IOTA Foundation Trinity Wallet Security Assessment

SIXGEN

A FULL-SPECTRUM CYBER SOLUTIONS COMPANY

Table of Contents

About SIXGEN

- SIXGEN cyber red teams provide real-world attack emulation designed to assess and improve the effectiveness of an entire information security program.
- Our goal is for an organization to understand and limit cybersecurity risk to organizational operations, assets, and individuals.
- Team members include veterans from the world's premier military and cybersecurity organizations.

VETERAN-OWNED SMALL BUSINESS (VOSB)
CAGE: 677F0
DUNS: 079336791
TOP-SECRET Facilities Clearance
SIXGEN, Inc.
844 West St. | Ste 200
Annapolis, MD 21401
www.SIXGEN.io

Assessment Information

SIXGEN conducted a security assessment of the desktop and mobile Trinity Wallet for the IOTA foundation from 02/18/19 to 03/15/19. The purpose of this testing was to evaluate and test risk associated with the Trinity wallet by emulating a real-world attacker.

Client Contact:

Mathew Yarger
Technical Lead and Security Specialist
Social Impact and Regulatory Affairs Team
The IOTA Foundation
+1 (443) 994-9110
mathew.yarger@iota.org

Lead Assessor:

Erik Hunstad
Adversary Emulation Lead
SIXGEN, Inc.
844 West St. | Ste 200
Annapolis, MD 21401
(443) 440-5034
erik@sixgen.io

Engagement Overview

SIXGEN conducts penetration tests, adversary emulation, and external exposure assessments. At SIXGEN, we specialize in manual assessments that go beyond basic automated tests to identify real attack vectors that can be used against your application or environment.

With decades of combined offensive experience, SIXGEN is at the forefront of application security, cloud security, and penetration testing. With a veteran team of subject matter experts, we staff experts that are authorities in their field.

Service Description

This security assessment focused on emulating real-world attacks using the same techniques as malicious actors. The Trinity wallet security assessment focused on the most likely and most dangerous attacks as well as a code review to identify any architectural weaknesses in the wallet.

Engagement Objectives

SIXGEN's consultants used the results of automated scanning tools, paired with their expert knowledge and experience to conduct a manual security analysis of the Trinity wallet. Our assessors attempted to exploit and gain unauthorized access to data through misconfigurations and exploits. The detailed results of the vulnerability scanning, manual testing, and code review are shown in this report.

Process and Methodology

SIXGEN used a comprehensive methodology to provide a security review of the Trinity wallet on all three major operating system (Windows, macOS, and Linux) as well as the mobile Trinity wallet on both iOS and Android. This process begins with detailed scanning and research into the architecture and environment, with automated testing for known vulnerabilities. Manual reconnaissance and tailored exploitation of weaknesses followed.

1. **Reconnaissance**

   The primary goal in this phase is to discover how the Trinity wallet processes and stores sensitive data (e.g. the user's seed) and how Trinity interacts with the operating system and network. This information provides the foundation for a tailored security assessment. Reconnaissance is carried out via automated scanners, dynamic analysis, and manual code review.

2. **Exploration and Verification**

   SIXGEN consultants use the results of the reconnaissance phase, paired with their expert knowledge and experience to conduct a manual security analysis of the Trinity wallet. The detailed results of this manual testing are shown in the tables in this report.

3. **Assessment Reporting**

   Throughout the engagement, SIXGEN delivered detailed analysis and threat reporting to the developers at the IOTA Foundation, including remediation steps. Our consultants set and industry standard for clear and concise reports, prioritizing the highest risk vulnerabilities first.

4. **Remediation Assessment**

   SIXGEN provided remediation retesting for all vulnerabilities reported as they were addressed. At the conclusion of the remediation testing the risk level determination for this report was made and all issues in this report were remediated or were being tracked for remediation.

Scoping and Rules of Engagement

While malicious actors have no limits on their actions, SIXGEN understands the need to scope assessments to complete the assessment in a timely manner and protect third parties not participating in the engagement. The following limitations were placed upon this engagement, as agreed upon by the IOTA Foundation:

- The Trinity wallet on Windows, macOS, Linux, iOS, and Android was in scope
- Network communications made by the Trinity wallet were in scope (e.g. tampered IRI responses)
- The IOTA network (e.g. iota.org and IRI nodes) was out of scope
- The IOTA ledger (“tangle”) and IOTA token were out of scope

Executive summary

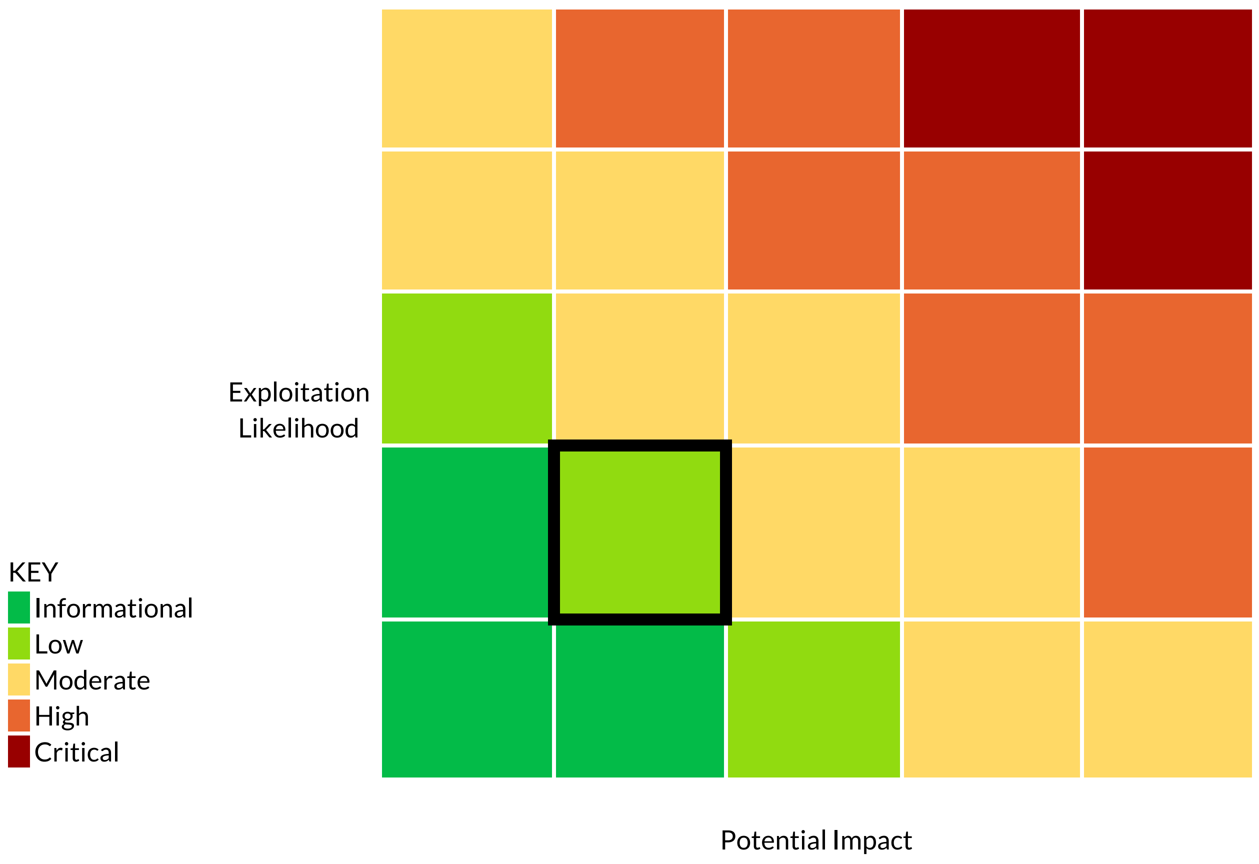
SIXGEN conducted a security assessment of the Trinity wallet for the IOTA Foundation. This test was performed to assess the defensive posture of the Trinity wallet and provide security assistance through proactively identifying vulnerabilities, validating their severity, and providing remediation steps to IOTA Foundation developers.

SIXGEN reviewed the security of the Trinity wallet and has determined a low risk of compromise from external attackers, as shown by the vulnerabilities detailed in this report. The detailed findings and remediation recommendations for these vulnerabilities may be found later in this report.

Trinity Wallet Risk Rating

SIXGEN calculates the risk to the Trinity wallet based on exploitation likelihood (ease of exploitation) and potential impact (potential business impact to the environment).

Overall Risk Rating: 🟩 Low



Summary of Strengths

While SIXGEN was tasked with finding issues and vulnerabilities in the Trinity wallet, it is useful to know when positive findings appear. Understanding the strengths of the Trinity wallet can reinforce security best practices and provide strategy and direction toward a robust defensive posture. The following traits were identified as strengths in the Trinity wallet:

1. **Trinity utilizes modern cryptography (e.g. Argon2i, AES-GCM) which protects secrets from brute force attacks.**
2. **Trinity is memory conscious, and clears secrets from memory when they are not needed to perform operations.**
3. **Trinity implements user education when enabling convenience features that may expose users to additional risks.**
4. **The Trinity development team considers security first when adding features to the wallet.**

Summary of Weaknesses

SIXGEN discovered and investigated multiple low and informational level vulnerabilities during the course of its assessment of the Trinity wallet. These vulnerabilities are categorized into general weaknesses below, and the IOTA Foundation team has implemented mitigations for these weaknesses or they are tracked for remediation.

1. **Like all complex software projects, the Trinity wallet relies on software packages from sources outside the IOTA Foundation. Due to the nature of the node.js software ecosystem, if any of these packages were to be compromised without detection (a “software supply chain” attack) all Trinity wallets could potentially be compromised.**
2. **The use of Electron introduces application integrity risks on some operating systems.**
3. **The use of convenience features such as deep links (e.g. links that begin with iota://) potentially exposes a Trinity user to additional phishing schemes.**
4. **Running any cryptocurrency wallet on a computer or device that may be compromised is a risk that needs to be understood by users.**

Assessment Narrative

This assessment included a source code review focused on code quality and exploit mitigation as well as a mobile application and Electron application security review covering the following key areas:

- Local Data Storage
- Network Communication
- Authentication and Authorization
- Interaction with the Operating System and Electron Security
- Environmental Resilience and Code Integrity
- Cryptography
- User Interface Security
- Repackaging Attacks

The assessment was conducted against the desktop wallet `develop` branch commit `9c6811a9033ebc61983a9adab12339776f13dd8e` (February 21, 2019), desktop version 0.4.6 (for integrity and signing verification), iOS Version 0.6.2 (42), and Android version 0.6.2 (49-1).

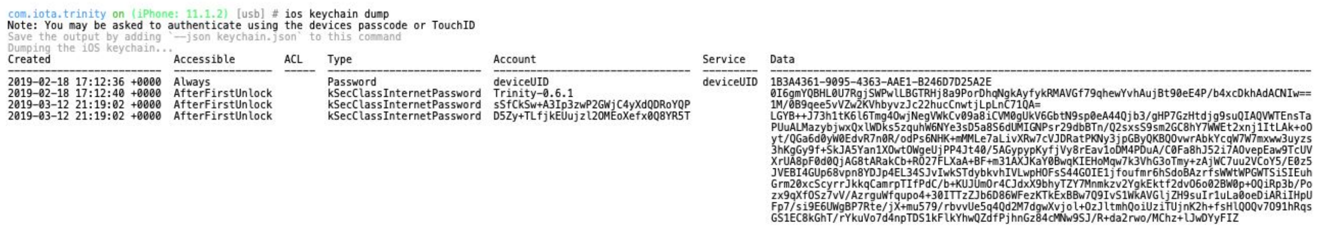
Throughout the assessment special consideration and focus was given to external attack vectors that could potentially compromise the Trinity wallet. Specifically the following were thoroughly investigated:

- Malicious SeedVaults
- Malicious Deeplinks/Deeplink Hijacking
- Seed Generation Randomness Issues
- Malicious IRI Responses
- Automatic Update Interception/Spoofing
- Application Modification
- Software Supply Chain Attacks

Local Data Storage

The secure storage of sensitive data such as user credentials or private keys is crucial to a wallet's security. Applications can accidentally expose sensitive data to cloud storage, backups, or the keyboard cache. Additionally, due to their size and portability, mobile devices are easily lost or stolen and an attacker may have physical access to a mobile device.

Trinity properly encrypts all sensitive locally stored data across all platforms. On macOS encrypted data is managed by the Keychain, on Linux it is managed by the Secret Service API/libsecret, and on Windows it is managed by Credential Vault. On iOS, trinity limits access to locally stored data with a keychain security group, preventing other applications from accessing the encrypted data. On Android, sensitive data was stored encrypted in a kdbx file.



```
com.iota.trinity on (iPhone: 11.1.2) [usb] # ios keychain dump
Note: You may be asked to authenticate using the devices passcode or TouchID
Save the output by adding `--json keychain.json` to this command
Dumping the iOS keychain...
Created                    Accessible        ACL    Type                        Account                           Service    Data
-------------------------  ----------------  -----  --------------------------  --------------------------------  ---------  ------------------------------------------------------------------------------
2019-02-18 17:12:36 +0000  Always                   Password                    deviceUID                         deviceUID  1B3A4361-9095-4363-AAE1-B246D7D25A2E
2019-02-18 17:12:40 +0000  AfterFirstUnlock         kSecClassInternetPassword   Trinity-0.6.1                                0I6gmYQBHL0U7RgjSWPwlLBGTRHj8a9PorDhqNgkAyfykRMAVGf79qhewYvhAujBt90eE4P/b4xcDkhAdACNIw==
2019-03-12 21:19:02 +0000  AfterFirstUnlock         kSecClassInternetPassword   sSfCkSw+A3Ip3zwP2GWjC4yXdQDRoYQP             1M/0B9qee5vVZw2KVhbyvzJc22hucCnwtjLpLnC71QA=
2019-03-12 21:19:02 +0000  AfterFirstUnlock         kSecClassInternetPassword   D5Zy+TLfjkEUujzl2OMEoXefx0Q8YR5T             LGYB++J73h1tK6l6Tmg4OwjNegVWkCv09a8iCVM0gUkV6GbtN9sp0eA44Qjb3/gHP7GzHtdjg9suQIAQVWTEnsTa
                                                                                                                             PUuALMazybjwxQxlWDks5zquhW6NYe3sD5a8S6dUMIGNPsr29dbBTn/Q2sxsS9sm2GC8hY7WWEt2xnj1ItLAk+oO
                                                                                                                             yt/QGa6d0yW0EdvR7n0R/odPs6NHK+mMMLe7aLivXRw7cVJDRatPKNy3jpGByQKBQOvwrAbkYcqW7W7mxww3uyzs
                                                                                                                             3hKgGy9f+SkJA5Yan1XOwtOWgeUjPP4Jt40/5AGypypKyfjVy8rEav1oDM4PDuA/C0Fa8hJ52i7AOvepEaw9TcUV
                                                                                                                             XrUA8pF0d0QjAG8tARakCb+RO27FLXaA+BF+m31AXJKaY0BwqKIEHoMqw7k3VhG3oTmy+zAjWC7uu2VCoY5/E0z5
                                                                                                                             JVEBI4GUp68vpn8YDJp4EL34SJvIwkSTdybkvhIVLwpHOFsS44GOIE1jfoufmr6hSdoBAzrfsWWtWPGWTSiSIEuh
                                                                                                                             Grm20xcScyrrJkkqCamrpTIfPdC/b+KUJUmOr4CJdxX9bhyTZY7Mnmkzv2YgkEktf2dvO6o02BW0p+OQiRp3b/Po
                                                                                                                             zx9qXfOSz7vV/AzrguWfqupo4+30ITTzZJb6D86WFezKTkExBBw7Q9IvS1WkAVGljZH9suIr1uLa0oeDiARiIHpU
                                                                                                                             Fp7/si9E6UWgBP7Rte/jX+mu579/rbvvUe5q4Qd2M7dgwXvjol+OzJltmhQoiUziTUjnK2h+fsHlQOQv7O91hRqs
                                                                                                                             GS1EC8kGhT/rYkuVo7d4npTDS1kFlkYhwQZdfPjhnGz84cMNw9SJ/R+da2rwo/MChz+lJwDYyFIZ
```

Figure 1: Encrypted Trinity Keychain Data from a Jailbroken iPhone

The proper use of encryption and the use of key storage APIs provided by the operating systems mitigates the risk of a malicious actor obtaining sensitive data store locally on a device.

Network Communication

End user devices, and mobile devices especially, communicate over a variety of networks, including those potentially shared with malicious actors. This reality creates the opportunity for a malicious actor to conduct a variety of network based attacks including man in the middle attacks where the malicious actor is able to inspect and modify all network traffic from the application to any endpoint.

All Trinity network communication is TLS encrypted on all platforms, which prevents interception and man-in-the-middle attacks. To emulate a malicious node on the IOTA network the desktop Trinity wallet was modified to disable TLS certificate checking and send all traffic through a local proxy. Using this proxy, assessors modified responses from non-malicious endpoints in an attempt to compromise the Trinity wallet. The Trinity wallet properly handled malicious traffic without issue and switched nodes or repeated API requests until valid responses were returned.

This use of standard cryptographic protocols (i.e. TLS) that ensure the identity of the remote endpoint and confidentiality of information passed over the network as well as proper error handling and retransmission mitigates the risk of a malicious actor being able to intercept or modify network traffic to affect the Trinity wallet.

Authentication and Authorization

Authentication and authorization are the foundation of most applications, and the Trinity wallet is no different. In order to protect the user's funds, the Trinity wallet forces a user to provide a wallet password before being able to access the functionality of the application. If this authentication check can be bypassed, a malicious actor could access user funds.

Trinity properly implements secure secret storage as well as secure coding practices that prevent access to the wallet without the proper user supplied password. One low risk issue found with the Trinity implementation of authentication is the way two factor authentication (2FA) works. Trinity supports 2FA, but decrypts the user's vault in memory before checking if 2FA is required (L1). This effectively prevents a malicious actor with local access to Trinity and a user's password from accessing sensitive data, but would not prevent a sophisticated malicious actor who is able to extract sensitive data from the Trinity processes' memory using only the wallet password, even if 2FA is enabled.

It is important to discuss the impossible task of securing an application running on a potentially compromised computing system. If a malicious actor is able to execute code on the same system a cryptocurrency wallet is running, no defensive mechanisms will be able to fully protect the sensitive data used by the wallet. Trinity stands out as taking many procations to ensure that even in the event of a system compromise, sensitive data is protected as much as possible. When using or storing significant amounts of cryptocurrency, a separate known-good system or hardware wallet where the private key never leaves the devices and requires physical interaction to confirm transactions are the best ways to keep funds secure.

Trinity has the ability to interface with the Ledger hardware wallet. This device generates and stores the seed as well as signs transactions on the device itself. With no key material on the user's computer to steal, a malicious actor would have to exploit the Ledger device itself in order to access cryptographic secrets and ultimately the user's funds.

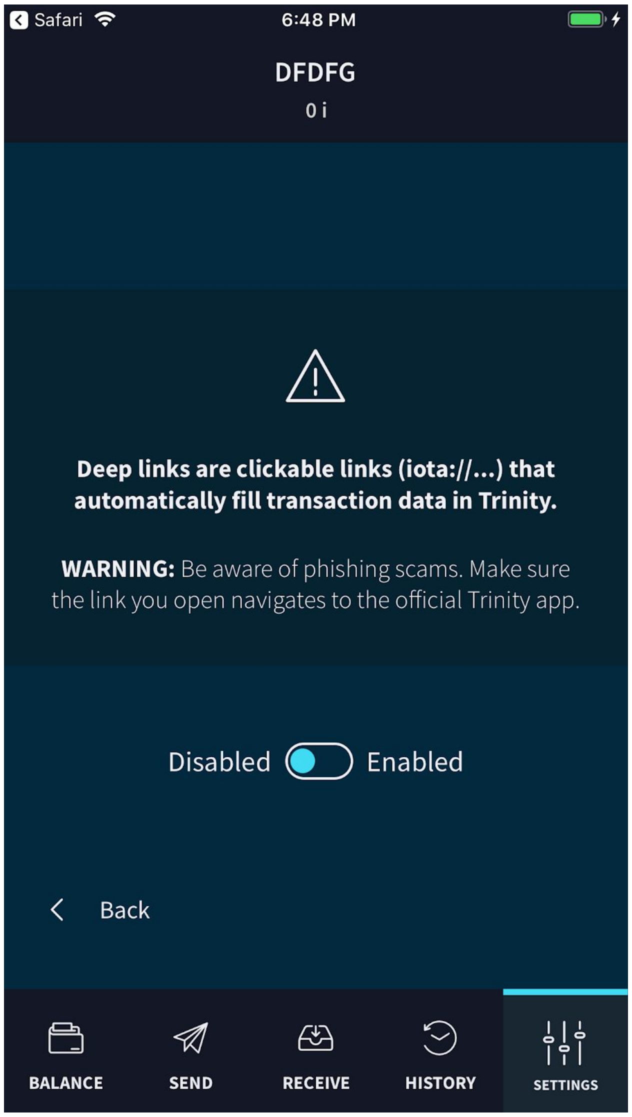
Interaction with the Operating System and Electron Security

The Trinity wallet is designed to work on multiple operating systems and while this feature allows for maximum user reach, it presents unique security concerns depending on the user's operating system. For example, while iOS forces applications to run in a sandbox, the Trinity Electron desktop application does not have that same base level of security provided by the operating system.

The Electron framework presents unique challenges where a Cross-Site Scripting (XSS) vulnerability can quickly become a Remote Code Execution (RCE) vulnerability[1]. The IOTA Foundation understands the risks of the Electron framework, and `nodeIntegration` and `auxclick` have been disabled for all webviews in Trinity. Trinity also disables the `eval` function globally, and has implemented a whitelist of allowed hostnames that can be opened as part of URLs by Trinity. To complete the security lockdown of Trinity, sandboxing and context isolation could be enabled in `webPreferences` (L2).

Trinity has the ability to launch and auto-populate transaction when a user clicks a link that begins with `iota://`. This functionality is called "deep-linking" and adds convenience but can also be a target for phishing attacks. Additionally, any application can register as the handler for a deep-link protocol. When enabling deep-links, Trinity appropriately warns users of the risks.

[1] https://statuscode.ch/2017/11/from-markdown-to-rce-in-atom



Deep links are clickable links (iota://...) that automatically fill transaction data in Trinity.

WARNING: Be aware of phishing scams. Make sure the link you open navigates to the official Trinity app.

Disabled ● Enabled

Figure 2: Deep-link Risk Warning

Environmental Resilience and Code Integrity

The Trinity wallet may be used on a computer or mobile device that has been previously compromised by a malicious actor. In order to maximize user trust, the Trinity wallet should provide reasonable resilience against a malicious actor with access to the operating system.

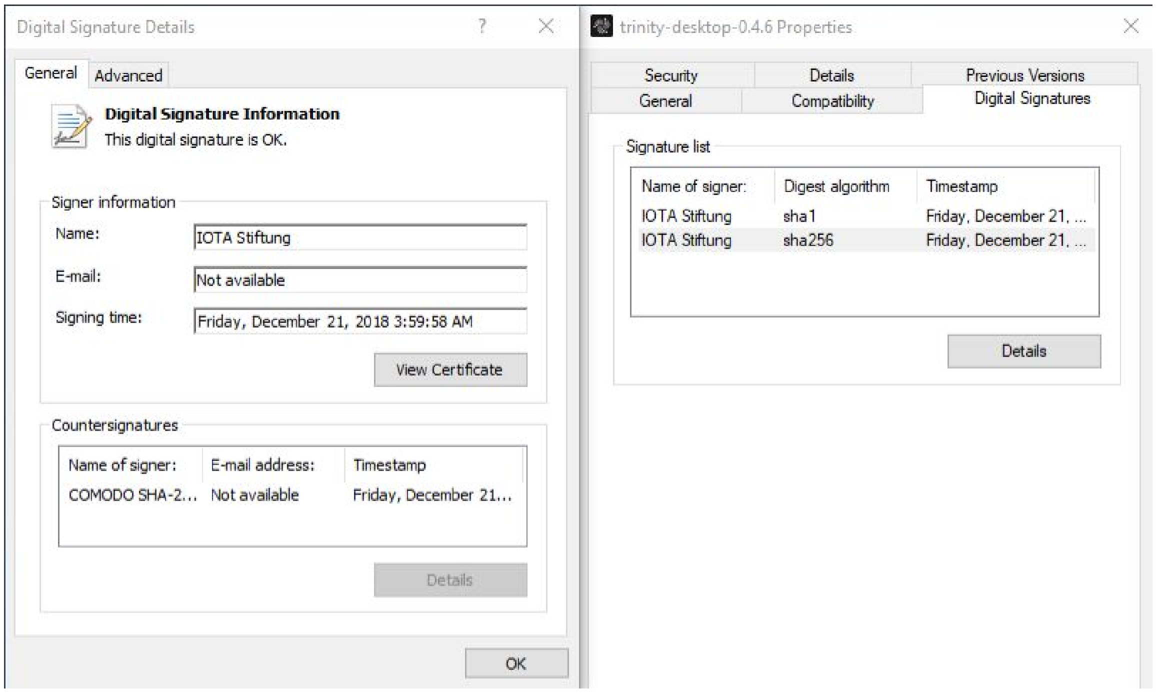
The desktop Trinity wallet utilizes the Electron framework, which packages application code into an asar archive. This archive contains all the application code, and if modified by a malicious actor could compromise the integrity of the wallet.

Additionally, the IOTA Foundation signs each binary release of the Trinity wallet. A summary of asar integrity verification and binary signing across the desktop operating systems can be seen below

OS	Binary Signed?	ASAR Integrity Verification?
Windows	Yes	No
macOS	Yes	Yes
Linux	Yes, external	Yes, external

Table 1: Binary Signing and ASAR Verification Status

On Windows, the Trinity binary is signed however there is no ASAR integrity verification. What this means in practice is that even if a user checks the signature status of Trinity manually and it is valid, there is no guarantee that the application has not been modified (I1). It is important to note that this issue is not unique to Trinity, but is an issue with all applications created using the Electron framework.


Digital Signature Details

General | Advanced

Digital Signature Information
This digital signature is OK.

Signer information

Name:	IOTA Stiftung
E-mail:	Not available
Signing time:	Friday, December 21, 2018 3:59:58 AM

Countersignatures

Name of signer:	E-mail address:	Timestamp
COMODO SHA-2...	Not available	Friday, December 21...

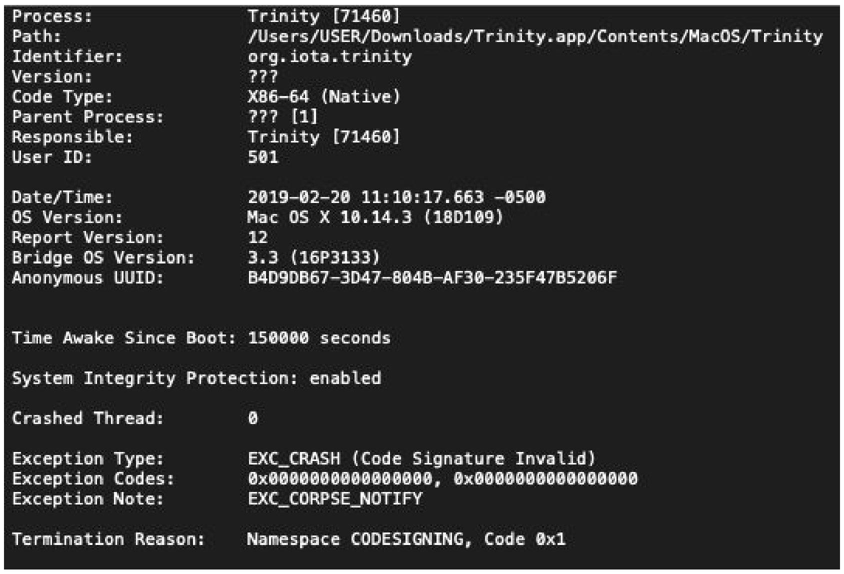
trinity-desktop-0.4.6 Properties

Signature list

Name of signer:	Digest algorithm	Timestamp
IOTA Stiftung	sha1	Friday, December 21, ...
IOTA Stiftung	sha256	Friday, December 21, ...

Figure 3: Binary Signing on Windows

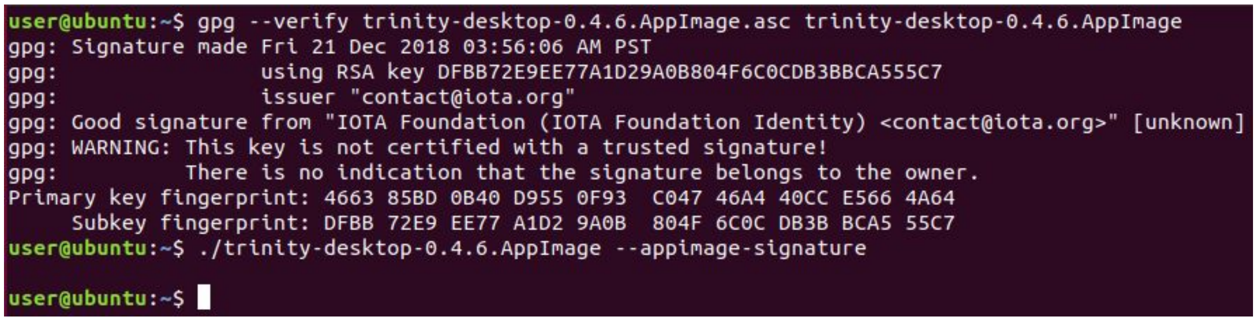
On macOS, any modification to the asar will cause the application to fail to launch. This code signing enforcement is built into the operating system. However, if a malicious actor can completely replace Trinity with an unsigned malicious version, macOS will not prevent it from launching.



```
Process:               Trinity [71460]
Path:                  /Users/USER/Downloads/Trinity.app/Contents/MacOS/Trinity
Identifier:            org.iota.trinity
Version:               ???
Code Type:             X86-64 (Native)
Parent Process:        ??? [1]
Responsible:           Trinity [71460]
User ID:               501

Date/Time:             2019-02-20 11:10:17.663 -0500
OS Version:            Mac OS X 10.14.3 (18D109)
Report Version:        12
Bridge OS Version:     3.3 (16P3133)
Anonymous UUID:        B4D9DB67-3D47-804B-AF30-235F47B5206F


Time Awake Since Boot: 150000 seconds

System Integrity Protection: enabled

Crashed Thread:        0

Exception Type:        EXC_CRASH (Code Signature Invalid)
Exception Codes:       0x0000000000000000, 0x0000000000000000
Exception Note:        EXC_CORPSE_NOTIFY

Termination Reason:    Namespace CODESIGNING, Code 0x1
```

Figure 4: Binary Signing Enforcement on macOS

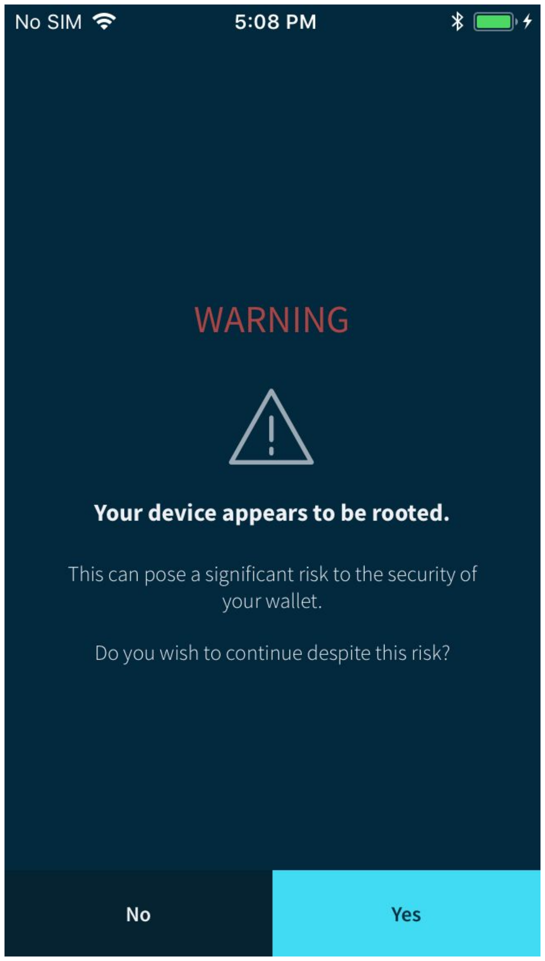
The Linux Trinity binary is signed using a GPG signature file, which must be externally verified by the user after importing the IOTA Foundation GPG key. Trinity is distributed as an AppImage on Linux, which has the ability to include an embedded signature. At the time of this report, Trinity did not include an embedded signature, but the IOTA Foundation was investigating adding one (I2).



```
user@ubuntu:~$ gpg --verify trinity-desktop-0.4.6.AppImage.asc trinity-desktop-0.4.6.AppImage
gpg: Signature made Fri 21 Dec 2018 03:56:06 AM PST
gpg:                using RSA key DFBB72E9EE77A1D29A0B804F6C0CDB3BBCA555C7
gpg:                issuer "contact@iota.org"
gpg: Good signature from "IOTA Foundation (IOTA Foundation Identity) <contact@iota.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 4663 85BD 0B40 D955 0F93  C047 46A4 40CC E566 4A64
     Subkey fingerprint: DFBB 72E9 EE77 A1D2 9A0B  804F 6C0C DB3B BCA5 55C7
user@ubuntu:~$ ./trinity-desktop-0.4.6.AppImage --appimage-signature

user@ubuntu:~$ 
```

Figure 5: External Binary Signing on Linux

Trinity on mobile operating systems is secured in part by the sandboxing and permission systems of both iOS and Android. However, on jailbroken or rooted phones respectively, these security boundaries are no longer effective. Trinity appropriately warns the user when run on a jailbroken or rooted device that security may be negatively impacted.



Your device appears to be rooted.

This can pose a significant risk to the security of your wallet.

Do you wish to continue despite this risk?

Figure 6: Security Alert on a Jailbroken iPhone

Cryptography

The Trinity wallet holds a user's private key, which in turn allows full access to a user's IOTA tokens. The proper generation (to ensure a malicious actor cannot generate the same private key as a legitimate user on a separate device), and storage of private keys is vital to the success of the Trinity wallet.

Trinity uses modern cryptography, specifically the Argon2i algorithm for password hashing and AES-GCM for vault encryption. When creating a new seed, Trinity uses a cryptographically sound pseudo-random number generator (PRNG)[2].

An informational level bug was found in the `randomBytes` function that when called without a `max` value, there is a comparison of a number against `NaN` which always will evaluate false and produces the intended behavior but may be confusing (I4).

Updates to Trinity are conducted over TLS, and old versions are blacklisted to help push users to newer, potentially patched versions.

[2] https://developer.mozilla.org/en-US/docs/Web/API/Crypto/getRandomValues

Trinity makes explicit calls to the garbage collector after cryptographically sensitive data is no longer needed, which clears it from memory. This practice limits the exposure of sensitive data and makes it more difficult for a malicious actor to scrape program memory for a user's seed, password, or other sensitive data.

User Interface Security

A malicious actor with access to a device where the Trinity wallet is running should be limited from extracting sensitive data from the user interface.

Trinity properly masks password input fields across all platforms. Additionally, Trinity pushes users to store their seed in a SeedVault (kdbx file), which uses AES-GCM to protect the seed. This "secure default" behavior is a positive security conscious user interface design choice.

An informational issue on iOS is the ability for a user to screenshot their wallet's seed. If this image is transferred to an insecure location (e.g. compromised icloud, Google photos, etc) the user's funds would be at risk (I3). On Android, screenshots were prevented when using normal functionality of the phone itself or via an `adb` shell as the root user.

Repackaging Attacks

The Trinity wallet is distributed as both source code and a compiled binary. This presents the opportunity for a malicious actor to copy the application, add malicious code, and present it as authentic to users.

The IOTA Foundation distributes Trinity from https://trinity.iota.org, which links to binary releases hosted on github.com. Both sites use TLS to verify their authenticity and encrypt traffic. An informational level issue with the distribution of Trinity is the lack of obvious checksums, signing keys, or verification steps near the binary download buttons (I5). These items exist, and can be found by searching the GitHub releases page and the trinity documentation, but presenting them up front with the binaries will greatly increase the number of users that verify their binary downloads.

The IOTA Foundation uses automated build systems which incorporate Snyk and Dependabot. These products check Trinity software dependencies for vulnerabilities and keep them up to date. These tools help mitigate the risk of a "software supply chain" attack.

Ultimately, a malicious actor can easily copy the Trinity source code, add malicious code, and distribute an application that looks identical to the official Trinity release. The only way to prevent such an attack is through user education. Users should be warned that trinity.iota.org is the only official source for Trinity, and any other source should not be trusted.

Insiders at the IOTA Foundation could potentially be a threat if a single person could push an update to end users. Currently, the process to push a release of Trinity involves opening a pull request on Github which must be approved by another person before it can be merged and a build triggered. In the event this processes is circumvented, the Trinity development team is monitoring Github releases and would be able to take immediate action in the event of an unauthorized build being posted.

The use of automated build systems with published cryptographic material (hashes, signatures, and public keys), dependency management, and user education help mitigate the risk of repackaging attacks.

Vulnerability Overview

The following vulnerabilities were found within each risk level. It is important to note that the total number of vulnerabilities is not a factor in determining risk level. Risk level depends upon the severity of the vulnerabilities found.

Vulnerability ID - Description	**Risk Level**
L1 - 2FA enforced after vault decryption	Low
L2 - Sandboxing and Context Isolation not enabled in Web Preferences	Low
I1 - ASAR integrity is not verified on Windows	Informational
I2 - Linux AppImage does not include embedded GPG signature	Informational
I3 - iOS application allows screenshots	Informational
I4 - `randomBytes` called without `max` value compares against `NaN`	Informational
I5 - Checksums, GPG signature, and GPG key not easily available with binary downloads	Informational

Vulnerability Findings and Recommendations

L1 - 2FA Enforced After Vault Decryption

Risk Level: 🟩 Low

Description

When a user enables two factor authentication (2FA) in Trinity, their 2FA key is stored inside the encrypted vault to protect it. However, in order to check the 2FA code, the vault must be decrypted. Thus, the 2FA protection blocks the Trinity user interface from going forward without the correct 2FA code, but the user's vault is decrypted regardless of a correct 2FA being provided. This presents an issue of how to implement 2FA without a trusted third party having custody of a user's seed, which is not a good solution. A hardware device or mobile application that can implement universal second factor[3] (U2F) which uses challenge-response authentication using public key cryptography would be a more secure solution than the current time based one time password 2FA used in Trinity. Is is important to note that the current 2FA implementation does provide security against an unsophisticated attacker that has gained access to Trinity and a user's password, but not a user's 2FA device.

Remediation

Users should be aware of the limitations of the current 2FA implementation in Trinity. The IOTA Foundation is actively investigating U2F as a 2FA solution for future versions of Trinity and for the time being has removed 2FA until a more secure solution is found.

[3] https://en.wikipedia.org/wiki/Universal_2nd_Factor

L2 - Sandboxing and Context Isolation not enabled in Web Preferences

Risk Level: 🟩 Low

Description

Trinity desktop is an Electron framework application, which uses a chromium based browser to render user interface elements. This framework can expose unique vulnerabilities usually limited to web applications, and therefore has options to limit the capabilities of application windows. Trinity properly implements most of the available restrictions available in the Electron framework, but could increase its security posture by adding the following to `webPreferences`:

```
sandbox: true
contextIsolation: true
```
[4] [5]

Remediation

The additional security measures suggested above are designed to handle untrusted code, and while Trinity currently does not load any untrusted resources, if these options can be added without affecting the user experience they will further increase the security of the Trinity desktop wallet in the event a `BrowserWindow` is compromised.

[4] https://electronjs.org/docs/api/sandbox-option
[5] https://electronjs.org/docs/tutorial/security#isolation-for-untrusted-content

I1 - ASAR integrity is not verified on Windows

Risk Level: 🟩 Informational

Description

This is an issue for all Windows applications built using the Electron framework[6], and not due to any flaw in the code of the Trinity wallet. On windows, even when the Trinity wallet application binary is signed, a malicious user can modify the application's asar archive to add malicious code without affecting the binary signature or its ability to run.

Remediation

Users should be aware that application signing does not protect the Trinity asar code archive on Windows. Users should take care to never use Trinity with significant amounts of IOTA on a computer that could potentially be compromised, regardless of code integrity checks. The IOTA Foundation is monitoring this issue and will implement any changes necessary to verify asar integrity on Windows when the issue with Electron is resolved.

[6] https://github.com/electron/asar/issues/123

I2 - Linux AppImage does not include embedded GPG signature

Risk Level: 🟩 Informational

Description

Trinity is distributed to Linux users as an AppImage[7] which is a single executable that contains all the code required to run Trinity. This executable is signed by the IOTA Foundation using a GPG key and external signature file. AppImages have the ability to embed signatures[8] which streamlines the binary verification process for the end user, and can optionally be validated every time the application is run if the user is running `appimaged`.

Remediation

The IOTA Foundation has incorporated the embedded signing of AppImage binaries into their automated build infrastructure, but due to an issue[9] with AppImage signing it is not enabled yet.

[7] https://appimage.org/
[8] https://docs.appimage.org/packaging-guide/signatures.html
[9] https://github.com/AppImage/AppImageKit/issues/949

I3 - iOS application allows screenshots

Risk Level: 🟩 Informational

Description

Trinity on iOS allows users to capture screenshots of their seed. An uninformed user may take a screenshot of their seed which lacks the protections offered by the Trinity iOS app. This screenshot may also be automatically uploaded to a user's iCloud or similar service, which if compromised would compromise the user's IOTA wallet.

Remediation

The IOTA Foundation is investigating possible user interface design changes which make it more difficult to screenshot a user's seed on iOS.

I4 - `randomBytes` called without `max` value compares against `NaN`

Risk Level: 🟩 Informational

Description

The `randomBytes` function when called without a `max` value compares each random byte against the numeric type `NaN` (Not a Number). This comparison always returns False, which produces the intended behavior of the function, but may be confusing for developers.

```
/**
 * Create random byte array
 * @param {number} Length - Random number array length.
 * @param {number} Max - Random byte max range
 * @returns {array} Random number array
 */
export const randomBytes = (size, max) => {
      if (size !== parseInt(size, 10) || size < 0) {
          return false;
      }

      const rawBytes = new Uint8Array(size);

      const bytes = global.crypto.getRandomValues(rawBytes);

      for (let i = 0; i < bytes.length; i++) {
          while (bytes[i] >= 256 - 256 % max) { // Issue here
            bytes[i] = randomBytes(1, max)[0];
          }
      }

      return Array.from(bytes);
};
```

Remediation

Setting a default value for `max` as 256 would not change the functionality of the function and add clarity. This fix has been implemented.[10]

[10] https://github.com/iotaledger/trinity-wallet/commit/0438753a345a1fe4061e800fd1b4b30a174469cc

I5 - Checksums, GPG signature, and GPG key not easily available with binary downloads

Risk Level: 🟩 Informational

Description

Trinity binaries are available for download at https://trinity.iota.org. On this page, there are no checksums, GPG signature files, or the IOTA Foundation GPG key. All of these cryptographic items are available on the GitHub releases page, but are not easy for users to find. Additionally, instructions on how to verify the Trinity binary downloads for each operating system are not provided on the same page as the downloads.

Remediation

Redesign the Trinity download page to clearly list the binary checksums, as well as any cryptographic items needed to verify the integrity of Trinity. Additionally, provide instructions to users for each major operating system Trinity supports on how to verify the Trinity binary. The IOTA foundation has put a link[11] adjacent to the download buttons that clearly explains how to verify the Trinity wallet downloads across all desktop operating systems.

[11] https://docs.iota.org/docs/trinity/0.1/how-to-guides/verify-trinity-desktop