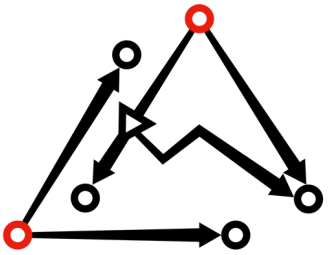

Least Authority

PRIVACY MATTERS

MetaMask Plugin System + LavaMoat

Security Audit Report

ConsenSys AG

Final Report Version: 4 March 2020

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Areas of Concern
- Findings
  - General Comments
  - Specific Issues & Suggestions
    - Issue A: [Plugin Beta] SES Realm Creation Enables the Error Stack
    - Issue B: [Plugin Beta] Restricted Method submitPassword
    - Issue C: [Plugin Beta] Method getState Returns Potentially Sensitive Data
    - Issue D: [Plugin Beta] Plugin State is Part of Main State
    - Issue E: [Plugin Beta] Bypass SES by Modifying global.process.env Properties
    - Issue F: [Plugin Beta] opts[requiredField] Will Return True if the Property is Declared but Undefined
    - Issue G: [LavaMoat] Prevent Access to __proto__ from deepGet
    - Issue H: [LavaMoat] Exported Factory Function Can Return Shared Object
    - Issue I: [LavaMoat] Code Injection via Label in wrapWithReturnCjsExports
    - Issue J: [LavaMoat] Child dependencies Can Access a Parent Module's Exports Before Harden is Applied
    - Suggestion 1: [Plugin Beta] Separate Logic for getSelectedAddress from getAccounts
    - Suggestion 2: [Plugin Beta] Avoid Using (foo in bar) Truth Checks
    - Suggestion 3: [LavaMoat] Detect Additional Primordials
    - Suggestion 4: [LavaMoat] Hide stacktraces
    - Suggestion 5: [LavaMoat] Stronger Magic Copy
- Recommendations
- About Least Authority
- Our Methodology

Overview

Background

ConsenSys AG has requested that Least Authority perform a security audit of MetaMask, a browser extension that enables interaction with applications built on Ethereum. MetaMask allows users to browse the web and interact with Ethereum applications, sign messages and transactions, and securely manage and store their private keys and assets.

The following components are in scope:

1. Plugin System
    a. SES-based plugin system
2. LavaMoat
    a. Browserify plugin system allowing the isolation of dependencies in Secure EcmaScript (SES) containers with the aim of removing the dangers of supply chain attacks (malicious code in the app dependency graph), ambient authority, and embodying the principle of least authority.

Project Dates

- **October 28 - November 18**: Code review *(Completed)*
- **November 21**: Delivery of Initial Audit Report *(Completed)*
- **February 24 - March 3:** Verification completed *(Completed)*
- **March 4:** Delivery of Final Audit Report *(Completed)*

Review Team

- Emery Hall, Security Researcher and Engineer
- Dominic Tarr, Security Researcher and Engineer
- Alexander Leitner, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the MetaMask Plugin System followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Plugin System
    - SES-based plugin system: https://github.com/MetaMask/metamask-plugin-beta/pull/77
- LavaMoat: https://github.com/LavaMoat/overview

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

```
metamask-plugin-beta@7d758d335279bd0d25e3a9c170fcf60709eb7828
```

```
lavamoat-browserify@9bd7fad6eddd54691caf55ee37a64b6f0bb1057a
```

For the verification, we examined the Git revision:

```
metamask-plugin-beta@0eaf8d282c2a06de5b7d8f18f4ca8a7a8f0f8218
```

```
lavamoat-browserify@0eaf8d282c2a06de5b7d8f18f4ca8a7a8f0f8218
```

All file references in this document use Unix-style paths relative to the project's root directory.

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component as well as secure interaction between the network components;
- Data privacy, data leaking, and information integrity;
- Key management implementation: secure private key storage and proper management of encryption and signing keys;
- Storing assets securely;
- Any attack that impacts funds, such as draining or manipulating of funds;
- Mismanagement of funds via transactions;
- Exposure of any critical information during user interactions with the blockchain and external libraries;
- General use of external libraries;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

MetaMask is exceptional and stands out in terms of secure architecture and paving the way for security best practices. We found the code to be comprehensible and robust and the underlying design to be clever and clearly thought out. Our concerns largely are characterized by areas where we felt that the code was not quite ready for production. We did discover some vulnerabilities in the LavaMoat code allowing execution outside of the secure container and noted several suggestions for further hardening the build system.

At the time of conducting our verification of the reported issues, the items we reported for the Plugin System remain unresolved, but are being tracked by the MetaMask team. MetaMask has stated their intent to address all of the outstanding issues prior to deploying the Plugin System into production.

Specific Issues & Suggestions

We list the issues we found in the code in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: [Plugin Beta] SES Realm Creation Enables the Error Stack	Unresolved
Issue B: [Plugin Beta] Restricted Method submitPassword	Unresolved

Issue C: [Plugin Beta] Method getState Returns Potentially Sensitive Data	Unresolved
Issue D: [Plugin Beta] Plugin State is Part of Main State	Unresolved
Issue E: [Plugin Beta] Bypass SES by Modifying global.process.env Properties	Unresolved
Issue F: [Plugin Beta] opts[requiredField] Will Return True if the Property is Declared but Undefined	Unresolved
Issue G: [LavaMoat] Prevent Access to __proto__ from deepGet	Resolved
Issue H: [LavaMoat] Exported Factory Function Can Return Shared Object	Unresolved
Issue I: [LavaMoat] Code Injection via Label in wrapWithReturnCjsExports	Resolved
Issue J: [LavaMoat] Child dependencies Can Access a Parent Module's Exports Before Harden is Applied	Unresolved
Suggestion 1: [Plugin Beta] Separate Logic for getSelectedAddress from getAccounts	Unresolved
Suggestion 2: [Plugin Beta] Avoid Using (foo in bar) Truth Checks	Unresolved
Suggestion 3: [LavaMoat] Detect Additional Primordials	Unresolved
Suggestion 4: [LavaMoat] Hide stacktraces	Resolved
Suggestion 5: [LavaMoat] Stronger Magic Copy	Unresolved

Issue A: [Plugin Beta] SES Realm Creation Enables the Error Stack

Location

app/scripts/controllers/plugins.js

Synopsis

The creation of the SES root realm enables passthrough of the error stack.

Impact

In the event of a thrown exception, the stack trace from the sandboxed realm is leaked and could potentially reveal information that was intended to be private.

Preconditions

The errorStackMode option is set to allow .

Remediation

Disable the errorStackMode option or only enable it when MetaMask is known to be running in a testing environment.

Status

There is a comment in the codebase to disable the error stack for production, however, the code currently still has this enabled. The MetaMask team has stated their intention to make the error stack enabled by an environment variable prior to pushing the code to production.

Verification

Unresolved.

Issue B: [Plugin Beta] Restricted Method submitPassword

Location

app/scripts/controllers/permissions/restrictedMethods.js

Synopsis

The method submitPassword is restricted and noted in the code as needing to be removed for production. There are still references to this method throughout other parts of the code indicating it may still be exposed to plugins.

Impact

If granted to a plugin, it would be able to potentially impersonate MetaMask and ask the user to unlock their wallet to intercept the user's password.

Preconditions

The method is exposed to plugins.

Remediation

Ensure that the submitPassword method is not usable or grantable to plugins.

Status

The inclusion of the method has not been removed, however, the MetaMask team has stated their intention to remove it prior to pushing the code to production.

Verification

Unresolved.

Issue C: [Plugin Beta] Method getState Returns Potentially Sensitive Data

Location

app/scripts/controllers/permissions/restrictedMethods.js

Synopsis

The method `getState` returns the entire wallet state in a JSON representation, revealing potentially sensitive information. It is noted in the code that it should be removed for production, however there are still references to it.

Impact

If granted to a plugin, it would allow the plugin to view MetaMask's internal state which may hold sensitive information that was not explicitly granted to the plugin.

Preconditions

The method is exposed to plugins.

Remediation

Ensure that the `getState` method is not usable or grantable to plugins.

Status

The inclusion of the method has not been removed, however, the MetaMask team has stated their intention to do so, as well as include tests around the functionality prior to pushing the code to production.

Verification

Unresolved.

Issue D: [Plugin Beta] Plugin State is Part of Main State

Location

app/scripts/controllers/permissions/restrictedMethods.js

Synopsis

The method `updatePluginState` can be used to manipulate the application state.

Impact

Plugins may be able to take advantage of the shared state storage in order to manipulate other plugins or the main extension's state.

Technical Details

More information and related issues:

https://github.com/MetaMask/metamask-snaps-beta/issues/88

https://github.com/MetaMask/metamask-extension/issues/7311

Remediation

Separate the state storage of plugins and the extension. Additional details and discussion is tracked in the links included in the technical details section.

Status

Inclusion of the method has not been removed. The MetaMask team has stated their intention to separate the plugin state from the main state and verify that Starkware will not use too much storage prior to pushing the code to production.

Verification

Unresolved.

Issue E: [Plugin Beta] Bypass SES by Modifying global.process.env Properties

Location

app/scripts/controllers/plugins.js

Synopsis

It is possible for the variables process.env.IN_TEST === 'true' and process.env.METAMASK_ENV to be modified to bypass loading SES.

Impact

Plugins will no longer be executed within a sandbox and dependencies are no longer validated.

Preconditions

Check if global.process.env.IN_TEST.

Technical Details

If a content script sets either process.env.IN_TEST === 'true' or process.env.METAMASK_ENV === 'test', then SES will not be enabled.

Remediation

Use Object.freeze(process.env).

Status

The IN_TEST global variable condition is still present. The MetaMask team has stated their intention to address the issue prior to pushing the code to production.

Verification

Unresolved.

Issue F: [Plugin Beta] opts[requiredField] Will Return true if the Property is Declared but undefined

Location

app/scripts/controllers/assets.js

Synopsis

Checking the opts[requiredField] will return true if the property is declared but undefined .

Impact

Improperly validated actions can cause null reference exceptions and similar issues because {foo:undefined} will pass the check for 'foo' in bar but not typeof bar[foo] !== 'undefined' or even just !!bar[foo].

Preconditions

`(foo in bar)` returns true if `bar[foo] === undefined` .

Remediation

Instead of (!(requiredField in opts)) use (typeof opts[requiredField] === 'undefined') , as well as adding more sophisticated validation for the specific fields.

Status

The code in question has been removed or relocated, however, the issue still exists elsewhere.

Verification

Unresolved.

Issue G: [LavaMoat] Prevent access to __ proto __ from deepGet

Location

https://github.com/LeastAuthority/lavamoat-browserify/blob/master/src/makeGetEndowmentsForConfig.js#L60-L73

Synopsis

The deepGet method looks up user provided paths in the configuration object before it's actually running inside of SES.

Impact

Unpredictable behavior.

Remediation

It should use Object.hasOwnProperty before checking the result so that it behaves more predictably.

Status

MetaMask implemented a different resolution than what was recommended by our team. Instead of using hasOwnProperty, it throws an error if any key in the path is __proto__ , which resolved the issue.

Verification

Resolved.

Issue H: [LavaMoat] Exported Factory Function Can Return Shared Object

Synopsis

Factory functions return a shared object if modules are cached.

Impact

If a factory function returns a shared object, that object can be modified by the receiver. Lavamoat@3.0.0 protects against this by not caching modules, however, Lavamoat@>=3.0.1 does not. Caching modules is needed to support circular references in the dependency graph.

Technical Details

Caching modules or not is a tricky design issue; caching was introduced to support recursive cyclic dependencies (If A depends on B, but B then depends on A). Completely removing caching would prevent

shared accesses, if B and C depend on A, B gets Ab and C gets Ac and Ab !== Ac. It also means that instanceof checks would not work for comparing Ab with Ac. If a module is a subject of a cyclic dependency, then it needs to be cached. Code to reproduce is available here: https://gist.github.com/dominictarr/740ed01c63174ec1d932cca98f51c684

Remediation

If caching was disabled by default, it could be enabled only when cyclic dependencies are used. Additionally, modules developers should be encouraged to Object.freeze their prototypes.

Another possibility that could avoid the need for configuration would be to pass a cache to the module loader, such that a module had a cache of only its parent modules. If A, B and C require D, they all get their own versions of D, but if X requires Y which requires X, Y gets the same X.

It is strongly recommended that module authors avoid cyclic dependencies. Having a helpful error on loading a cyclic dependency is a better default behavior for LavaMoat.

Status

MetaMask has noted that this is an area of ongoing research in order to identify the best strategy to mitigate this issue.

Verification

Unresolved.

Issue I: [LavaMoat] Code Injection via Label in wrapWithReturnCjsExports

Location

src/generatePrelude.js

Synopsis

A module could bust out of the wrapper if they put a new line and JavaScript code into a file name. The name is inserted into a line comment. If the file name contains a new line (which is allowable under unix), the label will expand outside of the line comment and the next line will be actual runnable code, and will see a different scope for module and exports as a result.

Impact

Modules could break out of the sandbox and run arbitrary code.

Remediation

Possible fixes:

- JSON.stringify(label) new lines will be escaped
- Remove the label altogether

It is possible that something else in the browserify system disallows unusual file names, but ensuring that browserify never makes those changes is not as simple as sanitizing the name.

Status

The wrapWithReturnCjsExports function was modified so that labels with newline characters will cause it to throw an exception.

Verification

Resolved.

Issue J: [LavaMoat] Child Dependencies Can Access a Parent Module's Exports Before Harden is Applied

Synopsis

Agoric's harden function recursively traverses a module and applies `Object.freeze` and wraps functions so that another module cannot modify that object. However, harden is called *after* the module returns, but any child modules are called *before* the module returns, thus child modules that have cyclic dependencies on the parent have access to the parent's exports before harden is called. See Issue H.

Impact

Child modules that explicitly access a parent module could modify it.

Mitigation

Recursive dependencies should be avoided. Unfortunately, common JavaScript style module systems currently just support cyclic dependencies silently.

Remediation

Require an explicit permission to get a cyclic reference to a parent module.

Status

MetaMask has noted that this is an area of ongoing research in order to identify the best strategy to mitigate this issue.

Verification

Unresolved.

Suggestions

Suggestion 1: [Plugin Beta] Separate Logic for `getSelectedAddress` from `getAccounts`

Location

app/scripts/metamask-controller.js

Synopsis

The `getAccounts` method accepts an origin parameter. If that parameter is MetaMask, then the function returns the currently selected address only.

Mitigation

Because it is unclear if an attacker could manipulate the origin parameter and the function effectively serves two purposes, we suggest moving the `if origin === 'metamask'` path of this method call to another method call altogether like `getSelectedAccount` and ensure that method is only accessible by the extension. This removes some ambiguity around whether or not other callers can manipulate the origin and whether or not it matters if they do.

Status

No changes were made to the code in question. MetaMask has stated their intention to deeply audit and ensure that sites and plugins cannot manipulate their origin strings prior to pushing the code into production.

Verification

Unresolved.

Suggestion 2: [Plugin Beta] Avoid Using (foo in bar) Truth Checks

Location

app/scripts/controllers/assets.js

Synopsis

Using (foo in bar) returns true, even if bar[foo] === undefined . The in check only checks that the property exists, not the value. If checks against undefined (or checks for general "truthiness" of the value) do not exist, it results with null reference exceptions and similar issues because {foo:undefined} will pass the check for 'foo' in bar.

Mitigation

Replace (foo in bar) checks with typeof bar[foo] !== 'undefined' or even just !!bar[foo] .

Status

The pattern in question still exists in the codebase. The MetaMask team has stated their intention to address the issue prior to pushing the code to production.

Verification

Unresolved.

Suggestion 3: [LavaMoat] Detect Additional Primordials

Synopsis

When inspectEnviroment is used by generateConfig to create a working configuration the first time that lavamoat-browserify is run if there is an assignment to Object or Array such as Object.prototype.foo = true then inspectEnviroment will set resources.<module_name>.enviroment = unfrozen in the config. This, however, does not detect other primordials. Looking at the configuration, which in a large program could also be very large, it is not immediately obvious that an unsafe permission has been set.

Mitigation

Add support for syntax such as --allowUnfrozen module_name and, if the need for an unfrozen environment is detected, a message indicating how to enable that would be printed, but the user must first explicitly consent to that.

Status

MetaMask has noted that this is an area of ongoing research in order to identify the best strategy to mitigate this issue.

Verification

Unresolved.

Suggestion 4: [LavaMoat] Hide stacktraces

Synopsis

Viewing Error#stacktrace should not be allowed except by special cases.

Remediation

Code should not be able to decipher if it is the unit tests that are running or the program. Of course, they should still be able to create errors, and if that error is passed to something that has stacktrace permission, it should be able to see it.

Status

MetaMask implemented the suggested remediation.

Verification

Resolved.

Suggestion 5: [LavaMoat] Stronger Magic Copy

Synopsis

A stronger magic copy would prevent a situation where a cached module X is used by Y and Z, and both Y and Z pass the same object into X, X would see it as different objects. An alternative would be to not cache the module, so Y gets YX and Z gets ZX.

Mitigation

Add a mode where everything crossing into the module is copied and everything it returns is copied again. This would be similar to how dnode copies all arguments but preserves functions. If the module exports a function, this mode would wrap that function with a function that copied all the arguments passed to it. This would add additional overhead. Some code expects to share a reference and that code would break, while other code would still work. This mode would be unlikely to be enabled by default because it is too expensive.

Status

MetaMask has noted that this is an area of ongoing research in order to identify the best strategy to mitigate this issue.

Verification

Unresolved.

Recommendations

We recommend that the unresolved *Issues* and *Suggestions* stated above are addressed as soon as possible and followed up with verification by the auditing team.

Overall, we encourage the MetaMask team to continue to prioritize security, along with support both internal and external code reviews. In addition, it is our understanding that the target revision we audited included a number of features that were clearly not intended to be introduced into production. Most of these areas were noted with comments, however this strategy for marking testing code for removal has

potential for being overlooked and making it into production. We would like to encourage using other methods for disabling code in production such as runtime environment variables, feature flags, or similar patterns.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create

an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.