PENETRATION TEST REPORT

Pali Wallet 2025

Prepared for: Syscoin

Prepared by: Tim De Wachter, CTO, Cyrex Ltd

2026-01-06

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

Cyrex was contracted by Syscoin to conduct a penetration test to determine its exposure to a targeted attack. All activities were conducted in a manner that simulated a malicious actor engaged in a targeted attack against the scope with the goals of:

- Identifying if a remote attacker could penetrate the scope of its defences.
- Determining the impact and possibility of a security breach.

Efforts were placed on the identification and exploitation of security weaknesses that could allow a remote attacker to gain unauthorized access to organizational data. The attacks were conducted with all levels of access that a general internet user would have.

As Syscoin provided Cyrex with full source code concerning the scope, we can label this kind of test as a white box penetration test. Cyrex was granted access to the extension in a standard end-user state.

What follows is a detailed explanation of all the different vulnerabilities, advised patches, and an accompanying risk analysis. We are confident that the penetration test and this report will help the customer to raise its security regarding their wallet extension to a higher level.

2. ABOUT CYREX

Cyrex is a native cybersecurity scale-up that specialises in penetration testing, load testing, and software development. Our focus is on online gaming, blockchain, and the SaaS world.

With more than 50 security engineers and 100 clients worldwide, Cyrex is known for its work on state-of-the-art technologies and innovation in multiple industries. With a constant drive for excellence, Cyrex has been recognized repeatedly for its quality of work, timely deliverables, and an outstanding operational performance.

Cyrex goes beyond the traditional penetration testing offering through a unique approach that integrates pair-hacking methodologies and manual security testing. As leaders in the industry for over 10 years, our teams are prepared for any type of project regardless of the technology stack, providing a full coverage of our partners' needs.

We are Cyrex, the gold standard in security, load testing and development.

3. SCOPE OF ENGAGEMENT

Cyrex performed a penetration test on Syscoin's Pali wallet extension. The following tests were performed by Cyrex:

Initial assessment (2025-11-17 - 2025-11-28).

Retest (2026-01-06 - 2026-01-07).

During the penetration test, strict protocols, guidelines, and a unique workflow have been followed. Different frameworks were integrated into this process flow, which are in line with the ethical hacking procedures. The process involved an active analysis of the application for any weaknesses, technical flaws, or vulnerabilities.

The focus of the test was to determine whether the applications were prone to any technical vulnerabilities that could affect the confidentiality and integrity of the data of Syscoin's customers, the availability of the platform, and the security of its users.

During the entire penetration testing life cycle, Cyrex performed the following actions to determine security issues within the application:

1. Analysis and testing of different endpoints
2. Tampering with different parameters within those requests
3. Identification of potential injection points, security flaws, and vulnerabilities
4. Exploitation to provide Proof of Concept (PoC)

With the regression testing, we make sure the vulnerabilities discovered during the penetration test are patched in a correct manner and no other vulnerabilities have been introduced during the patching process.

The following components were part of the tested scope:

- Browser Extension
- Source code

We want to thank Syscoin for putting trust in our know-how and expertise concerning ethical hacking specific to wallet security.

4. RISK ANALYSIS

The risk assessment conducted uses the 'Common Vulnerability Scoring System'. The Common Vulnerability Scoring System (CVSS) provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes.

CVSS is composed of three metric groups: Base, Temporal, and Environmental, each consisting of a set of metrics.

The Base metric group represents the intrinsic characteristics of a vulnerability that are constant over time and across user environments. It is composed of two sets of metrics: the Exploitability metrics and the Impact metrics.

- The Exploitability metrics reflect the ease and technical means by which the vulnerability can be exploited. That is, they represent characteristics of the thing that is vulnerable, which we refer to formally as the vulnerable component.
- The Impact metrics reflect the direct consequence of a successful exploit, and represent the consequence to the thing that suffers the impact, which we refer to formally as the impacted component.

The Temporal metric group reflects the characteristics of a vulnerability that may change over time but not across user environments. For example, the presence of a simple-to-use exploit kit would increase the CVSS score, while the creation of an official patch would decrease it.

The Environmental metric group represents the characteristics of a vulnerability that are relevant and unique to a particular user's environment. Considerations include the presence of security controls which may mitigate some or all consequences of a successful attack, and the relative importance of a vulnerable system within a technology infrastructure.

For more information:
https://www.first.org/cvss/

The risk assessment for a finding serves as an indication. It is intricate to estimate the exact (business) impact of a vulnerability. Ultimately, it is up to the client to determine the urgency of remedying a security weakness.

Additional report labels

Next to the risk classifications, several other indication labels are (possibly) used in the report:

- When there is no direct security impact related to the finding, but it is still recommended to follow the best practices as outlined in the recommendation section: INFO
- When a vulnerability found during the initial research has been retested, and is resolved correctly: ✓ Resolved
- When a vulnerability found during the initial research has not been resolved, but is marked as an acceptable risk by the client: 🤝 Accepted Risk

CRITICAL	HIGH	MEDIUM	LOW	INFO
No findings	No findings	No findings	1 finding	1 finding

Table 4.1: Finding counts by severity.

#	Title	Status	Severity	Page
001	Private Keys Encrypted With Hashed Password Instead of Password Browser Extension	✓ Resolved	HIGH (7.9)	15
003	Stealing Seedphrase through Clipboard Attacks Browser Extension	✓ Resolved	MEDIUM (6.8)	18
002	Low Password Complexity Threshold Browser Extension	✓ Resolved	MEDIUM (5.7)	31
004	Password Rate Limiting Bypass Browser Extension	✓ Resolved	MEDIUM (4.6)	9
007	Suboptimal Password Hashing Source code	☰ Partially Resolved	LOW (1.8)	22
005	Cached ENS Lookup in UI Browser Extension	✓ Resolved	INFO (0)	28
006	Vulnerable Dependencies Source code	☰ Partially Resolved	INFO (0)	25

Table 4.2: Findings ordered by severity and status.

5. CONCLUSION

The overall security maturity of the tested scope seems sufficient at this stage.

Syscoin demonstrated a commendable commitment to security through the timely identification, prioritization, and remediation of the reported vulnerabilities. The vast majority of findings were successfully addressed, with only one minor issue related to a 🔵 Vulnerable Dependencies remaining partially remediated. Overall, the response reflects a mature security posture and a strong dedication to continuously enhancing the protection of systems and users.

In addition, the availability of Android and iOS applications presents a valuable opportunity to further strengthen the ecosystem. Conducting dedicated penetration tests for both platforms would help ensure the same high standard of security is consistently maintained across all user touchpoints.

We are confident that we tested the whole scope and all functionalities that were available to us as in-depth as possible.

We want to thank Syscoin for putting trust in our know-how and expertise concerning ethical hacking.

6. VULNERABILITIES

This section lists all the vulnerabilities that were successfully determined within the target scope. These vulnerabilities are explained in depth, and for every type of vulnerability, a best practice is provided by Cyrex to mitigate these vulnerabilities in the future.

As this penetration test was performed on the Pali wallet extension, Cyrex followed strict protocols and guidelines. Frameworks were integrated into this process flow, which comply with the legislation regarding ethical hacking. A process of reporting will be redundant throughout the entire report for the sake of simplicity and ease of comprehension.

Each vulnerability is assigned a unique ID starting with SYSCOIN-PALI2025, which refers to Pali Wallet 2025. The unique identifier will increment by one for each subsequent vulnerability.

Whenever a vulnerability reported by Cyrex is unclear to the customer, Cyrex is eager to help and supply additional details concerning the vulnerability. Feel free to contact us at any time.

6.1. Brute Force Attack

Brute Force Attacks (CWE-799) are a type of security vulnerability that occurs when an attacker systematically tries every possible combination of usernames and passwords to gain unauthorized access to a system or network. This vulnerability can arise when proper authentication and session management techniques are not implemented, such as weak password policies, lack of account lockout mechanisms, and insufficient monitoring of login attempts.

Attackers use various methods to carry out brute force attacks, such as dictionary attacks, where attackers use pre-computed lists of commonly used passwords, or credential stuffing attacks, where attackers use previously leaked credentials from other services to try and gain access. Brute force attacks can also be used to exploit vulnerabilities in encryption algorithms or guess authentication tokens.

The impact of a successful brute force attack can be severe, as it can result in the compromise of sensitive data, the installation of malware or ransomware, and even the complete takeover of the system.

To mitigate Brute Force Attacks, organizations should implement strong password policies, including the use of multifactor authentication, account lockout mechanisms, and monitoring of login attempts. It is also important to educate users on the importance of using strong and unique passwords and avoiding the reuse of passwords across different services.

Additionally, organizations should regularly review their systems for potential vulnerabilities and implement security controls, such as rate limiting and throttling, to prevent brute force attacks and other types of attacks that rely on a large number of requests to overwhelm the system.

By addressing this vulnerability, organizations can enhance the security of their systems and protect against unauthorized access attempts through brute force attacks and other similar techniques.

Password Rate Limiting Bypass

OK

~~4.6~~
~~MEDIUM~~ ~~CVSS:3.1/AV:P/AC:H/PR:L/UI:N/S:U/C:L/I:H/A:N~~

ID: SYSCOIN-PALI2025-004
Target: Browser Extension

Description

Cyrex has determined a flaw in the wallet unlock process where the intended limit on repeated failed attempts exists only in memory. The counters that track failures and the time window for lockout reset whenever the background script reloads or the browser restarts. An attacker can repeatedly trigger these resets to avoid the intended delay between attempts and continue brute forcing without any real restriction.

Risk

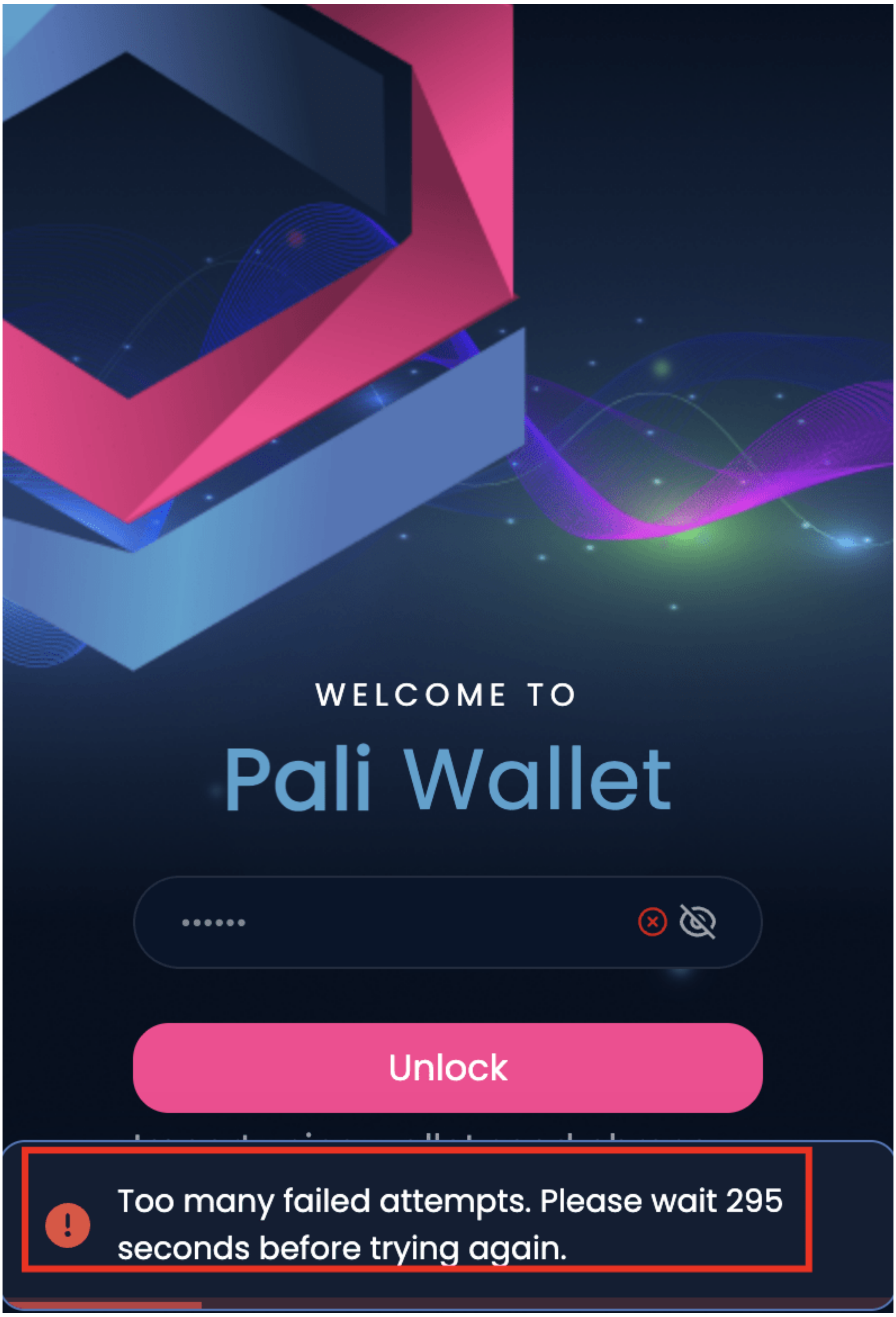
An attacker can bypass the unlock protection completely by forcing a script reload between attempts. This allows unlimited password guesses and places stored private keys and wallet contents at high risk of exposure.

Recommendation

It is recommended to store the rate limit state in persistent extension storage so that failure counters and lockout timers continue to apply even after script reloads and browser restarts. It is recommended to read the stored metadata before every unlock attempt and apply the lockout window consistently to prevent any form of bypass.

Proof

1. Repeatedly enter incorrect wallet passwords until the rate limit activates.



Too many failed attempts. Please wait 295 seconds before trying again.

Figure 6.1: Triggered Rate limit

2. Reload the extension and observe that the lockout resets and the rate limit no longer applies.

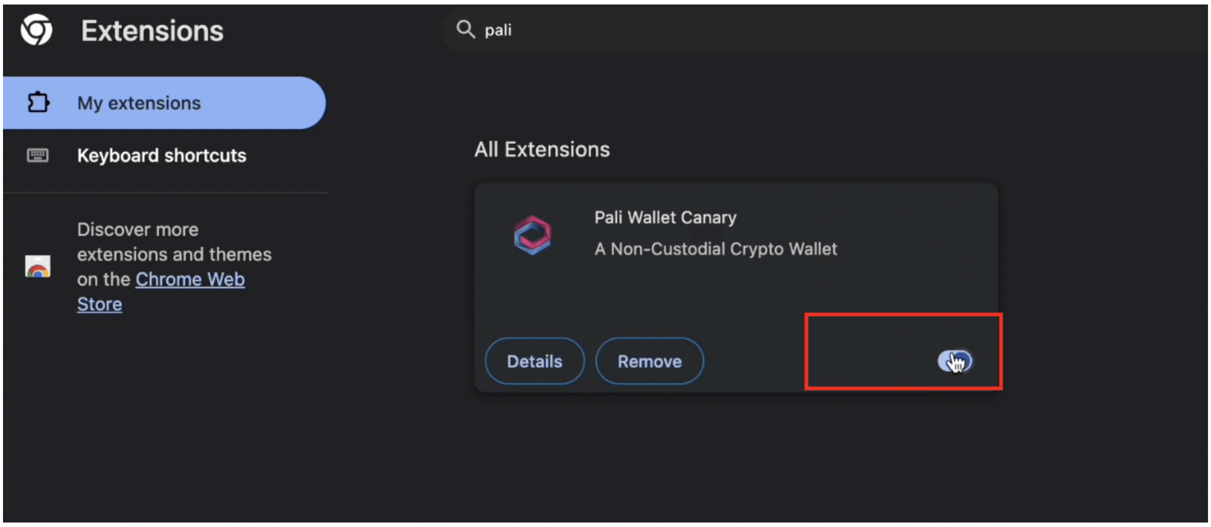


Figure 6.2: Reload Extension

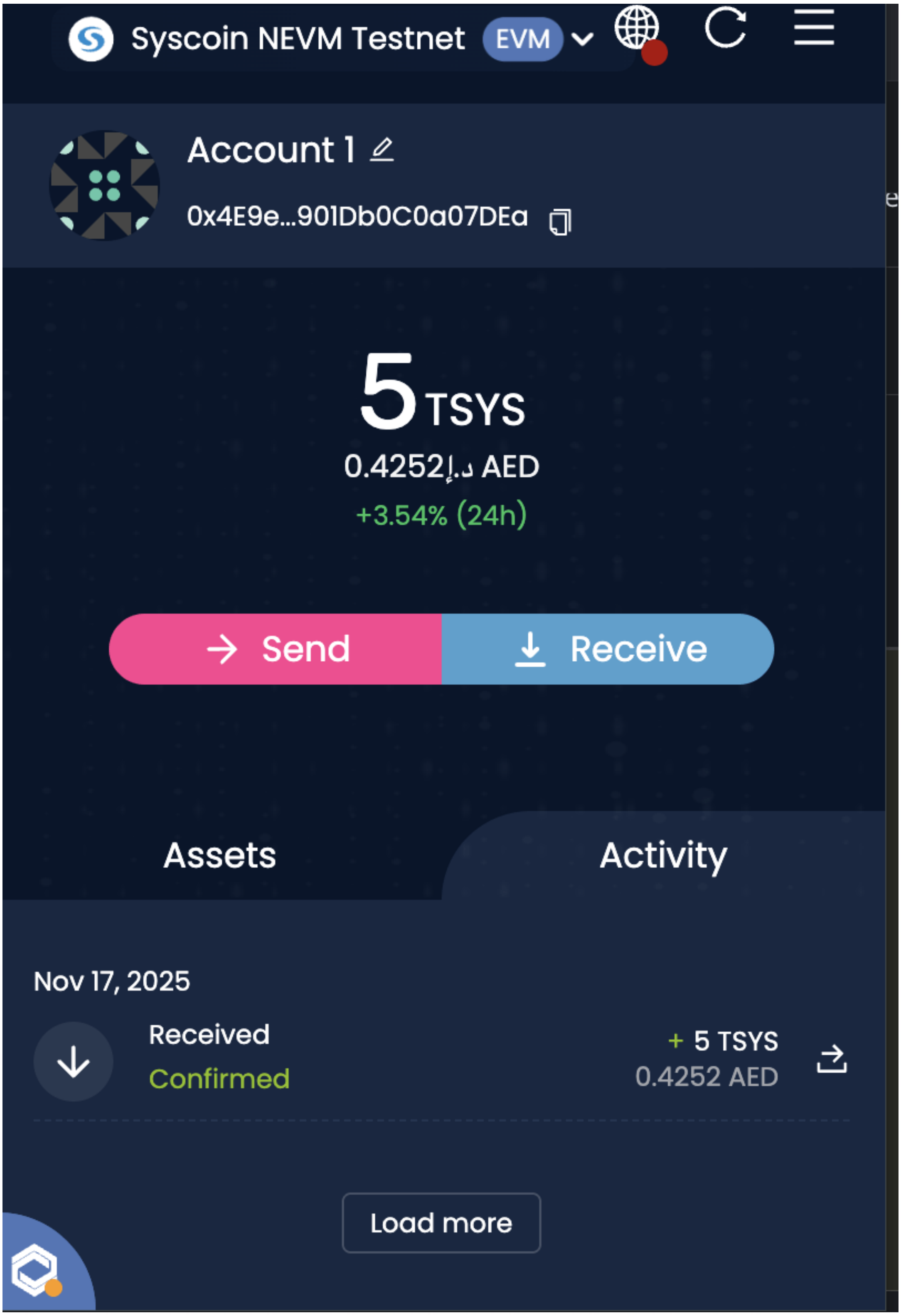
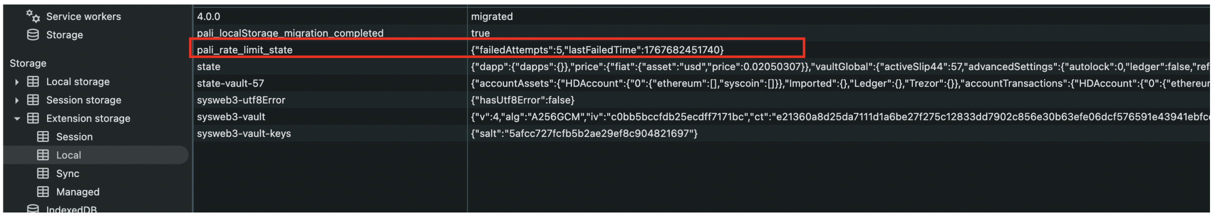


Figure 6.3: Rate limit Bypassed

POC Video

Retest 2026-01-06

OK



4.0.0	migrated
pali_localStorage_migration_completed	true
pali_rate_limit_state	{"failedAttempts":5,"lastFailedTime":1767682451740}
state	{"dapp":{"dapps":{}},"price":{"fiat":{"asset":"usd","price":0.02050307}},"vaultGlobal":{"activeSlip44":57,"advancedSettings":{"autolock":0,"ledger":false,"ref
state-vault-57	{"accountAssets":{"HDAccount":{"0":{"ethereum":[],"syscoin":[]}},"Imported":{},"Ledger":{},"Trezor":{}},"accountTransactions":{"HDAccount":{"0":{"ethereur
sysweb3-utf8Error	{"hasUtf8Error":false}
sysweb3-vault	{"v":4,"alg":"A256GCM","iv":"c0bb5bccfdb25ecdff7171bc","ct":"e21360a8d25da7111d1a6be27f275c12833dd7902c856e30b63efe06dcf576591e43941ebfc
sysweb3-vault-keys	{"salt":"5afcc727fcfb5b2ae29ef8c904821697"}

Figure 6.4: Rate Limiting Logic Relies on Persistent Extension Storage

6.2. Information Disclosure

Information Disclosure (CWE-200) is a security vulnerability that arises when an application or system reveals sensitive information unintentionally. This can occur due to various reasons, including improper error handling, misconfigured servers, or weak encryption.

Attackers can exploit the vulnerability by gaining access to sensitive information that they should not have access to, including personally identifiable information, financial data, or other confidential information. The information may be leaked through error messages, system logs, or other channels that provide too much information to attackers.

To mitigate Information Disclosure, developers should ensure that sensitive information is adequately protected and not unintentionally disclosed. This can be achieved by implementing appropriate security measures such as encryption to protect data in transit and at rest, ensuring that servers and applications are correctly configured to prevent unauthorized access, and implementing error-handling mechanisms that do not reveal sensitive information to attackers.

Developers should also regularly review their applications and systems for potential information disclosure vulnerabilities and promptly address any discovered vulnerabilities. This can help prevent attackers from gaining access to sensitive information and mitigate the impact of any successful attacks.

Private Keys Encrypted With Hashed Password Instead of Password

OK

~~7.9~~
~~HIGH~~ ~~CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:N~~

ID: SYSCOIN-PALI2025-001
Target: Browser Extension

Description

Cyrex has determined a flaw in the way account private keys are encrypted. They are encrypted with the user's hashed password rather than the password itself. Both values are stored in local storage, which is accessible when the wallet is locked.

Risk

Whenever a user obtains access to the local storage of the background process or the underlying filesystem, they can always decrypt the private keys.

Recommendation

The private keys should be encrypted with a key that comes from the user's password. That key must never be stored and should stay in memory only for the short time needed to unlock the wallet or export any secret. Use Argon2id or a similar, string algorithm as the KDF, with a unique salt and a pepper to slow down brute-force attempts and raise the cost for an attacker.

Proof

1. Install the Pali Wallet browser extension.
2. Create an account.
3. Go to the

`your keys` section and copy your private key.
4. Lock the wallet.

5. Check the local storage of the extension. Under `state-vault-{number}`, you can find the encrypted private key for that account. Under `sysweb3-vault-keys`, you will find the hashed password.
6. Now, run the following script with the values from step 5.

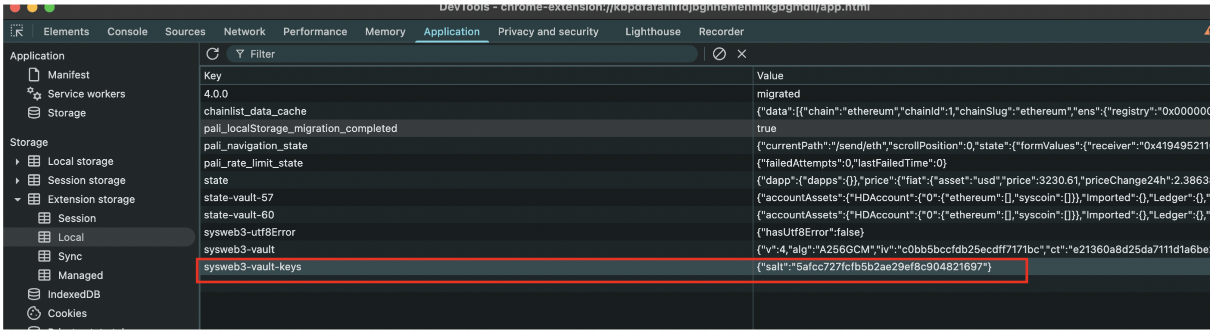
```
import CryptoJS from 'crypto-js';

const privKey = "U2FsdGVkX18i2Mp...AKPH1HDXCo=";
const password = 'f81b220c29f...a178f6c6e5f16c6fe93de24d57e515a153b';

try {
    const decryptedPrivKey = CryptoJS.AES.decrypt(
        privKey,
        password
    ).toString(CryptoJS.enc.Utf8);

    if (!decryptedPrivKey) {
        throw new Error('Decryption failed: Invalid password or corrupted data.');
    }

    console.log('decrypted : ', decryptedPrivKey);
} catch (error) {
    console.error('Error during decryption:', error.message);
}
```

Listing 6.1:

7. Compare the output with your copied private key from step 3.

Retest 2026-01-06

OK



Key	Value
4.0.0	migrated
chainlist_data_cache	{"data":[{"chain":"ethereum","chainId":1,"chainSlug":"ethereum","ens":{"registry":"0x000000
pali_localStorage_migration_completed	true
pali_navigation_state	{"currentPath":"/send/eth","scrollPosition":0,"state":{"formValues":{"receiver":"0x419495211
pali_rate_limit_state	{"failedAttempts":0,"lastFailedTime":0}
state	{"dapp":{"dapps":{}},"price":{"fiat":{"asset":"usd","price":3230.61,"priceChange24h":2.3863
state-vault-57	{"accountAssets":{"HDAccount":{"0":{"ethereum":[],"syscoin":[]}},"Imported":{},"Ledger":{},"
state-vault-60	{"accountAssets":{"HDAccount":{"0":{"ethereum":[],"syscoin":[]}},"Imported":{},"Ledger":{},"
sysweb3-utf8Error	{"hasUtf8Error":false}
sysweb3-vault	{"v":4,"alg":"A256GCM","iv":"c0bb5bccfdb25ecdff7171bc","ct":"e21360a8d25da7111d1a6be
sysweb3-vault-keys	{"salt":"5afcc727fcfb5b2ae29ef8c904821697"}

Figure 6.5: Removed hash from vault-keys completely

```
const sessionPasswordKey = await this.encryptSHA512Async(password, salt);
await getDecryptedVault(sessionPasswordKey);
```

Listing 6.2: Password Validation via Decryption not directly from hash

```
const encryptVaultWebCrypto = async (
  plaintextJson: string,
  keyHex: string
): Promise<VaultGcmEnvelopeV4> => {
  const subtle = (globalThis as any).crypto.subtle as SubtleCrypto;
  const iv = new Uint8Array(12);
  (globalThis as any).crypto.getRandomValues(iv);

  const keyBytes = hexToBytes(keyHex);
  const key = await subtle.importKey(
    'raw',
    keyBytes as unknown as BufferSource,
    { name: 'AES-GCM' },
    false,
    ['encrypt']
  );

  const pt = new TextEncoder().encode(plaintextJson);
  const ct = await subtle.encrypt(
    { name: 'AES-GCM', iv: iv as unknown as BufferSource },
    key,
    pt as unknown as BufferSource
  );

  return {
    v: 4,
    alg: 'A256GCM',
    iv: bytesToHex(iv.buffer),
    ct: bytesToHex(ct),
  };
};
```

Listing 6.3: More Secure Storage

Stealing Seedphrase through Clipboard Attacks

OK

~~6.8~~
~~MEDIUM~~ ~~CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:N~~

ID: SYSCOIN-PALI2025-003
Target: Browser Extension

Description

Cyrex has determined a flaw where a hostile site or decentralized application can interact with the system clipboard during user actions or by using overly permissive clipboard features. This behavior enables the site to capture or replace sensitive information such as seed phrases, putting wallet data at risk.

Risk

An attacker can silently read sensitive values copied by the user or push a forged value into the clipboard. This can lead to full compromise of the wallet, loss of funds, and control of the account by an unauthorized party.

Recommendation

It is recommended to remove the copy button from the interface where seed phrases are shown so users are not encouraged to place sensitive values on the clipboard. Display these secrets in a view-only form that avoids automatic or easy copying.

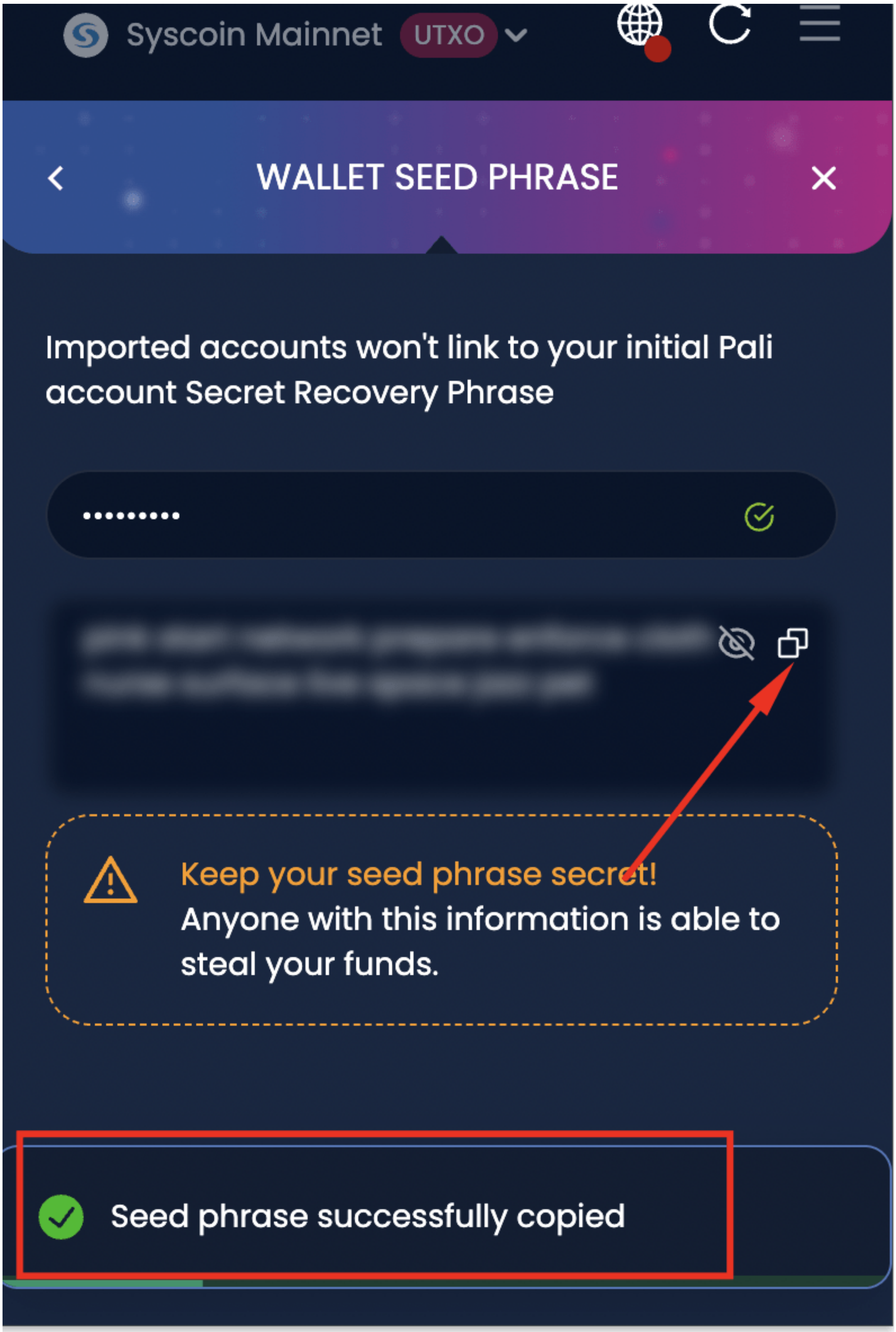
Proof

1. Host a proof of concept page designed to read clipboard data during a user gesture.

```
<!DOCTYPE html>
<html>
<body>

<h3>Clipboard Test</h3>

<button id="read">Read Clipboard</button>
<script>
// Read on click (simulates malicious read after a gesture)
document.getElementById("read").addEventListener("click", async () => {
  try {
    const content = await navigator. clipboard.readText()
    console.log("Clipboard read:", content)
    alert("Clipboard contains: " + content)
  } catch (e) {
    console.log("Read blocked", e)
  }
})
</script>
</body>
</html>
```

Listing 6.4: POC Page

2. Open the page and grant clipboard permissions when prompted.
3. Switch to the wallet extension and copy your own seed phrase using the copy option.



Imported accounts won't link to your initial Pali account Secret Recovery Phrase

••••••••••

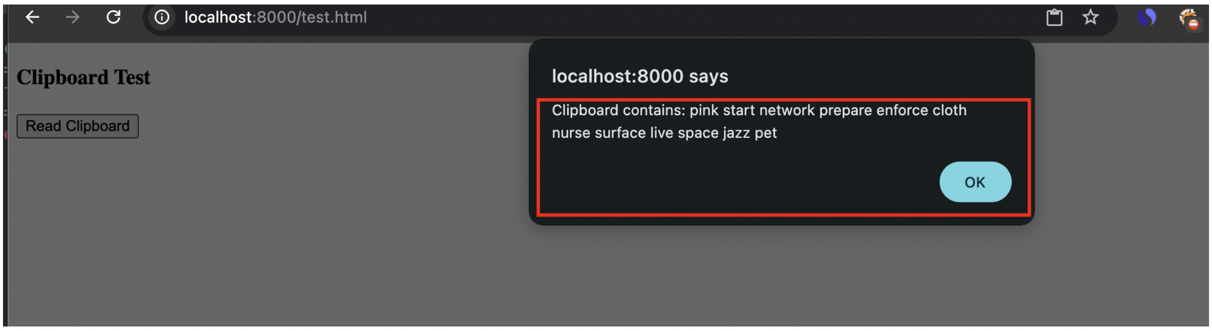
[redacted]

Keep your seed phrase secret! Anyone with this information is able to steal your funds.

Seed phrase successfully copied

Figure 6.6: Copied Seed Phrase

4. Return to the proof of concept website and click on the Read Clipboard button.

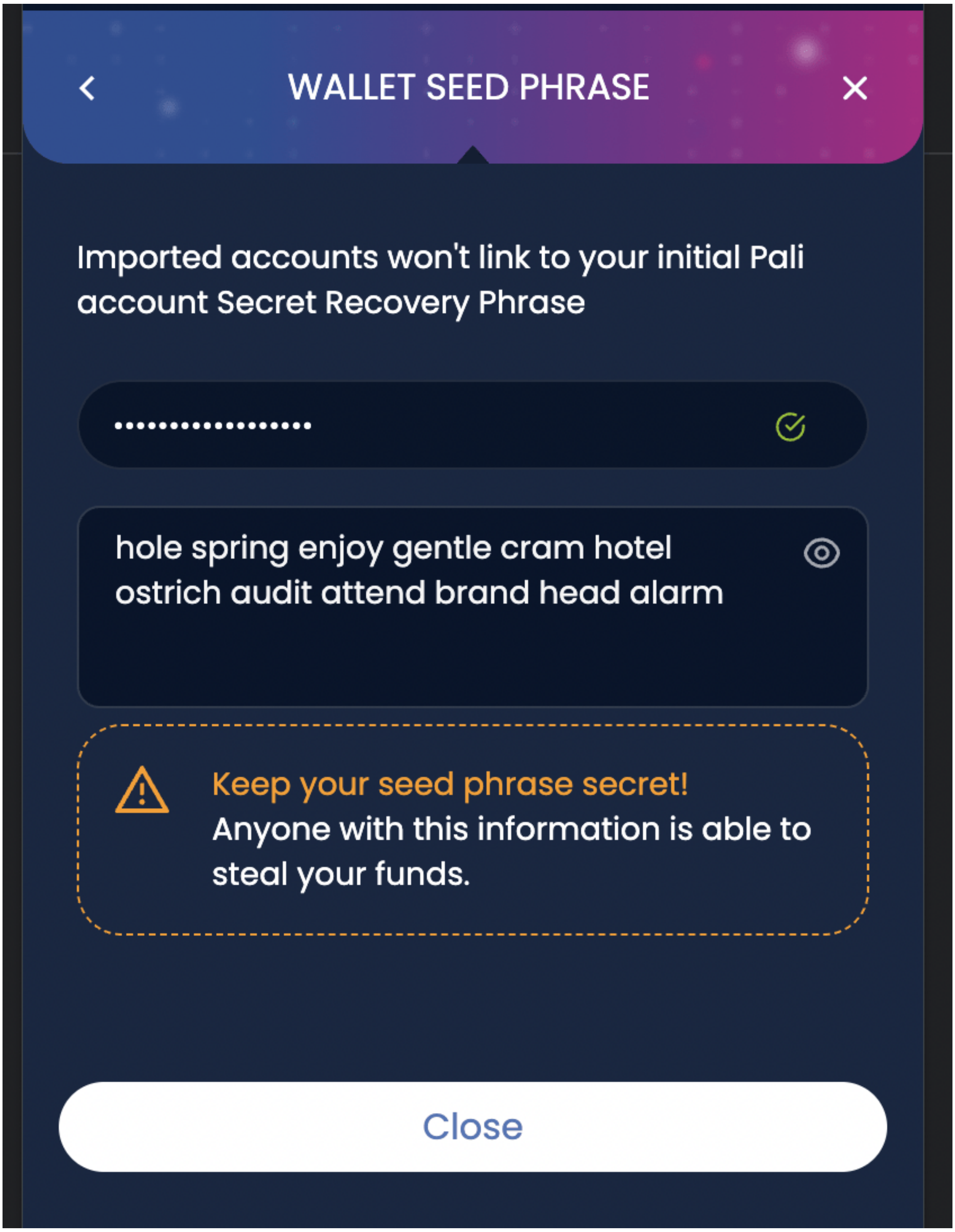


Clipboard contains: pink start network prepare enforce cloth nurse surface live space jazz pet

Figure 6.7: Stolen Seed Phrase

Retest 2026-01-06

OK



Imported accounts won't link to your initial Pali account Secret Recovery Phrase

••••••••••••••••••

hole spring enjoy gentle cram hotel ostrich audit attend brand head alarm

Keep your seed phrase secret!
Anyone with this information is able to steal your funds.

Figure 6.8: Disabled Copy Functionality

6.3. Security Misconfiguration

Security Misconfiguration (CWE-1349) is a type of vulnerability that occurs when a system or application is configured insecurely or incorrectly, leaving it vulnerable to potential attacks. This type of weakness can occur at various levels, such as hardware, software, applications, operating systems, network devices and infrastructure, and cloud services.

The vulnerability arises due to human error, lack of awareness about security practices, and improper installation or configuration of software. Misconfigured systems can lead to potential attacks, such as unauthorized access, data breaches, or launching denial-of-service (DoS) attacks.

Attackers can exploit a misconfiguration vulnerability in several ways, such as gaining unauthorized access to sensitive information, executing arbitrary code, and launching DoS attacks. For example, an attacker may exploit a misconfigured firewall that does not block unwanted traffic, leaving the system open to attacks.

To mitigate Security Misconfigurations, developers should ensure that they follow secure configuration guides and best practices provided by software or hardware vendors. Regular audits and security assessments can also help identify potential misconfiguration vulnerabilities and address them before they are exploited.

Some common examples of misconfiguration include default passwords, running unused services, failing to apply security patches and updates, using weak encryption algorithms or settings, and leaving system files and directories unprotected.

It is important to note that misconfiguration is not limited to specific types of systems or applications and can affect any system or service that relies on proper configuration.

The business impact of misconfigurations can vary depending on the protection needs of the application and data. Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, and more to gain unauthorized access or knowledge of the system. Therefore, it is critical to ensure that systems and applications are configured securely and that any potential misconfigurations are quickly identified and remediated.

Suboptimal Password Hashing

1.8
LOW
CVSS:3.1/AV:P/AC:H/PR:L/UI:N/S:U/C:N/I:L/A:N

~~4~~
~~MEDIUM~~
~~CVSS:3.1/AV:P/AC:H/PR:L/UI:N/S:U/C:N/I:H/A:N~~

ID: SYSCOIN-PALI2025-007
Target: Source code

Description

Cyrex confirmed a flaw where SHA512 is used with a single iteration. This makes the hash far too fast and reduces the work needed for an attacker to run brute force attacks.

Risk

If an attacker obtains the password hash, they can attempt guesses at high speed, which increases the likelihood of a successful password break.

Recommendation

It is recommended to adopt Argon2id with a robust configuration - at minimum 19 MiB of memory, 2 iterations, and 1 degree of parallelism - or an equivalent strong password hashing algorithm to significantly increase the computational cost of brute-force attacks. Maintain the use of a unique salt for each password and consider incorporating a server-side pepper to further enhance security.

Proof

1. Navigate to the

`keyring-manager.ts`

2. Observe that the password is hashed only once using

`SHA512`

```
  /**
   *
   * @param password
   * @param salt
   * @returns hash: string
   */
  private encryptSHA512 = (password: string, salt: string) =>
    crypto.createHmac('sha512', salt).update(password).digest('hex');
...
```

Listing 6.5: sysweb3-main\packages\sysweb3-keyring\src\keyring-manager.ts

References

- https://www.baeldung.com/cs/password-salt-pepper
- https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Retest 2026-01-06

1.8
LOW

CVSS:3.1/AV:P/AC:H/PR:L/UI:N/S:U/C:N/I:L/A:N

It is recommended increasing iterations to 200,000 if possible.

```
private encryptSHA512Auth = (password: string, salt: string): string => {
    // PBKDF2 with SHA-512, 24,000 iterations (prod default), 512-bit output
    // NOTE: Tests override iterations to keep CI fast.
    const isTestEnv =
      typeof process !== 'undefined' && process?.env?.NODE_ENV === 'test';
    const iterations =
      Number.parseInt(process?.env?.SYSWEB3_PBKDF2_AUTH_ITERS || '', 10) ||
      (isTestEnv ? 1_200 : 24_000);
    const hash = CryptoJS.PBKDF2(password, CryptoJS.enc.Hex.parse(salt), {
      keySize: 512 / 32, // 512 bits
      iterations,
      hasher: CryptoJS.algo.SHA512,
    });
    return hash.toString();
  };
```

Listing 6.6: PBKDF2 with 24k Iterations

References:

- https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Vulnerable Dependencies

0
INFO

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

ID: SYSCOIN-PALI2025-006
Target: Source code

Description

Cyrex has determined that several packages used in the project contain known vulnerabilities. Scanning identified 10 matches across dependencies like

`axios` ,

`node-fetch` ,

`cookie` ,

`playwright` , and

`tmp` . Some vulnerabilities are rated high severity, and while most have fixes available, the project is currently using older, vulnerable versions.

Risk

Exploitable vulnerabilities in widely used libraries can allow attackers to compromise the application, steal sensitive data, or execute arbitrary code in the environment where the dependencies run. High-severity flaws in packages like

`axios` and

`node-fetch` are particularly concerning for server-side or client-facing operations.

Recommendation

It is recommended to upgrade each vulnerable package to the latest version and keep it updated. In some cases, dependencies on third-party libraries contain vulnerabilities. In that case, it is recommended to check the changelog and upgrade if the dependency is updated.

Proof

1. Install the Grype tool, if not already installed
2. Navigate to the project directory that you wish to scan.
3. Run the following Grype command to scan for outdated dependencies:

```
grype .
```

Listing 6.7: Command

4. Wait for Grype to complete the scan and observe the output in the terminal.

```
  pali-wallet-develop grype .
  Indexed file system
↪
↪                                                .
  Cataloged contents
↪                                                                            cdb4ee2aea69cc6
↪ a83331bbe96dc2caa9a299d21329efb0336fc02a82e1839a8
```

```
       Packages                        [1,744 packages]
       File digests                    [5 files]
       File metadata                   [5 locations]
       Executables                     [0 executables]
  Scanned for vulnerabilities     [18 vulnerability matches]
     by severity: 0 critical, 12 high, 4 medium, 2 low, 0 negligible
     by status:   18 fixed, 0 not-fixed, 0 ignored
[0000]  WARN no explicit name and version provided for directory source, deriving artifact ID
↪ from the given path (which is not ideal)
NAME        INSTALLED  FIXED-IN  TYPE  VULNERABILITY        SEVERITY
axios       0.21.4     0.30.0    npm   GHSA-jr5f-v2jv-69x6  High
axios       0.21.4     0.30.2    npm   GHSA-4hjh-wcwx-xvwj  High
axios       0.21.4     0.28.0    npm   GHSA-wf5p-g6vw-rhxx  Medium
axios       1.11.0     1.12.0    npm   GHSA-4hjh-wcwx-xvwj  High
axios       1.7.7      1.8.2     npm   GHSA-jr5f-v2jv-69x6  High
axios       1.7.7      1.12.0    npm   GHSA-4hjh-wcwx-xvwj  High
cookie      0.4.2      0.7.0     npm   GHSA-pxg6-pf52-xh8x  Low
glob        10.4.5     10.5.0    npm   GHSA-5j98-mcp5-4vw2  High
js-yaml     3.14.1     3.14.2    npm   GHSA-mh29-5h37-fv8m  Medium
js-yaml     4.1.0      4.1.1     npm   GHSA-mh29-5h37-fv8m  Medium
node-fetch  1.7.3      2.6.7     npm   GHSA-r683-j2x4-v87g  High
node-forge  1.3.1      1.3.2     npm   GHSA-5gfm-wpxj-wjgq  High
node-forge  1.3.1      1.3.2     npm   GHSA-554w-wpv2-vw27  High
node-forge  1.3.1      1.3.2     npm   GHSA-65ch-62r8-g69g  Medium
playwright  1.55.0     1.55.1    npm   GHSA-7mvr-c777-76hp  High
tmp         0.0.33     0.2.4     npm   GHSA-52f5-9888-hmc6  Low
valibot     0.37.0     1.2.0     npm   GHSA-vqpr-j7v3-hqw9  High
valibot     0.38.0     1.2.0     npm   GHSA-vqpr-j7v3-hqw9  High
```

Listing 6.8: Output

```
  sysweb3-main grype .
  Indexed file system
↪
↪                                                  .
  Cataloged contents
↪                                                                              cdb4ee2aea69cc6
↪ a83331bbe96dc2caa9a299d21329efb0336fc02a82e1839a8
       Packages                        [1,807 packages]
       File digests                    [6 files]
       File metadata                   [6 locations]
       Executables                     [0 executables]
  Scanned for vulnerabilities     [22 vulnerability matches]
     by severity: 0 critical, 17 high, 5 medium, 0 low, 0 negligible
     by status:   22 fixed, 0 not-fixed, 0 ignored
[0000]  WARN no explicit name and version provided for directory source, deriving artifact ID
↪ from the given path (which is not ideal)
NAME        INSTALLED  FIXED-IN  TYPE  VULNERABILITY        SEVERITY
axios       0.21.4     0.30.0    npm   GHSA-jr5f-v2jv-69x6  High
axios       0.21.4     0.30.2    npm   GHSA-4hjh-wcwx-xvwj  High
axios       0.21.4     0.28.0    npm   GHSA-wf5p-g6vw-rhxx  Medium
axios       1.11.0     1.12.0    npm   GHSA-4hjh-wcwx-xvwj  High
axios       1.7.7      1.8.2     npm   GHSA-jr5f-v2jv-69x6  High
axios       1.7.7      1.12.0    npm   GHSA-4hjh-wcwx-xvwj  High
glob        11.0.3     11.1.0    npm   GHSA-5j98-mcp5-4vw2  High
js-yaml     3.14.1     3.14.2    npm   GHSA-mh29-5h37-fv8m  Medium
js-yaml     4.1.0      4.1.1     npm   GHSA-mh29-5h37-fv8m  Medium
node-fetch  1.7.3      2.6.7     npm   GHSA-r683-j2x4-v87g  High
valibot     0.37.0     1.2.0     npm   GHSA-vqpr-j7v3-hqw9  High
valibot     0.38.0     1.2.0     npm   GHSA-vqpr-j7v3-hqw9  High
```

Listing 6.9: Output

Retest 2026-01-06

0
INFO

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

```
   pali-wallet-master grype .

   Indexed file system
↪                                                                                           .
   Cataloged packages              [1874 packages]
   Scanned for vulnerabilities     [3 vulnerability matches]
      by severity: 0 critical, 2 high, 0 medium, 1 low, 0 negligible
      by status:   2 fixed, 1 not-fixed, 0 ignored
[0000]  WARN no explicit name and version provided for directory source, deriving artifact ID
↪ from the given path (which is not ideal)
NAME  INSTALLED  FIXED-IN  TYPE        VULNERABILITY        SEVERITY
atty  0.2.14               rust-crate  GHSA-g98v-hv3f-hcfr  Low
qs    6.13.0     6.14.1    npm         GHSA-6rw7-vpxm-498p  High
qs    6.14.0     6.14.1    npm         GHSA-6rw7-vpxm-498p  High
```

Listing 6.10: Outdated Dependencies Identified

Cached ENS Lookup in UI

OK

~~0~~
~~INFO~~ ~~CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N~~

ID: SYSCOIN-PALI2025-005

Target: Browser Extension

Description

Cyrex has determined a flaw where the interface displays an ENS name using a stored value from an earlier lookup. When the ownership or records of that name change, the interface still shows the older address from the cache. This can confuse the user since the presentation in the interface does not reflect the current state of the name on the chain.

Risk

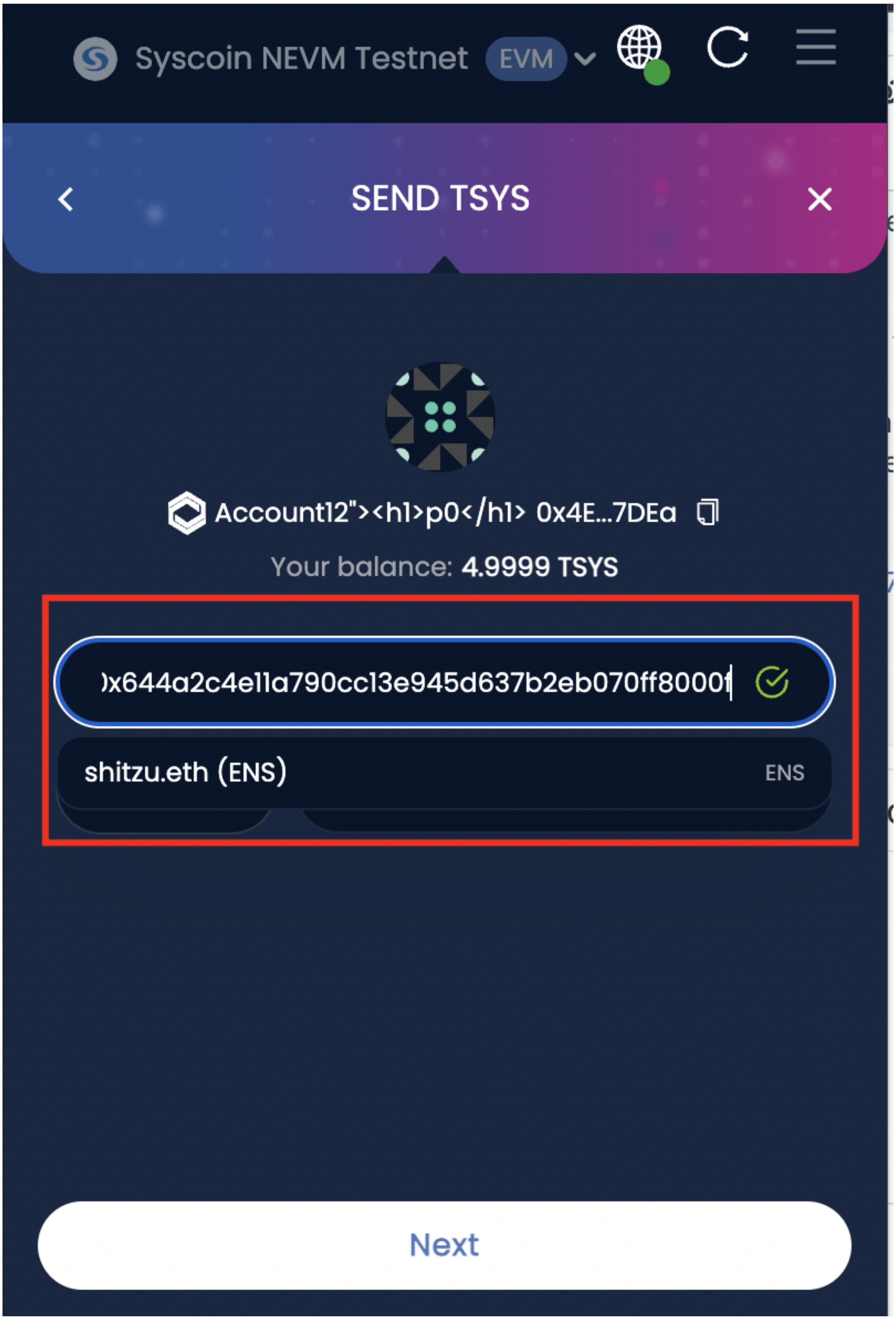
A user may believe they are interacting with the address shown in the interface when, in reality, the ENS name now points to a different address. Although the transaction flow updates to the correct address during the final step, the mismatch in the early interface view can lead to misunderstanding or incorrect assumptions during review.

Recommendation

It is recommended to refresh ENS data as soon as the user enters the ENS name in the send field. This ensures that the address shown in every part of the interface reflects the current on-chain state. It is recommended to avoid relying on older lookup data for anything visible to the user.

Proof

1. Open the wallet and click on send, enter the ENS name, and note the resolved address.



Syscoin NEVM Testnet EVM

SEND TSYS

Account12"><h1>p0</h1> 0x4E...7DEa

Your balance: **4.9999 TSYS**

)x644a2c4e11a790cc13e945d637b2eb070ff8000f

shitzu.eth (ENS) ENS

Figure 6.9: Resolved ENS

2. Transfer your ENS NFT to another account, then set the new address as primary on https://app.ens.domains/.

3. Return to the wallet, click on send again, and enter the same ENS name, and note that it still resolves to the older address.

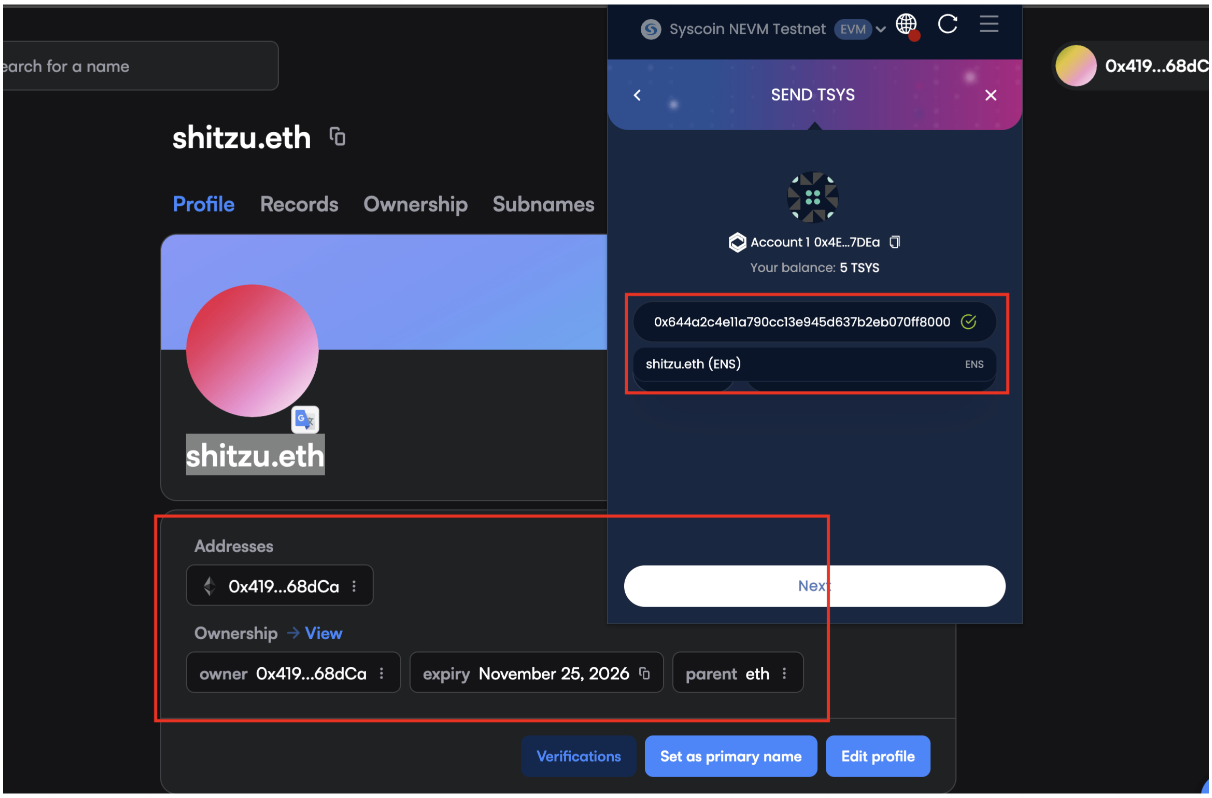


Figure 6.10: Cached ENS Lookup in UI

Retest 2026-01-06

OK

```
export const ENS_CACHE_TTL_MS = 5 * 60 * 1000;
export const isEnsCacheEntryValid = (
  entry: { name: string; timestamp: number },
  now: number = Date.now()
): boolean => now - entry.timestamp < ENS_CACHE_TTL_MS;
```

Listing 6.11: Added Cache TTL Constant and Created Validity Check Function

6.4. Weak Authentication

Weak Authentication (CWE-1390) is a security vulnerability that occurs when a web application does not properly authenticate users who attempt to access protected resources or functionality. This can lead to attackers impersonating legitimate users and gaining unauthorized access to sensitive data or functionality.

The vulnerability typically arises when an application's authentication mechanisms are weak or poorly implemented. This includes the application using weak passwords, storing passwords in plain text, or failing to enforce password complexity requirements. Another issue could be the use of insecure authentication protocols, such as HTTP Basic authentication, that are vulnerable to interception and replay attacks.

To mitigate Weak Authentication, developers should follow secure coding practices and implement robust authentication mechanisms that are resistant to attacks. This includes enforcing password complexity requirements, using secure password storage and encryption techniques, and using secure authentication protocols, such as OAuth or SAML. Moreover, developers should ensure that authentication is enforced on all access points to the application, including APIs and other entry points that may not be immediately apparent.

Developers should also be aware of and mitigate various weaknesses associated with authentication such as automated attacks such as credential stuffing, brute force or other automated attacks. They should also avoid the use of default, weak, or well-known passwords. Moreover, developers should use effective credential recovery and forgot-password processes and avoid plain text, encrypted, or weakly hashed passwords that may expose sensitive data.

In addition, it is recommended that developers implement multifactor authentication and properly manage user sessions and authentication tokens. User sessions or authentication tokens should be properly invalidated during logout or a period of inactivity to prevent the exposure of sensitive data. Developers should also monitor user authentication and authorization activities for suspicious behaviour, such as multiple login attempts from different locations or devices.

In summary, Weak authentication can lead to severe security breaches if not properly addressed. Developers should ensure that robust authentication mechanisms are implemented and apply secure coding practices to protect against these vulnerabilities. By implementing the mentioned mitigation techniques, developers can protect against automated attacks, insecure password storage, and usage, as well as prevent unauthorized access to sensitive data or functionality.

Low Password Complexity Threshold

OK

~~5.7~~
~~MEDIUM~~ ~~CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:L/I:H/A:N~~

ID: SYSCOIN-PALI2025-002
Target: Browser Extension

Description

Cyrex has identified a flaw in which the wallet accepts weak passwords such as

`password1` . The current password check only looks for a minimum length of eight characters, along with at least one number. This light level of validation makes it easy for weak passwords to pass, which reduces the overall protection of private keys and wallet data.

Risk

Attackers can guess or brute force such passwords with little effort. This can expose sensitive wallet contents and may allow full access to stored keys, which puts user funds and account integrity at risk.

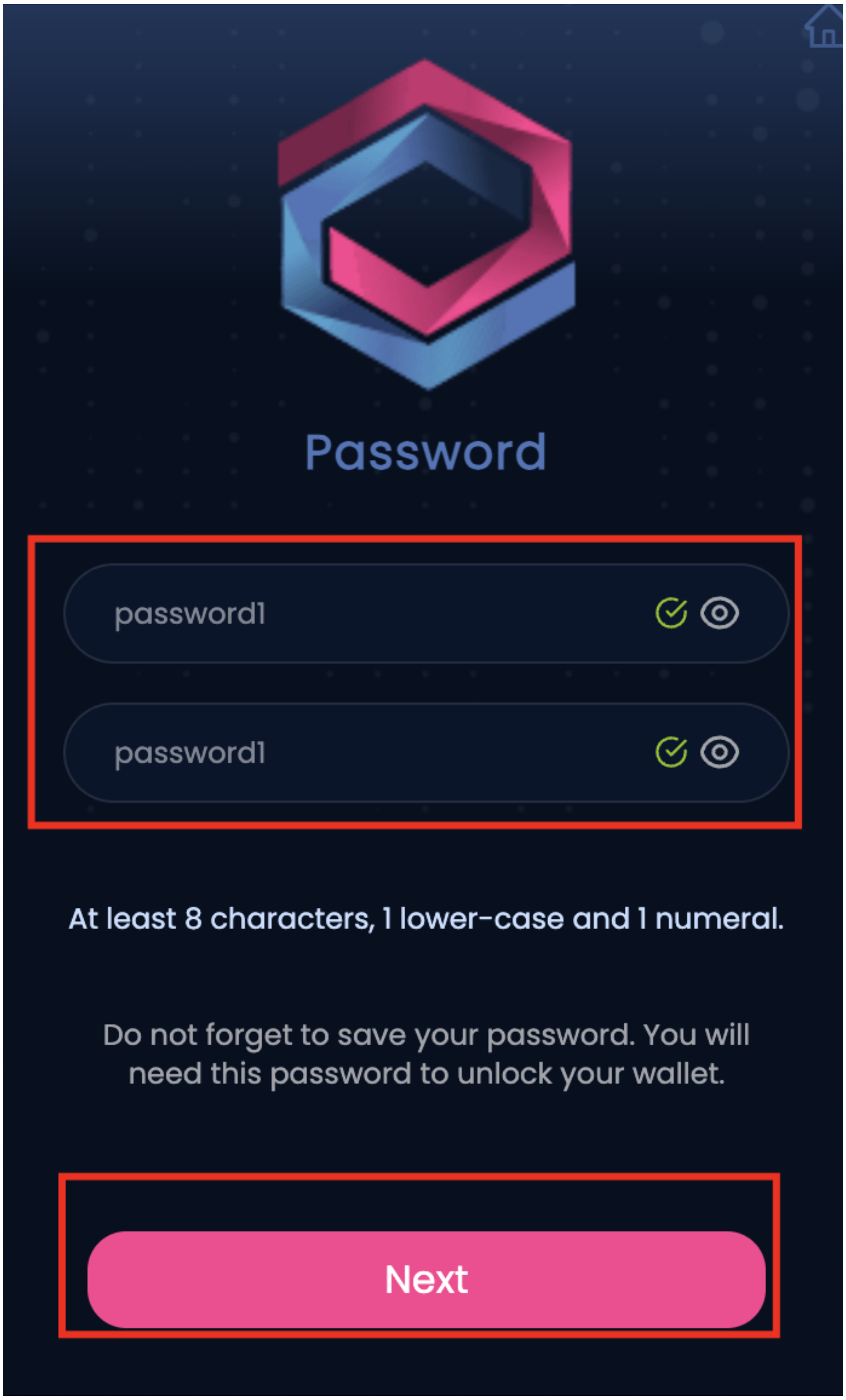
Recommendation

It is recommended to enforce a stronger password policy that requires a mix of uppercase letters, lowercase letters, numbers, and special characters. It is also recommended to introduce checks for common weak patterns to prevent users from setting predictable passwords.

Proof

1. Set up the wallet with a weak password such as

`password1` and observe that it is accepted.



Password

password1

password1

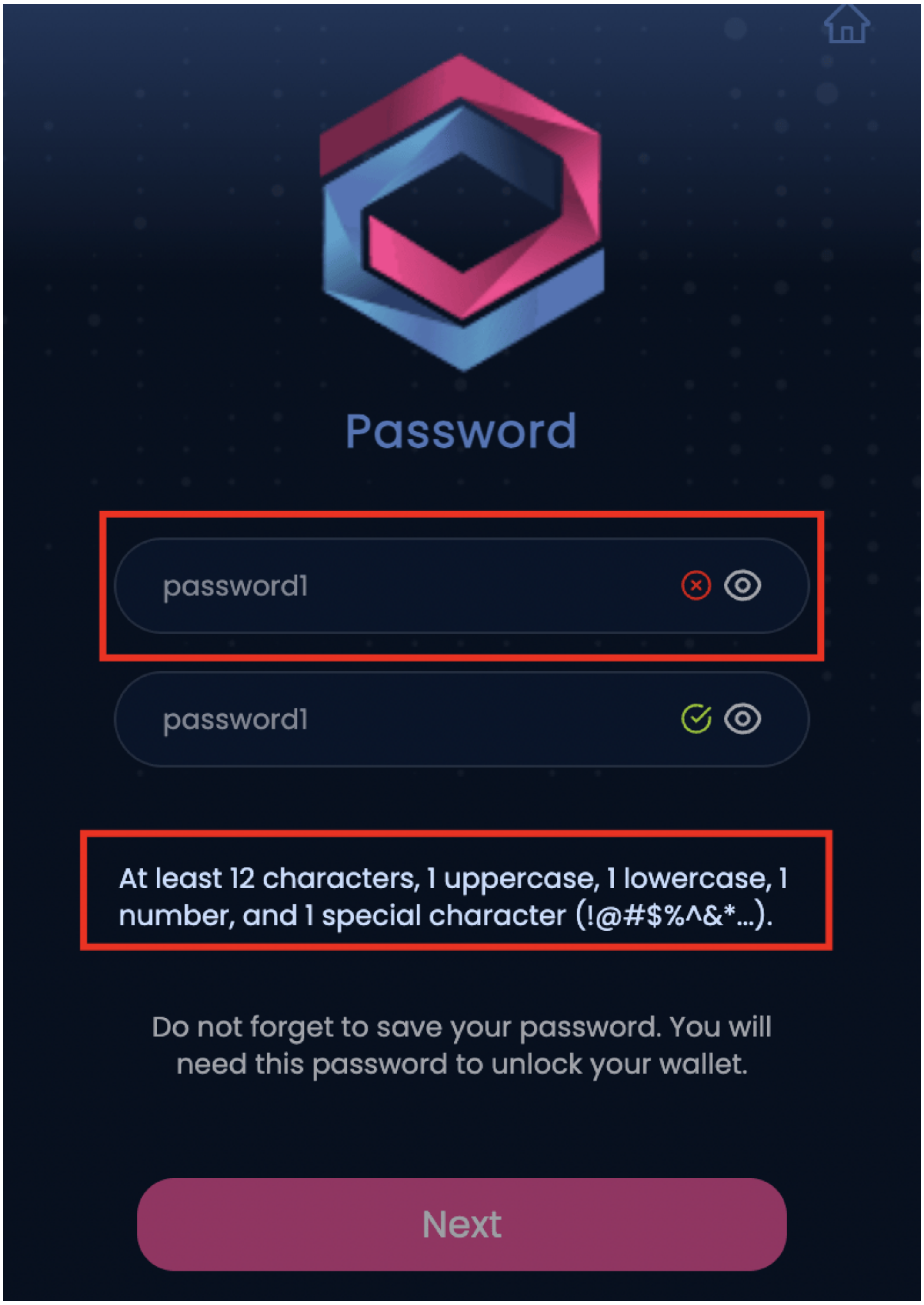
At least 8 characters, 1 lower-case and 1 numeral.

Do not forget to save your password. You will need this password to unlock your wallet.

Figure 6.11: Low Password Complexity Threshold

Retest 2026-01-06

OK



password1

password1

At least 12 characters, 1 uppercase, 1 lowercase, 1 number, and 1 special character (!@#$%^&*...).

Do not forget to save your password. You will need this password to unlock your wallet.

Figure 6.12: Enforcement of Strong Password Complexity

We would like to thank you on behalf of the entire CYREX team for the opportunity to work on your project and for placing your trust in our expertise.

Visit us at
cyrex.tech

Mail us
hello@cyrextech.net

Follow us on Linkedin
cyrextechnet

Tweet us
Cyrextech