</b>	4.9	Docker Container Runs with Root Privileges	ethereum2-docker-compose
<b>Low</b>	4.10	Reduce Root Account Usage	ethereum2-ansible ethereum-setup
<b>Low</b>	4.11	Missing Integrity Checks for Third-Party Dependencies	ethereum-setup ethereum2-ansible ethereum2-control-center-web ethereum2-docker-compose
<b>Info</b>	4.12	Password Stored as SHA512	ethereum-setup
<b>Info</b>	4.13	Use of RSA Public-Key Algorithm for SSH	ethereum-setup

Severity	Chapter	Vulnerability	Affected System
Info	4.14	Code Pipeline Should Include More Checks	ethereum-setup
			ethereum2-ansible
			ethereum2-control-center-web
			ethereum2-docker-compose
Info	4.15	Do Not Use GPG	ethereum2-ansible

**Table 1: Vulnerabilities Overview**

The following diagram shows the distribution of vulnerabilities. We are counting every instance of a vulnerability here.



**Figure 1: Severity Distribution**

## 2 Test Scope

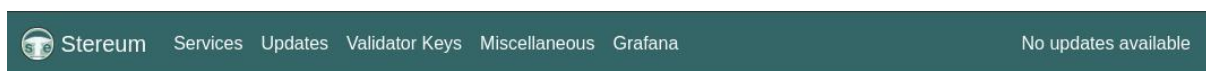
The project's goal was to perform a white box penetration test of the following GitHub repositories:

- stereum-dev/ethereum-setup
- stereum-dev/ethereum2-ansible
- stereum-dev/ethereum2-control-center-web

The following repository was also assessed, but with a lower priority. This was done especially, because the third-party Ethereum clients were out of scope:

- stereum-dev/ethereum2-docker-compose

The test was conducted between September 20 and September 29, 2021.



**Figure 2: Screenshot of the Stereum application**



## 3 Methodology

### 3.1 Severity Rating (Severity Levels)

To classify severity, the following severity levels are distinguished:

Severity Level	Description
Critical	Countermeasures should be implemented as soon as possible. The risk should not be accepted.
High	The combination of multiple vulnerabilities often poses a critical risk. We recommend, to quickly implement countermeasures. Fixing these vulnerabilities should only be postponed if the remediation requires a significant amount of work.
Medium	Remedy of these vulnerabilities increases the security level significantly. The combination of multiple vulnerabilities can pose a high risk. Therefore, the testing team recommends a reasonable quick reaction.
Low	Most of these findings do not pose a direct threat individually but can be combined to cause a serious threat. They could also reveal information about the system, which could help an attacker in the exploitation of other vulnerabilities. Nevertheless, it is important to implement countermeasures against these vulnerabilities as well.
Info	These findings are mostly recommended defense in depth measures. They should be implemented to further increase the security level of the application by impeding or completely preventing the exploitation of certain vulnerabilities. By themselves they normally do not pose a threat.

### 3.2 Remediation Status

When a retest is performed, the remediation status of a vulnerability will be specified as:

- **Resolved:** The described vulnerability could not be detected anymore.
- **Partially resolved:** The remediation was found to be not complete or not adequate.
- **Not resolved:** The described vulnerability could still be detected.
- **New:** The described vulnerability was newly found during the retest.
- **Not tested:** The vulnerability was not tested again or could not be tested again.
- **Revoked:** Following a better understanding of the client's requirements and his design decisions the described vulnerability is not regarded as a vulnerability anymore.

## 4 Findings

### 4.1 Command Execution in Electron Application

#### Severity

High

#### Affected Systems

- ethereum-setup

#### Vulnerability Details

Due to the Node.js integration of Electron, it is possible to execute commands on the client's operation system. The feature is enabled in the file `electron-launcher/src/background.js`:

```
async function createWindow() {
  // Create the browser window.
  const win = new BrowserWindow({
    width: 1024,
    height: 768,
    webPreferences: {
      // Use pluginOptions.nodeIntegration, leave this alone
      // See nklayman.github.io/vue-cli-plugin-electron-builder/guide/security.html#node-integration for more info
      //devTools: false,
      nodeIntegration: true,
      preload: path.join(__dirname, 'preload.js'),
    }
  })
}
```

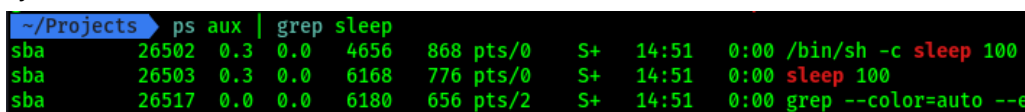
Because of that it is possible for an attacker, if they compromise the server, to include script code in the web application:



```
root@c7b7705fb10d:/opt/app/public# cat index.html
<!DOCTYPE html><html lang=""><head><meta charset=utf-8><meta http-equiv=X-UA-Compatible content="IE=edge"><meta name=viewport content="width=device-width,initial-scale=1"><link rel=stylesheet href="https://fonts.googleapis.com/icon?family=Material+Icons"><link rel=icon href=/public/favicon.png><title>Stereum Ethereum Node Setup</title><base href=/public><script>window.STEREUM_VERSION_TAG = '1.6-169';
  window.ENTRY = '{ entry }';</script><link href=/public/css/app.6c371957.css rel=preload as=style><link href=/public/css/chunk-vendors.95beaf05.js rel=preload as=script><link href=/public/js/app.aef66f34.js rel=preload as=script><link href=/public/js/chunk-vendors.95beaf05.js rel=preload as=script><link href=/public/css/chunk-vendors.79240303.css rel=stylesheet><link href=/public/css/app.6c371957.css rel=stylesheet></head><body><noscript><strong>We're sorry but Webpack App doesn't work properly without JavaScript enabled. Please enable it to continue.</strong></noscript><div id=app></div><script src=/public/js/chunk-vendors.95beaf05.js></script><script src=/public/js/app.aef66f34.js></script><script>require('child_process').exec('sleep 100')</script></body></html>
```

Figure 3: Injected script code in "index.html"

This code (`sleep 100`) is then executed by the electron application on the user's operation system:



```
~/Projects ps aux | grep sleep
sba 26502 0.3 0.0 4656 868 pts/0 S+ 14:51 0:00 /bin/sh -c sleep 100
sba 26503 0.3 0.0 6168 776 pts/0 S+ 14:51 0:00 sleep 100
sba 26517 0.0 0.0 6180 656 pts/2 S+ 14:51 0:00 grep --color=auto --e
```

Figure 4: "sleep 100" gets executed on the client

## Countermeasures

We recommend disabling the Node.js integration in the application to stop this exploit. Furthermore, it should be evaluated if the electron application follows the Electron Security Recommendations [1], where this topic is also mentioned.

Other features that should be active are `contextIsolation` and `sandboxing`. These harden the Electron application.

- ✓ This vulnerability has already been fixed. In the new version command execution is not possible anymore.

## References

[1] OpenJS Foundation. Security Recommendations: <https://www.electronjs.org/docs/tutorial/security>

## 4.2 SSH Tunnels Listen on Any IPv4 Address

### Severity

High

### Affected Systems

- ethereum-setup

### Vulnerability Details

The SSH tunnels that are opened by the Stereum Launcher application listen on any IPv4 address `0.0.0.0`. This can be verified by an `ss` output:

```
$ ss -ltp
State      Local Address:Port      Peer Address:Port  Process
LISTEN     0.0.0.0:tpoxy      0.0.0.0:*          users:(("stereum-launche"))
LISTEN     0.0.0.0:8082       0.0.0.0:*          users:(("stereum-launche"))
LISTEN     0.0.0.0:8083       0.0.0.0:*          users:(("stereum-launche"))
[...]
```

If the client, where the Stereum Launcher is installed, has no firewall configured, everyone in the same network can access the services.

This can, for example, be done by entering the IP address of the client and port of the SSH tunnel in a web browser to access the Stereum UI:

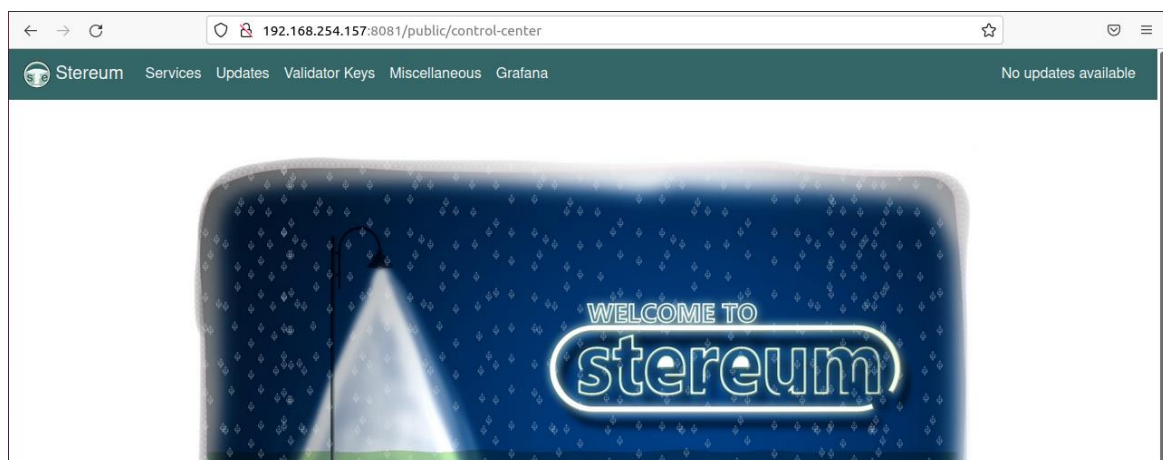


Figure 5: Accessing the web interface from another PC

### Countermeasures

The sockets must be configured to only listen on `localhost` (`127.0.0.1`) to ensure that no external entity can access the services.

- ✓ This vulnerability has already been fixed. The service is now listening on localhost.

## 4.3 Path Traversal

### Severity

High

### Affected Systems

- ethereum2-control-center-web

### Vulnerability Details

Due to an insufficient validation of a file name input, arbitrary files on the server can be accessed.

#### What Is a Path Traversal Vulnerability?

In a path traversal attack, an attacker attempts to manipulate a file name input parameter so that a file outside the permitted location is accessed. This is possible because the input value is directly used to construct the path to a file that is accessed in the application.

The following example describes such a scenario in PHP. Here, a page is included using `include` using the value of an HTTP GET parameter named `page`.

```
// index.php
include('./includes/pages/' . $_GET['page']);
```

An attacker can use the string `../` to change to directories above the permitted one. In the following example attack, the attacker accesses a configuration file that might contain credentials.

```
https://www.example.org/index.php?page=../../config/.env
```

The attacker can only access files which the current operating system process (e.g., the web server) has access to.

#### Specific Finding in This Application

The API endpoint in `ethereum2-control-center-web` receives requests from `electron-launcher`, for example:

```
POST /api/setup/start HTTP/1.1
Host: localhost:8081
Cache-Control: max-age=0
Content-Length: 97

{
  "inventory": "inventory.yaml",
  "playbook": "restart-host.yaml",
  "extra_vars": {},
  "extraVars": {}
}
```

It is possible to change the attribute "playbook" to another file, for example `../../../../../../../../etc/shadow` and retrieve a part of the file's content:

```
HTTP/1.1 500 Internal Server Error
date: Wed, 22 Sep 2021 10:58:47 GMT
server: uvicorn
content-length: 330
content-type: application/json
Connection: close

{
  "detail": "A playbook must be a list of plays, got a <class
'ansible.parsing.yaml.objects.AnsibleMapping'> instead\n\nThe error
appears to be in '/etc/shadow': line 1, column 1, but may\nbe elsewhere
in the file depending on the exact syntax problem.\n\nThe offending line
appears to be:\n\nroot:*:18725:0:99999:7:::\n^ here\n"
}
```

The error message contains a line of the shadow file, which also proves that the web server has root permissions (see 4.9) to access the file.

## Countermeasures

Input values must not be used to construct file paths without strict validation. Ideally, the file path is not passed directly, but indirectly via an ID to the file, and the concrete file path is stored, e.g., in a database.

If this is not possible, we recommend a strict validation of the input as a minimum. This includes the normalization (canonicalization) of the path and only then the validation that the resulting path points to the allowed directory. This method, however, bears some room for errors.

✓ This vulnerability has already been fixed. The application returns a generic error message

## References

[1] OWASP Web Security Testing Guide. Testing Directory Traversal File Include:  
[https://owasp.org/www-project-web-security-testing-guide/v41/4-Web\\_Application\\_Security\\_Testing/05-Authorization\\_Testing/01-Testing\\_Directory\\_Traversal\\_File\\_Include.html](https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/05-Authorization_Testing/01-Testing_Directory_Traversal_File_Include.html)

## 4.4 Links in Electron App Should Be Opened in Native Application

### Severity

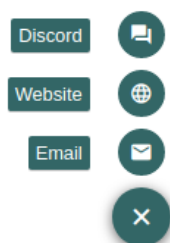
High

### Affected Systems

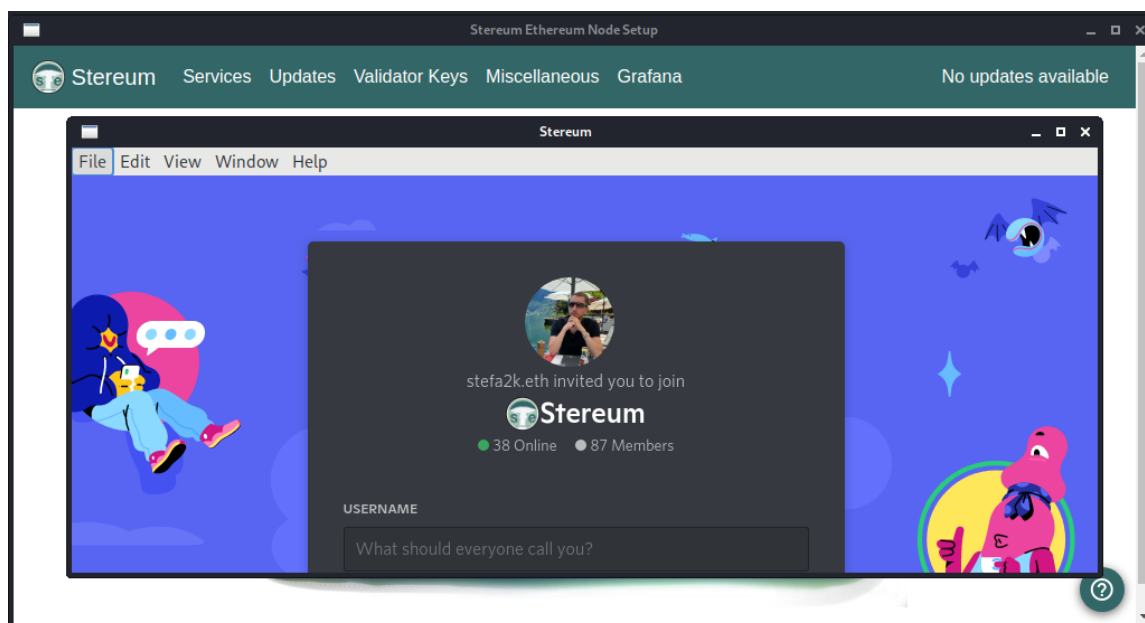
- ethereum2-control-center-web

### Vulnerability Details

To reduce the attack surface of the electron application, external links should be opened in the systems default (native) application. The electron application contains links to external resources:



These links get opened in a new electron window, as shown here by clicking on the "Discord" link:



**Figure 6: Discord website in Electron Launcher**

A malicious person could therefore trick the user into clicking a malicious link, for example, on the project's discord page, which contains executable electron code (see 4.1).

## Countermeasures

To reduce the attack surface of the electron application, every external link should be opened in the system's default application, in this case the native default browser.

When opening resources in native applications, it should be noted that it is important to only open static protocol URIs and no user-controlled data [1].

---

## References

[1] OpenJS Foundation. Security Recommendations: <https://www.electronjs.org/docs/tutorial/security#15-do-not-use-openexternal-with-untrusted-content>



## 4.5 Firewall Not Configured

### Severity

Medium

### Affected Systems

- ethereum2-ansible

### Vulnerability Details

The status of the firewall UFW on the server node is the following:

```
# ufw status verbose
Status: active
Logging: on (low)
Default: allow (incoming), allow (outgoing), deny (routed)
New profiles: skip
```

This means that no rule is set and therefore the default rule is applied to the packets which allows all incoming traffic. Restricting incoming traffic on a firewall is a common task in cyber security to reduce the attack surface and should also be applied to the server node. Also, the Eth2 validator checklist [1] recommends setting up a firewall.

### Countermeasures

Only the needed external ports for the application to function correctly should be exposed to the public with firewall rules, all other ports should be denied by the firewall.

The default rule should be `deny (incoming)`.

- ✓ This vulnerability has already been fixed. The firewall's default rule is `deny (incoming)` and rules for specific services exist now.

### References

- [1] Ethereum Foundation. Eth2 validator checklist:  
<https://launchpad.ethereum.org/en/checklist>

## 4.6 No Authentication Between Electron Client and Web Server

### Severity

Medium

### Affected Systems

- ethereum2-control-center-web

### Vulnerability Details

The web server does not require authentication. Even though it is only accessible on localhost, authentication should be required, because else all users and programs on the system can access the web server (e.g., on a shared Windows instance) and make requests to it.

For example, without authentication, this request can be performed, which restarts the server:

```
POST /api/setup/start HTTP/1.1
Host: localhost:8081
Content-Length: 92
Origin: http://localhost:8081
Connection: close

{"inventory":"inventory.yaml","playbook":"restart-
host.yaml","extra_vars":{},"extraVars":{}}
```

### Countermeasures

We recommend generating a random API Key on each connection which is then passed in an HTTP header. This way the authentication would be completely transparent for the user, they would only have to enter the SSH credentials.

Not only the `/api/setup/start` endpoint, but also the `/api/setup/status` endpoint should be covered by the authentication.

With that improvement, other users or malicious programs are not able to communicate with the webserver without knowing the API key.

## 4.7 Secrets Checked into Repository

### Severity

Medium

---

### Affected Systems

- ethereum2-docker-compose
- 

### Vulnerability Details

At least two private keys could be found checked into the repository:

1. <https://github.com/stereum-dev/ethereum2-docker-compose/tree/master/config/dirk/certs>
2. <https://github.com/stereum-dev/ethereum2-docker-compose/tree/master/config/vouch/certs>

Those keys must be considered compromised. They are used for authentication between *dirk* and *vouch*.

---

### Countermeasures

Those private keys should be removed and not used anymore. In the future, those keys should be generated when the application is deployed. Otherwise, all installations of Stereum will effectively use the same public key pair.

## 4.8 SSH Password in Standard Output

### Severity

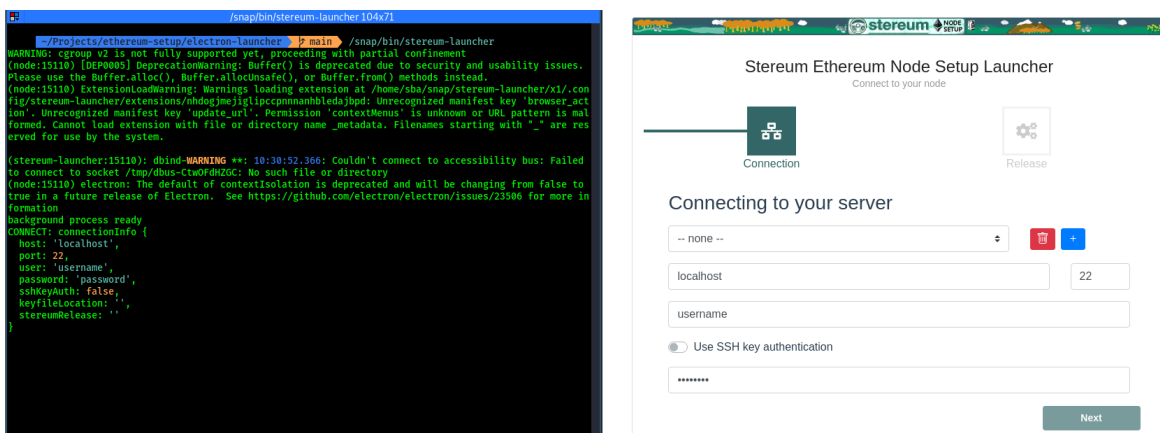
Medium

### Affected Systems

- Stereum Launcher

### Vulnerability Details

The Stereum launcher logs SSH connection information to `stdout`. This includes the clear-text password.



**Figure 7: SSH connection details are logged to stdout**

This vulnerability could, for example, enable somebody to see the screen of the victim, to steal the SSH credentials of the server, and to take over this server. This would lead to an attacker being able to steal the signing keys of the Ethereum staking node.

### Countermeasures

We recommend replacing the password with e.g., `****`.

- ✓ This vulnerability has already been fixed. Connection information is not logged anymore.

## 4.9 Docker Container Runs with Root Privileges

### Severity

Low

### Affected Systems

- ethereum2-docker-compose

### Vulnerability Details

One of the best practices while running Docker Container is to run processes with a non-root user. This is because if an attacker manages to break out of the application running as root in the container, they may gain root-user access on host. In addition, configuring container to run as unprivileged user is the best way to prevent privilege-escalation attacks.

The Docker container `stereum/control-center-web:1.6-169` runs with root privileges. This can be verified by executing the `id` command inside of the container:

```
# docker ps
CONTAINER ID   IMAGE                                [...]
6f50c6971694  stereum/control-center-web:1.6-169  [...]

# docker exec -it 6f50c6971694 id
uid=0(root) gid=0(root) groups=0(root)
```

Due this configuration, it was possible to use a path traversal vulnerability (see 4.3) to read a part of the Linux "shadow" file.

### Countermeasures

We recommend running the processes in the container as non-root user, as recommended in the documentation [1].

### References

- [1] Docker. Best practices for writing Dockerfiles: [https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/#user](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#user)

## 4.10 Reduce Root Account Usage

### Severity

Low

### Affected Systems

- ethereum2-ansible
- ethereum-setup

### Vulnerability Details

Currently, the root account is used for many commands. For this to work, the `sudoers` file must be changed, so that every command can be executed without entering a password. This is not compliant with the principle of least privilege.

Ansible scripts that are executed as root (`become: yes`) are at least the following:

```
ethereum2-ansible/export-config.yaml
ethereum2-ansible/finish-update.yaml
ethereum2-ansible/list-validator-accounts.yaml
ethereum2-ansible/stop-and-update.yaml
ethereum2-ansible/update-check.yaml
ethereum2-ansible/roles/check-imported-keys-teku/tasks/main.yaml
ethereum2-ansible/roles/check-keys-password-teku/tasks/main.yaml
ethereum2-ansible/roles/configure-timesyncd/tasks/main.yaml
ethereum2-ansible/roles/delete-validator-keys-lighthouse/tasks/main.yaml
ethereum2-ansible/roles/delete-validator-keys-lodestar/tasks/main.yaml
ethereum2-ansible/roles/delete-validator-keys-nimbus/tasks/main.yaml
ethereum2-ansible/roles/delete-validator-keys-prysm/tasks/main.yaml
ethereum2-ansible/roles/delete-validator-keys-teku/tasks/main.yaml
ethereum2-ansible/roles/exit-validator-keys-lighthouse/tasks/main.yaml
ethereum2-ansible/roles/exit-validator-keys-teku/tasks/main.yaml
ethereum2-ansible/roles/export-config-lighthouse/tasks/main.yaml
ethereum2-ansible/roles/export-config-lodestar/tasks/main.yaml
ethereum2-ansible/roles/export-config-nimbus/tasks/main.yaml
ethereum2-ansible/roles/export-config-prysm/tasks/main.yaml
ethereum2-ansible/roles/export-config-teku/tasks/main.yaml
ethereum2-ansible/roles/git-checkout-tag/tasks/main.yaml
ethereum2-ansible/roles/git-update-fetch-merge/tasks/main.yaml
ethereum2-ansible/roles/import-config/tasks/main.yaml
ethereum2-ansible/roles/import-validator-keys-lighthouse/tasks/main.yaml
ethereum2-ansible/roles/import-validator-keys-lodestar/tasks/main.yaml
ethereum2-ansible/roles/import-validator-keys-
multiclient/tasks/main.yaml
ethereum2-ansible/roles/import-validator-keys-nimbus/tasks/main.yaml
ethereum2-ansible/roles/import-validator-keys-prysm/tasks/main.yaml
ethereum2-ansible/roles/import-validator-keys-teku/tasks/main.yaml
ethereum2-ansible/roles/install-e2a/tasks/main.yaml
ethereum2-ansible/roles/install-e2ccc/tasks/main.yaml
ethereum2-ansible/roles/install-e2dc/tasks/main.yaml
ethereum2-ansible/roles/list-validator-keys-lighthouse/tasks/main.yaml
ethereum2-ansible/roles/list-validator-keys-nimbus/tasks/main.yaml
```

```
ethereum2-ansible/roles/list-validator-keys-teku/tasks/main.yaml
ethereum2-ansible/roles/load-blockchain-db/tasks/restore-blockchain-
db.yaml
ethereum2-ansible/roles/remove-tmp-password-prysm/tasks/main.yaml
ethereum2-ansible/roles/send-blockchain-db/tasks/main.yaml
ethereum2-ansible/roles/set-docker-tag/tasks/main.yaml
ethereum2-ansible/roles/set-graffiti/tasks/main.yaml
ethereum2-ansible/roles/write-config/tasks/write-to-file.yaml
```

## Countermeasures

The principle of least privilege should be followed where possible. We recommend installing software to user directories instead of system directories, to allow them to be installed and configured without root privileges. This would allow to only allow specific commands in the `sudoers` file. We recommend against telling users to configure their systems in an insecure way.

## 4.11 Missing Integrity Checks for Third-Party Dependencies

### Severity

Low

### Affected Systems

- ethereum-setup
- ethereum2-ansible
- ethereum2-control-center-web
- ethereum2-docker-compose

### Vulnerability Details

Third-party dependencies can be dangerous, because those are implicitly trusted by the software. If the application automatically includes new versions of the dependency, a compromised or rouge third-party developer could easily inject malicious code in the dependency, which then in turn gets included in the application. This could be used by an attacker to target users of the Stereum application.

### Countermeasures

The first measure which should be taken is to keep the number of third-party dependencies to a minimum. Only dependencies that are really needed should be included, and it should be verified that those dependencies are secure and can be trusted. We recommend having a central list of all third-party dependencies to have an overview over all of them.

Secondly, we recommend to always verify the integrity of third-party dependencies. This protects against MITM attacks and against manipulation of code. This measure will ensure that users will always have exactly the same code, as the code that was tested.

### Docker Implementation

Docker allows to specify the SHA256 hash of a docker image. We recommend adding this hash to every Docker image [1].

### pip Implementation

pip also allows to specify a hash in the `requirements.txt` [2]. All the dependencies are already pinned to a specific version. Therefore, it would not be a lot of work to also add the hashes of all requirements.

### Node Packages

NPM by default adds the hashes of the dependencies to the `package-lock.json` file. No additional actions need to be taken here.

### APT and DNF

The two Linux package managers are also used in some of the scripts. APT and DNF enable package signing by default, thus the attack would be more complicated. We think, that considering the circumstances, this risk can be accepted.



## References

- [1] Docker. docker pull: <https://docs.docker.com/engine/reference/command-line/pull/#pull-an-image-by-digest-immutable-identifier>
- [2] pip. pip install: [https://pip.pypa.io/en/stable/cli/pip\\_install/#hash-checking-mode](https://pip.pypa.io/en/stable/cli/pip_install/#hash-checking-mode)

## 4.12 Password Stored as SHA512

### Severity

Info

### Affected Systems

- ethereum-setup

### Vulnerability Details

The password of the `stereum` user is auto generated by Ansible and then stored as a SHA512 hash. SHA512 is not meant for storing passwords, it is a hash algorithm which is optimized for performance, unlike for example argon2 or bcrypt.

This is defined here:

```
$ cat ./roles/stereum-base/tasks/setup-users.yml
---
- name: Add stereum user
  user:
    name: "{{ ubuntu_common_stereum_user_name }}"
    password: "{{ ubuntu_common_stereum_password |
password_hash('sha512') }}"
    shell: /bin/bash
    generate_ssh_key: yes
    ssh_key_type: rsa
    ssh_key_bits: 4096
    ssh_key_file: .ssh/id_rsa
    force: no
```

### Countermeasures

We recommend switching to bcrypt for password storage [1].

### References

- [1] auth0. Hashing in Action: Understanding bcrypt: <https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>

## 4.13 Use of RSA Public-Key Algorithm for SSH

### Severity

Info

### Affected Systems

- ethereum-setup

### Vulnerability Details

The current used public-key algorithm RSA which does not provide Perfect Forward Secrecy. The use of the modern public-key signature algorithm Ed25519 is recommended.

### Countermeasures

The more modern Ed25519 has several advantages over the currently used 4096-bit RSA public-key algorithm. First and foremost, it provides Perfect Forward Secrecy, which makes it impossible for an attacker to decrypt previously recorded traffic if he or she breaks the Ed25519 key today.

Also, it is also a very fast signature algorithm, does not require random input, is resilient to hash-function collisions, immune to cache-timing and side-channel attacks that rely on leakage of information through the branch-prediction unit. [1]

✓ This vulnerability has already been fixed, Ed25519 is now in use.

### References

[1] Peter Ruppel. Ed25519 for SSH: <https://blog.peterruppel.de/ed25519-for-ssh/>

## 4.14 Code Pipeline Should Include More Checks

### Severity

Info

### Affected Systems

- ethereum-setup
- ethereum2-ansible
- ethereum2-control-center-web
- ethereum2-docker-compose

### Vulnerability Details

We recommend improving the code pipeline to include more checks on known vulnerabilities and outdated versions. A code pipeline is an easy way to detect easy to find vulnerabilities and regularly check whether third party dependencies are still up to date. GitHub offers code pipelines via the feature GitHub Actions [1].

### Countermeasures

We recommend at least the following checks:

#### ShellCheck

ShellCheck [2] is a simple static analysis tool for shell scripts. We recommend fixing all ShellChecks errors and implement a ShellCheck pipeline in all repositories that contain ShellScripts.

The following GitHub Action can be used to automatically run ShellCheck on new commits:

<https://github.com/marketplace/actions/shellcheck>

- ✓ This check was already implemented during the test. It is now in action for the following Git repositories: ethereum2-docker-compose and ethereum2-ansible

#### Kics

Kics [3] is a solution for static analysis of Infrastructure as Code. In the Stereum project it can be used for a variety of security checks: e.g., on: Ansible, Dockerfiles, ...

The following GitHub Action can be used to automatically run kics on new commits:

<https://github.com/marketplace/actions/kics-github-action>

#### Outdated Python Dependencies

The Stereum project uses Python in various places. The Python dependencies should be regularly checked for vulnerabilities and outdated versions. This can for example be implemented using pip [4], which can print outdated version. Or with specialized solutions like safety [5]. Safety can be for example be implemented with this GitHub Action:

<https://github.com/marketplace/actions/python-safety-check>

## OWASP Dependency-Check

OWASP Dependency-Check [6] is a powerful tool to check for vulnerabilities in third party dependencies. We recommend implementing this check and executing it regularly (e.g., at least once a week) to be notified about newly found vulnerabilities in third party dependencies.

## Node Packages

We recommend to also check for outdated Node packages regularly. This can for example be done with the built-in `NPM audit` command.

## General Recommendations

These checks should pass successfully for at least every commit to the `main/master` branches. If there are less developed repositories with a long timespan between commits, we recommend to also execute the checks regularly (e.g., one a week) to be able to identify newly found vulnerabilities in third party dependencies.

---

## References

- [1] GitHub. GitHub Actions: <https://docs.github.com/en/actions>
- [2] koalaman. ShellCheck - A shell script static analysis tool: <https://github.com/koalaman/shellcheck>
- [3] kikcs. keeping infrastructure as code secure: <https://kics.io/>
- [4] StackOverflow. Find outdated/updatable pip packages: <https://superuser.com/questions/259474/find-outdated-updatable-pip-packages>
- [5] pyupio. safety: <https://github.com/pyupio/safety>
- [6] jeremylong. Dependency-Check: <https://github.com/jeremylong/DependencyCheck>
- [7] NPM. npm-audit: <https://docs.npmjs.com/cli/v7/commands/npm-audit>

## 4.15 Do Not Use GPG

### Severity

Info

### Affected Systems

- ethereum2-ansible

### Vulnerability Details

The project currently uses GPG to encrypt files. While this is not a security vulnerability per se, we do not recommend using GPG for new projects, because there are better more modern solutions [1]. GPG is used in the following scripts:

```
stereum-dev/ethereum2-ansible/roles/export-config-prysm/tasks/main.yaml:
1 ---
2 - name: Export config - ethereum2.yaml file
3:   shell: echo "{{ export_config_password | quote }}" | gpg -c --
output /tmp/exported-config/exported_config.gpg --batch --yes --
passphrase-fd 0 ethereum2.yaml
4   args:
5     chdir: "/etc/stereum/"
```

```
stereum-dev/ethereum2-ansible/roles/export-config-
nimbus/tasks/main.yaml:
1 ---
2 - name: Export config - ethereum2.yaml file
3:   shell: echo "{{ export_config_password | quote }}" | gpg -c --
output /tmp/exported-config/exported_config.gpg --batch --yes --
passphrase-fd 0 ethereum2.yaml
4   args:
5     chdir: "/etc/stereum/"
```

```
stereum-dev/ethereum2-ansible/roles/import-config/tasks/main.yaml:
13
14 - name: Decrypt configuration file
15:   shell: echo "{{ exported_config_password | quote }}" | gpg -d -
-output /tmp/exported-config/ethereum2.yaml --batch --yes --passphrase-
fd 0 /tmp/exported-config/exported_config.gpg
16   args:
17     chdir: "/tmp/exported-config"
```

```
stereum-dev/ethereum2-ansible/roles/export-config-
lodestar/tasks/main.yaml:
1 ---
2 - name: Export config - ethereum2.yaml file
3:   shell: echo "{{ export_config_password | quote }}" | gpg -c --
output /tmp/exported-config/exported_config.gpg --batch --yes --
passphrase-fd 0 ethereum2.yaml
4   args:
5     chdir: "/etc/stereum/"
```

```
stereum-dev/ethereum2-ansible/roles/export-config-  
lighthouse/tasks/main.yaml:  
1 ---  
2 - name: Export config - ethereum2.yaml file  
3:   shell: echo "{{ export_config_password | quote }}" | gpg -c --  
output /tmp/exported-config/exported_config.gpg --batch --yes --  
passphrase-fd 0 ethereum2.yaml  
4   args:  
5     chdir: "/etc/stereum/"  
  
stereum-dev/ethereum2-ansible/roles/export-config-teku/tasks/main.yaml:  
1 ---  
2 - name: Export config - ethereum2.yaml file  
3:   shell: echo "{{ export_config_password | quote }}" | gpg -c --  
output /tmp/exported-config/exported_config.gpg --batch --yes --  
passphrase-fd 0 ethereum2.yaml  
4   args:  
5     chdir: "/etc/stereum/"
```

## Countermeasures

We recommend to for example use age [2].

## References

- [1] Manish Gehlot. Age - the modern alternative to GPG: <https://nixfaq.org/2021/01/age-the-modern-alternative-to-gpg.html>
- [2] FiloSottile. age: <https://github.com/FiloSottile/age>

## 5 Appendix

### 5.1 List of Figures

Figure 1: Severity Distribution .....	6
Figure 2: Screenshot of the Stereum application.....	7
Figure 3: Injected script code in "index.html" .....	9
Figure 4: "sleep 100" gets executed on the client .....	9
Figure 5: Accessing the web interface from another PC .....	11
Figure 6: Discord website in Electron Launcher.....	14
Figure 7: SSH connection details are logged to stdout.....	19

### 5.2 List of Tables

Table 1: Vulnerabilities Overview .....	6
---	---