APP WALLET AUDIT REPORT

for

BHOP CONSULTANTING PTE. LTD

Prepared By: Shuxiao Wang

Jan. 06, 2021

Document Properties

Client	BHOP Consultanting Pte. Ltd
Title	App Wallet Audit Report
Target	HBTC Wallet
Version	1.0
Author	Huaguo Shi
Auditors	Huaguo Shi, Xin Li
Reviewed by	Chiachih Wu, Shuxiao Wang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author	Description
1.0	Jan. 06, 2021	Huaguo Shi	Final Release
1.0-rc1	Dec. 28, 2020	Huaguo Shi	Release Candidate #1
0.3	Dec. 25, 2020	Huaguo Shi	Additional Findings #2
0.2	Dec. 21, 2020	Huaguo Shi	Additional Findings #1
0.1	Dec. 20, 2020	Huaguo Shi	Initial Draft

Contact

For more information about this document and its contents, please contact PeckShield Inc. [10] 。

Name	Shuxiao Wang
Phone	+86 173 6454 5338
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related implementation of HBTC Wallet, we have accordingly audited the client applications under three different platforms: `Android`, `iOS`, and `Web`. We outline in this report our systematic method to evaluate potential security issues in current implementation, expose possible semantic inconsistencies between the source code and the design specification, and provide additional suggestions and recommendations for improvement. Our results show that the given implementation of HBTC Wallet can be further improved due to the presence of several issues related to either security or performance. This document describes our audit results in detail.

1.1 About HBTC Wallet

HBTC Chain presents the next-generation blockchain-based technology for decentralized asset custody and clearing. HBTC Wallet is a decentralized, cross-chain, digital custody wallet that has a native support of the HBTC Chain. It also contains the development of a number of auxiliary tools and extensions to facilitate or streamline the use of the HBTC Chain by end-users. The basic information of HBTC Wallet is shown in Table 1.1

Table 1.1: Basic Information of HBTC Wallet

Item	Description
Issuer	BHOP Consultanting Pte. Ltd
Website	https://hbtcchain.io/
Type	Wallet App
Platform	Android/iOS/Web
Coding Language	C/Java/Type Script
Audit Method	White-box
Latest Audit Report	Jan. 06, 2021

In the following, we show the Git repositories of reviewed files and the commit hash values used in this audit.

- Android: https://github.com/hbtc-chain/wallet-android (f4e75f8)
- iOS: https://github.com/hbtc-chain/wallet-ios (9a28e11)
- Web: https://github.com/hbtc-chain/wallet-web (0d06d72)

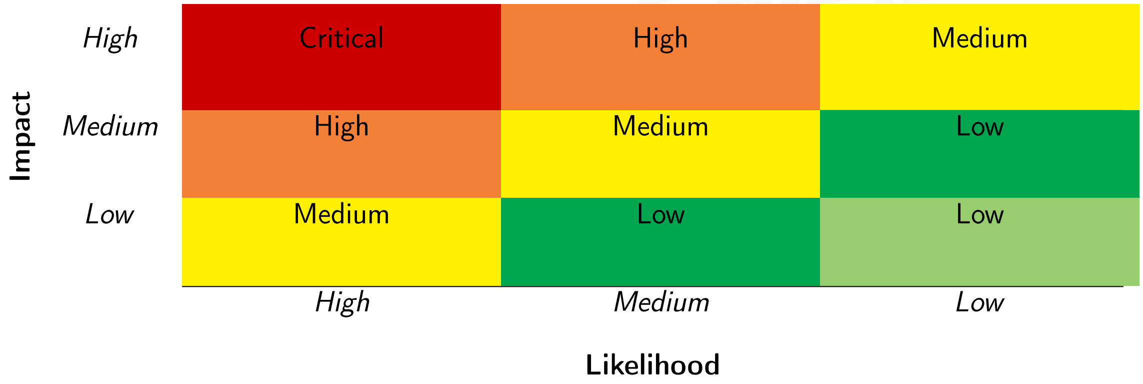
And these are the commit IDs after all fixes for the issues found in the audit have been checked in:

- Android: https://github.com/hbtc-chain/wallet-android (e7f4e87)
- iOS: https://github.com/hbtc-chain/wallet-ios (9052e84)
- Web: https://github.com/hbtc-chain/wallet-web (832614a)

1.2 About PeckShield

PeckShield Inc. is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products including security audits. We are reachable at Telegram (https://t.me/peckshield), Twitter (twitter), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification



Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact
Likelihood

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [9]:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact: measures the technical loss and business damage of a successful attack;
- Severity: demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, and *Low* shown in Table 1.2.

To evaluate the risk, we go through a checklist of items with a platform category. For one check item, if our tool or analysis does not identify any issue, it is considered safe regarding the check item. For any discovered issue, we might further run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. This audit covers applications on multiple platforms, so different reviews are conducted on the characteristics and application scenarios of each platform. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following audit steps:

- Automated Static Analysis: We first begin the analysis by detecting common code and application-level vulnerabilities with a home-made automated static analysis tool. This tool is helpful to quickly identify known coding bugs, and we will then manually verify (reject or confirm) those issues found by our tool. Specifically, throughout the vulnerability scanning process, we will reproduce each issue based on the error log files generated by the vulnerability analysis tool. For each vulnerability case, we will further analyze the root cause and check if it is indeed a vulnerability. Once a risk is confirmed, we will analyze it further as part of a white-box audit, with a better understanding of the associated business logic and context.
- Business Logic Analysis: We next understand business logics, review system-wide operations, examine the interactions between different components, and place wallet-related logic under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of applications from the perspective of proven programming practices for the targeted platforms.

To better describe each issue we identified, we also categorize the findings based on Common Weakness Enumeration (CWE-699) [8], which is a community-developed list of software weakness types to better classify and organize weaknesses around concepts frequently encountered in software development. We use the CWE categories in Table 1.4 to classify our findings.

Table 1.3: The Full Audit Checklist

Platform	Category	Check Item
Android	**System Components**	Export of Component Activity
		Export of Component Service
		Export of Component Broadcast Receiver
		Export of Component Content Provider
	Android Secure Development	Un-obfuscated Java Code
		Arbitrarily Debugging
		Intent Scheme URLs Attack
		Dynamically and Safely Loading DEX File
		Executing Command Function in DLL
	System Security Detection	Signature Validation
		Detection on root/hook
		App Self-protection
iOS	**Known Vulnerability**	URL Schema Vulnerability Detection
		AFNetworking SSL Vulnerability Detection
		XcodeGhost Virus Detection
		"Youmi" Malicious SDK Detection
		iBackDoor Backdoor Detection
	System Security Detection	Detection on Jailbreak
Android/ iOS/Web	**Third-party Plug-in**	Security of using third-party plugins
	Common Secure Development	Correct Random Number
		Sensitive Information Printed by Log
	Communication Security	Network Communication Security
		Network Proxy Security
	Data Protection	Program Data Arbitrarily Backup
		Global File Security
		Configuration File Security
		Clipboard Security
		Anti-screenshot
	Business Logic	Private Key Generating
		Encryption Algorithm
		Password Strength
		Mnemonic Words Generating
		Private Key and Mnemonic Words Storage
		Local Sensitive Data Storage
		Mnemonic Words Import
		Private Key Import
		User Input Security
		Transaction Logic
		Wallet Communication
		Password Strength
		Password Updating
		Server Interaction Logic
		Functionality Integrity

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, **Return Values,** **Status Codes**	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of blockchain software. Last but not least, this security audit should not be used as investment advice.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the HBTC Wallet implementation. During the first phase of our audit, we study the wallet source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) those issues reported by our tool. We further manually review the business logic, examine system operations, and analyze the security issues of private key storage and signature verification, and place wallet-related logic under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	6	■■■■■■
Low	2	■■
Informational	3	■■■
Total	11	

We have so far identified a list of potential issues, which include some issues that can be overlooked from the wallet developer's perspective such as the App's platform security. Other security issues are mainly related to store privatekey and using third-party plugins. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. The detailed discussions of each of them are in the next section.

2.2 Key Findings

Overall, the HBTC Wallet is well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 6 medium-severity vulnerabilities, 2 low-severity vulnerabilities and 3 informational recommendations.

Table 2.1: Key Audit Findings

ID	Severity	Platform	Title	Category	Status
PVE-001	Informational	Android	Weak AES Encryption in Plug-in bitcoinj	Coding Practices	Confirmed
PVE-002	Medium	Android	Possible Privilege Escalation in Plug-in xuexiangjys	Coding Practices	Fixed
PVE-003	Informational	Android	ZipperDown Vulnerability in Plug-in xuexiangjys	Coding Practices	Fixed
PVE-004	Informational	Android	Implicit Intent Invocation in Plug-in xuexiangjys	Coding Practices	Fixed
PVE-005	Low	Android	Arbitrary Backup of Private Data	Business Logic	Fixed
PVE-006	Medium	Android/ iOS/Web	Possible Private Key Leakage through Clipboard	Business Logic	Confirmed
PVE-007	Medium	Android/ iOS	Flawed System Security Validation	Business Logic	Fixed
PVE-008	Medium	iOS	Unsafe Storage of Private Key and Mnemonic Words	Security Features	Fixed
PVE-009	Medium	Web	Vulnerability in Outdated Web Plug-in resolve-url-loader	Coding Practices	Fixed
PVE-010	Low	Android/ iOS/Web	No Strength Validation on Password Setting	Business Logic	Confirmed
PVE-011	Medium	Android/ iOS	Lack of Network Proxy Detection	Business Logic	Fixed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the actual deployment. The risk-control mechanisms should kick in at the very moment when the application or software is deployed. Please refer to Section 3 for details.

3 Detailed Results

3.1 Weak AES Encryption in Plug-in bitcoinj

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

- Platform: Android
- Target: `org.bitcoinj.crypto`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1104 [1]

Description

The core functionality of the wallet is implemented by `bitcoinj-core-0.15.8`. Our analysis shows that the used APIs in `bitcoinj` are appropriate. However, we notice an algorithm issue in the implementation of `encryptNoEC()/decryptNoEC()`, a part of the plug-in `org.bitcoinj.crypto.BIP38PrivateKey`. To be specific, the encryption process of AES is in the `AES/ECB/NoPadding` mode, which uniformly encrypts all the blocks of divided files. Therefore, to break the encryption, an adversary will have to decrypt one arbitrary block. Though the flawed API is not used in the current implementation, the developers still need to be warned to not to use this specific API.

```java
    private ECKey decryptNoEC(String normalizedPassphrase) {
        try {
            byte[] derived = SCrypt.generate(normalizedPassphrase.getBytes(
                StandardCharsets.UTF_8), addressHash, 16384, 8, 8, 64);
            byte[] key = Arrays.copyOfRange(derived, 32, 64);
            SecretKeySpec keyspec = new SecretKeySpec(key, "AES");

            DRMWorkaround.maybeDisableExportControls();
            Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding");

            cipher.init(Cipher.DECRYPT_MODE, keyspec);
            byte[] decrypted = cipher.doFinal(content, 0, 32);
            for (int i = 0; i < 32; i++)
                decrypted[i] ^= derived[i];
            return ECKey.fromPrivate(decrypted, compressed);
        } catch (GeneralSecurityException x) {
            throw new RuntimeException(x);
        }
    }
```

Listing 3.1: org.bitcoinj.crypto.BIP38PrivateKey

Recommendation Appropriately raise the potential issue in the documentation, or utilize the patched source code of plug-in `bitcoinj`.

Status This issue has been acknowledged by the team. The `bitcoinj-core-0.15.8` is necessary for wallet development, and there is no easy fix because the `bitcoinj-core` is still contained the issue in the newer version. The flawed API is not used in the current implementation, so the team decides to ignore the issue.

3.2 Possible Privilege Escalation in Plug-in xuexiangjys

- ID: PVE-002
- Severity: Medium
- Likelihood: Low
- Impact: High

- Platform: Android
- Target: `com.xuexiang.xutil.common.ShellUtils`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1104 [1]

Description

The implementation of QR code scanning imports the third-party plug-in `com.github.xuexiangjys.XUtil:xutil-core v1.1.5`, where the function `com.xuexiang.xutil.common.ShellUtils.execCommand` has unsafely invoked `exec` of `Runtime`. Moreover, the `exec` also tries to call the command `su`. Though we have not found the actual risk caused by this behavior, we strongly recommend not to use this plug-in.

```
    /**
     * execute shell commands
     *
     * @param commands         command array
     * @param isRoot           whether need to run with root
     * @param isNeedResultMsg  whether need result msg
     * @return <ul>
     * <li>if isNeedResultMsg is false, {@link CommandResult#successMsg}
     * is null and {@link CommandResult#errorMsg} is null.</li>
     * <li>if {@link CommandResult#result} is -1, there maybe some
     * excepiton.</li>
     * </ul>
     */
    public static CommandResult execCommand(String[] commands, boolean isRoot,
                                            boolean isNeedResultMsg) {
```

```
        int result = -1;
        if (commands == null  commands.length == 0) {
            return new CommandResult(result, null, null);
        }

        Process process = null;
        BufferedReader successResult = null;
        BufferedReader errorResult = null;
        StringBuilder successMsg = null;
        StringBuilder errorMsg = null;

        DataOutputStream os = null;
        try {
            process = Runtime.getRuntime().exec(
                    isRoot ? COMMAND_SU : COMMAND_SH);
            os = new DataOutputStream(process.getOutputStream());
            for (String command : commands) {
                if (command == null) {
                    continue;
                }

                // donnot use os.writeBytes(commmand), avoid chinese charset
                // error
                os.write(command.getBytes());
                os.writeBytes(COMMAND_LINE_END);
                os.flush();
            }
            os.writeBytes(COMMAND_EXIT);
            os.flush();

            result = process.waitFor();
```

Listing 3.2: com.xuexiang.xutil.common.ShellUtils

Recommendation Remove the plug-in `com.github.xuexiangjys.XUtil:xutil-core v1.1.5`, or revise the current version to get rid of any unsafe usage.

Status The issue has been fixed by this commit: `e7f4e87`.

3.3 ZipperDown Vulnerability in Plug-in xuexiangjys

- ID: PVE-003
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Platform: Android
- Target: com.xuexiang.xutil.file.ZipUtils
- Category: Coding Practices [6]
- CWE subcategory: CWE-1104 [1]

Description

The implementation of QR code scanning imports the third-party plug-in com.github.xuexiangjys.XUtil:xutil-core v1.1.5, in which we found the ZipperDown loophole. To be specific, during the unzip process, the function getName() (line 318 and line 324) has not validated the name of the to be unzipped file. The adversary could construct a malicious zip file with a well-designed name. Therefore, through the ZipperDown loophole, the file could be unzipped to an arbitrary directory, possibly overlapped with an existing file. Though the function has not been invoked in the current version of implementation, it is still necessary to keep in mind the potential risk of this plug-in.

```
307     public static List<File> unzipFileByKeyword(final File zipFile,
308                                                 final File destDir,
309                                                 final String keyword)
310             throws IOException {
311         if (zipFile == null  destDir == null) return null;
312         List<File> files = new ArrayList<>();
313         ZipFile zf = new ZipFile(zipFile);
314         Enumeration<?> entries = zf.entries();
315         if (isSpace(keyword)) {
316             while (entries.hasMoreElements()) {
317                 ZipEntry entry = ((ZipEntry) entries.nextElement());
318                 String entryName = entry.getName();
319                 if (!unzipChildFile(destDir, files, zf, entry, entryName)) return files;
320             }
321         } else {
322             while (entries.hasMoreElements()) {
323                 ZipEntry entry = ((ZipEntry) entries.nextElement());
324                 String entryName = entry.getName();
325                 if (entryName.contains(keyword)) {
326                     if (!unzipChildFile(destDir, files, zf, entry, entryName)) return
                            files;
327                 }
328             }
329         }
330         return files;
331     }
```

Listing 3.3: com.xuexiang.xutil.file.ZipUtils

Recommendation Remove the plug-in `com.github.xuexiangjys.XUtil:xutil-core v1.1.5`, or revise the current version to fix the above `ZipperDown` loophole.

Status The issue has been fixed by this commit: `e7f4e87`.

3.4 Implicit Intent Invocation in Plug-in xuexiangjys

- ID: PVE-004
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

- Platform: Android
- Target: `com.xuexiang.xutil`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1104 [1]

Description

The implementation of the QR code scanning of the wallet imports the third-party plug-in `com.github.xuexiangjys.XUtil:xutil-core v1.1.5`. Our analysis shows the presence of (multiple) invocations of the implicit intent, i.e., only specifying the action without a designated receiver. Therefore, the intent might be hijacked by the unexpected applications, which could lead to the leak of privacy content. Example invocations of the implicit intent are shown below:

```java
public static void shutdown() {
    ShellUtils.execCommand("reboot -p", true);
    Intent intent = new Intent("android.intent.action.ACTION_REQUEST_SHUTDOWN");
    intent.putExtra("android.intent.extra.KEY_CONFIRM", false);
    XUtil.getContext().startActivity(intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
}
```

Listing 3.4: com.xuexiang.xutil.system.DeviceUtils

```java
public static void openWirelessSettings() {
    XUtil.getContext().startActivity(new Intent(android.provider.Settings.ACTION_WIRELESS_SETTINGS)
            .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
}
```

Listing 3.5: com.xuexiang.xutil.net.NetworkUtils

```java
public static void openAppSettings() {
    Intent intent = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS");
    intent.setData(Uri.parse("package:" + XUtil.getContext().getPackageName()));
    XUtil.getContext().startActivity(intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
}
```

Listing 3.6: com.xuexiang.xutil.system.PermissionUtils

Recommendation Remove the plug-in `com.github.xuexiangjys.XUtil:xutil-core v1.1.5`, or modify and replace the current version of it.

Status The issue has been fixed by this commit: `e7f4e87`.

3.5 Arbitrary Backup of Private Data

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: Medium
- Platform: Android
- Target: `AndroidManifest.xml`
- Category: Coding Practices [6]
- CWE subcategory: CWE-281 [3]

Description

In Android, the `android:allowBackup` attribute defines whether application data can be backed up and restored by a user who has enabled `usb` debugging. If this particular attribute is set to `true`, it allows an attacker to take the backup of the application data via `adb` even if the device is not rooted. Therefore, applications that handle and store sensitive information such as wallets, card details, passwords etc. should have this setting explicitly set to `false` because by default it is set to true to prevent such risks.

However, our analysis show that the `android:allowBackup` attribute in file `AndroidManifest.xml` is currently `true`, which should be set to `false`. As mentioned earlier, due to the sensitivity and privacy requirement of wallets, the wallet-related data should not be allowed to be exported, especially without user authorization. Therefore,

Recommendation Set the `android:allowBackup` in `AndroidManifest.xml` as `false`.

Status The issue has been fixed by this commit: `e7f4e87`.

3.6 Possible Private Key Leakage through Clipboard

- ID: PVE-006
- Severity: Medium
- Likelihood: Low
- Impact: High

- Platform: Android/iOS/Web
- Target: N/A
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

During the process of storing and exporting the private key, we have found a potential risk which may result in the compromise of the private key. Specifically, the wallet allows users to save their private keys to other applications by copying them to the clipboard for easy access. However, a malicious application that has access to the clipboard is able to obtain the private key without being perceived by the user.

Recommendation Forbid the behavior of copying the private key. However, for the sake of user experience, we recommend that the user can be clearly informed of the possible risks before choosing to copy the private key, and provide a function to clear the clipboard after the copy.

Status The issue has been fixed in the iOS version by this commit: `9052e84` and the issue has been confirmed in the Android and Web version. Like in the common practice of other wallets, it should not clear the clipboard of the system.

3.7 Flawed System Security Validation

- ID: PVE-007
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Platform: Android/iOS
- Target: N/A
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

Once the mobile platform's underlying operating system has been `rooted` or somehow `hooked`, the adversary could easily obtain all the local information of device, e.g., the user's password. Moreover, it is also possible to retrieve the private key temporarily stored in the memory through known code injection techniques.

Recommendation We highly recommend for the wallet to validate whether there is a possibility that the user's device is `rooted` or `hooked`, and clearly inform the potential risk to the user.

Suggested Detection Method There are many related items in `Android`, please refer to the link for details: https://github.com/hamada147/AndroidRootChecker/blob/master/RootChecker.java. In `iOS`, please refer to the following code snippet:

```
+ (BOOL) isJeilBreakingByPath {
    NSArray *jailbreak_tool_paths = @[@"/Applications/Cydia.app",@"/Library/MobileSubstrate/MobileSubstrate.dylib",@"/bin/bash",@"/usr/sbin/sshd",@"/etc/apt"];
    for (int i=0; i<jailbreak_tool_paths.count; i++) {
        if ([[NSFileManager defaultManager] fileExistsAtPath:jailbreak_tool_paths[i]]) {
            NSLog(@"The device is jail broken by path: %@!", jailbreak_tool_paths[i]);
            return YES;
        }
    }
    return NO;
}
```

Listing 3.7: Detecting jailbreaking behavior in iOS

Status The issue has been fixed by these two commits: `e7f4e87` and `9052e84` .

3.8 Unsafe Storage of Private Key and Mnemonic Words

- ID: PVE-008
- Severity: Medium
- Likelihood: Low
- Impact: High

- Platform: iOS
- Target: N/A
- Category: Security Features [5]
- CWE subcategory: CWE-260 [2]

Description

Possessing the private key and the mnemonic words is equivalent to having the right to dispose the asset in corresponding wallet. We performed a check about the implementation of storing private key on the `iOS` platform, and found out that the private key generation function `createAction()` invokes `encryptSecretStorageJSON()` (line 80). The storage of private key follows the core `KeyStore` standard and the `eclipse encryption` algorithm, which basically eliminates the possibility of brute force cracking for private keys.

```
63 - (void)createAction {
64     if (![self.textFieldView.textField.text isEqualToString:KUser.localPassword]) {
65         Alert *alert = [[Alert alloc] initWithTitle:LocalizedString(@"TwoPasswordInconsistent") duration:kAlertDuration completion:^{
66         }];
67         [alert showAlert];
68            return;
69        }
70        [MBProgressHUD showActivityMessageInView:nil];
71        NSLog(@"%@    %@",KUser.localPrivateKey,KUser.localPhraseString);
72        if (!IsEmpty(KUser.localPhraseString)) {
73            self.account = [Account accountWithMnemonicPhrase:KUser.localPhraseString];
74        } else if (!IsEmpty(KUser.localPrivateKey)) {
75            SecureData * data = [SecureData secureDataWithHexString:KUser.localPrivateKey];
76            self.account = [Account accountWithPrivateKey:data.data];
77        }  else {
78            self.account = [Account randomMnemonicAccount];
79        }
80        [self.account encryptSecretStorageJSON:KUser.localPassword callback:^(NSString *json
              ) {
81            [self backupKeystore:json];
82        }];
83    }
```

Listing 3.8: wallet-ios/Bluehelix/Bluehelix/Class/CreateWallet/XXRepeatPasswordVC.m

However, when we further examined the `backupKeystore()` call to store the model object into `sqlite` database, we found that model object stores not only the `KeyStore` data, but also the `MD5` hash value of the password (line 91), private key, and mnemonic words. After we continued our analysis, we found that the private key and mnemonic words are encrypted and stored by `AES(privatekey/mnemonicPhrase, password)`. Therefore, if the password is not very strong with complex combinations and the attacker has access to the `sqlite` database data, he may brute force to uncover the private key through a `rainbow table` attack.

```
85    - (void)backupKeystore:(NSString *)json {
86        XXAccountModel *model = [[XXAccountModel alloc] init];
87        model.privateKey = [AESCrypt encrypt:self.account.privateKeyString password:KUser.
              localPassword];
88        model.publicKey = self.account.pubKey;
89        model.address = self.account.BHAddress;
90        model.userName = KUser.localUserName;
91        model.password = [NSString md5:KUser.localPassword];
92        model.keystore = json;
93        if (self.account.mnemonicPhrase && IsEmpty(KUser.localPhraseString)) {
94            NSString *mnemonicPhrase = [AESCrypt encrypt:self.account.mnemonicPhrase
                  password:KUser.localPassword];
95            model.mnemonicPhrase = mnemonicPhrase;
96            model.backupFlag = NO;
97        } else {
98            model.mnemonicPhrase = @"";
99            model.backupFlag = YES;
100       }
101       model.symbols = [NSString stringWithFormat:@"btc,eth,usdt,%@",kMainToken];

103       if (KUser.accounts) {
104           for (XXAccountModel *a in KUser.accounts) {
105               if ([a.address isEqualToString:model.address]) {
```

```
                Alert *alert = [[Alert alloc] initWithTitle:LocalizedString(@"PrivateKeyRepetition") duration:kAlertDuration completion:^{
                }];
                [alert showAlert];
                [[XXSqliteManager sharedSqlite] deleteAccountByAddress:model.address];
            }
        }
    }
    [[XXSqliteManager sharedSqlite] insertAccount:model];
    KUser.address = model.address;
    [MBProgressHUD hideHUD];
    if (model.backupFlag) {
        Alert *alert = [[Alert alloc] initWithTitle:LocalizedString(@"ImportSuccess") duration:kAlertDuration completion:^{
            KWindow.rootViewController = [[XXTabBarController alloc] init];
            [self showBiometricAlert];
        }];
        [alert showAlert];
    } else {
        XXCreateWalletSuccessVC *successVC = [[XXCreateWalletSuccessVC alloc] init];
        successVC.text = KUser.localPassword;
        [self.navigationController pushViewController:successVC animated:YES];
        [self showBiometricAlert];
    }
    KUser.localPassword = @"";
    KUser.localUserName = @"";
    KUser.localPhraseString = @"";
    KUser.localPrivateKey = @"";
}
```

Listing 3.9: wallet–ios/Bluehelix/Bluehelix/Class/CreateWallet/XXRepeatPasswordVC.m

Recommendation Remove the redundant backup process as the `KeyStore` has stored the private key. If the backup of mnemonic words is necessary, we recommend to apply the same way by taking advantage of `KeyStore`.

Status The issue has been fixed by this commit: `9052e84` .

3.9 Vulnerability in Outdated Web Plug-in resolve-url-loader

- ID: PVE-009
- Severity: Medium
- Likelihood: Low
- Impact: High

- Platform: Web
- Target: N/A
- Category: Coding Practices [6]
- CWE subcategory: CWE-1104 [1]

Description

It is known that during the web development, it is necessary to import a number of third-party or unofficial libraries. Therefore, validating the security of dependent libraries is critical and necessary. During our analysis, we found that the imported library `resolve-url-loader v3.1.1` contains several known vulnerabilities, the most severe one could lead to `denial-of-service` to block normal services. More details can be found at `https://npmjs.com/advisories/1556`.

Recommendation Upgrade the dependent `resolve-url-loader v3.1.1` to version `v3.1.2` or above. Moreover, we highly recommend executing `npm audit fix` to check the existence of any 0day vulnerabilities after the change of dependent libraries.

Status The issue has been fixed by this commit: `832614a`

3.10 No Strength Validation on Password Setting

- ID: PVE-010
- Severity: Low
- Likelihood: Low
- Impact: Low

- Platform: Android/iOS/Web
- Target: N/A
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

The password setting in the `Web` wallet does not perform a validation on the chosen password strength. It is possible that a user may choose a password that is simple, but cannot survive the rainbow table attack. In the meantime, we note that the password is stored by following the `KeyStore` standard, which greatly reduces the risk of brute force cracking. Therefore, we consider this issue as low risk.

Recommendation Increase the strength requirement when a password is chosen and saved. The application can limit the minimum number of digits and force the inclusion of upper and lower case letters and numbers (as well as special characters).

Status The issue has been fixed in the web version by this commit: `832614a`, and the issue has been confirmed in the mobile version (Android/iOS).

3.11 Lack of Network Proxy Detection

- ID: PVE-011
- Severity: Medium
- Likelihood: Low
- Impact: High
- Platform: Android/iOS
- Target: N/A
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

The applications on iOS or Android platforms do not use plain-text communication and all traffic is encrypted by default via HTTPS. Such encryption needs to be enforced to effectively defeat man-in-the-middle attacks. However, there are several network sniffer applications (e.g., Fiddler, Charles) that are able to sniff and capture the packet encrypted by HTTPS, or even modify the packet content (if with the access to the certificate on the mobile device).

With that, it is important to reliably detect the presence of a network proxy and avoid it as much as possible in wallet-like applications. Our analysis shows that the detection of the presence of possible network proxies is currently missing.

Recommendation Currently, the Android version of the wallet allows the user-installed certificate. We highly recommend modifying it to throw an exception after the system certificate verification fails (line 222).

```
214     @Override
215     public void checkServerTrusted(X509Certificate[] chain, String authType) throws
            CertificateException
216     {
217         try
218         {
219             defaultTrustManager.checkServerTrusted(chain, authType);
220         } catch (CertificateException ce)
221         {
222             throw new CertificateException("error in validating certificate", ce);
223         }
224     }
```

Listing 3.10: wallet-android/Lib/Lib_Network/src/main/java/com/bhex/network/utils

The proxy detection under iOS platform is for reference only:

```
(BOOL)isVPNOn{
    BOOL flag = NO;
    CFDictionaryRef dicRef = CFNetworkCopySystemProxySettings();
    const CFStringRef proxyCFstr = (const CFStringRef) CFDictionaryGetValue(dicRef,
        (const void*)kCFNetworkProxiesHTTPProxy);
    NSString* proxy = (__bridge NSString *) proxyCFstr;
    if (proxy != NULL){
            flag = YES;
        }
        return flag;
    }
```

Status The issue has been fixed by these two commits: `e7f4e87` and `9052e84` .

4 Conclusion

In this audit, we have analyzed the design and implementation of HBTC Wallet under three different platforms, i.e., `Android`, `iOS`, and `Web`. The audited system does involve various intricacies in both design and implementation. And the current code base is well structured and neatly organized. Those identified issues are promptly confirmed and fixed.

In the meantime, we emphasize that even if a wallet is well-designed and securely implemented, it does not guarantee you will not be vulnerable to other attacks, such as social engineering, physical threats, or human errors. Therefore, we strongly encourage users to use common sense and apply basic security methods to keep their digital assets safe.

References

[1] MITRE. CWE-1104: Use of Unmaintained Third Party Components. https://cwe.mitre.org/data/definitions/1104.html.

[2] MITRE. CWE-260: Password in Configuration File. https://cwe.mitre.org/data/definitions/260.html.

[3] MITRE. CWE-281: Improper Preservation of Permissions. https://cwe.mitre.org/data/definitions/281.html.

[4] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

[5] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[6] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[7] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[8] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[9] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[10] PeckShield. PeckShield Inc. https://www.peckshield.com.