CYREX

Penetration Test Report

—

Prepared for: **João Marcos Warmling Dudy, COO, Elint**
Prepared by: **Tim De Wachter**, CTO, *Cyrex Ltd*

24/09/2021

Table of Contents

Executive summary

Penetration test

Cyrex was contracted by Elint to conduct a penetration test in order 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 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 Elint provided Cyrex the source code, we can label this kind of test as a white box penetration test. Cyrex was granted access to the application with all regular user privileges.

Sanity and regression test

With the sanity and 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.

What follows is a conclusion concerning the overall security maturity of the application and the tested vulnerability types.

We are confident that the penetration test and this report helps the customer to raise its security of the Pali Wallet extension to a higher level, this by enforcing the principles of confidentiality, integrity and availability.

Conclusion

Overall

Cyrex determined that the overall security maturity of this application is great and will meet the risk appetite of any end user. All suggested patches were implemented in a correct manner but more importantly the browser extension was tested and validated thoroughly by Cyrex' application security experts.

Architecture

The Pali Wallet extension is developed using the cross-browser extension APIs and frameworks. Standard best practices when it comes to browser extensions have been implemented. Mitigating common security pitfalls during development and enforcing the security maturity of the application by design.

Security Controls

User input is one of the main injection points for malicious end-users for any application, Pali Wallet has applied validations for the parameters used within the exposed controller methods. These validations are handled in an effective and secure manner whenever the methods are tampered with. In this way, malicious payloads are rendered useless and will not affect the system or its data.

Next to this, only necessary data is returned from within the extension. This means that users are not able to gain access to private data that should remain secret within the wallet. A significant number of possible scenarios and use cases have been tested and we can state that it is not possible for a user to unintentionally view or modify data within the wallet without access to an unlocked wallet of the user. In this way, the end-user's data remains confidential and integer at all times.

Layered security is implemented in the application, which is the recommended security best-practice for any application, meaning whenever a new vulnerability would be exploited it will not impact the end-user as much as it potentially could, or it will be a lot harder for a malicious actor to determine a new vulnerability within the target scope. A good example of this is the lockout timer that has been implemented.

Scope of test

Cyrex performed a penetration test on the Pali Wallet extension, this test was performed starting the 1st of September 2021 up till and including the 3th of September 2021.

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.

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

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

We are confident the maturity level of the application meets the security requirements of any end user; therefore, the application can be publicly exposed and be put into a production environment.

We want to thank Elint for putting trust in our knowhow and expertise concerning ethical hacking specific to applications.

Fixed vulnerabilities

This section lists all the vulnerabilities that Cyrex determined to be successfully fixed within the scope of this sanity and regression test.

ID	Title
PW001	xpriv and encriptedMnemonic are disclosed in getWalletState()
PW002	Call getWalletState() on locked and/or disconnected wallets
PW003	Wallet password stored in plaintext within memory
PW004	xprv stored unencrypted in localStorage
PW005	connectedTo array disclosed on getWalletState()
PW006	connectedTo array disclosed on getConnectedAccount() call
PW007	Spoof token symbol when sending tokens
PW008	Account connected before confirmation
PW009	Prototype pollution
PW010	DOS when providing incorrect parameters on signPSBT()
PW011	DOS when providing incorrect parameters on signAndSend()
PW012	xprv is encrypted in storage using an unsecure, static key
PWR001	Auto-Lock Timer
PWR002	Use storage API instead of localStorage to save sensitive data
PWR003	Show token symbol when issuing tokens
PWR004	Show asset GUID when transferring tokens
PWR005	Show NFT name when minting an NFT
PWR006	Show NFT name when transfering NFT

Tested Vulnerability Types

The application has been tested for the following types of vulnerabilities:

Remote Code Execution (RCE)

In computer security, arbitrary code execution (ACE) is an attacker's ability to execute arbitrary commands or code on a target machine or in a target process. An arbitrary code execution vulnerability is a security flaw in software or hardware allowing arbitrary code execution. The ability to trigger arbitrary code execution over a network (especially via a wide-area network such as the Internet) is often referred to as **remote code execution** (RCE).

On its own, an arbitrary code execution exploit will give the attacker the same privileges as the target process that is vulnerable. For example, if exploiting a flaw in a web browser, an attacker could act as the user, performing actions such as modifying personal computer files or accessing banking information, but would not be able to perform system-level actions (unless the user in question also had that access).

SQL Injection

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can **read** sensitive **data** from the database, **modify** database **data**, execute administration operations on the database, recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.

SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

The main consequences of SQL Injection vulnerabilities are:

- Loss of confidentiality, since the database generally holds sensitive data.
- No limitation to authentication
- No limitation to authorization and privileges
- Loss of integrity due to modification of data

Access Control Flaws

Within the application's core security mechanisms, access controls are logically built upon authentication and session management. The application needs a way of deciding whether it should **permit** a given request to perform its attempted action or access the resources that it is requesting.

Access controls are a **critical defence mechanism** within the application because they are responsible for making these key decisions.
When they are defective, an attacker can often:
- Compromise the entire application
- Take control of administrative functionality
- Access sensitive data belonging to every other user.

Broken access controls are among the most commonly encountered categories of web application vulnerabilities.

Brute Force Attacks

A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analysing the response. For the sake of efficiency, an attacker may use a dictionary attack or a traditional brute-force attack.

Brute-force attacks are often used for attacking authentication and discovering content/pages within a web application. These attacks are usually sent via GET and POST requests to the server. In regard to authentication, brute force attacks are often mounted when an account lockout policy is not in place.

Cross-Site Scripting

XSS is a vulnerability that lets an attacker control some of the content of a web application. By exploiting a Cross Site Scripting, the attacker can target the web application users.

By performing an XSS attack an attacker is able to:
- Modify the content of the site at run-time
- Inject malicious contents.
- Steal the cookies, thus the session, of a user.
- Perform actions on the web application as if it was a legitimate user.
- …

A vulnerable web application is what makes XSS attacks possible.
XSS vulnerabilities happen when a web application uses **unfiltered user input** to build the output content displayed to its end users.

This lets an attacker control the output HTML and JavaScript code, thus attacking the application users.

Broken Authentication

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords;
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords (see A3:2017-Sensitive Data Exposure).
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login.
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

Information Disclosure

Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Applications can also leak internal state via how long they take to process certain operations or via different responses to differing inputs, such as displaying the same error text with different error numbers.

Web applications will often leak information about their internal state through detailed or debug error messages. Often, this information can be leveraged to launch or even automate more powerful attacks.

Denial of Service (DoS)

The **Denial of Service (DoS)** attack is focused on making a resource (site, application, server) unavailable for the purpose it was designed. There are many ways to make a service unavailable for legitimate users by manipulating network packets, programming, logical, or resources handling vulnerabilities, among others. If a service receives a very large number of requests, it may cease to be available to legitimate users. In the same way, a service may stop if a programming vulnerability is exploited, or the way the service handles resources it uses.

Sometimes the attacker can inject and execute arbitrary code while performing a DoS attack in order to access critical information or execute commands on the server. Denial-of-service attacks significantly degrade the service quality experienced by legitimate users. These attacks introduce large response delays, excessive losses, and service interruptions, resulting in direct impact on availability.

Business Logic Flaws

Most security problems are weaknesses in an application that result from a broken or missing security control (authentication, access control, input validation, etc. ...). By contrast, business logic flaws are ways of using the legitimate processing flow of an application in a way that results in a negative consequence to the organization.

Unrestricted File Upload

Uploaded files pose a significant risk if not handled correctly. The consequences of unrestricted file upload van vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks or simple defacement, dependent on what the application does with the uploaded file and especially where it is stored.

The impact of this vulnerability is high, supposed code can be executed in the server context or on the client side. The likelihood of detection for the attacker is high. The prevalence is common. As a result, the severity of this type of vulnerability is high.

Server-Side Request Forgery (SSRF)

In a Server-Side Request Forgery (SSRF) attack, the attacker can abuse functionality on the server to read or update internal resources.

The attacker can supply or modify a URL which the code running on the server will read or submit data to, and by carefully selecting the URLs, the attacker may be able to read server configuration such as AWS metadata, connect to internal services like http enabled databases or perform post requests towards internal services which are not intended to be exposed.

Improper Session Validation

In order to keep the authenticated state and track the users progress within the web application, applications provide users with a session identifier (session ID or token) that is assigned at session creation time and is shared and exchanged by the user and the web application for the **duration** of the session

Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack. This can be seen as a control against preventing other attacks like Cross Site Scripting and Cross Site Request Forgery. Such attacks have been known to rely on a user having an authenticated session present. Not having a secure session termination only increases the attack surface for any of these attacks.

A secure session termination requires at least the following components:
- Availability of user interface controls that allow the user to manually log out
- Session termination after a given amount of time without activity (session timeout).
- Proper invalidation of server-side session state

Security Misconfigurations

Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc

Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise.

The business impact depends 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, etc to gain unauthorized access or knowledge of the system.

Open Redirect Flaws

Open redirect is a security flaw in an app or a web page that causes it to redirect users to potentially malicious URLs

When apps and web pages have requests for URLs, they are **supposed verify that those URLs are part of the intended pages domain**. Open redirect is a failure in that process that makes it possible for attackers to steer users to malicious third-party websites. Sites or apps that fail to authenticate URLs can become a vector for malicious redirects to convincing fake sites for identity theft or sites that install malware.

Normally, redirection is a technique for shifting users to a different web page than the URL they requested. Webmasters use redirection for valid reasons, such as dealing with resources that are no longer available or have been moved to a different location.

Improper Input Validation

Input validation is a frequently used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe processing within the code, or when communicating with other components. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

It is important to emphasize that the distinctions between input validation and output escaping are often blurred, and developers must be careful to understand the difference, including how input validation is not always sufficient to prevent vulnerabilities, especially when less stringent data types must be supported, such as free-form text.

Path Traversal Attacks

A path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the web root folder. By manipulating variables that reference files with dot-dot-slash (../) sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files.

JSON Injection

JSON injection occurs when data enters a program from an untrusted source, or the data is written to a JSON stream.

Applications typically use JSON to store data or send messages. When used to store data, JSON is often treated like cached data and may potentially contain sensitive information. When used to send messages, JSON is often used in conjunction with a RESTful service and can be used to transmit sensitive information such as authentication credentials.

The semantics of JSON documents and messages can be altered if an application constructs JSON from unvalidated input. In a relatively benign case, an attacker may be able to insert extraneous elements that cause an application to throw an exception while parsing a JSON document or request. In a more serious case, such as ones that involves JSON injection, an attacker may be able to insert extraneous elements that allow for the predictable manipulation of business-critical values within a JSON document or request. In some cases, JSON injection can lead to cross-site scripting or dynamic code evaluation.

XML Injection

XML injection manipulates or compromises the logic of an XML application or service. The injection of unintended XML content and/or structures into an XML message can alter the intended logic of an application, and XML Injection can cause the insertion of malicious content into resulting messages/documents.

With a successful XML Injection attack, the attacker can steal the entire database, or can even log in as the administrator of the website.

SMTP Header Injection

SMTP header injection vulnerabilities arise when user input is placed into email headers without adequate sanitization, allowing an attacker to inject additional headers with arbitrary values. This behaviour can be exploited to send copies of emails to third parties, attach viruses, deliver phishing attacks, and often alter the content of emails. It is typically exploited by spammers looking to leverage the vulnerable company's reputation to add legitimacy to their emails.

This issue is particularly serious if the email contains sensitive information not intended for the attacker, such as a password reset token.