EthStaker Deposit CLI

Security Assessment

December 13, 2024

Prepared for:
Rémy Roy
EthStaker

Prepared by: **Alexander Remie, Bo Henderson, Sam Alws, and Emilio López**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mike Shelton, Project Manager
mike.shelton@trailofbits.com

The following engineering directors were associated with this project:

Keith Hoodlet, Engineering Director, Application Security
keith.hoodlet@trailofbits.com

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Bo Henderson, Consultant
bo.henderson@trailofbits.com

Sam Alws, Consultant
sam.alws@trailofbits.com

Emilio López, Consultant
emilio.lopez@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 24, 2024	Pre-project kickoff call
October 8, 2024	Delivery of report draft; readout meeting
October 28, 2024	Delivery of final comprehensive report
December 13, 2024	Delivery of final comprehensive report with fix review

Executive Summary

Engagement Overview

EthStaker engaged Trail of Bits to review the security of `ethstaker-deposit-cli`, its deposit command-line tool for creating keys and files required for staking in Ethereum. It is a fork of Ethereum's `staking-deposit-cli`, with more functionality and improved performance.

A team of four consultants conducted the review from September 30 to October 4, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on preventing the mnemonic or other key material from leaking and the overall security of the development and release processes. With full access to source code and documentation, we performed static and dynamic testing of the EthStaker deposit command-line tool, using automated and manual processes.

Observations and Impact

Our review of the EthStaker deposit command-line tool uncovered several issues related to encryption, key and supply chain management, and the user experience, most of which are highly difficult to exploit. Some of these issues, if exploited, could lead to users inadvertently exposing the generated keys to a third party.

In particular, we found two issues related to the project supply chain and release process: the automated build system uses unversioned and unpinned components (TOB-ETHSTAKER-1), which may become compromised, and it signs its releases with GPG (TOB-ETHSTAKER-2), which can be confusing for users.

On the encryption and key management front, we found that the tool does not correctly randomize salts and IVs (TOB-ETHSTAKER-6), and it fails to properly clear out mnemonics from the terminal scrollback buffer (TOB-ETHSTAKER-7). The tool also potentially exposes keys while storing them on disk (TOB-ETHSTAKER-3). The clipboard cleanup process may also lead to the inadvertent loss of the mnemonic if the user is distracted (TOB-ETHSTAKER-5).

All in all, the codebase is well organized and significantly tested, which aided our review. However, some of the newer features could benefit from more thorough verification. Further manual testing could have discovered the lack of scrollback clearing (TOB-ETHSTAKER-7); additionally, we identified incompatibilities in the staking launchpad, so more manual testing should be performed to ensure compatibility.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that EthStaker take the following steps prior to encouraging public use:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **On POSIX machines, save keystore files with 400 permissions, rather than 440.** This would disallow users in the same group from accessing the files.
- **Use automated tooling such as Dependabot to keep dependencies up to date.** This would allow EthStaker to more quickly update dependencies in response to security fixes, and automate away some of the manual labor that developers currently have to do themselves.
- **Increase the automation in the binary release process.** The current process calls for a manual download of artifacts and a re-upload of said assets when preparing the GitHub release. It should be possible to automate this, so that when a tag is pushed to the repository, a draft release is prepared and any required artifacts are built and attached to it. This also reduces the risk of incorrect or malicious artifacts getting uploaded.
- **Lock the version of third-party dependencies whenever possible.** There are instances in workflows, apart from the use of third-party actions and Dockerfile base images (TOB-ETHSTAKER-1), where software gets installed in an ad-hoc manner and without version pinning, which opens the system to the possibility of supply chain compromise through changes upstream.
- **Perform manual end-to-end tests.** Some aspects of this tool, such as the issue underlying TOB-ETHSTAKER-7, do not lend themselves to automated testing. Manually test the tool across different operating systems, terminal emulators, and staking scenarios. In addition, manually test that the files will be accepted and interpreted correctly by the staking launchpad provided by the Ethereum Foundation. Incompatibilities are present and will require further investigation to resolve.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	2
Low	1
Informational	2
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Configuration	2
Data Exposure	1
Data Validation	1
Undefined Behavior	2

Project Goals

The engagement was scoped to provide a security assessment of the EthStaker Deposit CLI. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are there any error-prone steps that could cause an honest user to leak their mnemonic, password, or other sensitive data?
- Has the CI and overall development process been set up securely?
- Could any errors arise that cause critical data to be lost?
- Could an attacker subvert filesystem path searches in a way that could lead to a crash or the use of an incorrect translation file?
- Is the right cryptography used, and is it used correctly?

Project Targets

The engagement involved a review and testing of the following target.

ethstaker-deposit-cli

Repository	https://github.com/eth-educators/ethstaker-deposit-cli
Version	`66054f57382200a28478dd755b00e1bf49d44e48` (tag `v0.2.1`)
Type	Python
Platform	Windows, macOS, Linux, Docker

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- A full manual review of EthStaker's Python code located in the `ethstaker_deposit` directory
- Manual testing of deposit CLI functionalities
    - Although we deployed a Holešky execution/consensus/validator stack, it finished syncing near the end of our engagement, and we later discovered incompatibilities in the Ethereum launchpad. Therefore, our manual testing focused on the generation of mnemonics, deposit files, and exit transactions rather than on their correct interpretation by the network and staking launchpad.
- A review of the build and release processes, with a focus on supply chain risks

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The `generate-bls-to-execution-change-keystore` feature was not manually tested nor was a review of its source code considered high priority. This feature depends on protocol-level changes to Ethereum that have not been finalized. We recommend conducting a code review of this feature after the required updates have been implemented and the upgrade fork has been scheduled.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
actionlint	A static checker for GitHub Actions workflow files	Appendix C
bandit	A static analysis tool designed to find common security issues in Python code	Appendix C
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	Appendix C
CodeQL	A code analysis engine developed by GitHub to automate security checks	Appendix C

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The deposit tool does not perform much arithmetic. We did not find any issues with any of EthStaker's arithmetic code.	**Satisfactory**
Auditing	When a problem occurs, users can troubleshoot it using command-line messages, which is sufficient.	**Satisfactory**
Authentication / Access Controls	Keystore files are secured with a password. On POSIX machines, keystores are saved into files with 440 permissions. We found that keystore files may be accessible to a local attacker briefly while they are being saved (TOB-ETHSTAKER-3). In addition, EthStaker should switch to 400 permissions by default so users in the same group cannot gain access to the file.	**Moderate**
Complexity Management	The EthStaker codebase is well organized and readable.	**Satisfactory**
Cryptography and Key Management	EthStaker outsources its cryptographic calculations to libraries when possible. Otherwise, it follows EIP specs. EthStaker uses the GPG program for release signing, which is considered an insecure practice (TOB-ETHSTAKER-2). Additionally, we found that certain cryptographically relevant function parameters are randomized at program initialization rather than when the function is called, leading to significantly weakened encryption on keystores (TOB-ETHSTAKER-6).	**Weak**

	Key management is generally done in a spec-compliant way. Keys are derived from randomly generated data represented as a mnemonic, and keystores are secured with a password; management of the mnemonic and the password is left to the user. A minimum password length of 12 is enforced. On POSIX machines, keystores are saved into files with 440 permissions. We found that keystore files may be accessible to a local attacker briefly while they are being saved (TOB-ETHSTAKER-3).	
Configuration	The project contains configuration files used to compile documentation and produce binaries as part of automated CI builds on GitHub Actions. Many of the build steps are duplicated across different operating system targets. Some of these workflows also use unpinned third-party actions (TOB-ETHSTAKER-1). The release process could benefit from further automation.	**Moderate**
Data Handling	We did not identify any issues with EthStaker's data validation on user inputs or files.	**Satisfactory**
Documentation	Most Python functions feature doc strings that describe their functionality, and sensitive logic is accompanied by in-line comments. High-level documentation is provided online and in the repository's `docs` folder; this includes installation and usage instructions along with clear warnings that describe the risks associated with certain actions.	**Satisfactory**
Maintenance	EthStaker pins its Python dependencies to version hashes. However, its GitHub Actions dependencies are not pinned (TOB-ETHSTAKER-1). EthStaker does not use any automation to update its dependencies; instead, they are updated manually. EthStaker should use a tool such as Dependabot to automatically update dependencies.	**Weak**
Memory Safety and Error Handling	Python is a memory safe language, so memory safety does not apply here. We did not find any issues with the way EthStaker performs error handling. Generally, errors result either in the user being re-prompted or in the CLI closing	**Satisfactory**

	gracefully and the error being reported to the user.	
Testing and Verification	The repository features thorough end-to-end unit tests for each command provided by the EthStaker deposit CLI tool. The average test coverage across all files was 95%, with many components having 100% coverage.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Use of unpinned third-party Docker image and actions on workflows	Configuration	**Medium**
2	Use of GPG for release signing and verification	Data Validation	**Informational**
3	Sensitive files are incorrectly assigned permissions and ownership	Access Controls	**High**
4	Error-prone path handling	Configuration	**Informational**
5	Emphasize critical warning regarding clipboard clearing	Undefined Behavior	**Medium**
6	Encryption function random parameters are set at program init	Undefined Behavior	**High**
7	Terminal buffer is not cleared on iTerm2	Data Exposure	**Low**

Detailed Findings

1. Use of unpinned third-party Docker image and actions on workflows

Severity: **Medium**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-ETHSTAKER-1

Target: `ethstaker-deposit-cli/.github/workflows/docs.yml`, `ethstaker-deposit-cli/.github/workflows/docker.yml`, `ethstaker-deposit-cli/.github/workflows/build.yml`, `Dockerfile`

Description

Several workflows in the repository directly use third-party actions, such as `peaceiris/actions-mdbook@v2`, `peaceiris/actions-gh-pages@v4`, `docker/login-action@v3`, `docker/metadata-action@v5`, `docker/build-push-action@v6`, and `crazy-max/ghaction-import-gpg@v6`. Some of these workflows have privileged access to secrets, such as the GPG private key and its passphrase.

```
- name: Import GPG key
  uses: crazy-max/ghaction-import-gpg@v6
  id: import-gpg-key
  with:
    gpg_private_key: ${{ secrets.GPG_PRIVATE_KEY }}
    passphrase: ${{ secrets.PASSPHRASE }}
    trust_level: 5
```

Figure 1.1: The `ci-build` workflow uses a third-party action (`ethstaker-deposit-cli/.github/workflows/build.yml#95–101`)

Git tags are malleable. This means that while, for example, `crazy-max/ghaction-import-gpg` is pinned to `v6`, the upstream may silently change the reference that `v6` points to. Attackers can make reference changes to include malicious software in an action, which may cause EthStaker's workflow to suddenly start executing malicious code.

GitHub's security hardening guidelines for third-party actions encourage developers to pin third-party actions to a full-length commit hash. Generally excluded from this are "official" actions under the `actions` organization; however, these other actions are not developed by GitHub.

The `Dockerfile` in the repository exhibits an analogous issue: the `FROM` line indicates an image tag name that may change on the server side (figure 1.2). To prevent these images from silently changing, these should be pinned with a SHA256 hash.

```
FROM python:3.12-slim-bookworm
```

Figure 1.2: The Dockerfile references a mutable tag (`ethstaker-deposit-cli/Dockerfile#1`)

Exploit Scenario

An attacker (or compromised maintainer) silently replaces the `v6` tag on the `crazy-max/ghaction-import-gpg` repository with a malicious action that leaks the GPG keys provided and injects malware into binaries. When the `ci-build` workflow is run, the GPG key is leaked and malicious binaries are published.

Recommendations

Short term, replace the current version tag references used in the workflow for third-party actions with full-length commit hashes. Similarly, pin Docker base images by their full hashes.

Long term, use Dependabot's support for Github Actions to keep third-party action commit hashes up to date, complemented with maintainer reviews to ensure their safety. Incorporate static analysis tools such as Semgrep into the CI pipeline to detect issues earlier on.

2. Use of GPG for release signing and verification

Severity: **Informational**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-ETHSTAKER-2
Target: `ethstaker-deposit-cli/.github/workflows/docker.yml`, `ethstaker-deposit-cli/.github/workflows/build.yml`	

Description

The EthStaker deposit CLI tool currently uses GPG for signing releases. While this was an improvement over the previous unsigned releases, GPG is not considered a good tool for verifying software releases in new projects. GPG has a history of cryptographic and memory safety vulnerabilities, poor default configurations, and a challenging user experience that makes it difficult for users to verify signatures correctly. Additionally, it requires maintaining and protecting a signing key that may become compromised (e.g., via TOB-ETHSTAKER-1).

For example, the current verification instructions suggest that the user import the key to be used for verification from a public keyserver by looking it up by email. This identifier may not be unique (i.e., multiple keys may be published for a single email), and it could lead to a user mistakenly importing an attacker-controlled key.

```
2. Make sure you have the `edc-security@ethstaker.cc` public key by running
```sh
gpg --keyserver keys.openpgp.org --search-keys 'edc-security@ethstaker.cc'
```
```

Figure 2.1: Part of the verification instructions in the quick setup guide (`ethstaker-deposit-cli/docs/src/quick_setup.md#32–35`)

EthStaker has already implemented artifact attestations to establish provenance for builds produced on GitHub Actions, which should be a suitable replacement to release signing. However, documentation showcases only how to validate GPG signatures.

Exploit Scenario

An attacker who has compromised the release distribution channel publishes a malicious release. A user of the EthStaker deposit CLI tool unknowingly downloads the malicious release and attempts to verify its GPG signature. Due to the complexity of GPG usage, the user makes a mistake in the verification process and believes they have successfully verified the signature using EthStaker's key when they have not. The user proceeds to execute the malicious release.

Recommendations

Update the documentation to cover how to verify releases using sigstore. Cover all relevant use cases, including offline verification and verification of Docker containers. Consider deprecating GPG release signatures in favor of sigstore verification. If providing verification through self-held keys is a desirable property for the EthStaker team, consider using a simpler signing solution, such as `minisign`. If GPG release signatures are not deprecated, adjust the documentation instructions to use a full key ID, and include proof that the EthStaker team controls said key (e.g., via posts on social networks or similar channels).

References

- GitHub: Using artifact attestations to establish provenance for builds
- GitHub: Verifying attestations offline

3. Sensitive files are incorrectly assigned permissions and ownership

Severity: **High**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-ETHSTAKER-3
Target: `ethstaker_deposit/utils/deposit.py`, `ethstaker_deposit/credentials.py`, `ethstaker_deposit/utils/exit_transaction.py`, `ethstaker_deposit/bls_to_execution_change_keystore.py`, `ethstaker_deposit/key_handling/keystore.py`	

Description

In various locations in the EthStaker codebase, a file is created, written with sensitive data, and then assigned permissions in the manner shown in figure 3.1. There are two problems with the current approach: it allows an attacker to read the file in the time after it has been written but before the permissions have been assigned, and it allows an attacker to create a file with himself as the owner before EthStaker writes to the path.

```
with open(filefolder, 'w') as f:
    f.write(self.as_json())
if os.name == 'posix':
    os.chmod(filefolder, int('440', 8))  # Read for owner & group
```

Figure 3.1: Example of incorrect file creation (`ethstaker_deposit/key_handling/keystore.py:95-102`)

Here are all the instances of this issue:

- `ethstaker_deposit/utils/deposit.py:7-10`
- `ethstaker_deposit/credentials.py:367-370`
- `ethstaker_deposit/utils/exit_transaction.py:58-61`
- `ethstaker_deposit/bls_to_execution_change_keystore.py:67-70`
- `ethstaker_deposit/key_handling/keystore.py:99-102`

Exploit Scenario 1

A user uses EthStaker to generate and save a keystore. A malicious user on an unprivileged account reads the keystore file after it is written but before its permissions are set, and uses the keystore information to steal the eth.

Exploit Scenario 2

A user uses EthStaker to generate and save a keystore. Before the keystore is written, a malicious user on an unprivileged account had already created a file with that path owned

by his own account, with permissions that allow EthStaker to write to the file. EthStaker writes to the file, and its data is now accessible to the malicious user.

In order to prevent the `os.chmod` call from causing an exception, the attacker would have to read the file and then call `chown` on it after the file is written to by EthStaker but before the `os.chmod` call, which may be difficult. Alternatively, this attack can be carried out on a Windows machine, where the `os.chmod` call does not take place, but due to Windows' permissioning system, this may be less viable.

Recommendations

Short term, change all the instances of this issue so that they use the `os.open` function, with the `O_EXCL` flag set, and a mode parameter of 440.

Long term, centralize file creation behavior by creating a utility function that creates and opens a file using the `os.open` function. Also, consider using 400 instead of 440 permissions.

4. Error-prone path handling

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-ETHSTAKER-4
Target: `ethstaker_deposit/utils/intl.py`	

Description

The `load_text` function looks at the stack to determine the absolute path to the current file and tries to modify it to find the internationalization text associated with the given file.

```
# Determine path to json text
file_path_list = os.path.normpath(file_path).split(os.path.sep)
rel_path_list = file_path_list[file_path_list.index('ethstaker_deposit') + 1:]
json_path = resource_path(os.path.join(INTL_CONTENT_PATH, lang, *rel_path_list))
```

Figure 4.1: The `load_text` function in `intl.py#L60-63`

`file_path_list.index('ethstaker_deposit')` gets the index of the first occurrence of `/ethstaker_deposit/` in the absolute path to this file. If the absolute path contains multiple instances of this directory name, the path we end up operating on will be incorrect.

Exploit Scenario

Alice has a username of `ethstaker_deposit`, which causes this function to treat her home directory as the root of its search for string translations. The translation files are unlikely to exist at this path, causing the CLI tool to fail.

Recommendations

Short term, determine the `rel_path_list` value from the last occurrence of the `ethstaker_deposit` string instead of from the first.

Long term, consider consolidating language translations into one mapping that does not rely on stack inspection and path manipulations such as gettext.

5. Emphasize critical warning regarding clipboard clearing

Severity: **Medium**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ETHSTAKER-5
Target: `ethstaker_deposit/cli/new_mnemonic.py`	

Description
A critical warning regarding the clipboard being cleared is stated halfway through one long, unformatted line of text that users are at a risk of missing.

```
Please type your mnemonic (separated by spaces) to confirm you have written it down.
You MUST write the mnemonic down, as the clipboard will be cleared after this step.
Note: you only need to enter the first 4 letters of each word if you'd prefer.

:
```

Figure 5.1: The mnemonic confirmation screen

Note that the clipboard is not cleared, but is replaced with a space. Some password managers, such as 1Password, display concealed passwords with a fixed length.

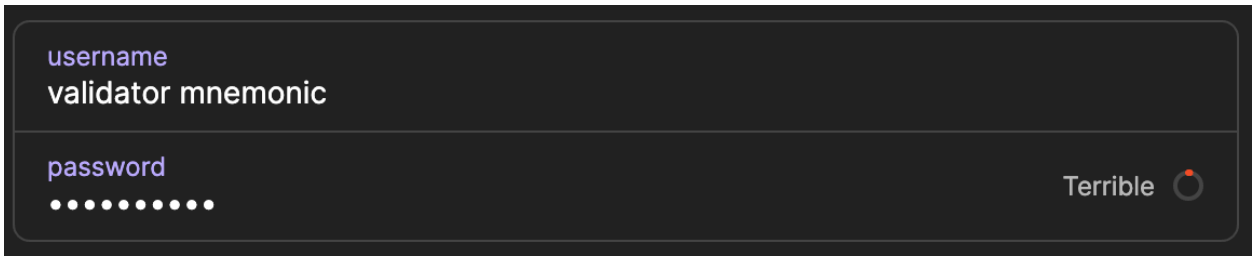

Figure 5.2: The 1Password view if the password field contains a single space

Exploit Scenario
Alice has acquired 32 ETH and performs multiple testnet deposits until she gets familiar with the process. When she performs her mainnet deposit, she knows what to do and proceeds through the flow quickly. She copies the mnemonic, pastes it into the confirmation screen and hits enter, pastes a space into her password manager, and quickly hits enter to save the entry. She then deposits 32 ETH, unknowingly burning her funds.

Recommendations
Short term, warn users that the clipboard will also be cleared during the first step, giving them another chance to read and comprehend it. Additionally, break the warning into multiple lines to improve readability.

Long term, consider adding a second confirmation screen after the user is requested to re-enter their mnemonic. This screen should have one warning that emphasizes that the clipboard is about to be cleared. Require the user to hit enter to confirm, or to type something like “clear my clipboard,” to ensure that they understand what is about to happen.

6. Encryption function random parameters are set at program init

Severity: **High**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ETHSTAKER-6
Target: `ethstaker_deposit/key_handling/keystore.py`	

Description

The `kdf_salt` and `aes_iv` default parameters of the `encrypt` method (figure 6.1) are chosen at program initialization, rather than when the function is called. This means that duplicate calls to the function will produce the same `kdf_salt` and `aes_iv` values unless they are manually specified in the function call.

This means, for example, that a call to `new-mnemonic` will produce multiple keystores with the same IV and salt values. This seriously weakens the encryption strength of these keystore files.

```
@classmethod
def encrypt(cls, *, secret: bytes, password: str, path: str = '',
            kdf_salt: bytes = randbits(256).to_bytes(32, 'big'),
            aes_iv: bytes = randbits(128).to_bytes(16, 'big')) -> 'Keystore':
    """
    Encrypt a secret (BLS SK) as an EIP 2335 Keystore.
    """
    keystore = cls()
    keystore.uuid = str(uuid4())
    keystore.crypto.kdf.params['salt'] = kdf_salt
    decryption_key = keystore.kdf(
        password=cls._process_password(password),
        **keystore.crypto.kdf.params
    )
    keystore.crypto.cipher.params['iv'] = aes_iv
    cipher = AES_128_CTR(key=decryption_key[:16], **keystore.crypto.cipher.params)
    keystore.crypto.cipher.message = cipher.encrypt(secret)
    keystore.crypto.checksum.message = SHA256(decryption_key[16:32] +
keystore.crypto.cipher.message)
    keystore.pubkey = bls.SkToPk(int.from_bytes(secret, 'big')).hex()
    keystore.path = path
    return keystore
```

Figure 6.1: `encrypt` method (`ethstaker_deposit/key_handling/keystore.py:129-149`)

Recommendations

Short term, set these default parameter values to None. Add `if` statements in the function body to check if the value is None, and if it is, assign a random value.

7. Terminal buffer is not cleared on iTerm2

Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-ETHSTAKER-7
Target: `ethstaker_deposit/utils/terminal.py`	

Description

The `tput reset` command does not clear the iTerm2 scrollback buffer, increasing the risk of sensitive data exposure.

```
def clear_terminal() -> None:
    ...

    if sys.platform == 'win32':
        # Special-case for asyncio pytest on Windows
        if os.getenv("IS_ASYNC_TEST") == "1":
            click.clear()
        elif shutil.which('clear'):
            subprocess.run(['clear'])
        else:
            subprocess.run('cls', shell=True)
    elif sys.platform == 'linux' or sys.platform == 'darwin':
        if shutil.which('tput'):
            subprocess.run(['tput', 'reset'], env=clean_env)
        elif shutil.which('reset'):
            subprocess.run(['reset'], env=clean_env)
        elif shutil.which('clear'):
            subprocess.run(['clear'], env=clean_env)
        else:
            click.clear()
    else:
        click.clear()
```

Figure 7.1: The second half of the `clear_terminal` function in `ethstaker_deposit/utils/terminal.py#L13-L39`

The `tput` command ships with many unix-like systems, including Linux and MacOS, so the highlighted checkers in figure 7.1 will rarely run any command other than `tput`. Note that the standalone `reset` and `clear` commands frequently call into `tput` under the hood.

Exploit Scenario

Alice generates a new mnemonic in her MacOS iTerm2 terminal. Her terminal scrolls down and hides her mnemonic after the confirmation step. She does not realize that her

mnemonic is still present, so she leaves her terminal session open. Some time later, an attacker accesses her computer, scrolls up, and gains access to her mnemonic.

Recommendations

Short term, replace the highlighted `elif` statements in figure 7.1 with an `if` to execute *all* of the clearing commands that are present on the system.

Long term, manually test terminal manipulation logic across multiple operating systems and multiple terminal emulators.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Automated Analysis Tool Configuration

This appendix describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, they detect certain categories of issues, such as memory leaks, misspecified format strings, and the use of unsafe APIs, with essentially perfect precision. We recommend periodically running these static analysis tools and reviewing their findings.

actionlint

We installed actionlint by following actionlint's quick start guide. We also installed its two external dependencies, shellcheck and pyflakes, using their corresponding installation guides.

After installing actionlint, we ran the following command to analyze the repository:

```
actionlint
```

Bandit

To install Bandit, we used `pip` by running `python3 -m pip install bandit`.

After installing Bandit, we used the following command to analyze the repository:

```
bandit -r repository-folder/
```

As this may result in many findings, it is possible to ignore some of the code by using the `-x` flag in Bandit. The following example ignores the test files, as well as the default Bandit ignore set:

```
bandit -r audit-ethstaker-deposit-cli/ -x
".svn,CVS,.bzr,.hg,.git,__pycache__,.tox,.eggs,*.egg,tests,*/test
_*.py"
```

Semgrep

To install Semgrep, we used `pip` by running `python3 -m pip install semgrep`.

To run Semgrep on the codebase, we ran the following command in the root directory containing the project (running multiple predefined rules simultaneously by providing multiple `--config` arguments):

```
semgrep --config "p/trailofbits" --config "p/ci" --config "p/python"
--config "p/security-audit" --metrics=off
```

We also used `semgrep-rules-manager` to fetch and run other third-party rules.

We recommend integrating Semgrep into the project's CI/CD pipeline. To thoroughly understand the Semgrep tool, refer to the Trail of Bits Testing Handbook, where we aim to streamline the use of Semgrep and improve security testing effectiveness. Also, consider doing the following:

- Limit results to error severity only by using the `--severity` ERROR flag.
- Focus first on rules with high confidence and medium- or high-impact metadata.
- Use the SARIF format (by using the `--sarif` Semgrep argument) with the SARIF Explorer for Visual Studio Code extension. This will make it easier to review the analysis results and drill down into specific issues to understand their impact and severity.

CodeQL

We installed CodeQL by following CodeQL's installation guide.

After installing CodeQL, we ran the following command to create the project database for the repository:

```
codeql database create ethstaker.db --language=python
```

We then ran the following command to query the database:

```
codeql database analyze ethstaker.db --format=sarif-latest
--output=codeql_res.sarif -- python-lgtm-full
python-security-and-quality python-security-experimental
```

For more information about CodeQL, refer to the CodeQL chapter of the Trail of Bits Testing Handbook.

D. Code Quality Recommendations

The following recommendations are not associated with any specific vulnerabilities. However, they will enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Review the use of `eval` in the build workflow.** The `eval` function is used in a subshell to obtain the first seven characters of the commit hash, but its use appears to be unnecessary.
- **Review and resolve `shellcheck` warnings in the CI workflows.** The `shellcheck` tool points out multiple warnings in the shell script snippets included in the CI workflows. Many of those are instances of potentially undesirable globbing and word splitting due to the lack of double quoting.
- **Review the return in the `__init__` function.** The `JITOption.__init__` function contains a return statement. The `__init__` function is used to initialize objects and not create them, and as such does not return a value.
- **Consider specifying the encoding to be used in `open(...)` calls.** The program opens multiple files in text mode (see TOB-ETHSTAKER-3 for examples). These open calls do not specify an encoding for these files. By default, `open(...)` uses device locale encodings, which may corrupt files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode.
- **Lock the version of test and linting tools.** Several unversioned test tools are installed in the runner workflow, such as `coverage` and `@prantlf/jsonlint`.

E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On December 2, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the EthStaker team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, all seven issues described in this report were resolved by the EthStaker team. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Use of unpinned third-party Docker image and actions on workflows	**Resolved**
2	Use of GPG for release signing and verification	**Resolved**
3	Sensitive files are incorrectly assigned permissions and ownership	**Resolved**
4	Error-prone path handling	**Resolved**
5	Emphasize critical warning regarding clipboard clearing	**Resolved**
6	Encryption function random parameters are set at program init	**Resolved**
7	Terminal buffer is not cleared on iTerm2	**Resolved**

Detailed Fix Review Results

TOB-ETHSTAKER-1: Use of unpinned third-party Docker image and actions on workflows
Resolved in PR 188, PR 193, and PR 198. PR 188 pins the dependency to the Python Docker image. PR 193 removes the dependency to the `crazy-max/ghaction-import-gpg` GitHub Action. PR 198 pins the remaining third-party GitHub Actions dependencies.

TOB-ETHSTAKER-2: Use of GPG for release signing and verification
Resolved in PR 193. This PR removes GPG signature generations from the `build.yml` GitHub workflow. Instead, the documentation has been updated to describe how to use the GitHub CLI tool to verify release attestations.

TOB-ETHSTAKER-3: Sensitive files are incorrectly assigned permissions and ownership
Resolved in PR 208. This PR creates a file opener that opens files using the `O_EXCL` flag and 400 permissions, and changes all of the code locations mentioned in the issue so that they use this opener.

TOB-ETHSTAKER-4: Error-prone path handling
Resolved in PR 211. This PR changes the code used to calculate the `rel_path_list` variable so that the last occurrence of `ethstaker_deposit` in the path is used, rather than the first.

TOB-ETHSTAKER-5: Emphasize critical warning regarding clipboard clearing
Resolved in PR 311. This PR changes the warning text so that it is clearer that the clipboard is about to be cleared. It also adds a confirmation dialog requiring the user to press any key to acknowledge the clipboard being cleared.

TOB-ETHSTAKER-6: Encryption function random parameters are set at program init
Resolved in commit 6a47170. This commit sets the default function parameter to None, and adds extra code in the function body to handle the case where the default parameter is used.

TOB-ETHSTAKER-7: Terminal buffer is not cleared on iTerm2
Resolved in PR 189 and PR 242. Instead of only running one of the available buffer clearing commands, PR 189 introduces a change that will run all available buffer clearing commands. PR 242 wraps these commands in a `for` loop that runs twice, since some terminals require two clear commands in a row to clear all text from the buffer.

F. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.