DOCUMENT NUMBER MTR120646
MITRE TECHNICAL REPORT

iOS iPad Mobile Application Security Audit

Sponsor: MITRE MIP MSR
Dept. No.: E547 / G021
Project No.: 1912M154-GA

Gregg Ganley, John Butterworth, Shawn Valle

December 2012 (v15)

Bedford, MA

Approved For Public Release

This page intentionally left blank.

Introduction

This report contains information regarding the forensic analysis and penetration testing of an iPad application running on multiple Apple iOS devices. The application under test is an open source MITRE research prototype mobile patient health reader that has a requirement to increase its application security in a measured way, called hReader. The application securely connects to a server to retrieve health information which it then secures locally for offline usage. In a production environment, the application could contain sensitive patient health information that implicitly needs enhanced security to ensure the data is protected. The development team requested a red-team to measure the security level and to iteratively audit the application using test patient data also known as "synthetic patient data".

The ultimate goal of this effort is to create guidance that will help developers and testers to enable any iOS mobile application with robust security while keeping the iPad application functionality acceptable to end-users.

As a baseline, the audit was performed assuming that an attacker has physical access to the device running the application under test. The audit team was able to forensically inspect the application and its data content, pre- and post-jailbreaking. Initially, patient information was found easily, which could be valuable to an attacker. Through iterative testing and reengineering, the application has become increasingly hardened against malicious attacks and is better able to protect sensitive user data.

Security audit, forensics and controls were researched and developed as part of a MITRE Research initiative called iMAS – iOS Mobile Application Security. iMAS is an iOS secure application framework, that endeavors to raise application security beyond the Apple provided security model through an open source community.

To measure the security a DISA/DoD Application STIG (Security Technical Implementation Guide) was used. Before the iMAS security controls were implemented, the application was 16% compliant with the STIG criteria. Upon final test, the application's STIG compliance increased to 90% compliant, with 72 out of 80 applicable criteria met. This represents a 5X or 500% compliance improvement and one could argue a significant increase in security hardening. As a further measure, this iterative security audit and development effort cost approximately 3 staff months.

As of May 2013, further security testing is underway and an addendum with additional hReader vulnerabilities and mitigations will be released later this summer/fall.

In this document, each round of audit testing will be discussed in turn in the sections that follow.

This page intentionally left blank.

Table of Contents

1 Application under test 1
2 Audit Version 1 (V1) Findings – iOS Forensics Discovery 1
2.1 iOS iPad Keychain and hReader Passcode 1
2.1.1 Passcode Recommendations 4
2.2 Home Button Screen Capture 4
2.3 Keyboard Cache Data Leaks 5
2.3.1 Keyboard Recommendations 5
2.4 hReader Patient Data 5
2.4.1 Application Data File Recommendations 7
2.5 hReader Developer Identity Data 7
2.5.1 hReader Developer Data Mitigation 7
2.6 hReader Authentication Bypass 7
2.6.1 hReader Authentication Bypass CWE Linking 8
2.6.2 hReader Authentication Bypass Mitigations 8
2.7 hReader Reverse Engineering Revealing Attack Vectors 8
2.7.1 hReader Reverse Engineering Revealing Attack Vector Mitigations 10
2.8 iOS General Security 10
3 Audit Version 2 (V2) Findings – iOS Forensics Discovery 11
3.1 iOS Bruteforce Passcode 11
3.2 Audit v2 Findings – iOS Keychain and hReader Passcode 12
3.3 Audit v2 Findings – Home Button Screen Capture 13
3.4 Audit v2 Findings – Patient Data 14
3.4.1 Recommendations 15
3.5 Audit v2 Findings – Developer Identity Data 15
3.6 Audit v2 Findings – hReader Authentication Bypass 15
3.7 Audit v2 Findings – Jailbreak Detection Avoidance 15
3.7.1 Recommendations 16
3.8 Audit v2 Findings – Runtime Analysis 16
3.9 Audit v2 – hReader Network Analysis 17
4 Audit Version 3 (V3) Findings 19
4.1 Audit v3 Findings – hReader SQLite Cipher Analysis 19
4.1.1 hReader SQLite Cipher Analysis Attack Vectors 20
4.2 Audit v3 Findings – hReader Source Code Analysis 20

4.3 Audit v3 Findings – hReader STIG Compliance (Initial Review) 20
5 Audit Version 4 (V4) Findings 21
5.1 hReader DISA/DoD Applications STIG Compliance (Final Review) 21
6 Bibliography 21
Appendix A STIG Details A-1

List of Figures

Figure 1 Keychain Message 1
Figure 2 Passcode and Questions and Answers 2
Figure 3 Passcode and Questions and Answers 3
Figure 4 hReader Home Button Screen Capture – Test Passed 4
Figure 5 hReader Keyboard Cache Leak 5
Figure 6 hReader Patient Data 6
Figure 7 hReader Patient Data (detail) 6
Figure 8 Information Identifying One of the hReader Developers 7
Figure 9 The "_HRAppDelegate_PINCodeViewController_isValidPIN_" Function as it appears in the hReader disassembly 9
Figure 10 Class-Dump-Z Utility Output 10
Figure 11 Bruteforce Algorithm Acquiring Passcode 12
Figure 12 iOS Keychaing Text File Showing hReader Sections 13
Figure 13 Family Member List Screen Capture 14
Figure 14 hReader/TestFlight Key Exchange 17
Figure 15 "Little" Johnny Smith in Ciphertext 18
Figure 16 SQLite encrypted with SQLCipher – unable to read database 19

This page intentionally left blank.

1 Application under test

This report contains information regarding the forensic analysis and penetration testing of the hReader iPad application running on multiple Apple iOS devices. hReader is an open source MITRE research prototype mobile patient health reader that has a requirement to increase its application security in a measured way. The development team requested a red-team to measure the security level and to iteratively audit the application using test patient data also known as "synthetic patient data".

2 Audit Version 1 (V1) Findings – iOS Forensics Discovery

hReader was installed on an iPad which was then subsequently jailbroken. Through forensic testing the following vulnerabilities and application test data were discovered. Steps to mitigate each issue are detailed for each vulnerability.

2.1 iOS iPad Keychain and hReader Passcode

The iOS keychain was used to store hReader's application password and password recovery questions. When the iPad is locked and the system PIN is needed to unlock the iPad, the iOS keychain is secure and information cannot be read from the keychain using hacker tools. When an attacker attempts to access the keychain, the following message (Figure 1) is displayed:

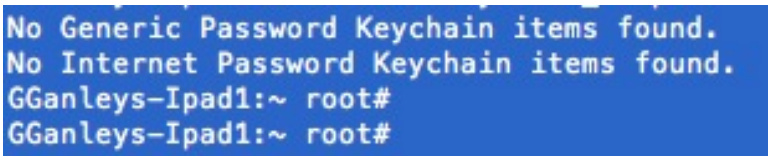


Figure 1 Keychain Message

When the iPad PIN is known and entered, the keychain and many other secure application assets become available. Using forensic procedures and tools, the application password and the associated questions and answers were revealed in plain text.

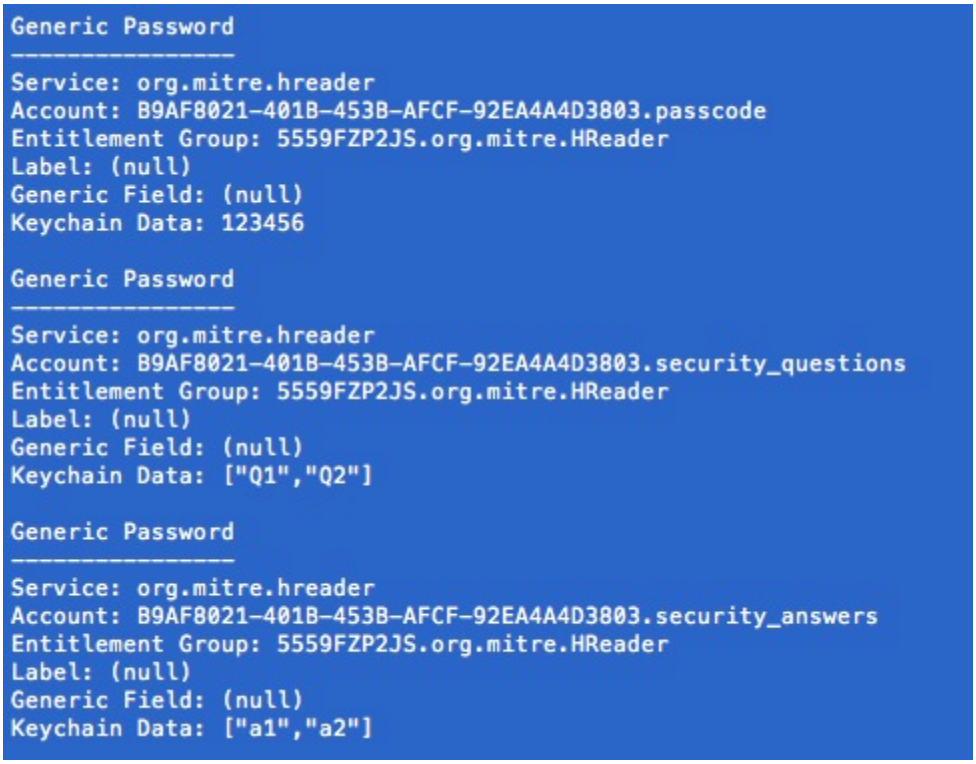


Figure 2 Passcode and Questions and Answers

Figure 2 shows the passcode section with "Keychain Data" containing the passcode as 123456. Also, the passcode reset questions and answers are listed, specifically, "Q1", "Q2", "a1", and "a2". Typically these items would represent a real question and answer in full text, however, for testing purposes, short questions and answers were used instead. With access to this information and attacker can easily login in to the hReader application and infiltrate/access patient data.

Next, the hReader application was uninstalled and deleted from the iPad and then re-installed. It was found that a second set of hReader passcode and questions/answers were found, as shown below in Figure 3. This was in addition to the original passcode from the first install. Lastly, after hReader was completely uninstalled from the iPad, the keychain continued to hold the hReader login information.

```
Generic Password
----------------
Service: org.mitre.hreader
Account: B9AF8021-401B-453B-AFCF-92EA4A4D3803.passcode
Entitlement Group: 5559FZP2JS.org.mitre.HReader
Label: (null)
Generic Field: (null)
Keychain Data: 123456

Generic Password
----------------
Service: org.mitre.hreader
Account: B9AF8021-401B-453B-AFCF-92EA4A4D3803.security_questions
Entitlement Group: 5559FZP2JS.org.mitre.HReader
Label: (null)
Generic Field: (null)
Keychain Data: ["Q1","Q2"]

Generic Password
----------------
Service: org.mitre.hreader
Account: B9AF8021-401B-453B-AFCF-92EA4A4D3803.security_answers
Entitlement Group: 5559FZP2JS.org.mitre.HReader
Label: (null)
Generic Field: (null)
Keychain Data: ["a1","a2"]

Generic Password
----------------
Service: com.apple.managedconfiguration
Account: Private
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: (null)

Generic Password
----------------
Service: org.mitre.hreader
Account: 2F40B29E-59B5-48D1-B62D-CB4AB0487D03.passcode
Entitlement Group: 5559FZP2JS.org.mitre.HReader
Label: (null)
Generic Field: (null)
Keychain Data: 123456

Generic Password
----------------
Service: org.mitre.hreader
Account: 2F40B29E-59B5-48D1-B62D-CB4AB0487D03.security_questions
Entitlement Group: 5559FZP2JS.org.mitre.HReader
Label: (null)
Generic Field: (null)
Keychain Data: ["Q11","Q22"]

Generic Password
----------------
Service: org.mitre.hreader
Account: 2F40B29E-59B5-48D1-B62D-CB4AB0487D03.security_answers
Entitlement Group: 5559FZP2JS.org.mitre.HReader
Label: (null)
Generic Field: (null)
Keychain Data: ["a11","a22"]
```

Figure 3 Passcode and Questions and Answers

2.1.1 Passcode Recommendations

Even though the iPad PIN code was needed to unlock the iPad, once unlocked, hReader's application data was accessible, leaving it vulnerable to attack. To reduce this risk, the following is recommended:

1. Do not store the application passcode in the iOS keychain or anywhere else on the device, instead use it to decrypt a known token phrase, that if verified allows entry to the application
2. Encrypt, as part of the application, all data that is stored in the iOS keychain
3. On application delete, remove all hReader items from the keychain. This may not be possible, so we recommend storing this information in a file as part of the application bundle within the application sandbox.
4. Ensure hReader items are not double-stored in the keychain leaving a trail of past passcodes

2.2 Home Button Screen Capture

During hReader use, the home button was pressed several times returning the user to the iPad desktop page. Upon repeating this several times, forensic tools were used to discover hReader screen shots. Fortunately, security measures were taken to ensure that actual hReader patient information screens were not being captured; instead a blank, black screen was captured. This is a positive result, and one should consider this potential security issue resolved.

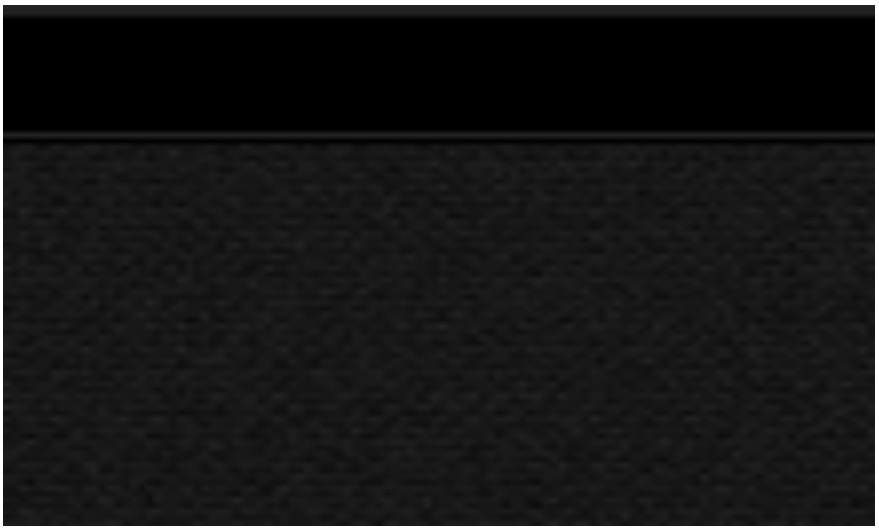


Figure 4 hReader Home Button Screen Capture – Test Passed

2.3 Keyboard Cache Data Leaks

During hReader use, the feedback feature was used to type several test messages using the iPad standard keyboard. The data typed was a simple phrase, "The lazy dog crossed the river" with the subject of "now". Using forensic techniques, the iPad's keyboard cache was found and searched with the results shown in Figure 5.

```
[ gganley ~/projs/imas/hreader ] $ strings mobile_dynamic-text.dat | grep -i lazy
lazy
[ gganley ~/projs/imas/hreader ] $ strings mobile_dynamic-text.dat | grep -i dog
dogpatch
[ gganley ~/projs/imas/hreader ] $ strings mobile_dynamic-text.dat | grep -i crossed
crossed
[ gganley ~/projs/imas/hreader ] $ strings mobile_dynamic-text.dat | grep -i river
river
[ gganley ~/projs/imas/hreader ] $ strings mobile_dynamic-text.dat | grep -i now
know
know
nowoncampus
```

Figure 5 hReader Keyboard Cache Leak

The figure above reveals the phrase was found in the keyboard cache, indicating a data leak. If more patient data entry text boxes are used in the application (e.g.; application password Q/A fields), it is possible that they could leak sensitive information in the clear to the iPad keyboard cache.

2.3.1 Keyboard Recommendations

Programmatically, disable the autocorrect functionality for each hReader text field that is considered sensitive.

2.4 hReader Patient Data

Using forensic tools and techniques, the patient data was located. The data was found in the hReader.xml file, a portion of its contents are shown in Figures 6 and 7.

Figure 6 hReader Patient Data

Figure 7 hReader Patient Data (detail)

Figure 7 shows a detail of the XML data recovered from hReader. Anyone viewing this data can clearly see the places in the file indicating a patient's BMI, blood pressure, and other pieces of health data.

2.4.1 Application Data File Recommendations

It is recommended that sensitive application data be stored in an encrypted file that is secured using the application's passcode. Another option is SQLCipher in which an encrypted layer of security is placed on top of a SQLite database.

2.5 hReader Developer Identity Data

When attempting to debug the hReader application using the GNU debugger (GDB), a developer's name was found. The attempt resulted in a number of errors. Each of the errors is shown in Figure 8; each containing the developer's name in the directory path.

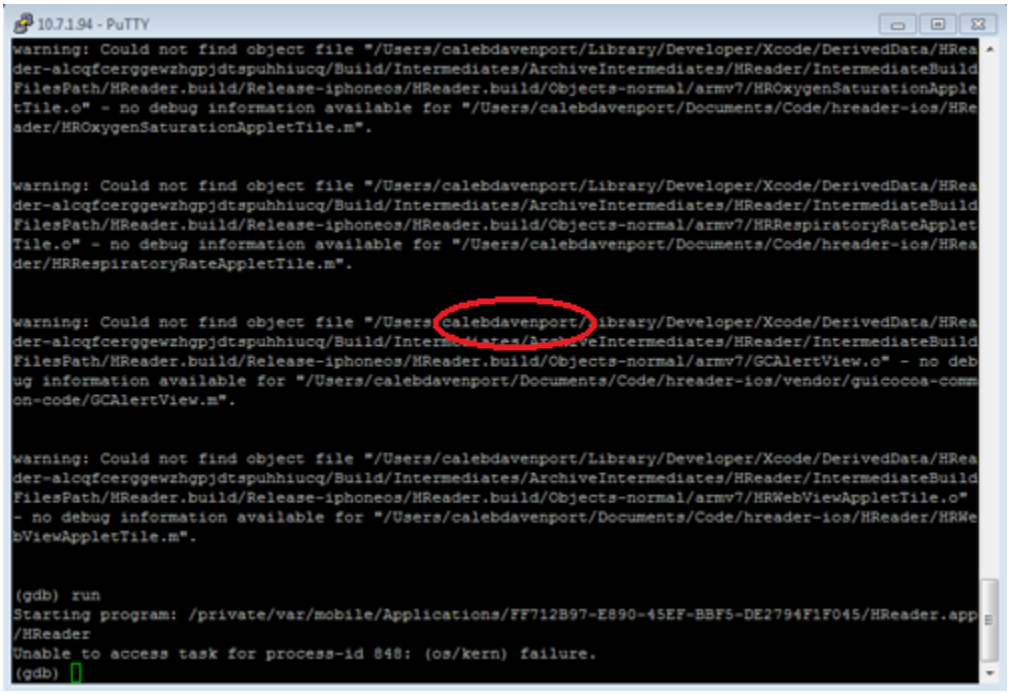


Figure 8 Information Identifying One of the hReader Developers

The developer's name appearing is problematic as it provides an opportunity for a social engineering attack. Once a malicious user has the developer's name, they can simply use Google to obtain the developer's online resume and identify his place of employment.

2.5.1 hReader Developer Data Mitigation

Developers should ensure that any personally identifiable information (PII) is excluded from their working development path.

2.6 hReader Authentication Bypass

The hReader application password and security questions were able to be bypassed.

When an attacker logs onto the iPad via SSH (e.g., as root) and executes an instance of hReader from the command-line, hReader is unable to find the credentials for root in the iOS keychain. hReader assumes this is a new installation and prompts the attacker to set their own application password and security/challenge questions, providing full access to the patient data.

To duplicate the bypass process, use the following steps:

1. On a jailbroken iPad, SSH to the device and login as root.
    a. Optionally, navigate to the directory where the hReader application resides.
2. Start an instance of hReader (the "&" is also optional, but frees hReader from the shell):
    a. # ./hReader &
3. On the iPad, use the touchscreen to bring up the hReader application. Technically the root-user instance is already started but appears to be minimized.
4. Enter a PIN and challenge questions for the root user.

2.6.1 hReader Authentication Bypass CWE Linking

The PIN and challenge question bypass loophole most closely matches CWE-305: Authentication Bypass by Primary Weakness. CWE-305 is summarized as follows:

> "The authentication algorithm is sound, but the implemented mechanism can be bypassed as the result of a separate weakness that is primary to the authentication error." (The MITRE Corporation 2012)

hReader correctly applies authentication using the iOS keychain, however, it should be noted that the iOS keychain mechanism is vulnerable.

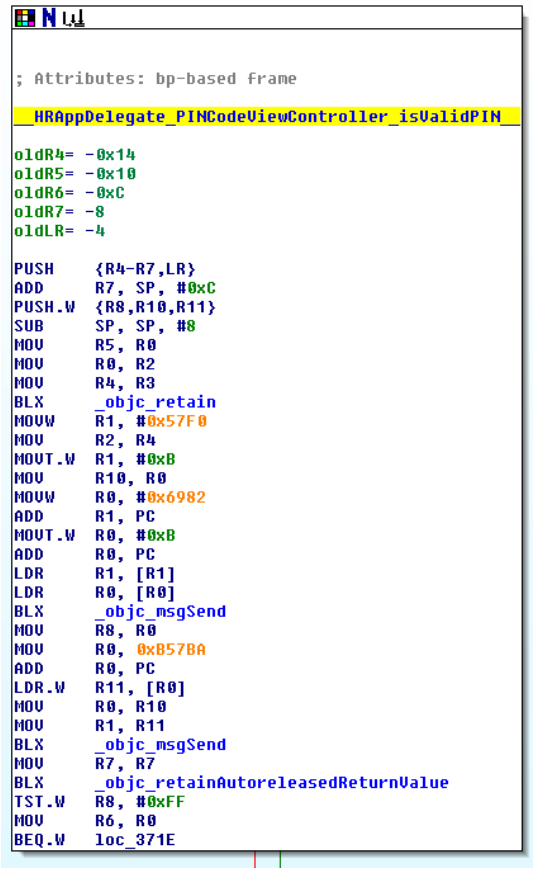
2.6.2 hReader Authentication Bypass Mitigations

The mitigation for this vulnerability is the same as the one provided in section 1.1.1 (Passcode Recommendations): Do not store the passcode in the iOS keychain.

2.7 hReader Reverse Engineering Revealing Attack Vectors

An attempt to gather sensitive data on how the hReader binary image could be modified maliciously was performed using the IDA Pro disassembler. It was apparent that this version was a debug build because function names were present in the disassembly. When this debug information is stripped out of the binary, the nature of these functions become obfuscated to the point where finding the same information, as provided below, would require an extensive amount of time and effort.

In Figure 9, it shows that the function "__HRAppDelegate_PINCodeViewController_isValidPIN__" was found.



```
; Attributes: bp-based frame

__HRAppDelegate_PINCodeViewController_isValidPIN_

oldR4= -0x14
oldR5= -0x10
oldR6= -0xC
oldR7= -8
oldLR= -4

PUSH    {R4-R7,LR}
ADD     R7, SP, #0xC
PUSH.W  {R8,R10,R11}
SUB     SP, SP, #8
MOV     R5, R0
MOV     R0, R2
MOV     R4, R3
BLX     _objc_retain
MOVW    R1, #0x57F0
MOV     R2, R4
MOVT.W  R1, #0xB
MOV     R10, R0
MOVW    R0, #0x6982
ADD     R1, PC
MOVT.W  R0, #0xB
ADD     R0, PC
LDR     R1, [R1]
LDR     R0, [R0]
BLX     _objc_msgSend
MOV     R8, R0
MOV     R0, 0xB57BA
ADD     R0, PC
LDR.W   R11, [R0]
MOV     R0, R10
MOV     R1, R11
BLX     _objc_msgSend
MOV     R7, R7
BLX     _objc_retainAutoreleasedReturnValue
TST.W   R8, #0xFF
MOV     R6, R0
BEQ.W   loc_371E
```

Figure 9 The "_HRAppDelegate_PINCodeViewController_isValidPIN_" Function as it appears in the hReader disassembly.

An attacker might assume that this function returns a Boolean value and attempt to modify the application binary hex-code to return both a value that tells the caller that "yes a valid PIN has been entered" or simply bypass the function-call and return-value check. This will only work on a jailbroken iPad that has code-signing turned off. Even under this condition, it is not an easy task to perform.

Running the "Class-Dump-Z" utility on the binary code revealed additional information about hReader's authentication-related functions (Figure 10).

```
@property(retain, nonatomic) UIWindow* window;
+(id)managedObjectContext;
+(id)persistentStoreCoordinator;
-(void).cxx_destruct;
-(void)resetPasscode:(id)passcode;
-(id)securityQuestions;
-(void)securityQuestionsController:(id)controller didSubmitQuestions:(id)questions answers:(id)answers;
-(unsigned)numberOfSecurityQuestions;
-(BOOL)PINCodeViewController:(id)controller isValidPIN:(id)pin;
-(void)PINCodeViewController:(id)controller didCreatePIN:(id)pin;
-(unsigned)PINCodeLength;
-(void)applicationProtectedDataDidBecomeAvailable:(id)applicationProtectedData;
-(void)applicationProtectedDataWillBecomeUnavailable:(id)applicationProtectedData;
-(void)applicationDidBecomeActive:(id)application;
-(void)applicationWillEnterForeground:(id)application;
-(void)applicationDidEnterBackground:(id)application;
-(void)applicationWillResignActive:(id)application;
-(BOOL)application:(id)application didFinishLaunchingWithOptions:(id)options;
-(void)dismissModalViewController;
-(void)presentPasscodeVerifyController;
@end
```

Figure 10 Class-Dump-Z Utility Output

It was determined that once these function names were stripped from the binary, it decreased the attacker's ability to find additional application information, making future attacks much more time consuming.

2.7.1 hReader Reverse Engineering Revealing Attack Vector Mitigations

No mitigations are required since this is a debug build; function and class names will be stripped from the release version of the binary. Without function names present, there is no indication given to an attacker as to where the authentication-related functions are located. This means the amount of work required to find and attempt to exploit this functionality increases greatly.

2.8 iOS General Security

The test setup for iOS General Security:

- Physical access to an iPad
- Jailbroken iPad 1 running iOS 5.0.1
- Cydia application installed. This app allows the user to find and install software packages on jailbroken devices
- OpenSSH installed. This allows network connections from the command line enabling files to be copied to and from the device

The iPad that was used for this test had a known system level passcode that was used to unlock the keychain data. More application information could be revealed once this passcode was entered. It has been reported that over 95% of iPhone and iPad users do not have a passcode or use the simple 4 digit passcode. This passcode can be cracked with relative ease, typically taking less than 30 minutes, and can then be used to login to a device, thus disabling major security

components of iOS. Therefore, it is recommended that an iOS application (i.e., hReader) only be installed if the iOS passcode is set to complex, or that it has at least 6 digits. Studies have shown that it will take considerably more time to brute force crack a longer passcode. This is a significant first step to securing an iOS application and is something that can help immediately.

3 Audit Version 2 (V2) Findings – iOS Forensics Discovery

hReader was audited a second time after additional security controls were coded into the application. The results of this "second pass" are described throughout this section, labeled "Audit v2."

3.1 iOS Bruteforce Passcode

Using tools offered by the iOS security research community, an iPad or iPhone can easily be rebooted, jailbroken, and a passcode cruncher installed in less than 30 minutes. Figure 11, shows a bruteforce algorithm marching through passcodes, testing each, and finally arriving at "0011" as the passcode. When entered on the iPad user interface, the passcode successfully unlocked the device. With the passcode unlocked all applications that use the iOS CoreData library and the iOS keychain are unlocked allowing for open forensic discovery.

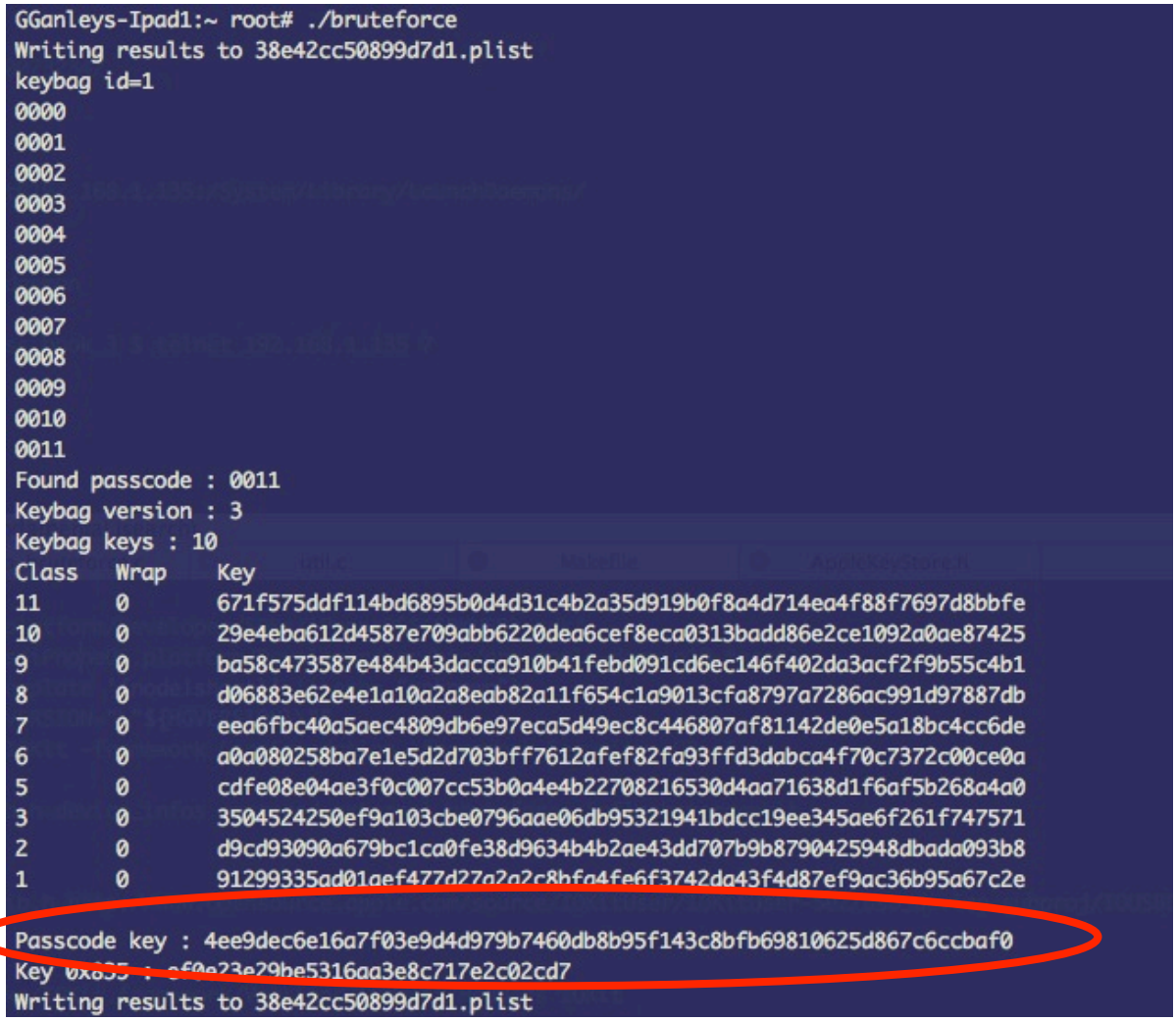


Figure 11 Bruteforce Algorithm Acquiring Passcode

This test illustrates why the developer cannot rely on the security of the iOS device itself. Assuming the attacker will inevitably access the device, the developer should use encryption and strong passwords to protect their own application and its sensitive resources.

3.2 Audit v2 Findings – iOS Keychain and hReader Passcode

In Figure 12, the iOS keychain shown lists the hReader pertinent sections. In this output, one observes that most, if not all of the sensitive data is no longer visible. This is a marked improvement over the prior security audit. However, the passcode recovery questions are still in plain text allowing for possible social engineering attacks. It is recommended that the questions be converted into a SHA-256 hash for obfuscation purposes.

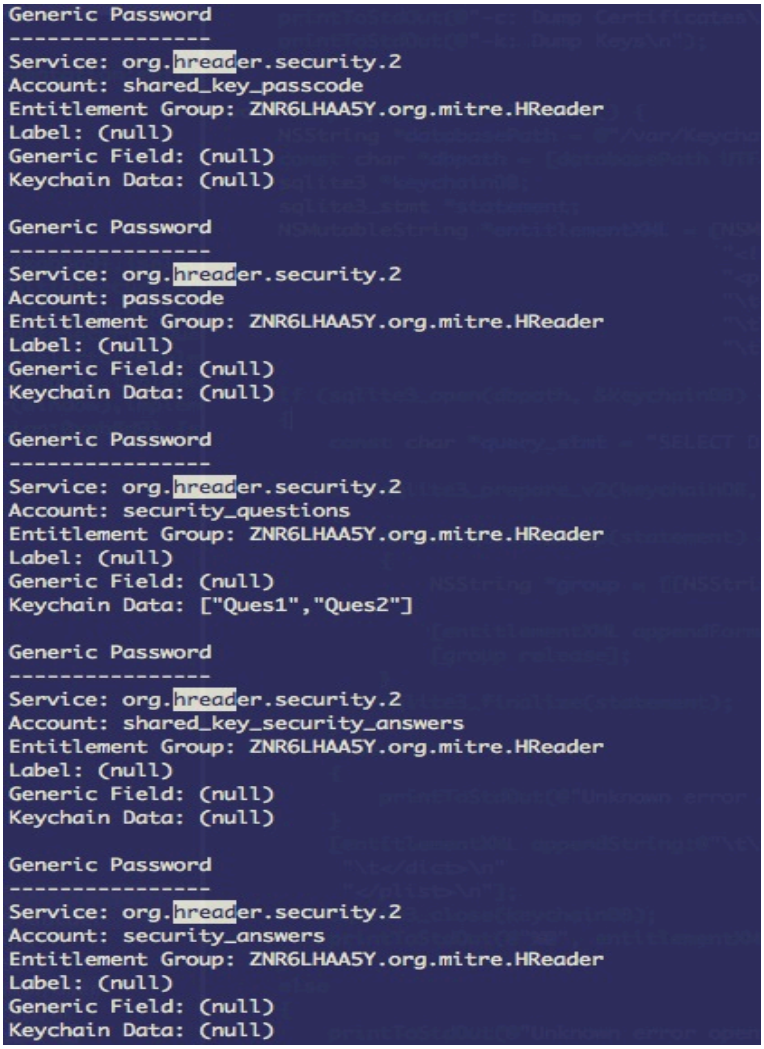


Figure 12 iOS Keychaing Text File Showing hReader Sections

Keychain Delete

Working with a jailbroken device with root access, a connection was established to the keychain SQLite DB and the hReader entries were deleted. Upon hReader restart, the user was asked to enter a new passcode and enter new questions. After entering new passcode/questions, the user was authenticated to the backend data server and was then able to see all patients and their synthetic medical data.

It is recommended that hReader not use the iOS keychain for this purpose because of this security vulnerability. Or at a minimum create a technical solution that handles the case of deleted keychain passcode/question data.

3.3 Audit v2 Findings – Home Button Screen Capture

During hReader use, the home button was pressed several times, returning the user to the iPad desktop page. Upon repeating this process again, forensic tools were used to discover hReader screen shots. A retest of this potential vulnerability, revealed proper concealment, but did find

this one case with the sub-menu open and pressing the home button revealed the family member list (Figure 13).

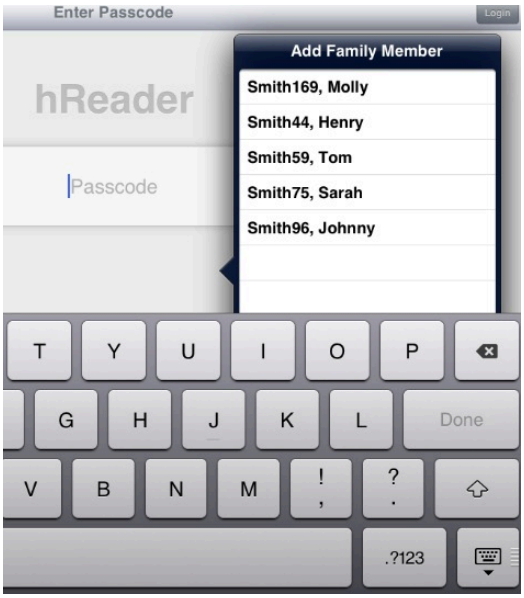


Figure 13 Family Member List Screen Capture

3.4 Audit v2 Findings – Patient Data

A SQLite DB was found amongst the hReader application files. Upon accessing the database, all 429 records were easily readable and revealed synthetic patient information.

Also, many unencrypted files with synthetic patient data in plain text were in the hReader application folder, including:

```
C32-Smith-Henry.html, johnny, molly, sarah, tom
release_notes.txt - description of development changes
README.txt - description of Testfligh config

GGanleys-Ipad1:/var/mobile/Applications/375EB619-70BA-4E85-8E5D-
FFFC89D5031A/HReader.app root# ls *json*
female-bmi-chart.json  js-synthetic.json    ms-synthetic.json
ss-synthetic.json
hs-synthetic.json      male-bmi-chart.json  ns-synthetic.json
ts-synthetic.json

timeline directory - many .xml files
0_Jamie_Russell.xml
single.xml

./hReader.momd/VersionInfo.plist
<key>hReader</key>
            <dict>
```

```
        <key>GCManagedObject</key>
        <data>
        nHgbd2Rt1Vym0DUUGwA6tvHPz8wd+dk4tH6O/Zj8ELU=
        </data>
        <key>HRMEntry</key>
        <data>
        69lHJMcZ+GNQX941DYI1GNz3/YN9wDmGyv4GkMMkx4Q=
        </data>
        <key>HRMPatient</key>
        <data>
        ivJpBhsm+EJu5KMKPRjeCNCqwqT02XyGgvYX4AELLpQ=
        </data>
</dict>
```

3.4.1 Recommendations

Pursue use of SQLCipher as a secure data store for hReader.
Remove or encrypt all patient data in plain text files.

3.5 Audit v2 Findings – Developer Identity Data

This vulnerability has been resolved. No information was found that could provide the identity of a developer.

3.6 Audit v2 Findings – hReader Authentication Bypass

This vulnerability has been resolved. Running the application from the command line, via SSH, as root no longer starts and runs the application.

3.7 Audit v2 Findings – Jailbreak Detection Avoidance

This portion of the audit was performed using an iPad v1. Upon download of the latest hReader application, hReader no longer functioned, that is it would start up then immediately close. After conferring with the developer that this was not a bug, but an intended effect, it was suspected that jailbreak detection software was in use. Consulting several iOS forensic books, one section referenced a list of files that an iOS application can check at run-time. If the files exist, then the device has been compromised. The audit team started at the top of the list looking for these files on the test device. Fortunately, the first file on the list "/Library/MobileSubstrate/MobileSubstrate.dylib" was found. As a test, the file was renamed and then hReader was re-executed. hReader functioned correctly and did not crash. Conferring with other auditors, who had an iPad v2 running an apparent different jailbreak exploit, was not affected by this jailbreak detection security control. In summary, the hReader jailbreak detection security control worked on the first generation iPad, but did not work on the second generation iPad.

3.7.1 Recommendations

Add more jailbreak file checks into the hReader application.

3.8 Audit v2 Findings – Runtime Analysis

The hReader application was examined using several different approaches such as, string analysis, class dumping, cycrypt libs, and GDB. Each is discussed below.

Strings analysis (using the "strings" utility) revealed that many text strings indicate password and passcode.

Class dumping the executable also revealed a bevy of method names and objects.
Memory dumping of hReader at runtime revealed similar output as the class and string analysis.

Using Cycrypt (an open source effort with a java like syntax that can be used to explore objective-c applications at runtime), further class and method extraction was possible, including this list of methods on the HRAppDelegate.

```
printMethods(HRAppDelegate)
{selector:@selector(createInitialPasscode::),implementation:0xabba9},
{selector:@selector(presentPasscodeVerificationController:),implementation:0x
{selector:@selector(performLaunchSteps),implementation:0xab289},
{selector:@selector(createInitialSecurityQuestions:::),implementation:0xabcf
{selector:@selector(verifyPasscodeOnLaunch::),implementation:0xac005},
{selector:@selector(resetPasscodeWithSecurityQuestions),implementation:0xac2
{selector:@selector(resetPasscodeWithSecurityQuestions:::),implementation:0xa
{selector:@selector(resetPasscode::),implementation:0xac565},
{selector:@selector(updateSecurityQuestions:::),implementation:0xacab5},
{selector:@selector(securityQuestions),implementation:0xacb3d},
{selector:@selector(numberOfSecurityQuestions),implementation:0xacb29},
{selector:@selector(verifyPasscodeOnPasscodeChange::),implementation:0xac669}
{selector:@selector(verifyPasscodeOnQuestionsChange::),implementation:0xac8cd
{selector:@selector(PINCodeLength),implementation:0xacb15},
{selector:@selector(managedObjectContextDidSave:),implementation:0xab85d},
{selector:@selector(.cxx_destruct),implementation:0xacb99},
{selector:@selector(window),implementation:0xacb51},
{selector:@selector(applicationDidEnterBackground:),implementation:0xabac5},
{selector:@selector(application:didFinishLaunchingWithOptions:),implementatio
{selector:@selector(setWindow:),implementation:0xacb6d}]
```

Attempts were made to call these functions directly; all failed or had little effect. Several of them displayed a text message indicating that it was not possible to run two instances of this class or method. From this list further investigation was conducted using the gdb debugger.

Using gdb, it was possible to set breakpoints on some methods but not all. When at a breakpoint the auditors were unable to consistently step in or over functions calls, essentially making gdb useless. After further observation, it was determined that the hReader image was stripped of symbols at build time making debugging difficult.

In summary, the run-time was secured more deeply than the prior version of hReader. Stripping symbol tables and limiting access to the objective-c class methods above was very effective. One recommendation would be to obfuscate the names of all security related methods and classes such that run-time snooping is more difficult. Lastly, more expensive tools (i.e., Ida Pro decompiler) which convert an iOS executable into readable C-code would be next in terms of more robust iOS forensic discovery, however these tools cost several thousands of dollars making the level of forensics well above consumer grade protection.

3.9 Audit v2 – hReader Network Analysis

Network traffic was observed during different hReader application use scenarios. The traffic was captured and analyzed for Information Assurance (IA) vulnerabilities.

In all instances of packet capture the following observations are true:

1. Intended to simulate someone passively monitoring (eavesdropping) the traffic between hReader and its authentication and data servers.
2. Packet capture was obtained using the tcpdump utility on the jailbroken iPad. Packets were captured in .pcap form for later viewing in Wireshark.
3. Packet capture included the entire packet (including payload).
4. iPad was connected to OuterNET wireless for internet connectivity.
5. In Wireshark, the outer layer of AES encryption provided by the OuterNET's use of WPA2 was removed, providing the "bare" packets.

Usage Scenarios

1. Authenticating with passcode:
    a. The bottom line: it was observed that all user authentication data is encrypted. Authentication is performed as necessary and does not identify any vulnerability.
    b. hReader is using RSA key exchange, as seen in Figure 14, so this connection can only be decrypted if the attacker has the RSA keys (requiring access to either the client or the server).

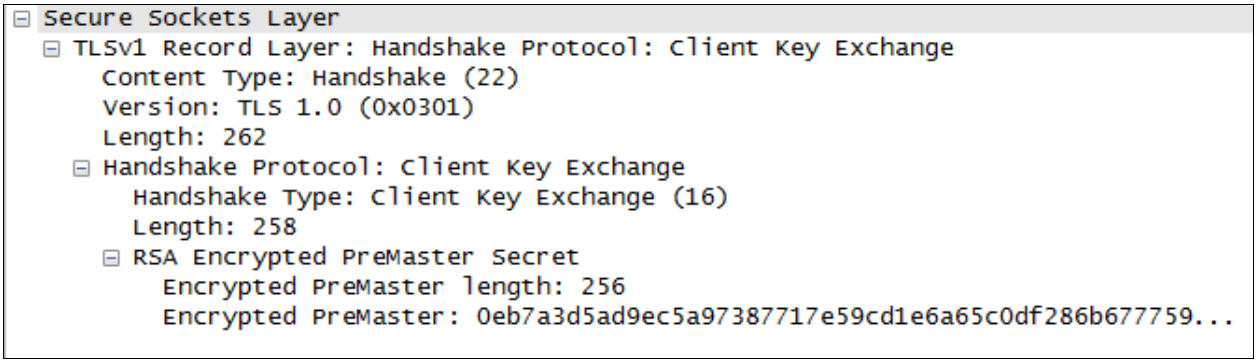


Figure 14 hReader/TestFlight Key Exchange

c. Observation: it appears that authentication is made to TestFlight (110.173.143.197). Analysis of that server is beyond the scope of this tasking; suffice it to say that the integrity of TestFlight is an important aspect to explore further.

2. Packets captured while using the "Add Family Member" option:

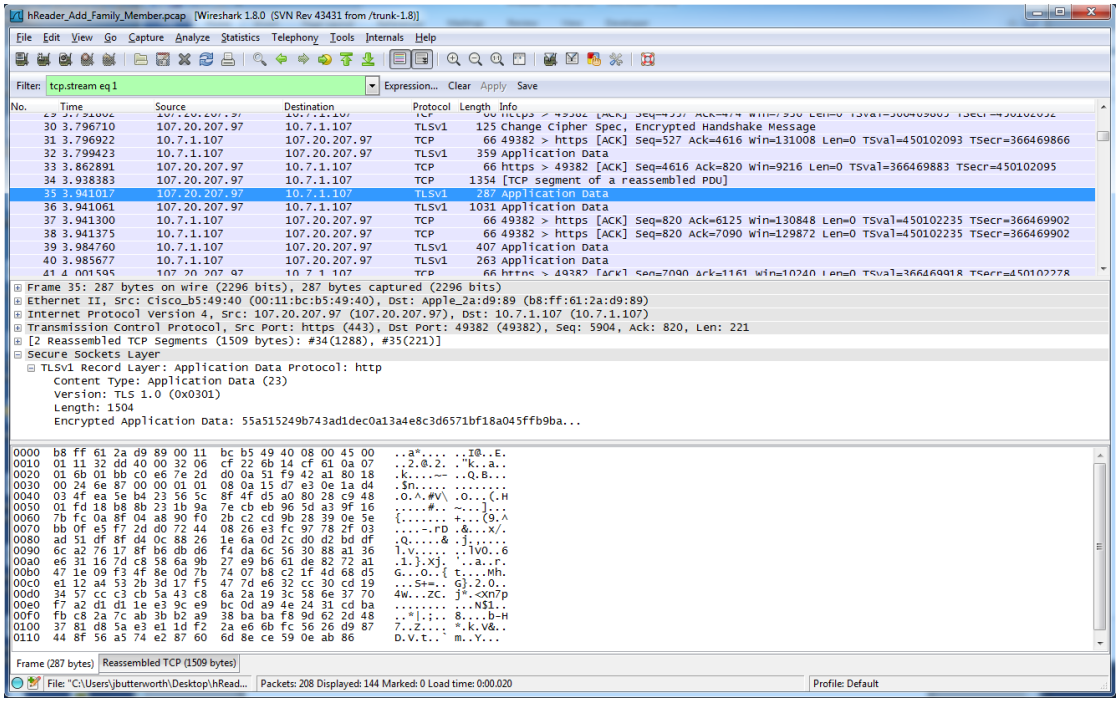


Figure 15 "Little" Johnny Smith in Ciphertext

a. The bottom line: Same as item 1, all data retrieved from the server (using Amazon web services) is encrypted. No sensitive clear-text data was transmitted.

Vulnerability Analysis to Man-In-The-Middle Attacks

An analysis of hReader's vulnerability to both active and passive MITM attacks on its SSL traffic was conducted. The findings are discussed below.

Passive MITM

The analysis determined that hReader is not vulnerable to passive MITM attacks. A passive MITM attack is one in which an attacker simply eavesdrops on TCP traffic. Regarding SSL, if an application is using a "weak" key exchange, then the entire SSL conversation can be decrypted. hReader is using a strong key exchange, as seen in Figure 15 above, so this is not a threat.

Passively decrypting patient data would require the private key from either the client or the server. Without physical access to the iPad this is essentially impossible. Even if the attacker gained physical access to the device, he/she would not be able to authenticate the logon credentials. It is recommended that the administrator of the TestFlight server ensure that the private key is well-protected.

Active MITM

Analysis has determined that hReader could be vulnerable to an active MITM attack if it is not verifying the server's certificate(s). An active MITM attack inserts an attacker-controlled device in-between the client and the server to act as a proxy, thus fooling both the client and server. This is not a theoretical attack; it is performed both legitimately and illegitimately in the real world.

To prevent this (or make it exceedingly difficult), hReader should verify the server certificate. This forces an attacker to gain physical access to the device in order to modify the trusted certificates.

As noted above, physical access to the device does not guarantee an attacker access to hReader and its data due to the complex passcode requirement. Verifying the server certificate should be a feature provided in the OpenSSL API.

In summary, the implementation of SSL uses a strong key exchange and provides no sensitive data to be passed in the clear. The only networking-related vulnerability comes from an active man-in-the-middle attack being performed. The success of this depends on the attacker's ability to acquire physical access to the device and bypass authentication. It is recommended that the client verify the server certificate to decrease the likelihood of this kind of attack succeeding.

4 Audit Version 3 (V3) Findings

hReader was audited a third time to finalize the DISA/DoD Application STIG (Security Technical Implementation Guide) review as well as test the SQLCipher implementation. The results of this "third pass" audit are described throughout the document with sections starting with "Audit v3."

4.1 Audit v3 Findings – hReader SQLite Cipher Analysis

hReader uses SQLCipher, an open-source extension to SQLite that provides 256-bit AES encryption for database files. The implementation was tested on the hReader patient database file.

```
John-Butterworths-iPad:/User/Applications/A24408BC-19E2-4C88-9C35-0F9646E8F5FE/L
ibrary/Application Support root# sqlite3 database.sqlite3.encrypted
SQLite version 3.7.7
Enter ".help" for instructions
sqlite> .tables
Error: file is encrypted or is not a database
sqlite>
```

Figure 16 SQLite encrypted with SQLCipher – unable to read database

To ensure that the sqlite3 utility had not failed for reasons other than encryption, the database file was viewed in a hex-editor as a "sanity-check". It was confirmed that all data was ciphertext. This test was repeated after the user was logged onto the application and at no time were we able to maliciously decrypt the database.

4.1.1 hReader SQLite Cipher Analysis Attack Vectors

During the V3 audit, no SQLite Cipher attack vectors were found. The database on disk remains encrypted at all times. The database is unlocked only after a successful login using the passcode.

4.2 Audit v3 Findings – hReader Source Code Analysis

This was not a static source code analysis, but rather an attempt to use the source to better understand the application so that more focused attacks could be performed.

It was determined that the user/attacker has very little control over the data inputted to the application. In these cases, third-party libraries are being used which implies that any penetration testing performed on this input is actually a penetration test on the third party library, rather than hReader. That being said, the following attack patterns were implemented to attempt to cause unexpected/undesirable behavior: Passcode of extraordinary length was inputted (as well as security challenge question response length); about 100 instances of family members were added to the hReader database; and the SQLite database data was substituted with random bytes. In the first two cases, the input resulted in no adverse behavior. In the case where the SQLite database contents were substituted with random bytes, the hReader application quit and returned the user to the iOS desktop. This could be desirable behavior, depending upon the developer's wishes. But as stated previously, this ended up being more of a test of SQLite's file input than anything regarding hReader.

4.3 Audit v3 Findings – hReader STIG Compliance (Initial Review)

hReader's compliance with the DISA/DoD Application STIG (Security Technical Implementation Guide) was analyzed with the following statistical results as shown in Table 1. The findings are found in Appendix A.

Table 1 Initial STIG Review Statistical Results

Result of Check	Amount	% Applicable Checks (166)
Yes (Compliant)	27	16%
No (Not Compliant)	10	6%
Not Applicable	89	-
Unknown	129	78%
Total Applicable	166	-

There are four states that hReader can fall into for each STIG criteria: "Yes" means that hReader does fulfill the requirement while "No" means it does not. The "Unknown" category indicates that more research is needed to understand whether the requirement should be necessary to a mobile application. The "Not Applicable" result of the finding indicates that the requirement does not apply to this particular mobile application (or any mobile application, as the case may be).

5 Audit Version 4 (V4) Findings

hReader has gone through three previous audits that are contained within this document. The hReader application was audited a fourth time to review the changes that were made to hReader based on the findings of the DISA/DoD Application STIG (Security Technical Implementation Guide) review.

5.1 hReader DISA/DoD Applications STIG Compliance (Final Review)

hReader's compliance with the DISA/DoD Application STIG was analyzed again after the development team had time to address the last found and noted vulnerabilities.

Out of the 255 total criteria defined by the STIG, 80 were found to be applicable to hReader as a mobile application prototype. The final assessment resulted in hReader passing 72 of these criterion checks and failing 8. The remaining 175 were not applicable to hReader.

Table 2 Final STIG Review Statistical Results

Result of Check	Amount	% Applicable Checks (80)
Yes (Compliant)	72	90%
No (Not Compliant)	8	10%
Not Applicable	175	-
Unknown	0	0%
Total Applicable	80	-

Overall, hReader's security measurement increased to 90% compliant with 72 out of 80 criteria met. This represents a 5X or 500% improvement in security. As a further measure, this iterative security audit and development effort cost approximately 3 staff months.

6 Bibliography

The MITRE Corporation. "CWE - Common Weakness Enumeration." *CWE - Common Weakness Enumeration.* June 4, 2012. cwe.mitre.org (accessed June 4, 2012).

Appendix A STIG Details

Criteria	Description	Assessment	Note
(APP3590.1: CAT I)	The Designer will ensure the application does not have buffer overflows.	**YES**	The question turns to Apple - and has Apple ensured their libraries are "fuzz" tested. hReader sanitizes and bounds checks all user input.
(APP3590.2: CAT II)	The Designer will ensure the application does not use functions known to be vulnerable to buffer overflows.	**YES**	The question turns to Apple - and has Apple ensured their libraries are "fuzz" tested. hReader sanitizes and bounds checks all user input.

(APP3130: CAT I)	The Designer will ensure the application follows the secure failure design principle.	**YES**	3.1.10 Secure Failure
			In order to minimize error handling and ensure proper return code, exception handling is implemented throughout the application. The secure failure principle should be implemented.
			This principle helps ensure predictable and secure application behavior in case of application failures. Applications often perform checks on the validity of data, user permissions, and resource existence before performing a function. Secure failure is defined if a check fails for any reason, the behavior of the application remains in a secure state. The following pseudo-code provides a simple example of a failure which is not secure:
			If
			CheckAccessDenied()
			Display Error Message ()
			DenyAccess()
			Else
			Perform Privileged Action ()
			Endif
			In the example above, if the Check Access Denied function encounters an error while running and does not return “TRUE” (for example, the function returns an “Out of memory” error), the privileged action is performed.

			While this example is very simple, complex behaviors are often modeled in an application, and it is important the most secure code branch is executed in the event of an application error. The application code should provide exception handling; thereby, leaving the application in a secure state. The principle of secure failure design is intended to account for all possible exceptions leaving the application in a vulnerable state. Coding constructs vary from programming language to programming language. The example above only illustrates the basic concept of secure failure design.
(APP3600: CAT II)	The Designer will ensure the application has no canonical representation vulnerabilities.	**YES**	hReader does not rely on the presence of files to alter the action of security processes.

(APP3320.1: CAT II)	The Designer will ensure the application has the capability to require account passwords having a minimum of 15 alphanumeric characters in length.	**YES**	Alternatively, the General Mobile Device (Non-Enterprise Activated) STIG states: The security requirements for a mobile device are found in the appropriate device STIG. For the non-enterprise activated mobile device, the only technical security requirements that must be met are related to the device unlock password/passcode: • Device unlock password/passcode available. • Device unlock password can be set to at least 8 characters. • The device can be set to lock the screen after a set period of inactivity and the screen will not unlock until the password/passcode is entered. The current build of hReader satisfies this requirement, and hReader is being deployed on a mobile device, so it's reasonable to abide by this requirement rather than more stringent one provided by the Application Security and Development STIG
(APP3060: CAT II)	The Designer will ensure the application does not store configuration and control files in the same directory as user data.	**YES**	latest hReader v3 audit mitigates this-XML files need to be removed - retest

(APP3220.1: CAT II)	The Designer will ensure sensitive data held in memory is cryptographically protected when not in use if required by the information owner.	**YES**	hReaderapp will ensure autocomplete is disabled in all text fields - retest
(APP3320.2: CAT II)	The Designer will ensure the application has the capability to require account passwords contain a mix of upper case letters, lower case letters, numbers, and special characters.	**YES**	see (APP3320.1: CAT II)
(APP3320.8: CAT II)	The IAO will configure the application to ensure account passwords conform to DoD password policy.	**YES**	hReader complies with this STIG - DISA general mobile device, non-enterprise activated STIG, says passcode, at least 8 chars, and inactive screen
(APP3100: CAT II)	The Designer will ensure the application removes temporary storage of files and cookies when the application is terminated.	**YES**	GG: keychain files need to be removed on uninstall webview images need to be removed 9/19/12 hReader does not use and temp files, so somehwhat not applicable
(APP3590.3: CAT II)	The Designer will ensure the application does not use signed values for memory allocation where permitted by the programming language.	**YES**	This might not really be applicable to an iPad app.
(APP3630.1: CAT II)	The Designer will ensure the application is not vulnerable to race conditions.	**YES**	
(APP3630.2: CAT III)	The Designer will ensure the application does not use global variables when local variables could be used.	**YES**	

(APP3630.3: CAT II)	The Designer will ensure a multi-threaded application uses thread safe functions when threads are accessing the same object or data.	**YES**	
(APP3630.4: CAT II)	The Designer will ensure global resources are locked before being accessed by the application.	**YES**	
(APP3050: CAT II)	The Designer will ensure the application does not contain source code that is never invoked during operation, except for software components and libraries from approved third-party products, which may include un-invoked code.	**YES**	
(APP3430: CAT I)	The Designer will ensure the application removes authentication credentials on client computers after a session terminates.	**YES**	
(APP5080: CAT II)	The Test Manager will ensure a code review is performed before the application is released.	**YES**	
(APP3530: CAT II)	The Designer will ensure the web application assigns the character set on all web pages.	**YES**	default for iOS is UTF8, all good
(APP3110: CAT II)	The Designer will ensure the application installs with unnecessary functionality disabled by default.	**YES**	
(APP3150.1: CAT II)	The Designer will ensure the application uses FIPS 140-2 validated cryptographic modules if the application implements encryption, key exchange, digital signature, and hash functionality.	**YES**	following apple and also security best practices for iOS apps
(APP3150.2: CAT II)	The Designer will ensure the application uses a FIPS 140-2 validated random number generator to support cryptographic functions.	**YES**	following apple and also security best practices for iOS apps
(APP6240: CAT III)	The IAO will ensure all user accounts are disabled which are authorized to have access to the application but have not authenticated within the past 35 days	**YES**	RHEx expires OATH tokens each month, which requires a new login
(APP2130.2: CAT II)	The Program Manager will ensure a mechanism is in place to notify users of security flaws and to provide users with the availability of patches.	**YES**	through github
(APP2130.3: CAT II)	The Program Manager will ensure a comprehensive vulnerability management process, including systematic identification and mitigation of software vulnerabilities is in place.	**YES**	through github

(APP2135: CAT I)	The Program Manager will ensure all products are supported by the vendor or the development team.	**YES**	through gituhub
(APP2060.3: CAT II)	The Designer will follow the established coding standards established for the project.	**YES**	following apple and also security best practices for iOS apps
(APP2060.4: CAT II)	The Designer will not use unsafe functions documented in the project coding standards.	**YES**	following apple and also security best practices for iOS apps
(APP3070: CAT II)	The Designer will ensure the user interface services are physically or logically separated from data storage and management services.	**YES**	
(APP3080: CAT II)	The Designer will ensure the application does not contain invalid URL or path references.	**YES**	
(APP3140: CAT II)	The Designer will ensure application initialization, shutdown, and aborts are designed to keep the application in a secure state.	**YES**	
(APP3180: CAT II)	The Designer will ensure private keys are accessible only to administrative users.	**YES**	no priv keys in the app, all entered by the user
(APP3350: CAT I)	The Designer will ensure the application does not contain embedded authentication data.	**YES**	
(APP3405: CAT I)	The Designer will ensure the application supports detection and/or prevention of communication session hijacking.	**YES**	
(APP3410.1: CAT II)	The Designer will ensure the application provides a capability to limit the number of logon sessions per user.	**YES**	

(APP3410.2: CAT II)	The Designer will ensure the application provides a capability to limit the total number of logon sessions for the application.	**YES**	
(APP3415: CAT II)	The Designer will ensure the application provides a capability to automatically terminate a session and logout after a system defined session idle time limit is exceeded.	**YES**	
(APP3440: CAT II)	The Designer will ensure the application is capable of displaying a customizable click-through banner at logon which prevents further activity on the information system unless and until the user executes a positive action to manifest agreement by clicking on a box indicating "OK".	**YES**	
(APP3460: CAT I)	The Designer will ensure the application does not rely solely on a resource name to control access to a resource.	**YES**	
(APP3510: CAT I)	The Designer will ensure the application validates all input.	**YES**	
(APP3540.1: CAT I)	The Designer will ensure the application is not vulnerable to SQL injection.	**YES**	
(APP3540.2: CAT II)	The Designer will ensure the application uses prepared or parameterized statements.	**YES**	
(APP3540.3: CAT II)	The Designer will ensure the application does not use concatenation or replacement to build SQL queries.	**YES**	
(APP3540.4: CAT II)	The Designer will ensure the application does not directly access the tables in a database.	**YES**	
(APP5090: CAT II)	The Test Manager will ensure flaws found during a code review are tracked in a defect tracking system.	**YES**	

(APP2130.1: CAT II)	The Program Manager will ensure users are provided with a means of obtaining updates for the application.	**YES**	Actually, not going in the app store - will be open sourced. Github will be updated as newer security controls are added
(APP3020.2: CAT II)	The Designer will identify potential mitigations to identified threats.	**YES**	
(APP3020.3: CAT II)	The Designer will ensure appropriate mitigations are implemented to threats based on their risk analysis.	**YES**	
(APP3120: CAT II)	The Designer will ensure the application is not subject to error handling vulnerabilities.	**YES**	
(APP3170: CAT II)	The Designer will ensure the application uses encryption to implement key exchange and authenticate end-points prior to establishing a communication channel for key exchange.	**YES**	
(APP3190: CAT II)	The Designer will ensure the application does not connect to a database using administrative credentials or other privileged database accounts.	**YES**	
(APP3250.1: CAT I)	The Designer will ensure unclassified, sensitive data transmitted through a commercial or wireless network is protected using NIST-certified cryptography.	**YES**	
(APP3280.1: CAT II)	The Designer will ensure applications requiring user authentication are PK-enabled.	**YES**	

(APP3310: CAT I)	The Designer will ensure the application does not display account passwords as clear text.	**YES**	
(APP3330: CAT I)	The Designer will ensure the application transmits account passwords in an approved encrypted format.	**YES**	
(APP3340: CAT I)	The Designer will ensure the application stores account passwords in an approved encrypted format.	**YES**	
(APP3360: CAT II)	The Designer will ensure the application protects access to authentication data by restricting access to authorized users and services.	**YES**	
(APP3420: CAT II)	The Designer will ensure the application provides a capability to terminate a session and logout.	**YES**	
(APP3450.1: CAT II)	The Designer will ensure application resources are protected with permission sets which allow only an application administrator to modify application resource configuration files.	**YES**	
(APP3480.1: CAT I)	The Designer will ensure access control mechanisms exist to ensure data is accessed and changed only by authorized personnel.	**YES**	
(APP3570: CAT I)	The Designer will ensure the application does not allow command injection.	**YES**	
(APP3580: CAT I)	The Designer will ensure the application does not have XSS vulnerabilities.	**YES**	
(APP3585: CAT II)	The Designer will ensure the application does not have CSRF vulnerabilities.	**YES**	
(APP3610: CAT I)	The Designer will ensure the application does not use hidden fields to control user access privileges or as a part of a security mechanism.	**YES**	
(APP3620: CAT II)	The Designer will ensure the application does not disclose unnecessary information to users.	**YES**	
(APP5010: CAT III)	The Test Manager will ensure at least one tester is designated to test for security flaws in addition to functional testing.	**YES**	
(APP5030: CAT II)	The Test Manager will ensure the application does not modify data files outside the scope of the application.	**YES**	

(APP5110: CAT II)	The Test Manager will ensure security flaws are fixed or addressed in the project plan.	**YES**	
(APP3020.4: CAT II)	The IAO will ensure identified mitigations to identified threats are implemented.	**YES**	
(APP6050: CAT II)	The IAO will ensure the system and installed applications have current patches, security updates, and configuration settings.	**YES**	
(APP6060: CAT I)	The IAO will ensure the application is decommissioned when maintenance or support is no longer available.	**YES**	
(APP6070: CAT III)	The IAO will ensure provisions are in place to notify users when an application is decommissioned.	**YES**	
(APP3320.3: CAT II)	The Designer will ensure the application has the capability to require account passwords be changed every 60 days or more frequently.	**NO**	inappropriate for a mobile device

(APP3210.1: CAT II)	The Designer will ensure NIST-certified cryptography is used to protect stored sensitive information if required by the information owner.	NO	RHEx expires OATH tokens each month, which requires a new login - security team to look into this further I have not found any useful information on whether iOS Crypto or SQLCipher are NIST-certified, although if SQLCipher uses OpenSSL, then you could certainly make the case that this requirement is satisfied says OpenSSL is very NIST-certified. 9/19/12 IOS Crypto is in step 2 / 5 for NIST certification. OpenSSL 1.0.1C, likely not NIST certified. Could switch keychain content crypto over to FIPS / NIST compliant lib. SQLCipher would need to be recompiled/ported NIST/compliant lib. Open SSL 2.0 and 2.0.1 are currently NIST certified. Open SSL 1.0.0-20.EL6 is the FIPS validated
(APP3020.1: CAT II)	The Designer will ensure threat models are documented and reviewed for each application release and updated as required by design and functionality changes or new threats are discovered.	NO	
(APP3230.1: CAT II)	The Designer will ensure the application properly clears or overwrites all memory blocks used to process sensitive data if required by the information owner.	NO	not now, perhaps down the road

(APP3390: CAT I)	The Designer will ensure users' accounts are locked after three consecutive unsuccessful logon attempts within one hour.	**NO**	alert box today; and after 3 logins, Q/A screen is shown; Somewhat do not want to follow this to the letter - hence yellow no
(APP2060.1: CAT II)	The Program Manager will ensure the development team follows a set of coding standards.	**NO**	best practices are being followed - an app style guide is needed and to be built
(APP2060.2: CAT II)	The Program Manager will ensure the development team creates a list of unsafe functions to avoid and document this list in the coding standards.	**NO**	see 2060.1
(APP3010: CAT II)	The Designer will create and update the Design Document for each release of the application identifying the following: − all external interfaces (from the threat model) − the nature of information being exchanged − categories of sensitive information processed or stored and their specific protection plans − the protection mechanisms associated with each interface − user roles required for access control − access privileges assigned to each role − unique application security requirements − categories of sensitive information processed or stored and specific protection plans (e.g., Privacy Act, Health Insurance Portability and Accountability Act (HIPAA), etc.) − restoration priority of subsystems, processes, or information	**NO**	Somewhat of a nice to have for a prototype and an agile process. Will be documenting in the code and an applet build document. May not be following the complete letter of the law. May be even more complicated when open sourced. Open source wiki page could apply toward this
(APP2010.1: CAT II)	The Program Manager will ensure an SSP is established describing the technical, administrative, and procedural IA program and policies governing the DoD information	**N/A**	

	system, and identifying all IA personnel and specific IA requirements and objectives.		
(APP2020.3: CAT II)	The Program Manager will ensure development systems, build systems, and test systems have a standardized environment and are documented in the Application Configuration Guide.	**N/A**	
(APP2050: CAT II)	The Program Manager will ensure the system has been assigned specific MAC and Confidentiality levels.	**N/A**	
(APP2070.1: CAT III)	The Program Manager will ensure any IA or IA-enabled products used by the application are NIAP approved or in the NIAP approval process.	**N/A**	
(APP2080.1: CAT II)	The Program Manager will ensure COTS IA and IA-enabled products, which are used to protect publicly released information, comply with National Security Agency (NSA)-endorsed Protection Profiles.	**N/A**	
(APP2080.2: CAT II)	The Program Manager will ensure COTS IA and IA-enabled products which are used to protect sensitive information when the information transits non DoD-owned networks, or the system handling the information is accessible by individuals who are not authorized to access the information on the system, comply with NSA-NIAP approved Protection Profiles.	**N/A**	
(APP2080.3: CAT II)	The Program Manager will ensure COTS IA and IA-enabled products which are used to protect classified information when the information transits networks, which are at a lower classification level than the information being transported, comply with NSANIAP approved Protection Profiles.	**N/A**	
(APP2090.1: CAT II)	The Program Manager will obtain DAA acceptance of risk for all public domain, shareware, freeware, and other software products/libraries with both (1) no source code to review, repair, and extend, and (2) limited or no warranty, but are required for mission accomplishment.	**N/A**	

(APP2160.1: CAT II)	The Program Manager will ensure development systems, build systems, test systems, and all components comply with all appropriate DoD STIGS, NSA guides, and all applicable DoD policies.	**N/A**	
(APP2070.2: CAT III)	The Designer will ensure any IA or IA-enabled products used by the application are NIAP-approved or in the NIAP approval process.	**N/A**	
(APP2090.2: CAT II)	The Designer will document for DAA approval all , public domain, shareware, freeware, and other software products/libraries with both (1) no source code to review, repair, and extend, and (2) limited or no warranty, but are required for mission accomplishment.	**N/A**	
(APP2100.2: CAT II)	The Designer will ensure the application design complies with the DoD Ports and Protocols guidance.	**N/A**	
(APP2110.2: CAT II)	The Designer will ensure the application is registered with the DoD Ports and Protocols database.	**N/A**	
(APP3210.2: CAT II)	The Designer will ensure NIST-certified cryptography is used to store classified non-Sources and Methods Intelligence (SAMI) information if required by the information owner.	**N/A**	
(APP3210.3: CAT II)	The Designer will ensure a classified enclave containing SAMI data is encrypted with NSA-approved cryptography.	**N/A**	
(APP3250.2: CAT I)	The Designer will ensure classified data, transmitted through a network that is cleared to a lower level than the data being transmitted, is separately protected using NSA approved cryptography.	**N/A**	no classified data to be used
(APP3250.3: CAT II)	The Designer will ensure information in transit through a network at the same classification level, but which must be separated for need-to-know reasons, is protected minimally with NIST-certified cryptography.	**N/A**	no classified data to be used
(APP3250.4: CAT II)	The Designer will ensure SAMI information in transit through a network at the same classification level is protected with NSA-approved cryptography.	**N/A**	

(APP3260: CAT II)	The Designer will ensure the application uses mechanisms assuring the integrity of all transmitted information (including labels and security parameters).	**N/A**	no classified data to be used
(APP3280.2: CAT II)	The Designer will ensure applications requiring user authentication are designed and implemented to support hardware tokens (e.g., CAC for NIPRNet).	**N/A**	
(APP3290.1: CAT II)	The Designer will ensure PK-enabled applications are designed and implemented to use approved credentials authorized under the DoD PKI program.	**N/A**	
(APP3305: CAT I)	The Designer will ensure the application using PKI validates certificates for expiration, confirms origin is from a DoD-authorized CA, and verify certificate has not been revoked by CRL or OCSP, and CRL cache (if used) is updated at least daily.	**N/A**	
(APP3370: CAT II)	The Designer will ensure the application installs with unnecessary accounts disabled or deleted by default.	**N/A**	
(APP3400: CAT II)	The Designer will ensure locked users' accounts can only be unlocked by the application administrator.	**N/A**	
(APP3470.1: CAT II)	The Designer will ensure the application is organized by functionality and roles to support the assignment of specific roles to specific application functions.	**N/A**	
(APP3480.2: CAT II)	The Designer will ensure the access procedures enforce the principles of separation of duties and "least privilege".	**N/A**	
(APP3500: CAT II)	The Designer will ensure the application executes with no more privileges than necessary for proper operation.	**N/A**	
(APP3640: CAT II)	The Designer will ensure the application supports the creation of transaction logs for access and changes to the data.	**N/A**	building a prototype
(APP3650: CAT III)	The Designer will ensure the application has a capability to notify an administrator when audit logs are nearing capacity as specified in the system documentation.	**N/A**	building a prototype

(APP3680.1: CAT II)	The Designer will ensure the application design includes audits on all access to need-to-know information.	**N/A**	
(APP3680.2: CAT II)	The Designer will ensure the application logs all failed access attempts to need-to-know information.	**N/A**	
(APP3680.3: CAT II)	The Designer will ensure the application's publicly releasable data audit records include: - Userid - Successful and unsuccessful attempts to access security files - Date and time of the event - Type of event	**N/A**	
(APP3680.4: CAT II)	The Designer will ensure the application's sensitive data audit records include: - Userid - Successful and unsuccessful attempts to access security files - Date and time of the event - Type of event - Success or failure of event - Successful and unsuccessful logons - Denial of access resulting from excessive number of logon attempts - Blocking or blacklisting a userid, terminal or access port and the reason for the action - Activities that might modify, bypass, or negate safeguards controlled by the system	**N/A**	

(APP3680.5: CAT II)	The Designer will ensure the application's classified data audit records include: - Userid - Successful and unsuccessful attempts to access security file - Date and time of the event - Type of event - Success or failure of event - Successful and unsuccessful logons - Denial of access resulting from excessive number of logon attempts - Blocking or blacklisting a userid, terminal or access port, and the reason for the action - Activities that might modify, bypass, or negate safeguards controlled by the system - Data required to audit the possible use of covert channel mechanisms - Privileged activities and other system-level access - Starting and ending time for access to the system - Security relevant actions associated with periods of activity where security labels or categories of information are processed or changed	**N/A**	
(APP3680.6: CAT III)	The Designer will ensure the application creates an audit trail for addition, deletion, or change of the confidentiality or integrity labels as designated by the information owner.	**N/A**	
(APP3690.1: CAT II)	The Designer will ensure the audit trail is readable only by the application and auditors.	**N/A**	
(APP3690.2: CAT II)	The Designer will ensure the audit trail is protected against modification or deletion except by the application and auditors.	**N/A**	
(APP3700.1: CAT II)	The Designer will ensure unsigned Category 1A mobile code is not used in the application.	**N/A**	
(APP3700.2: CAT II)	The Designer will ensure Category 1A mobile code used in an application is signed with a DoD-approved code-signing certificate.	**N/A**	
(APP3700.3: CAT II)	The Designer will ensure signed Category 1A mobile code used in an application is obtained from a trusted source and is designated as trusted.	**N/A**	
(APP3710.1: CAT II)	The Designer will ensure signed Category 1A mobile code signature is validated before executing.	**N/A**	

(APP3700.4: CAT II)	The Designer will ensure Category 1X mobile code is not used in applications.	**N/A**	
(APP3720: CAT II)	The Designer will ensure unsigned Category 2 mobile code executing in a constrained environment has no access to local system and network resources.	**N/A**	
(APP3700.5: CAT II)	The Designer will ensure signed Category 2 mobile code used in an application is signed with a DoD-approved code-signing certificate.	**N/A**	
(APP3700.6: CAT II)	The Designer will ensure Category 2 mobile code not executing in a constrained execution environment is obtained from a trusted source over an assured channel using at least one of the following measures: 1. The mobile code was digitally signed with a code-signing certificate that was designated as trusted by the recipient' s component. 2. The mobile code was downloaded over an SSL connection from a trusted SSL web server using a DoD or trusted commercial SSL server certificate. 3. The mobile code was downloaded over a TLS connection from a trusted TLS web server using a DoD or trusted commercial TLS server certificate. 4. The mobile code was downloaded from a trusted web server over an encrypted IPSec connection that establishes mutual authentication using a DoD or trusted commercial certificate.	**N/A**	
(APP3710.2: CAT II)	The Designer will ensure the signed Category 2 mobile code signature is validated before executing.	**N/A**	
(APP3730: CAT II)	The Designer will ensure uncategorized or emerging mobile code is not used in applications.	**N/A**	
(APP3740: CAT II)	The Designer will ensure the application only embeds mobile code in e-mail that does not execute automatically when the user opens the e-mail body or attachment.	**N/A**	
(APP3750: CAT II)	The Designer will ensure development of new mobile code includes measures to mitigate the risks identified.	**N/A**	
(APP6010: CAT II)	The IAO will ensure if an application is designated critical, the application is not hosted on a general purpose	**N/A**	

	machine.		
(APP6020: CAT II)	The IAO shall ensure if a DoD STIG or NSA guide is not available, a third-party product will be configured by the following in descending order as available: (1) commercially accepted practices, (2) independent testing results, or (3) vendor literature.	**N/A**	
(APP2100.3: CAT II)	The IAO will ensure the application is configured to comply with the DoD Ports and Protocols guidance.	**N/A**	
(APP2110.3: CAT II)	The IAO will ensure the application and all associated PPS are registered with the DoD PPS database.	**N/A**	
(APP2150.2: CAT II)	The IAO will ensure procedures are implemented to assure physical handling and storage of information is in accordance with the data's sensitivity.	**N/A**	
(APP6030: CAT II)	The IAO will ensure unnecessary services are disabled or removed.	**N/A**	
(APP6040: CAT II)	The IAO will ensure at least one application administrator has registered to receive update notifications or security alerts when automated alerts are available.	**N/A**	
(APP6090: CAT III)	The IAO will ensure the system alerts an administrator when low resource conditions are encountered.	**N/A**	
(APP3450.2: CAT II)	The IAO will ensure application resources are protected with permission sets only allowing application administrator to modify these configurations and files.	**N/A**	
(APP3450.3: CAT II)	The IAO will ensure access to format strings used by the application are restricted to authorized users.	**N/A**	
(APP6100: CAT II)	The IAO will ensure production database exports have database administration credentials and sensitive data removed before releasing the export.	**N/A**	
(APP3290.2: CAT I)	The IAO will ensure the PK-enabled applications are configured to honor only approved DoD PKI certificates.	**N/A**	
(APP6110: CAT III)	The IAO will review audit trails periodically based on system documentation recommendations or	**N/A**	

	immediately upon system security events.		
(APP6120: CAT II)	The IAO will report all suspected violations of IA policies in accordance with DoD information system IA procedures.	**N/A**	
(APP6130: CAT III)	The IAO will ensure, for classified systems, application audit trails are continuously and automatically monitored, and alerts are provided immediately when unusual or inappropriate activity is detected.	**N/A**	
(APP6140: CAT II)	The IAO will ensure application audit trails are retained for at least 1 year for applications without SAMI data, and 5 years for applications including SAMI data.	**N/A**	
(APP3690.3: CAT II)	The IAO will ensure the audit trail is readable only by application administrators and auditors.	**N/A**	
(APP3690.4: CAT II)	The IAO will ensure the audit trail is protected against modification or deletion except by application administrators and auditors.	**N/A**	
(APP6160.1: CAT II)	The IAO will ensure recovery procedures and technical system features exist so recovery is performed in a secure and verifiable manner.	**N/A**	
(APP6160.2: CAT II)	The IAO will document circumstances inhibiting a trusted recovery.	**N/A**	
(APP6220: CAT I)	The IAO will ensure passwords generated for users are not predictable and comply with the organizations password policy.	**N/A**	
(APP6230: CAT II)	The IAO will ensure the applications users do not use shared accounts.	**N/A**	
(APP6250: CAT II)	The IAO will ensure unnecessary built-in application accounts are disabled.	**N/A**	
(APP6260: CAT I)	The IAO will ensure default passwords are changed.	**N/A**	
(APP3470.2: CAT II)	The IAO will ensure access to privileged accounts is limited to privileged users.	**N/A**	
(APP3470.3: CAT II)	The IAO will ensure non-privileged accounts are limited to non-privileged users.	**N/A**	

(APP3470.4: CAT II)	The IAO will ensure the application account is established and administered in accordance with a role-based access scheme to enforce least privilege and separation of duties.	**N/A**	
(APP3480.3: CAT II)	The IAO will ensure the access procedures enforce the principles of separation of duties and "least privilege".	**N/A**	
(APP2160.3: CAT II)	The IAO will ensure deployment systems and all components comply with all appropriate DoD STIGS, NSA guides, and all applicable DoD policies.	**N/A**	
(APP6270: CAT II)	The IAO will ensure connections between the DoD enclave and the Internet or other public or commercial wide area networks require a DMZ.	**N/A**	
(APP6290: CAT I)	The Designer and the IAO will ensure physical operating system separation and physical application separation is employed between servers of different data types in the web tier of Increment 1/Phase 1 deployment of the DoD DMZ for Internet-facing applications.	**N/A**	
(APP6310: CAT II)	The IAO will ensure web service inquiries to UDDI provide read-only access to the registry to anonymous users.	**N/A**	
(APP6320: CAT II)	The IAO will ensure if the UDDI registry contains sensitive information, read access to the UDDI registry is granted only to authenticated users.	**N/A**	
(APP3830.2: CAT II)	The IAO will ensure digital signatures exist on UDDI registry entries to verify the publisher.	**N/A**	
(APP3840.2: CAT II)	The IAO will ensure UDDI versions are used supporting digital signatures of registry entries.	**N/A**	
(APP3850.2: CAT II)	The IAO will ensure UDDI publishing is restricted to authenticated users.	**N/A**	
(APP6300: CAT II)	The IAO will ensure an XML firewall is deployed to protect web services.	**N/A**	no web services in hReader
(APP2160.2: CAT II)	The Test Manager will ensure both client and server machines are STIG compliant.	**N/A**	hReader only
(APP2010.2: CAT II)	The Program Manager will ensure all appointments to required IA roles are established in writing to include assigned duties and appointment criteria, such as training, security clearance, and IT designation.	**N/A**	building a prototype

(APP2020.1: CAT II)	The Program Manager will provide an Application Configuration Guide to the application hosting providers.	**N/A**	building a prototype
(APP2020.2: CAT II)	The Program Manager will provide a list of all potential hosting enclaves and connection rules and requirements.	**N/A**	
(APP2040.1: CAT II)	The Program Manager will ensure a Security Classification Guide exists containing data elements and their classifications if the system contains classified information.	**N/A**	
(APP2120.1: CAT II)	The Program Manager will ensure all levels of program management receive security training regarding the necessity, impact, and benefits of integrating secure development practices into the development lifecycle.	**N/A**	building a prototype
(APP2120.2: CAT II)	The Program Manager will ensure designers are provided training on secure design principles for the entire SDLC and newly-discovered vulnerability types on at least an annual basis.	**N/A**	building a prototype
(APP2120.3: CAT II)	The Program Manager will ensure developers are provided with training on secure design and coding practices on at least an annual basis.	**N/A**	building a prototype
(APP2120.4: CAT II)	The Program Manager will ensure testers are provided training on at least an annual basis.	**N/A**	building a prototype
(APP2140.1: CAT II)	The Program Manager will ensure a security incident response process for the application is established that defines reportable incidents and outlines a standard operating procedure for incident response.	**N/A**	building a prototype
(APP2150.1: CAT II)	The Program Manager will ensure procedures are implemented to assure physical handling and storage of information is in accordance with the data's sensitivity.	**N/A**	hReader on github will not contain any sensitive or PHI data

(APP2020.4: CAT II)	The Designer will ensure known security assumptions, implications, system-level protections, best practices, and required permissions are documented in the Application Configuration Guide.	**N/A**	building a prototype
(APP2020.5: CAT II)	The Designer will ensure deployment configuration settings are documented in the Application Configuration Guide.	**N/A**	**GG: building a prototype**
(APP3200: CAT III)	The Designer will ensure transaction-based applications implement transaction roll-back and transaction journaling.	**N/A**	
(APP3220.2: CAT II)	The Designer will ensure classified data held in memory is cryptographically protected when not in use.	**N/A**	no classified data to be used
(APP3230.2: CAT II)	The Designer will ensure the application properly clears or overwrites all memory blocks used to classified data.	**N/A**	no classified data to be used
(APP3270: CAT I)	The Designer will ensure the application has the capability to mark sensitive/classified output when required.	**N/A**	no classified data to be used
(APP3300: CAT II)	The Designer will ensure applications requiring server authentication are PK-enabled.	**N/A**	
(APP3320.5: CAT II)	The Designer will ensure the application has the capability to limit reuse of account passwords within the last 10 password changes.	**N/A**	
(APP3320.6: CAT II)	The Designer will ensure the application has the capability to limit user changes to their account passwords once every 24 hours with the exception of privileged or administrative users.	**N/A**	
(APP3320.7: CAT II)	The Designer will ensure the application has the capability to require new account passwords differ from the previous password by at least four characters when a password is changed.	**N/A**	
(APP3380: CAT II)	The Designer will ensure the application prevents the creation of duplicate accounts.	**N/A**	
(APP3550: CAT I)	The Designer will ensure the application is not vulnerable to integer arithmetic issues.	**N/A**	

(APP3560: CAT I)	The Designer will ensure the application does not contain format string vulnerabilities.	**N/A**	
(APP3660: CAT III)	The Designer will ensure the application has a capability to notify the user on login of date and time of the user's last unsuccessful logon, IP address of the user's last unsuccessful logon, date and time of the user's last successful logon, IP address of the user's last successful logon, and number of unsuccessful logon attempts since the last successful logon.	**N/A**	
(APP3670: CAT II)	The Designer will ensure the application has a capability to display the user's time and date of the last change in data content.	**N/A**	
(APP3760: CAT II)	The Designer will ensure web services are designed and implemented to recognize and react to the attack patterns associated with application-level DoS.	**N/A**	
(APP3770: CAT II)	The Designer will ensure the web service design includes redundancy of critical functions.	**N/A**	
(APP3780: CAT II)	The Designer will ensure web service design of critical functions is implemented using different algorithms to prevent similar attacks from a complete application level DoS.	**N/A**	
(APP3790: CAT II)	The Designer will ensure web services are designed to prioritize requests to increase availability of the system.	**N/A**	
(APP3800: CAT II)	The Designer will ensure execution flow diagrams are created and used to mitigate deadlock and recursion issues.	**N/A**	
(APP3810: CAT I)	The Designer will ensure the application is not vulnerable to XML injection.	**N/A**	
(APP3820: CAT I)	The Designer will ensure web services provide a mechanism for detecting resubmitted SOAP messages.	**N/A**	
(APP3830.1: CAT II)	The Designer will ensure digital signatures exist on UDDI registry	**N/A**	

	entries to verify the publisher.		
(APP3840.1: CAT II)	The Designer will ensure UDDI versions are used supporting digital signatures of registry entries.	**N/A**	
(APP3850.1: CAT II)	The Designer will ensure UDDI publishing is restricted to authenticated users.	**N/A**	
(APP3860: CAT II)	The Designer will ensure SOAP messages requiring integrity sign the following message elements: • Message ID • Service Request • Timestamp • SAML Assertion.	**N/A**	
(APP3870: CAT I)	The Designer will ensure when using WS-Security messages use timestamps with creation and expiration times.	**N/A**	
(APP3880: CAT I)	The Designer will ensure validity periods are verified on all messages using WS-Security or SAML assertions.	**N/A**	
(APP3890: CAT II)	The Designer shall ensure each unique asserting party provides unique assertion ID references for each SAML assertion.	**N/A**	
(APP3900: CAT II)	The Designer shall ensure encrypted assertions or equivalent confidentiality when assertion data is passed through an intermediary and confidentiality of the assertion data is required to pass through the intermediary.	**N/A**	
(APP3910: CAT I)	The Designer shall use the NotBefore and NotOnOrAfter when using the SubjectConfirmation element in a SAML assertion.	**N/A**	
(APP3920: CAT I)	The Designer shall use the both the NotBefore and NotOnOrAfter elements or OneTimeUse element when using the Conditions element in a SAML Assertion.	**N/A**	
(APP3930: CAT II)	The Designer shall ensure if a <OneTimeUse> element is used in an assertion, there is only one used in <Conditions> element of an assertion.	**N/A**	
(APP3940: CAT II)	The Designer will ensure the asserting party uses FIPS-approved random numbers in the generation of SessionIndex in the SAML Element <AuthnStatement>.	**N/A**	
(APP3950: CAT II)	The Designer shall ensure messages are encrypted when the SessionIndex is tied to privacy data.	**N/A**	

(APP3960: CAT II)	The Designer will ensure the application is compliant with all DISR IPv6 profiles.	**N/A**	
(APP3970: CAT II)	The Designer will ensure supporting application services and interfaces have been designed or upgraded for IPv6 transport.	**N/A**	
(APP3980: CAT II)	The Designer will ensure the application is compliant with IPv6 multicast addressing and features an IPv6 network configuration options as defined in RFC 4038.	**N/A**	
(APP3990: CAT II)	The Designer will ensure the application is compliant with the IPv6 addressing scheme as defined in with RFC 1884.	**N/A**	
(APP4010: CAT III)	The Release Manager will ensure the access privileges to the configuration management (CM) repository are reviewed every 3 months.	**N/A**	
(APP4030.1: CAT II)	The Release Manager will develop an SCM plan describing the configuration control and change management process of objects developed and the roles and responsibilities of the organization.	**N/A**	
(APP4030.2: CAT III)	The Release Manager will ensure the SCM plan identifies all objects created during the development process subject to configuration control.	**N/A**	
(APP4030.3: CAT II)	The Release Manager will ensure the SCM plan maintains procedures for identifying individual application components, as well as, entire application releases during all phases of the software development lifecycle.	**N/A**	
(APP4030.4: CAT III)	The Release Manager will ensure the SCM plan identifies and tracks all actions and changes resulting from a change request from initiation to release.	**N/A**	
(APP4030.5: CAT III)	The Release Manager will ensure the SCM plan contains procedures to identify, document, review, and authorize any change requests to the application.	**N/A**	
(APP4030.6: CAT III)	The Release Manager will ensure the SCM plan defines the responsibilities, the actions to be performed, the tools, techniques and methodologies, and defines an initial set of baseline software components.	**N/A**	
(APP4030.7: CAT III)	The Release Manager will ensure the SCM plan objects have security classifications labels.	**N/A**	

(APP4030.8: CAT II)	The Release Manager will ensure the SCM plan identifies tools and version numbers used in the software development lifecycle.	**N/A**	
(APP4030.9: CAT III)	The Release Manager will ensure the SCM plan identifies mechanisms for controlled access of simultaneous individuals updating the same application component.	**N/A**	
(APP4030.10: CAT II)	The Release Manager will ensure the SCM plan assures only authorized changes by authorized persons are possible.	**N/A**	
(APP4030.11: CAT III)	The Release Manager will ensure the SCM plan identifies mechanisms to control access and audit changes between different versions of objects subject to configuration control.	**N/A**	
(APP4030.12: CAT II)	The Release Manager will ensure the SCM plan identifies mechanisms to track and audit all modifications of objects under configuration control. Audits will include the originator and date and time of the modification.	**N/A**	
(APP4040.1: CAT II)	The Release Manager will establish a CCB managing the CM process.	**N/A**	
(APP4040.2: CAT II)	The Release Manager will ensure the IAM is a member of the CCB.	**N/A**	
(APP4040.3: CAT III)	The Release Manager will ensure the CCB meets at least every release cycle or more often.	**N/A**	
(APP5040: CAT II)	The Test Manager will ensure the changes to the application are assessed for IA and accreditation impact prior to implementation.	**N/A**	
(APP5050: CAT II)	The Test Manager will ensure tests plans and procedures are created and executed prior to each release of the application or updates to system patches.	**N/A**	
(APP5060: CAT II)	The Test Manager will ensure tests procedures are created and at least annually executed to ensure system initialization, shutdown, and aborts are configured to ensure the system remains in a secure state.	**N/A**	
(APP5100: CAT III)	The Test Manager will ensure fuzz testing is included in the test plans and procedures and performed for each application release based on application exposure.	**N/A**	

(APP5070: CAT III)	The Test Manager will ensure code coverage statistics are maintained for each release of the application.	**N/A**	
(APP2010.3: CAT II)	The IAO will ensure all appointments to required IA roles are established in writing to include assigned duties and appointment criteria such as training, security clearance, and IT designation.	**N/A**	
(APP2040.2: CAT II)	The IAO will ensure the classification guide for the application data exists and is available to users.	**N/A**	
(APP2020.6: CAT II)	The IAO will ensure the application is deployed in a manner consistent with the Application Configuration Guide provided by the developers.	**N/A**	Actually, not going in the app store - will be open sourced.
(APP2100.4: CAT II)	The IAO will ensure mitigations have been applied from the vulnerability assessments for all ports used in the application.	**N/A**	
(APP2140.2: CAT II)	The IAO will ensure a security incident response process for the application is followed.	**N/A**	
(APP6170: CAT II)	The IAO will ensure back-up copies of the applications software are stored in a fire-rated container and not collocated with operational software.	**N/A**	
(APP6180: CAT II)	The IAO will ensure procedures are in place to assure the appropriate physical and technical protection of the backup and restoration of the application.	**N/A**	
(APP6190.1: CAT II)	The IAO will ensure data backup is performed at least weekly.	**N/A**	
(APP6190.2: CAT II)	The IAO will ensure data backup is performed daily and recovery media is stored off-site at a location.	**N/A**	
(APP6190.3: CAT II)	The IAO will ensure data backup is accomplished by maintaining a redundant secondary system, not collocated, that can be activated without loss of data or disruption to the operation.	**N/A**	

(APP6200.1: CAT II)	The IAO shall ensure a disaster plan exists providing for the smooth transfer of all mission or business essential functions to an alternate site for the duration of an event with little or no loss of operational continuity.	**N/A**	
(APP6200.2: CAT II)	The IAO shall ensure a disaster plan exists providing for the resumption of mission or business essential functions within 24 hours of activation.	**N/A**	
(APP6200.3: CAT II)	The IAO shall ensure a disaster plan exists providing for the partial resumption of mission or business essential functions within 5 days of activation.	**N/A**	
(APP6210: CAT II)	The IAO will ensure an account management process is implemented, verifying only authorized users can gain access to the application and individual accounts designated as inactive, suspended, or terminated are promptly removed.	**N/A**	
(APP6280: CAT I)	The IAO will ensure web servers are on separate network segments from the application and database servers if it is a tiered application.	**N/A**	
(APP3240: CAT II)	The Designer will ensure all access authorizations to data are revoked prior to initial assignment, allocation or reallocation to an unused state.	**N/A**	
(APP3320.4: CAT II)	The Designer will ensure passwords do not contain personal information such as names, telephone numbers, account names, birthdates, or dictionary words.	**N/A**	
(APP6080: CAT II)	The IAO will ensure protections against DoS attacks are implemented.	**N/A**	