Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report imToken Wallet Code 05-07.2018

Cure53, Dr.-Ing. M. Heiderich, BSc. F. Fäßler, Dr. J. Magazinius, MSc. N. Kobeissi, Dipl.-Ing. A. Aranguren

Index

Introduction

> *“Your First Smart Digital Wallet. Secure Convenient Powerful”*
>
> From https://token.im/

This report documents the findings of a security assessment targeting the imToken product developed and maintained by ConsenLabs. The project was executed by Cure53 in two rounds in 2018 and yielded eight security-relevant discoveries. Carried out in May 2018 (Phase 1) and July 2018 (Phase 2), this assessment entailed a dedicated audit of the items placed in scope by the ConsenLabs entities.

In terms of scope, the main test target during Phase 1 was the wallet implementation to be used by the imToken mobile applications for Android and iPhone. Although Cure53 originally assumed that the test would cover the entire compound of the applications, this was clarified to only encompass the wallet item in the ConsenLabs product family. Consequently, changes were made in the Cure53 testing team setup in order to best

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

reflect the skillset and foci needed for the completion of the actual test requirements. For Phase 2, the Cure53 auditors primarily focused on the new features, specifically the SegWit and EOS support. Since the Cure53 team was already familiar with a considerable part of the codebase, the audit could be much more efficient and allowed for in-depth analyses.

As regards the resources, five Cure53 testers investigated the scope against a time budget of 11.5 days. The resources should be deemed as sufficient and the testing team believes to have reached a very good coverage of the agreed upon scope. The numbers are collated to reflect the resources dedicated to the project across both phases.

The methodology chosen for this assessment of the imToken wallet was the so-called white-box approach and this was upheld for both phases of the project. Under this premise, Cure53 was granted access to relevant sources and binaries. Further necessary insights and internal knowledge was also shared with Cure53 when necessary. In line with the white-box strategy, the communications channels remained open between the in-house and testing teams. The exchanges occurred via email. It should be emphasized that, aside from a slight confusion around the scope at the very beginning of the test, the investigations went smoothly. Good communications persisted during Phase 2, as the issues were once again live-reported by Cure53, thus enabling the ConsenLabs team to address them while the tests were still ongoing.

The testers managed to uncover five issues during Phase 1, demarcating four as security vulnerabilities and noting one as a general weakness. In Phase 2 of the audit, Cure53 increased the total number of findings to seven with two additional miscellaneous issues. Moreover, Cure53 decided to raise the severity of one issue unveiled in Phase 1 to *“High”* (see CL-01-005). While no discoveries posed a threat at a “*Critical*” level, a vulnerability spotted during Phase 1 was determined to be of a significant, “*High”*-impact. This means that, all in all, two problems were deemed as “*High”* in terms of possible negative implications after the completion of both rounds of testing. The issues were live-reported via email to the imToken maintainers to facilitate an early start to the process of crafting and deployment of fixes.

In the following sections, the scope objects are first discussed in a brief yet detailed manner. After that, the document provides a case-by-case discussion of findings from both Phase 1 and Phase 2, furnishing mitigation advice when applicable. Finally, the *Conclusion* section offers some broader impressions gained by Cure53 during this broader assignment and sheds light on the general verdict about the overall security posture of the tested imToken wallet implementation and its related features.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Round 1: ConsenLabs / imToken Mobile Apps**
  - Sources were shared with Cure53
  - Binaries were shared with Cure53
- **Round 2: Newly added ConsenLabs Wallet Features**
  - Sources were shared with Cure53

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *CL-02-001*) for the purpose of facilitating any future follow-up correspondence.

To distinguish issues stemming from Phase 1 from those uncovered later in Phase 2, the issues spotted in May 2018 have the heading of CL-01 and those unveiled in July 2018 begin with CL-02 heading in the ensuing documentation.

CL-01-001 Swift: Path traversal in *LocalFileStorage.swift* (Medium)

The public functions of the *LocalFileStorage* class, which implements the functionality responsible for loading, storing and deleting wallets, a concern around path traversal was raised. Specifically, a *walletID* is used to build a filesystem path and is neither sanitized nor checked for path traversals. In other words, the application using the library has to implement this missing check itself. This approach is especially dangerous because the exposed functions do not make it clear that the *walletID* is in fact a filename. This could allow an attacker to break out of the default *"/Users/<username>/Documents/wallet/"* path and write, delete or read arbitrary files.

Affected File
ios-token-core/Sources/Storage/LocalFileStorage.swift

Affected Code

```
public func loadWalletByIDs(_ walletIDs: [String]) -> [BasicWallet] {
    // [...]
}
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
public func deleteWalletByID(_ walletID: String) -> Bool {
    return deleteFile(walletID)
}

public func flushWallet(_ keystore: Keystore) -> Bool {
    // via the keystore.id
    return writeContent(keystore.dump(), to: keystore.id)
}
```

From the code examples, it can be inferred that the *walletIDs* should follow a specific structure, namely *xxxxxxxx-yyyy-zzzz-aaaa-bbbbbbbbbbbb.* This means the *LocalFileStorage* class could check that this format is followed or ensure that only files inside of the *walletsDirectory* can be accessed. The impact on iOS is low due to the application sandboxing though the code. It should be added that this approach could also be extrapolated and used on OSX. At the same time, it is still recommended to address this issue in a more comprehensive way in order to make the library more robust. The audited Java sources show a similar pattern and should also implement a protection against path traversal attacks.

Fix Notes: *The issue has been fixed by passing the walletID to a WalletIDValidator first.*

CL-01-003 Crypto: Use of malleable encryption modes *(Medium)*

The *TokenCore* library uses *AES* in either *CBC* or *CTR* encryption mode. Both modes lack authentication, meaning that ciphertext can, to a varying degree, be manipulated in a predictable and intelligible way. In the *CTR* mode a stream of pseudo-random numbers is generated and subsequently *xor:ed* with the plaintext to form the ciphertext. In essence, flipping a bit in the ciphertext will result in the same bit being flipped in the plaintext. Similarly, in the *CBC* mode, the plaintext is *xor:ed* with the ciphertext of the previous block. In this context, flipping a bit in a ciphertext block will render that block useless upon decryption. What is more, it will also reliably flip the corresponding bit in the ensuing block of plaintext.

If the plaintext is partially or fully known, an attacker can modify the ciphertext to produce a plaintext that is - to a large extent - under their control.

Affected File
android-token-core\app\src\main\java\org\consenlabs\tokencore\foundation\crypto\AES.java

Affected Code

```
public enum AESType {
    CTR, CBC
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
}

public static byte[] encryptByCTR(byte[] data, byte[] key, byte[] iv) {
  return doAES(data, key, iv, Cipher.ENCRYPT_MODE, AESType.CTR,
              "PKCS5Padding");
}

public static byte[] decryptByCTR(byte[] ciphertext, byte[] key, byte[] iv) {
  return doAES(ciphertext, key, iv, Cipher.DECRYPT_MODE, AESType.CTR,
              "PKCS5Padding");
}

public static byte[] encryptByCBC(byte[] data, byte[] key, byte[] iv) {
  return doAES(data, key, iv, Cipher.ENCRYPT_MODE, AESType.CBC,
              "PKCS5Padding");
}

public static byte[] decryptByCBC(byte[] ciphertext, byte[] key, byte[] iv) {
  return doAES(ciphertext, key, iv, Cipher.DECRYPT_MODE, AESType.CBC,
              "PKCS5Padding");
}

public static byte[] encryptByCTRNoPadding(byte[] data, byte[] key, byte[] iv) {
  return doAES(data, key, iv, Cipher.ENCRYPT_MODE, AESType.CTR, "NoPadding");
}

public static byte[] decryptByCTRNoPadding(byte[] ciphertext, byte[] key,
       byte[] iv) {
  return doAES(ciphertext, key, iv, Cipher.DECRYPT_MODE, AESType.CTR,
              "NoPadding");
}

public static byte[] encryptByCBCNoPadding(byte[] data, byte[] key, byte[] iv) {
  return doAES(data, key, iv, Cipher.ENCRYPT_MODE, AESType.CBC, "NoPadding");
}

public static byte[] decryptByCBCNoPadding(byte[] ciphertext, byte[] key,
       byte[] iv) {
  return doAES(ciphertext, key, iv, Cipher.DECRYPT_MODE, AESType.CBC,
              "NoPadding");
}
```

To mitigate this issue an authenticated block cipher mode, like *GCM*, should be used. This will effectively prevent an attacker from modifying a ciphertext and remaining undetected. Should the ciphertext be modified, it will fail to decrypt.

Fix Notes: *As these functions are only used to implement the Ethereum standard, and not for general use, ConsenLabs will make no changes.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

CL-01-004 Crypto: *PKCS5* padding scheme vulnerable to oracle attacks *(Medium)*

In the cryptographic implementation *PKCS5* is used as the padding scheme for the *AES* block cipher. It is known that the *PKCS5* padding scheme is vulnerable to oracle attacks. Specifically, if an attacker is able to repeatedly decrypt the ciphertext and observe the outcome, whether decryption succeeded or failed, this can be used to obtain the plaintext message.

Affected File

android-token-core\app\src\main\java\org\consenlabs\tokencore\foundation\crypto\AES.java

Affected Code

```
public static byte[] encryptByCTR(byte[] data, byte[] key, byte[] iv) {
  return doAES(data, key, iv, Cipher.ENCRYPT_MODE, AESType.CTR,
               "PKCS5Padding");
}

public static byte[] decryptByCTR(byte[] ciphertext, byte[] key, byte[] iv) {
  return doAES(ciphertext, key, iv, Cipher.DECRYPT_MODE, AESType.CTR,
               "PKCS5Padding");
}

public static byte[] encryptByCBC(byte[] data, byte[] key, byte[] iv) {
  return doAES(data, key, iv, Cipher.ENCRYPT_MODE, AESType.CBC,
               "PKCS5Padding");
}

public static byte[] decryptByCBC(byte[] ciphertext, byte[] key, byte[] iv) {
  return doAES(ciphertext, key, iv, Cipher.DECRYPT_MODE, AESType.CBC,
               "PKCS5Padding");
}
```

A modern cipher mode such as *GCM* does not require a padding scheme and is therefore not vulnerable to this type of attacks. It is recommended to stop using the *CBC* and *CTR* cipher modes and migrate to an authenticated cipher mode like *GCM* or similar.

Fix Notes: *As these functions are only used to implement the Ethereum standard, and not for general use, ConsenLabs will make no changes.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

CL-01-005 Crypto: Static keys hardcoded into library encryption features *(High)*

It was found that the *HDMnemonic Keystore* implementation in both the Android and Swift libraries expected hardcoded static symmetric keys to be encoded for all encryption operations. This approach is problematic for a number of reasons:

- Any user with a copy of the client application can simply extract the keys either be decompiling the application or by dumping its memory allocations.
- Changing the static keys seems to result in the included unit-tests failing, thereby making the chances of properly testing the application uncertain.

Affected File
android-token-core\app\src\main\java\org\consenlabs\tokencore\wallet\keystore\HDMnemonicKeystore.java

Affected Code

```
public static String XPubCommonKey128 = "B888D25EC8C12BD5043777B1AC49F872";
public static String XPubCommonIv = "9C0C30889CBCC5E01AB5B2BB88715799";
```

It is recommended, if possible, to deprecate this library functionality completely. Instead, it is advised to always focus on secure key generation from Mnemonic passphrases or from passwords, in the same fashion that this is handled in other parts of the library.

Fix Notes: *ConsenLabs states that the keys are to be replaced once the code goes live.*

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

CL-01-002 Swift: Failing silently on error conditions *(Medium)*

Several functions have error conditions which are handled differently. For example, *BigNumber* returns *"0"* for a parsing error and the *Data* extension of *tk_hexCharToBinary* returns the string *"-1"*. In case of the *BigNumber* implementation, the developers are aware of this issue as indicated by the *TODO* comment. Simultaneously, this ticket was created to point out that this matter should be urgently addressed for a library because the product may be handed over or used by other developers who are not aware of these unexpected behaviours.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected File
ios-token-core/Sources/Utils/BigNumber.swift

Affected Code

```
    } else {
        // Parse fail!
        value = BigUInt(0) // TODO: handle parse error
    }
}
return BigNumber(value: value, padding: padding, bytesLength: bytesLen)
```

Affected File
ios-token-core/Sources/Utils/Data+Extensions.swift

Affected Code

```
 static func tk_hexCharToBinary(char: Character) -> String {
    if let value = Int(String(char), radix: 16) {
      var bin = String(value, radix: 2)
      while bin.count < 4 {
        bin = "0" + bin
      }
      return bin
    } else {
      return "-1"
    }
  }
```

Error conditions should be clearly documented and consistent for all functions. It is also recommended to throw exceptions rather than use return values to carry error information.

Fix Notes: *The specific case of returning -1 or 0000 has been changed to return uniformly 0 values on errors.*

CL-02-001 Re-Test: Unfixed silent failure on several error conditions *(High)*

In the first phase of the audit, it was reported in CL-01-002 that the library functions generally fail silently on error conditions by returning inconsistent return values. In a worst-case scenario, an application using these functions could mistake an erroneously returned value as a correct value and use it. Therefore, it was recommended in Phase 1 guidelines to use proper exception features of the languages as this would make error conditions clear.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

As can be read in the fix notes accompanying said issue, ConsenLabs addressed the issue by changing the return value. It now adheres to a more uniform rule of returning "*zero*". Sadly, this still does not prevent accidental misuse. Consequently, it is believed that this is a serious problem, especially if the library was to be used by other developers, likely unaware of these unexpected behaviors. Cure53 decided to emphasize the importance of exceptions in Phase 2 by raising the severity of this bug from "*Medium"* to "*High"*.

Affected File:
ios-token-core/Sources/Utils/BigNumber.swift

Affected Code:

```
//[...]
   } else {
      // Parse fail!
      value = BigUInt(0)
   }
}
return BigNumber(value: value, padding: padding, bytesLength: bytesLen)
```

Affected File:
ios-token-core/Sources/Utils/Data+Extensions.swift

Affected Code:

```
/// Convert hex character to binary, if input is not a valid hex character,
return "0000".
  static func tk_hexCharToBinary(char: Character) -> String {
    if let value = Int(String(char), radix: 16) {
      var bin = String(value, radix: 2)
      while bin.count < 4 {
        bin = "0" + bin
      }
      return bin
    } else {
      return "0000"
    }
  }
```

Mixing valid data with error conditions, solely based on the value itself is generally a bad pattern, especially when the language itself supports exceptions. It is recommended to throw meaningful exceptions[1], rather than use return values to carry error information.

[1] https://docs.swift.org/swift-book/LanguageGuide/ErrorHandling.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Fix Notes: *ConsenLabs decided to cease the use of exceptions as recommended. It has been stated that errors are handled in a uniform way through return values.*

CL-02-002 Swift: Potentially insecure file storage *(Medium)*

While the wallet's private keys are encrypted and the application's sandbox on iOS is very strong, it is still recommended to implement file storage with the recommended APIs provided by Apple.

Affected File:
ios-token-core/Sources/Storage/LocalFileStorage.swift

Affected Code:

```
func writeContent(_ content: String, to filename: String) -> Bool {
  do {
    let filePath = walletsDirectory.appendingPathComponent(filename).path
    try content.write(toFile: filePath, atomically: true, encoding: .utf8)
    return true
  } catch {
    debugPrint("Error: \(error)")
    return false
  }
}
```

It is strongly advised to implement a file-system access via the *NSFileManager* that supports various levels of the *NSFileProtection* classes. iOS will then encrypt the files automatically. For example, with *NSFileProtectionComplete*, the system will derive a key from the user's passcode and discard the key from memory shortly after the device is locked, thus enabling strong data protection at rest. The iOS security guide can be consulted for more details about the different data protection classes[2].

Fix Notes: *ConsenLabs decided against changing the storage design at present.*

CL-02-003 Swift: *tk_dataFromHexString* might return wrong result *(Low)*

It was found that the utility function, namely *tk_dataFromHexString,* does not check if the hex string passed to it is properly padded. If an odd length hex string is passed to this function, the data will not be converted properly. Because C-strings are null-terminated, this off-by-one error does not cause any immediate issues but it might result in wrongly converted values.

[2] https://www.apple.com/business/docs/iOS_Security_Guide.pdf

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

PoC:
"0x1234" → parsed correctly as *<1234>*
"0x123" → parsed as *<1203>*, but expected is *<0123>*

Affected File:
ios-token-core/Sources/Utils/Hex.swift

Affected Code:

```
func tk_dataFromHexString() -> Data? {
    if Hex.hasPrefix(self) {
      return tk_substring(from: 2).tk_dataFromHexString()
    }

    guard let chars = cString(using: .utf8) else { return nil}
    let length = count

    guard let data = NSMutableData(capacity: length / 2) else { return nil }
    var byteChars: [CChar] = [0, 0, 0]
    var wholeByte: CUnsignedLong = 0

    for i in stride(from: 0, to: length, by: 2) {
      byteChars[0] = chars[i]
      byteChars[1] = chars[i + 1]
      wholeByte = strtoul(byteChars, nil, 16)
      data.append(&wholeByte, length: 1)
    }

    return data as Data
  }
```

It is recommended to ensure that the hex string is fully padded before attempting to extract the single bytes.

Fix Notes: *ConsenLabs has fixed the issue through an improved algorithm. The fix was verified by Cure53.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The overall results of this Cure53 two-stage assessment of the ConsensLabs' scope are rather positive. Five testers, who were tasked with investigating the ConsenLabs imToken wallet library and its later added new features in 2018, concluded that the majority of security goals communicated by the project have been sufficiently met.

To give some specific comments, it should be mentioned as a plus that the TokenCore library does not re-implement cryptographic primitives and relies on the known third-party libraries. While the Swift implementation uses mostly CoreBitcoin and CryptoSwift, the Java implementation hinges upon Spongy Castle and bitcoinj. It has been concluded that these cryptographic primitives have been deployed mostly in a safe manner.

Judging by the overall findings, there needs to be some caution moving forward because the actual issues that may emerge in production will largely depend on the usage of the library. In order to prevent security issues on certain setups, a broad recommendation is to consider more robust encryption modes and padding schemes than those currently in place. On a similar note, there is an undocumented option to configure hardcoded static keys in a library and this setting should be deprecated. There is no need to ship this to clients as part of mobile applications.

Based on Phase 1 results, it should be underscored that the safety of a cryptographic library does not only depend on the security issues of the implementation. In this context, it is slightly problematic that the library appears to be a "work in progress". There are multiple comments in the source code indicating a somewhat unstable development, for instance evident from warnings like "Don't use this key in production". At any rate, a proper documentation and notation of all shortcomings are crucial for understanding secure default values and handling of errors. These aspects must be resolved as they are indispensable for a safe use of the products by third-parties.

During Phase 2, the above concerns have been confirmed once again as valid and must be reiterated. In particular, no significant changes have been made to address the Cure53's worry about the lacking developer documentation. Similarly, consistency in error handling continued to exhibit shortcomings as well. As a result, Cure53 has decided to raise the severity of the issue related to this realm from "Medium" to "High" in order to emphasize its general importance. The key point that must be understood would be that the developers wishing to build applications with the tested library would be forced to grasp the "ins and outs" of the ConsensLabs library very well in order to safely use it.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To sum up, on the one hand a good indicator is that no issues of “Critical” severity have been found on the tested scope of the imToken wallet, neither in Phase 1 nor in Phase 2. In light of this conclusion, Cure53 believes that no major risks can hinder a public release of the source code. On the other hand, it is recommended to issue a warning that the project should not be blindly used without a deeper understanding of the implementation. By connecting such caution with an overall acceptable security posture of the code, it can be stated that the ConsenLabs’ imToken project meets adequate thresholds from a security standpoint.

Cure53 would like to thank Blue Yang and Elian Yu from the ConsenLabs/ imToken team for their excellent project coordination, support and assistance, both before and during this assignment.