

AUDIT OF WALLET IOS APPLICATION

zokyo

August 18th 2023 | v. 1.0

TECHNICAL SUMMARY

MarsDao engaged Zokyo to conduct a security assessment on their IOS Mobile application beginning on June 5th and ending on August 18th, 2023. MarsDao (MDAO) Wallet is the most user-friendly cryptocurrency wallet. Send, receive, and store Bitcoin and many other cryptocurrencies and digital assets safely and securely with the MDAO Wallet mobile app. The security assessment was scoped to the IOS mobile application (com.ttmbank.wallet.app). An audit of the security risk and implications regarding the changes introduced by the development team at MarsDAO prior to its production release, shortly following the assessment deadline. Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure Mobile application development.

Table of Contents

Auditing Strategy and Techniques Applied 3

Executive Summary 5

Structure and Organization of the Document 6

Complete Analysis 7

AUDITING STRATEGY AND TECHNIQUES APPLIED

Zokyo performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the penetration test. The majority of the time was spent evaluating its use of mnemonic seed Protection . The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of client wallet.
- Manual code read and analysis.
- Reverse engineering of the hashing and encryption functions used Inside the wallet.
- Scanning of code used to locate bugs or security flaws.(MOBSF)
- Proxying the traffic from the local client to the external Internet to determine the traffic and data leaving the system. (REDUX, POSTMAN, BURP SUITE)
- Multiple IOS mobile applications Pen-test tools like frida, objection etc .

Audit is focused on various aspects to ensure the security of the mobile application and includes:

- Implementation of correctness and adherence to industry best practices.
- Exposure of critical information during user interactions, including authentication mechanisms.
- Adversarial actions and attacks that could impact funds, such as draining or manipulating funds.
- Proper management of funds via transactions to prevent mismanagement.
- Identification and remediation of vulnerabilities in the code, as well as ensuring secure interaction between the related and network components.
- Secure management of encryption and storage of private keys, including the key derivation process.
- Prevention of inappropriate permissions and excess authority.
- Ensuring data privacy, prevention of data leakage, and maintaining information integrity.
- Identification and remediation of any other potential security risks, as identified during the initial analysis phase.

In summary, Zokyo identified a few security risks and recommends performing further testing to validate extended safety and correctness in context to the whole structure

SCOPE :

The following scope was audited by Zokyo team :
IOS Application : https://apps.apple.com/us/app/mdao-wallet/id1540851562
Application Name : MDAO-Wallet
Version : 2.2.4

Executive Summary

There was one medium issue found during the audit and some low severity.

They are described in detail in the “Complete Analysis” section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the MarsDAO team and the MarsDAO team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

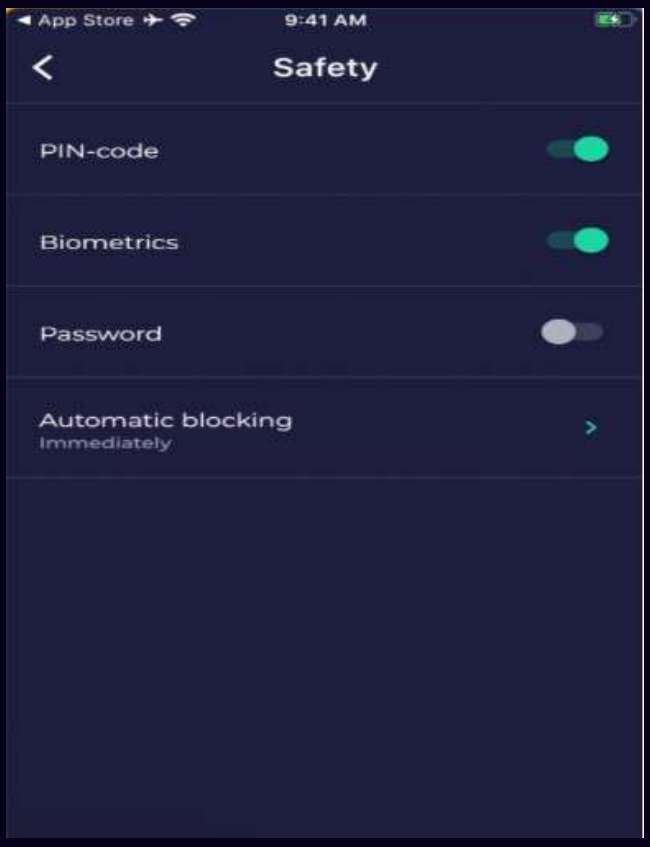
#	Title	Risk	Status
1	Touch ID authentication Bypass	Medium	Resolved
2	Lack of SSL Pinning	Low	Resolved
3	Lack of Jailbreak Detection	Low	Resolved
4	Snapshot Data Disclosure	Low	Resolved
5	Insufficient Debugging Protections	Low	Unresolved
6	Lack Of Token Name, Symbol And Decimal Validation	Low	Unresolved
7	Lack of _RESTRICT segment in the Application Binary	Low	Unresolved
8	Binary makes use of insecure API(s)	Low	Unresolved

Touch ID authentication Bypass

Description :
Many users rely on biometric authentication like Face ID or Touch ID to enable secure, effortless access to their devices. As a fallback option, and for devices without biometry, a passcode or password serves a similar purpose. Use the LocalAuthentication framework to leverage these mechanisms in your app and extend authentication procedures your app already implements.
To maximize security, your app never gains access to any of the underlying authentication data. You can't access any fingerprint images, for example. The Secure Enclave, a hardware-based security processor isolated from the rest of the system, manages this data out of reach even of the operating system. Instead, you specify a particular policy and provide messaging that tells the user why you want them to authenticate. The framework then coordinates with the Secure Enclave to carry out the operation. Afterwards, you receive only a Boolean result indicating authentication success or failure.

Problem Details :
When we run the ios ui biometrics_bypass command, a hook is executed that listens for invocations of the -[LAContext evaluatePolicy:localizedReason:reply:] selector. If the evaluate policy method is called, the hook will replace the success boolean to a True in the code block that is executed when a reply is received.

Proof of Concept
The following is needed in order to reproduce this issue:
Step 1 - Install the Lyra application and set up **TouchID.**



Step 2 - Connect your iDevice with the PC. Run the following command from frida configured terminal to list the opened applications name.

frida-ps -Ua

Step 3 - The apps are usually running by their own names “com.ttmbank.wallet.app” as shown.

```
sh-3.2# frida-ps -Ua
 PID  Name          Identifier
4  ----------  --------------------
3230  Calendar      com.apple.mobilecal
3999  Camera        com.apple.camera
4271  Cydia         com.saurik.Cydia
4267  MDAO Wallet   com.ttmbank.wallet.app
```

Step 4 - Run objection with the commands like: objection -g com.ttmbank.wallet.app explore

```
sh-3.2# objection -g com.ttmbank.wallet.app explore
Using USB device `iPhone`
Agent injected and responds ok!

       _   _         _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] #
```

Step 5 - Use ios ui biometrics_bypass command and observe the application prompted for TouchID. Click on Cancel.

```
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] # ios ui biometrics_bypass
(agent) Registering job 908758. Type: ios-biometrics-disable-evaluatePolicy
(agent) Registering job 896230. Type: ios-biometrics-disable-evaluateAccessControl
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] # (agent) [481801] OS authentication response: false
(agent) [481801] Marking OS response as True instead
(agent) [518594] OS authentication response: true
(agent) [518594] Biometrics bypass hook complete (evaluatePolicy)
(agent) [481801] Biometrics bypass hook complete (evaluatePolicy)
```

```
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] # (agent) [518594] Localized Reason for auth requirement (evaluatePolicy): Authorization
(agent) [481801] Localized Reason for auth requirement (evaluatePolicy): Authorization
(agent) [908758] Localized Reason for auth requirement (evaluatePolicy): Authorization
(agent) [442230] Localized Reason for auth requirement (evaluatePolicy): Authorization
(agent) [442230] OS authentication response: false
(agent) [442230] Marking OS response as True instead
(agent) [908758] OS authentication response: true
(agent) [481801] OS authentication response: true
(agent) [518594] OS authentication response: true
(agent) [518594] Biometrics bypass hook complete (evaluatePolicy)
(agent) [481801] Biometrics bypass hook complete (evaluatePolicy)
(agent) [908758] Biometrics bypass hook complete (evaluatePolicy)
(agent) [442230] Biometrics bypass hook complete (evaluatePolicy)
```



Precondition:
If an attacker manages to get physical access to the victim device, then they can bypass the lock screen and access the logged-in user data.

Recommendation:
A better way to securely save the data would be to save the data in the keychain and protect it with appropriate keychain attributes (for e.g **ksecattraccessiblewhenpasscodesetthisdeviceonly)**, which require touch ID or device passcode authentication to access the keychain content. This will make it harder for the attacker to get the data since to gather the information from the keychain the user would actually have to authenticate with Touch ID or enter the passcode, depending on which access control he applied, and also the logic is managed by the OS and not the application.

Biometric authentication via the Local Authentication framework is easy to implement, it is not recommended to be used for sensitive applications, such as banking or other financial apps.

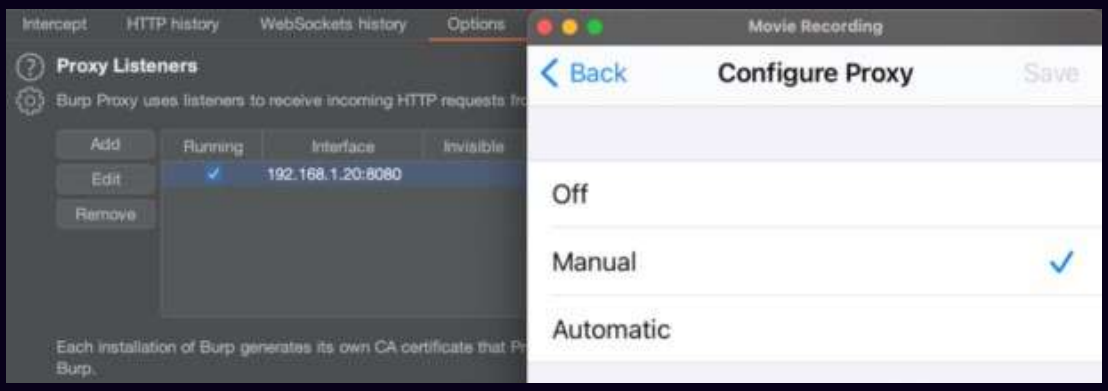
Lack of SSL Pinning

Description:
Certificate pinning is the process of associating the backend server with a particular X.509 certificate or public key instead of accepting any certificate signed by a trusted certificate authority. After storing ("pinning") the server certificate or public key, the mobile app will subsequently connect to the known server only. Withdrawing trust from external certificate authorities reduces the attack surface (after all, there are many cases of certificate authorities that have been compromised or tricked into issuing certificates to impostors).

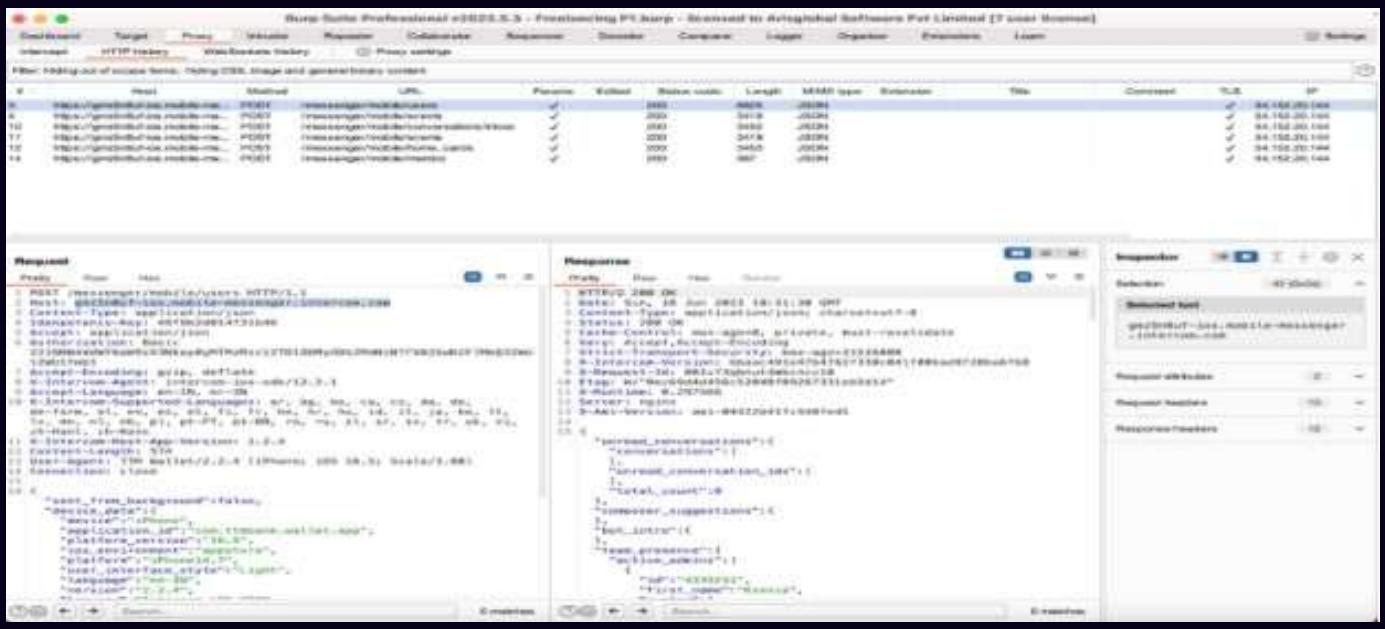
The certificate can be pinned and hardcoded into the app or retrieved at the time the app first connects to the backend. In the latter case, the certificate is associated with ("pinned" to) the host when the host is seen for the first time. This alternative is less secure because attackers intercepting the initial connection can inject their own certificates.

Problem Details
During analysis, it has been observed that SSL pinning is not enabled in the MDAO iOS application and it is possible to intercept all requests and responses. Please refer to the below proof of concept.

Proof of Concept
The following is needed in order to reproduce this issue:
Steps to Reproduce
Step 1 - Set up the web-proxy tool (Burp Suite) to listen on any port (8080).
Step 2 - Install the Burp Suite CA certificate onto the iOS device with the certificateinstaller.
Step 3 - Follow the WiFi setting and set the host system's IP as a proxy system on theiOS device.



Step 4 - Now start navigating through the iOS application, and at the same time, you can observe traffic in the burp.



If SSL pinning is not implemented, then all the communication between the application and the server could be intercepted by an attacker. Because of this, attackers can craft malicious requests to the server and look for multiple vulnerabilities. Automated attacks can be mounted against app with the help of a proxy tool.

Recommendation:
Following are the ways to implement SSL pinning:

An open-source SSL pinning library for iOS and OS X was released at Black Hat 2015, which provides an easy-to-use API for deploying pinning within an App: h ttps://github.com/datatheorem/TrustKit

When using NSURLConnection, iOS pinning is performed through a NSURLConnectionDelegate. The delegate must implement connection:canAuthenticateAgainstProtectionSpace: and connection:didReceiveAuthenticationChallenge:. In connection:didReceiveAuthenticationChallenge:, the delegate must call SecTrustEvaluate to perform customary X509 checks.

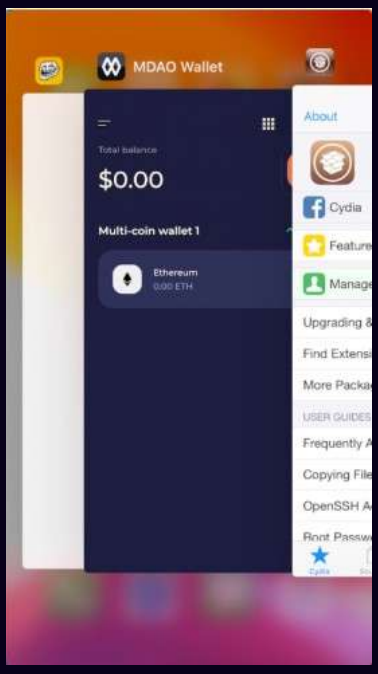
Lack of Jailbreak Detection

Description:
Jailbreak detection mechanisms are added to reverse engineering defense to make running the app on a jailbroken device more difficult. This blocks some of the tools and techniques reverse engineers like to use. Like most other types of defense, jailbreak detection is not very effective by itself, but scattering checks throughout the app's source code can improve the effectiveness of the overall anti-tampering scheme.

Problem Details
The source code review revealed that the mobile application does not perform any checks or validations to determine if it is being executed on a rooted device. This oversight leaves the application vulnerable to several security risks. A jailbroken device bypasses iOS's normal security model. iOS protects applications using iOS sandboxing features, but apps can break out of the sandbox on a jailbroken device. A malicious application on a jailbroken iOS device can gain access to sensitive data created or stored on the device and data written to device logs.

Proof of Concept
The following is needed in order to reproduce this issue:

As shown in the POC, the application is successfully installed in the jailbreak device as cydia is present in the device.



Severity
An attacker who has gained access to a victim's device will be able to access protected data stored in the application's sandbox.

Recommendation:
Filesystem-based Detection: The jailbreak process modifies the filesystem by adding, moving and changing files and directories. New Files Created- During the jailbreaking process, some additional files are created on the device. Looking for these files is a simple way to detect a jailbreak. List of files are as follows:
/private/var/lib/cydia/private/var/mobile/Library/SBSettings/
Themes/
/Library/MobileSubstrate/MobileSubstrate.dylib /System/Library/
LaunchDaemons/com.saurik.Cydia.Startup.plist/
/var/cache/apt/ /var/lib/apt/ /
var/lib/cydia/
/var/log/syslog/ /var/tmp/cydia.log/ /bin/bash/ /bin/sh/
/usr/sbin/sshd/ /usr/libexec/ssh-keysign/
/usr/sbin/sshd/
/usr/bin/sshd/ /usr/libexec/sftp-server/
/etc/ssh/sshd_config/ /etc/apt/
/Applications/Cydia.app/
/Applications/WinterBoard.app/
/Applications/SBSettings.app/

Directory permissions - Certain permissions on partitions and folders can also indicate a jailbroken device. During the jailbreaking process, access to the root partition is amended. If the root partition has read write permissions, the device has been jailbroken.

Writing files - On jailbroken devices, applications are installed the Applications folder and thereby given root privileges. A jailbroken device could be detected by having the app check whether it can modify files outside of its sandbox. This can be done by having the app attempt to create a file in, for example, the /private/ directory. If the file is successfully created, the device has been jailbroken.

API-based Detection:** Detecting a jailbroken device based on API calls can be both effective and difficult for a malicious individual to recognize and bypass. API calls like system(), fork(), dyld functions, etc are difficult to bypass.

Cydia Scheme Detection:** Most jailbroken devices have Cydia installed. While an attacker can change the location of the Cydia app, it's difficult to change the URL scheme for the Cydia app. If calling Cydia's URL scheme (Cydia://) from your application is successful, you can be sure that the device is jailbroken.

LOW-3 | RESOLVED

Snapshot Data Disclosure

Description:
Manufacturers want to provide device users with an aesthetically pleasing experience at application startup and exit, so they introduced the screenshot- saving feature for use when the application is backgrounded. This feature may pose a security risk. Sensitive data may be exposed if the user deliberately screenshots the application while sensitive data is displayed. A malicious application that is running on the device and able to continuously capture the screen may also expose data. Screenshots are written to local storage, from which they may be recovered by a rogue application (if the device is rooted) or someone who has stolen the device.

For example, capturing a screenshot of a banking application may reveal information about the user's account, credit, transactions, and so on.

Problem Details
The field-nation iOS application stores snapshots in local storage when the application is backgrounded. The snapshot includes sensitive data such as username and passwords An attacker with local access to the device, or one who is able to infect the device with malware, would be able to read this data.

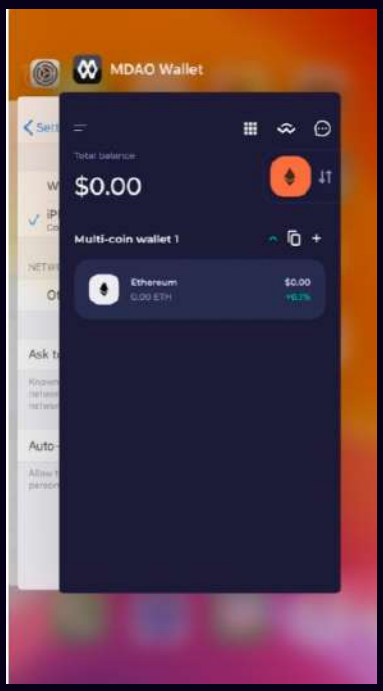
Proof of Concept

The following is needed in order to reproduce this issue:

- Jailbroken iDevice.
- Access to the application.

Step 1 - Start the MDAO iOS application and navigate to the dashboard.

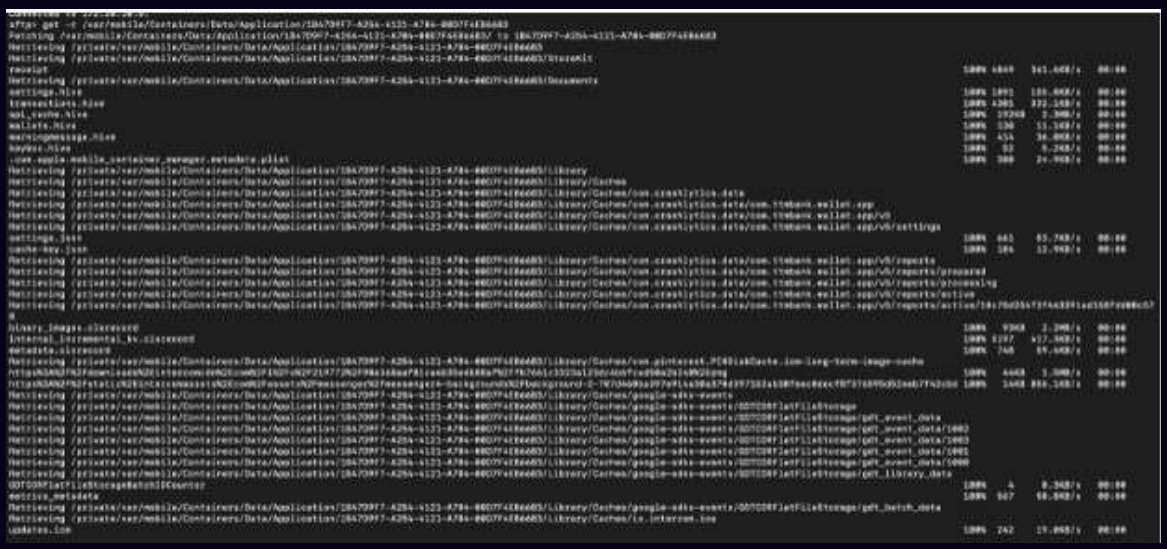
Step 2 - Switch to another application so that the application is backgrounded.



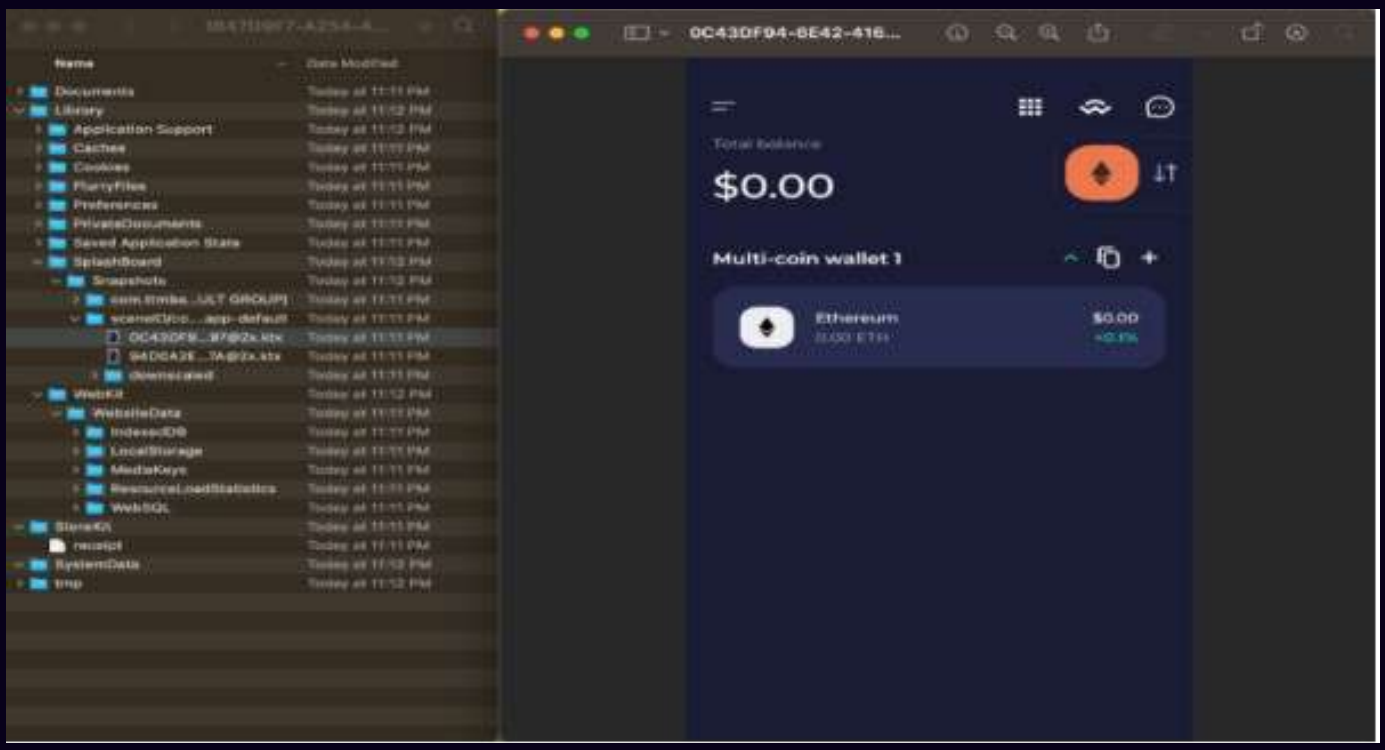
Step 3 - SSH into the jailbroken iOS device.

Step 4 - Identify the data directory (sandbox_id) for the MDAO application. Pull the application data to the host machine using the below command.

sftp > get -r /var/mobile/Containers/Data/Application/<sandbox_id>/



Step 5 - Navigate to SplashBoard and open ktx ext file.



Step 6 - Observe that the screenshot of the application is stored when the HOME button is pressed and stored in the application cache directory on the filesystem.

Severity
This vulnerability represents a minimal exposure to exploitation. Only the users of the mobile devices to which the attacker has access are affected by this vulnerability.

Recommendation:
As a best practice, consider preventing background screen caching if the application displays sensitive data.

Insufficient Debugging Protections

Description:
Anti-tamper techniques must be used to prevent attackers from back-dooring legitimate applications. Though none of the solutions are foolproof and a motivated attacker could bypass the protections given a sufficient amount of time, the mitigations do provide a barrier against attackers. Due to the sensitive nature of the application and the data that it accesses, defenses against attacks of this nature greatly increase the security posture of organizations that deploy applications.

Problem Details
The MDAO iOS application does not sufficiently defend itself against reverse engineering and runtime tampering tools. This allows attackers to attach debuggers and reversing tools with little to no modification of the application, significantly speeding up exploit discovery and development.

Proof of Concept

Step 1 - Start the Frida server on the iOS device.

```
sh-3.2# frida-ps -Ua
 PID  Name          Identifier
4  ------------  ----------------------
3230  Calendar      com.apple.mobilecal
3999  Camera        com.apple.camera
4271  Cydia         com.saurik.Cydia
4267  MDAO Wallet   com.ttmbank.wallet.app
```

Step 2 - Use frida-ps to verify that the Frida server is ready. Assuming that the iOS device is connected to the host machine via USB, use the following command on the host machine.

frida-ps -Ua

```
sh-3.2# objection -g com.ttmbank.wallet.app explore
Using USB device `iPhone`
Agent injected and responds ok!

     _   _         _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | |  _|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] #
```

Step 3 - Connect to the application using Objection, which will inject the Frida library into the application and allow runtime manipulation. Notice that the application is restarted, but shows no indication that the library injection has occurred.

#objection -g com.fieldnation.ios.mobile explore
 # ios list hooking list classes
 # ios list hooking search classes login

```
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] # ios hooking list classes login
AAAbsintheContext
AAAbsintheSigner
AAAbsintheSignerContextCache
AAAccount
AAAccountManagementUIResponse
AAAccountManager
AAAddEmailUIRequest
AAAppleIDSettingsRequest
AAAppleTVRequest
AAAttestationSigner
AAAuthenticateRequest
AAAuthenticationResponse
AAAutoAccountVerifier
AAAvailabilityRequest
AAAvailabilityResponse
AACertificatePinner
AAChildAccountCreationUIRequest
AACloudKitDevicesListRequest
AACloudKitDevicesListResponse
AACloudKitMigrationStateRequest
AACloudKitMigrationStateResponse
AACloudKitStartMigrationRequest
AACloudKitStartMigrationResponse
AACompleteEmailVettingRequest
AACompleteEmailVettingResponse
AADataclassManager
AADevice
AADeviceInfo
AADeviceList
AADeviceListRequest
AADeviceListResponse
AADeviceProvisioningRequest
AADeviceProvisioningResponse
AADeviceProvisioningSession
AAEmailVettingRequest
AAFMIPAuthenticateRequest
AAFMIPAuthenticateResponse
AAFamilyDetailsRequest
AAFamilyDetailsResponse
AAFamilyEligibilityRequest
AAFamilyEligibilityResponse
AAFamilyInvite
AAFamilyMember
AAFamilyMemberDetailsUIRequest
```

```
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] # ios hooking search classes crypto
TangemSdk.CryptoUtils
_TtCV9CryptoKit24CoreCryptoChaChaPolyImplP33_1EEA2C15408765EA775A43B1D63577DE7Context
_TtCV9CryptoKit17CoreCryptoGCMImplP33_A4BAEB898ACAD702674B710A7536C9C17Context
FConfigCryptoUtils
MCCrypto
MRCryptoPairingSessionBlockDelegate
MRCryptoPairingIdentity
MRCryptoPairingSession
MRCryptoPairingMessage
NEIKEv2Crypto
DESPFLEncryptor
WBSHistoryCrypto
_MRCryptoPairingMessageProtobuf

Found 13 classes
com.ttmbank.wallet.app on (iPhone: 13.3.1) [usb] #
```

Severity
This vulnerability is difficult to exploit. However, many off-the-shelf tools exist in order to implement attacks which leverage this vulnerability.

Recommendation:
Use anti-debugging techniques. Anti-debugging techniques, such as Android's Debug.isDebuggerConnected() available from the android.os.Debug class or using sysctl to check for the presence of a ptrace-based debugger on iOS, will defend the application against debugging, memory manipulation, and reverse engineering. Note that techniques such as using PT_DENY_ATTACH will not work as demonstrated on other BSD based systems, as the ptrace syscall itself is not in the public iOS API and will therefore be blocked from release by Apple.

Perform checks for common reverse engineering tools. Checking for the existence of open D-Bus ports (which are used by Frida), detecting code trampolines, in which the flow of code is diverted into attacker controlled code and is used commonly by Substrate, and scanning process memory for known artifacts of common runtime manipulation tools would allow the application to detect that runtime manipulation is occurring and take appropriate action.

Perform application signature checks. Ensure that the application performs a checksum check or some validation mechanism to detect tampering of the application. If the application is tampered with, the detection scheme should take a reactive approach and prevent malicious execution of the application.

Xcode - In Xcode, there are certain checks that an attacker can use to determine whether an application is being debugged or not. In Xcode, use the following piece of code wherever you want to put a check for a debugger.

```
#ifndef DEBUG
 SEC_IS_BEING_DEBUGGED_RETURN_NIL();
 #endif
```

Another technique to prevent these debuggers from attaching to your application is by using the ptrace function. Using this function with a specific parameter, you can just deny any other debugger the ability to attach to your application. The ptrace function is used by the debuggers like GDB and LLDB to attach to a process. Using the ptrace command with the parameter PT_DENY_ATTACH will tell the function to not allow this application to be traced..

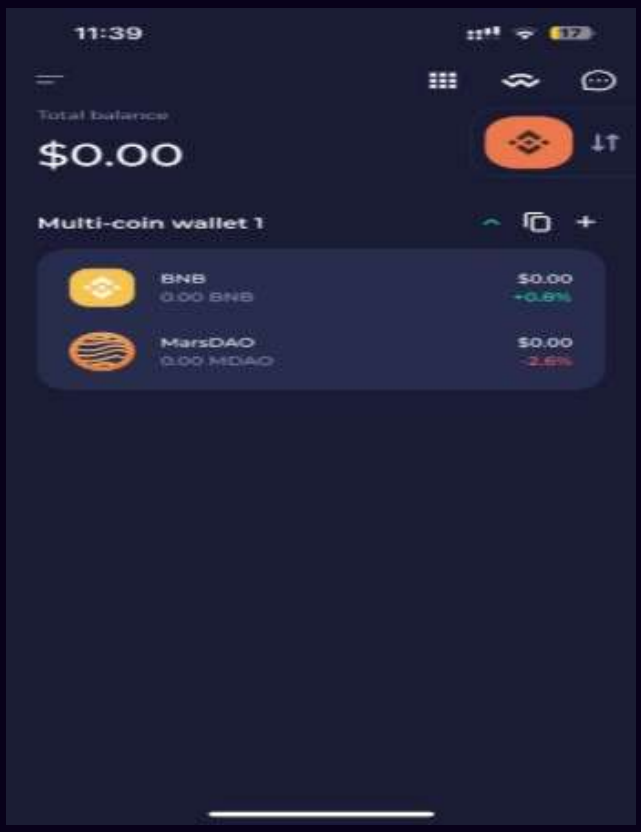
Lack Of Token Name, Symbol And Decimal Validation

Description:
The Lack of Token Name, Symbol, and Decimal Validation vulnerability poses a significant risk to the security and usability of the MDAO Wallet iOS application. This vulnerability allows for the creation and addition of tokens without proper validation of their name, symbol, and decimal places. Attackers can exploit this weakness to deceive users, manipulate token values, and potentially execute financial fraud.

Problem Details
The MDAO wallet iOS application allows users to import custom tokens, and display the token name and symbol in the wallet UI. The ERC20 token contract standard doesn't have any restrictions on any of the token properties, and anyone can deploy token contracts on the blockchain. Once a token is added, the wallet will fetch the information and display them in it's interlace.

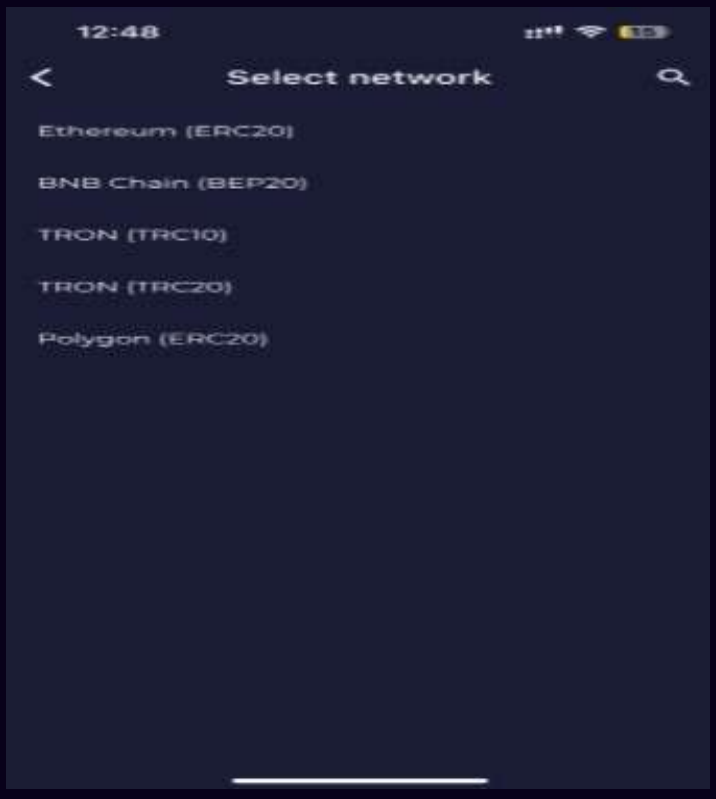
Proof of Concept

Step 1 - Navigate to the dashboard.

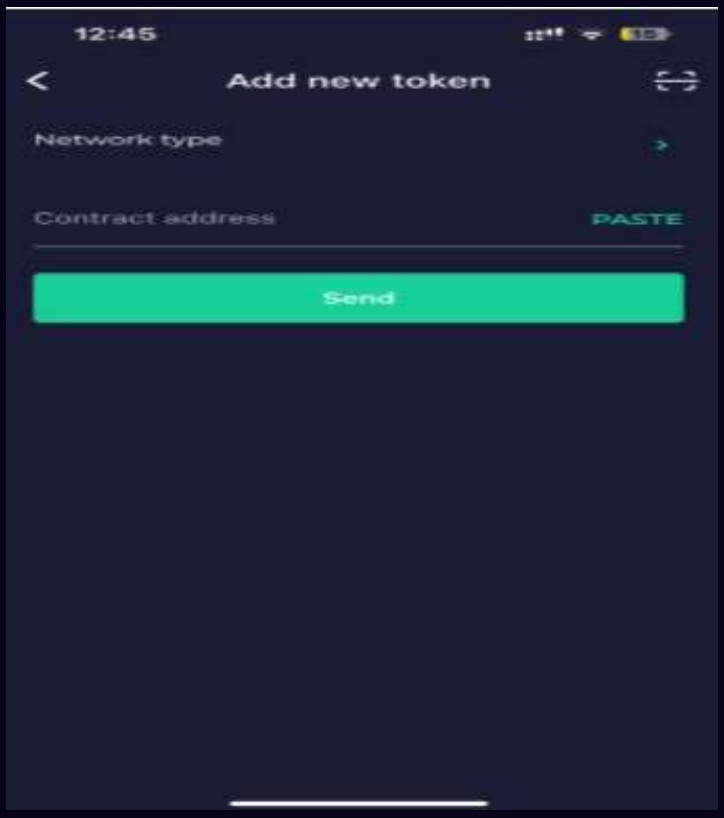


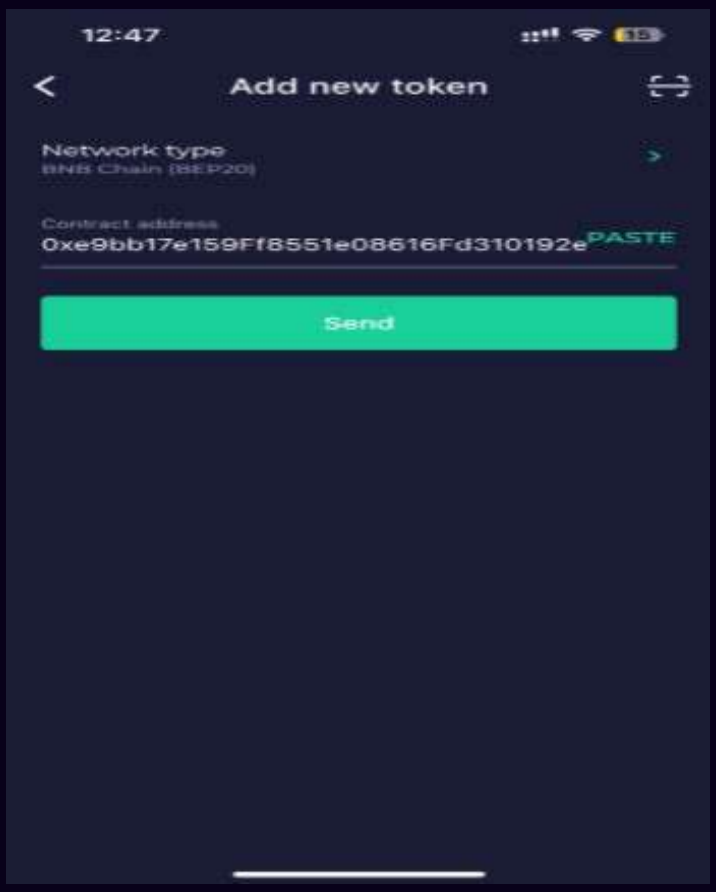
Step 2 - Add New Token and select network.



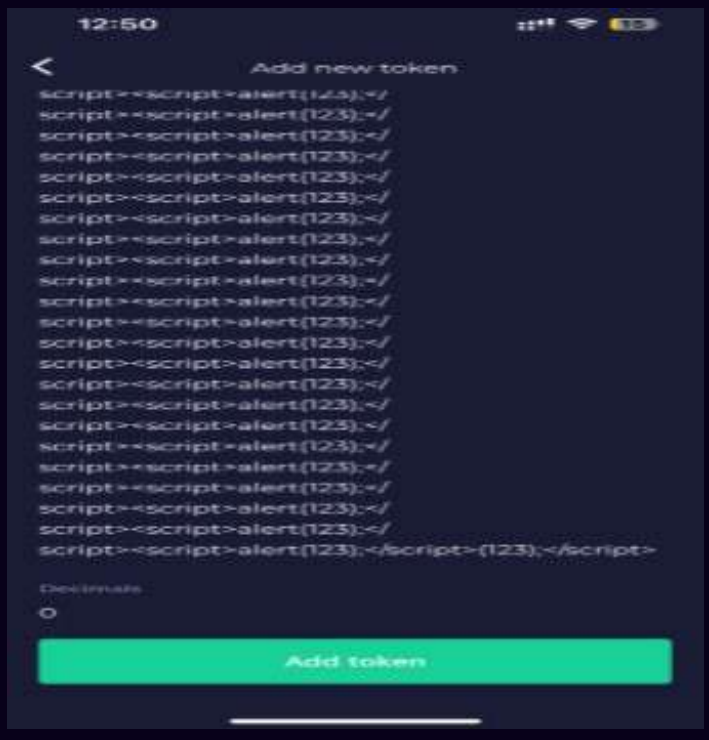


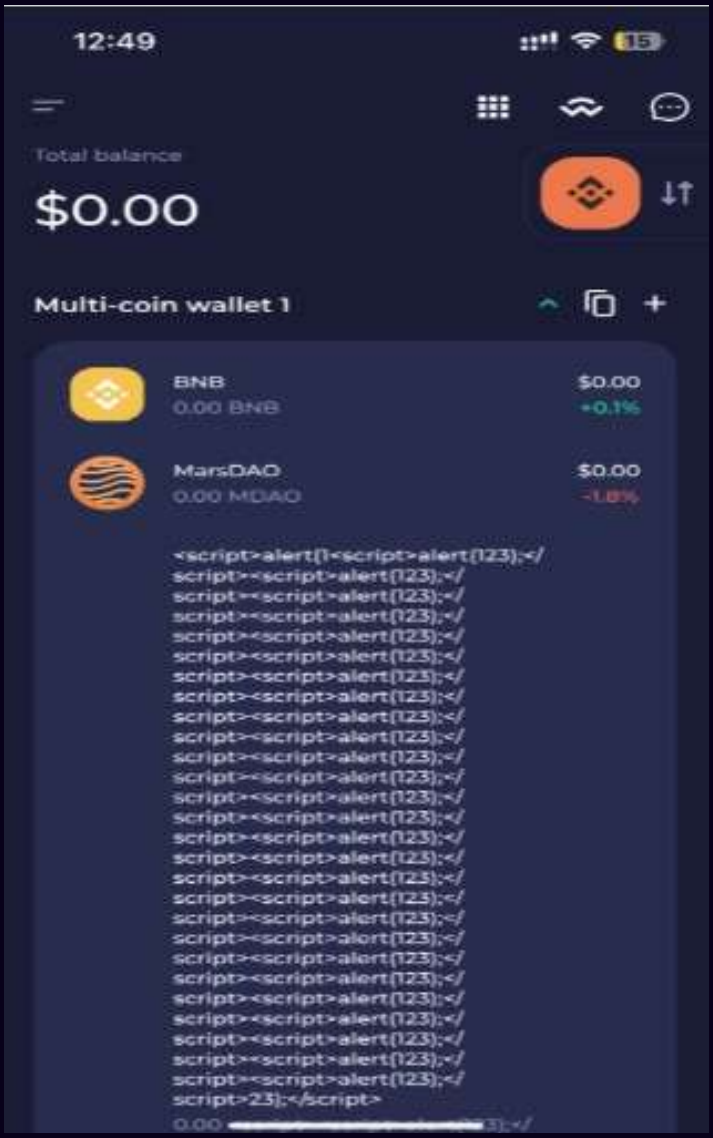
Step 3 - Import the following token in the BNB Chain.
“0xe9bb17e159Ff8551e08616Fd310192ea57BBe52b”








12:50
Add new token
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</
script><script>alert(123);</



Precondition
Importing a crafted malicious token can bring a negative user experience to the user, and the attacker can potentially craft a message to perform a phishing attack.

Misleading Token Information: Attackers can create tokens with deceptive names and symbols, leading users to believe they are interacting with a legitimate or well-known token.

Value Manipulation: By manipulating the decimal places associated with a token, attackers can alter the perceived value of the token, potentially leading to financial loss for unsuspecting users.

Recommendation
It's recommended to place restrictions on the token name, symbol, and decimals similar to the following:

Lack of _RESTRICT segment in the Application Binary

Description:
The dynamic linker (dyld) is the process that loads and runs binaries on OS X and iOS. This process also has some very special environment variables that can modify its normal behavior, one commonly used environment variable is DYLD_INSERT_LIBRARIES: This is commonly used to inject dylibs into applications that modify behavior or patch specific functionality.

So when an application is launched the binary is run through dyld and that processes the binary file. This finds what libraries it needs to load and link against to generate a complete symbol table. Doing this requires parsing through the binary header, while it does this it can trigger flags in dyld based on what segments are present in the binary.There is a special flag (_RESTRICT) that will be set for binaries that are marked as "restricted". This special flag means that the dynamic linker should ignore any set environment variables; absence of these given flags allows an attacker to modify/ patch the application in loading malicious dylib by abusing DYLD_INSERT_LIBRARIES env variable.

This makes the app vulnerable to code injection attacks in the following ways :

Prison Break Injection: Through Modification DYLD_INSERT_LIBRARIES The value of the environment variable to insert the dynamic library and executeNon-escape injection: Pack the custom Framwork or dylib library directly into APP and re-sign it.

Using yololib to modify MachO files and add library paths. Dyld loads and executes when the application starts.

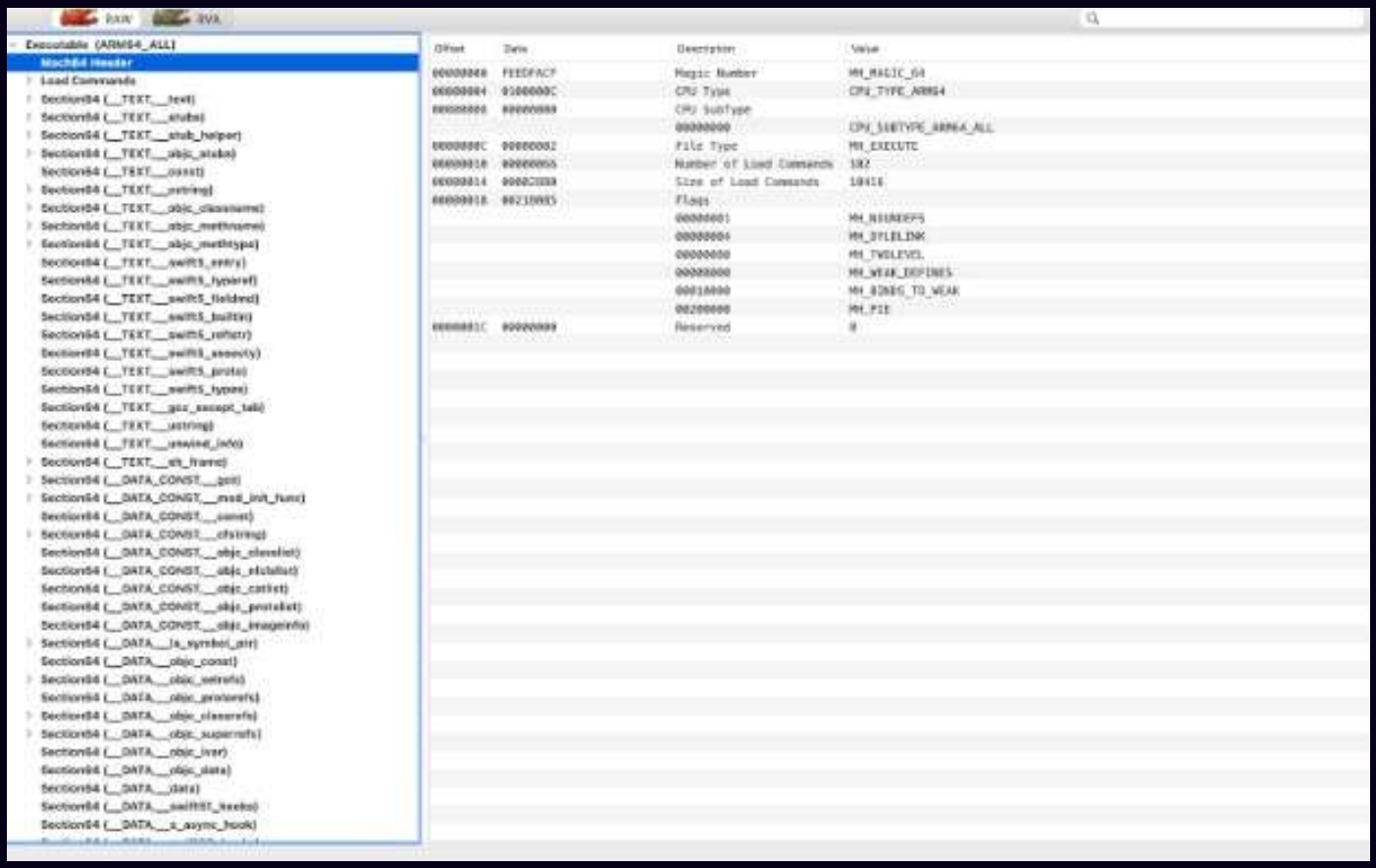
Problem Details
During testing, it was observed that the application has a Lack of
_RESTRICT
segment in the application binary

Proof of Concept
The following is needed in order to reproduce this issue:
- Access to the application file.
- MachOView file viewer

Steps to Reproduce

Step 1 - Analyze the application binary in MachOView.

Step 2 - Observe the Mach64 header lacks the _RESTRICT segment in the application binary.



Severity

An attacker can inject dylibs into applications that modify the behaviour of the application or inject arbitrary code during runtime by loading malicious dylib by abusing DYLD_INSERT_LIBRARIES

env variable.

Recommendation

Find Linker Flags in Engineering Build Settings and add fields - Wl,-sectcreate,__RESTRICT,__restrict,/dev/null

Binary makes use of insecure API(s)

Description:
The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.

This weakness covers three distinct situations. A "missing" protection mechanism occurs when the application does not define any mechanism against a certain class of attack. An "insufficient" protection mechanism might provide some defences - for example, against the most common attacks - but it does not protect against everything that is intended. Finally, an "ignored" mechanism occurs when a mechanism is available and in active use within the product, but the developer has not applied it in some code path. https://cwe.mitre.org/data/definitions/693.html

Problem Details
The M**DAO** iOS application uses insecure API(s), which can lead to a buffer overflow:

0×00000001000d04e0 422 ___memcpy_chk
0×00000001000d0744 496 _fprintf
0×00000001000d0858 548 _memcpy
0×00000001000d0c00 626 _snprintf
0×00000001000d0c0c 627 _sscanf
0×00000001000d0c30 630 _strlen
0×000000010011c660 422 ___memcpy_chk
0×000000010011c7f8 496 _fprintf
0×000000010011c8b0 548 _memcpy
0×000000010011cb20 626 _snprintf
0×000000010011cb28 627 _sscanf
0×000000010011cb40 630 _strlen

Proof of Concept

The following is needed in order to reproduce this issue:
- **Access to the application file.**
- **Otool, available at - https://github.com/Imposter/otool**

Steps to Reproduce

Step 1 - Navigate to the application directory.

Step 2 - Run the command and observe the list of unsafe functions which can lead to a buffer overflow.

otool -Iv TTM\ _Wallet | grep -E 'strncat|strcpy|vsnprintf|sscanf|strtok|scanf|strcat|sprintf|printf|strlen|memcpy|strncpy'

```
sh-3.2# otool -Iv TTM\ _Wallet | grep -E 'strncat|strcpy|vsnprintf|sscanf|strtok|scanf|strcat|sprintf|printf|strlen|memcpy|strncpy'
0x0000001000d04e0   422 ___memcpy_chk
0x0000001000d0744   496 _fprintf
0x0000001000d0858   548 _memcpy
0x0000001000d0c00   626 _snprintf
0x0000001000d0c0c   627 _sscanf
0x0000001000d0c30   630 _strlen
0x000000010011c660   422 ___memcpy_chk
0x000000010011c7f8   496 _fprintf
0x000000010011c8b0   548 _memcpy
0x000000010011cb20   626 _snprintf
0x000000010011cb28   627 _sscanf
0x000000010011cb40   630 _strlen
sh-3.2#
```

Severity

The likelihood of an attacker exploiting this issue successfully is low. An attacker that is able to exploit this issue, however, would gain full control over the elsa iOS application and could use it to compromise the underlying device.

Recommendation

Enable stack canaries flag in swift and make sure that -fstackprotector-all flag gets enabled.

We are grateful for the opportunity to work with the MarsDAO team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the MarsDAO team put in place a bug bounty program to encourage further analysis of the wallet application by third parties.



zokyo