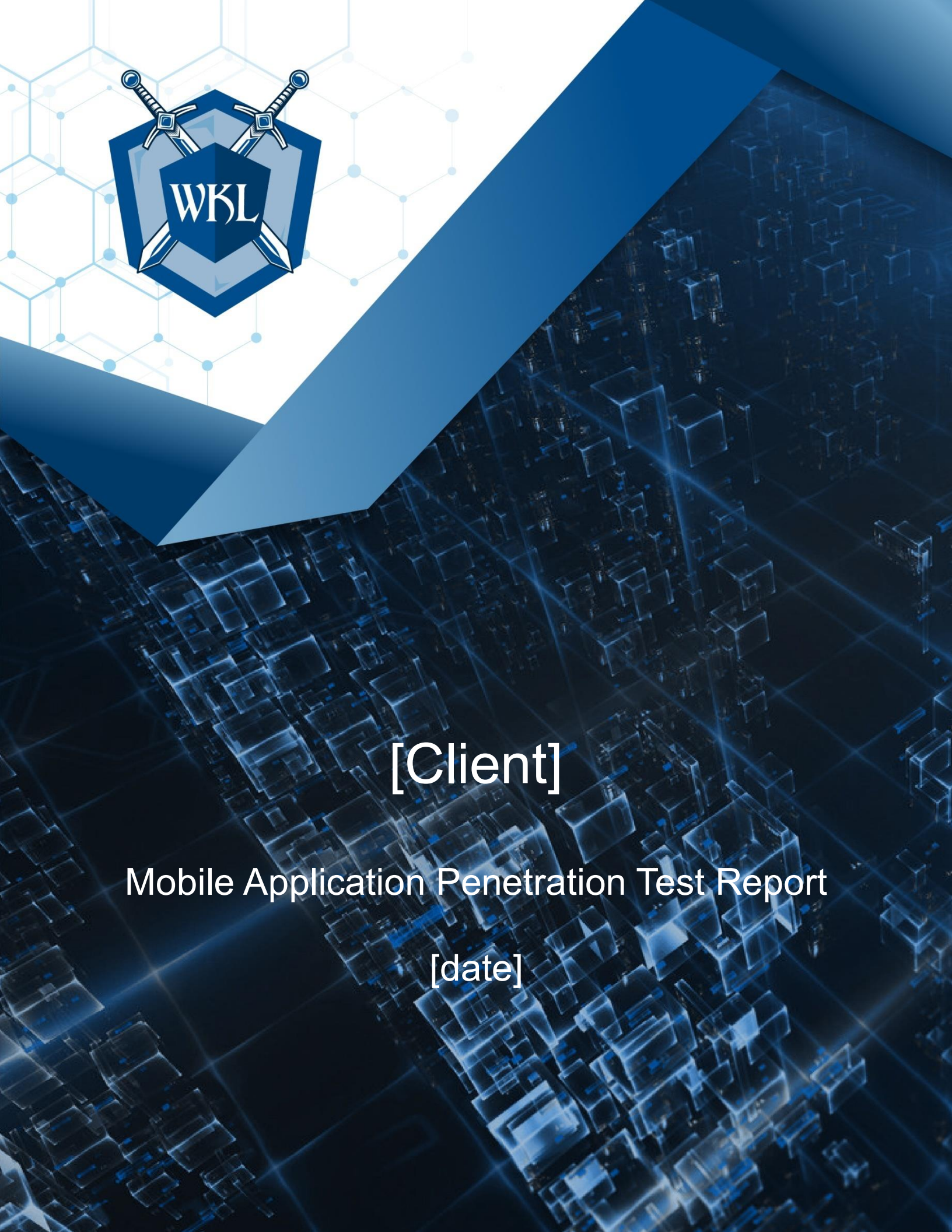


[Client]

Mobile Application Penetration Test Report

[date]

Confidentiality Statement

All information in this document is provided in confidence. It may not be modified by or disclosed to a third party (either in whole or in part) without the prior written approval of White Knight Labs (WKL). WKL will not disclose to any third-party information contained in this document without the prior written approval of [client].

Document Control

Date	Change	Change By	Issue
[date]	Document Created	[engineer]	V0.1
[date]	Document Edited	[editor]	V1.0
[date]	Document Published	[engineer]	V1.1

Document Distribution

Name	Company	Format	Date
[client contact]	[company]	PDF	[date]

White Knight Labs Contact Details

Address	**White Knight Labs** 10703 State Highway 198 Guys Mills PA 16327
Contact	**Tel:** +1 (877) 864-4204 **Mob:** +1 (814) 795-3110 **Email:** info@whiteknightlabs.com

Table of Contents

Executive Summary

Security is a journey, not a destination. Companies must remain vigilant and strive towards a robust security posture. The threat landscape is ever-changing and malicious actors are always innovating. As the internet becomes more hostile, defenders must enhance their capabilities and continue to invest in security.

[Client] engaged White Knight Labs (WKL) to conduct a mobile application penetration test of their mobile application. Client's mobile application provides [company details]. Over the course of this test, WKL was tasked with proactively identifying any vulnerabilities, validating their severity, and providing recommended remediation steps. [Client] seeks to improve its defensive posture and better protect its sensitive information and infrastructure from potential attacks.

The testing was performed between [date] and [date]. This testing represents a point-in-time look at the security posture of the mobile application.

Scoping and Rules of Engagement

While malicious actors are not constrained, WKL understands the need to establish a scope for each assessment. This ensures that work can be completed in a timely manner while protecting third-parties not participating in the engagement. WKL conducted a black box mobile application test with several sets of user credentials. The following briefly elaborates on these techniques:

- **Black-Box Testing**: In a black-box engagement, the consultant does not have access to any internal information such as source code, APIs, extensive details on the technology stack, and is not granted internal access to the client's network or web servers. It is the job of the consultant to perform all reconnaissance to obtain the sensitive knowledge needed to proceed, which places them in a role as close to the typical attacker as possible.
- **User Credentials:** [Client] provided WKL with several sets of user credentials with the permissions and functionality normally granted to its clients. This mimics scenarios where malicious actors (1) may obtain user credentials by compromising a client account or (2) may exploit vulnerabilities related to vertical or horizontal business logic in order to perform unauthorized changes. These credentials, in conjunction with MFA access, allowed WKL to assess [client]'s resilience to authenticated attacks in addition to unauthenticated attacks.
- **Registration Test Data:** [Client] provided WKL with several sets of test data, such as [examples] in order to register new accounts. This allowed consultants to test not only authentication and authorization but also registration from the perspective of a new user with valid data.

WKL evaluated the iOS application version [#] pointing to the test API located at [address]. This IPA was sideloaded onto a jailbroken iPhone 8 running iOS 15.8. The client application was thoroughly assessed, as well as the supporting backend API endpoints. The backend API consisted of approximately [#] API endpoints, of which all requests and responses were encrypted with [specific] encryption. When decrypted, all messages were [specific type], and plain responses.

In addition, the source code was provided. WKL performed static analysis and a manual source code review. Unless otherwise noted, the following components were determined by [client] to be out of scope:

- [component]
- [component]
- [component]
- [component]
- [component]

The following timeline details the entire engagement of the [client] application:

- **Kickoff Call**: [date]
- **Engagement Testing:** [date] – [date]
- **Report Delivery:** TBD
- **Debrief Call** – TBD

WKL's security assessment includes a detailed approach that merges our standard testing methodology with client-specific use cases. This strategy is crafted to deliver a nuanced assessment, ensuring thorough coverage of general security vulnerabilities as well as the particular aspects that concern the client.

Client-Specific Use Cases Testing Table:

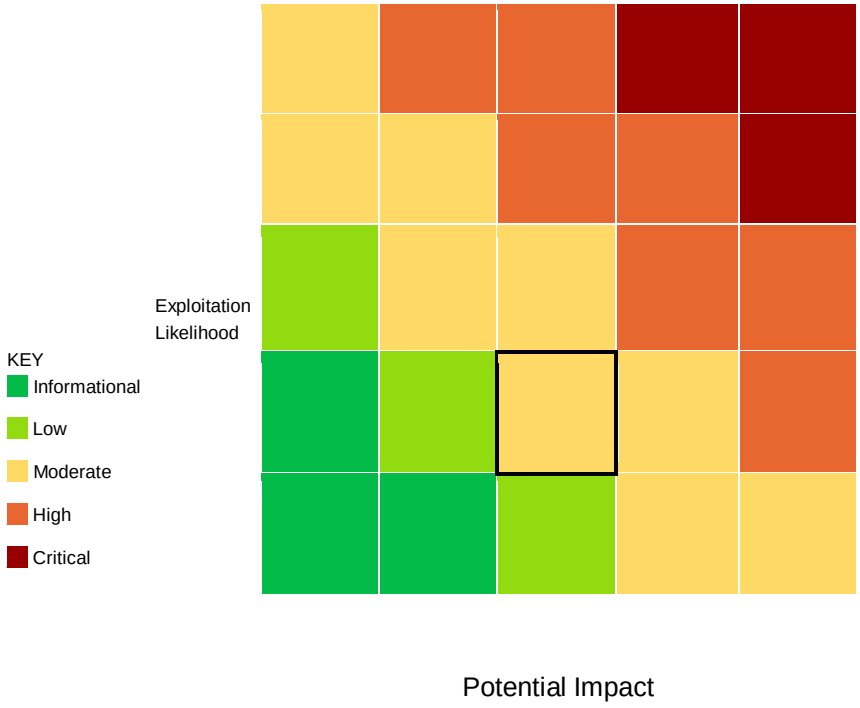
Test Case	Description	Tested (Yes/No)
Horizontal Privilege Escalation	Ensure that business logic cannot be exploited to allow users to modify other client accounts or perform transactions without authorization.	Yes
Insecure Direct Object Reference	Ensure that numerical user IDs cannot be modified in order to perform actions or enumerate accounts attackers do not have access to.	Yes

Data in Transit	Ensure that certificate validation, encryption, and iOS application transport security (ATS) are enforced. Data in transit protections need to be in to place to prevent MITM attacks.	Yes
Data at Rest	Ensure that moderate and highly sensitive data are either not stored at rest, or are properly stored in the Keychain.	Yes
Client-Side Weaknesses	Ensure that the client enforces binary protections and does not suffer from remote code execution (RCE) or any other logical flaw that relies on client instead of server-side logic.	Yes
Improper Cryptography	Ensure that the client does not hard-code cryptographic keys or make use of weak ciphers, hashes, or proprietary cryptographic algorithms.	Yes

[Client] Risk Rating

WKL calculated the risk to [client] based on exploitation likelihood (ease of exploitation) and potential impact (potential business impact to the environment). Overall, the application is at a relatively low level of risk and is in line with typical configurations for mobile applications. Some areas of improvement are important to address, data storage practices in particular. [These types of] applications should be held to the highest level of scrutiny for security, due to the sensitive nature of [data] handled. WKL recommends that the two medium-severity issues be prioritized, then the low-severity issues be addressed as defense-in-depth best practices to harden the overall security posture of the [client] mobile application.

Overall Risk Rating: Moderate



Summary of Findings

The security assessment conducted has identified key areas where [client] could enhance its security posture. WKL recommends that investments be directed towards the following initiatives for improvements.

Sensitive Data Storage:

- The application makes use of several insecure locations for data storage of moderately sensitive data. It also uses hard-coded cryptographic key values that can be extracted from the application at runtime or through static analysis. The development team should first consider whether the data needs to be stored on the device, then use the Keychain for all moderate or sensitive data. Keys should never be hard-coded, but if they need to be stored on the device, they should also use the Keychain.

Defense in Depth Protections:

- The application lacks defense-in-depth protections that will harden the overall security posture. These protections include certificate pinning and anti-tampering/anti-jailbreak protections. These will mitigate the opportunity for attackers to exploit MITM attacks via a compromised CA, or to obtain sensitive PCI or credentials on a rooted device.

TLS weaknesses:

- The application API serves TLS 1.0 and TLS 1.1, as well as outdated and potentially vulnerable ciphers. These are long out of compliance for PCI and contain known vulnerabilities.

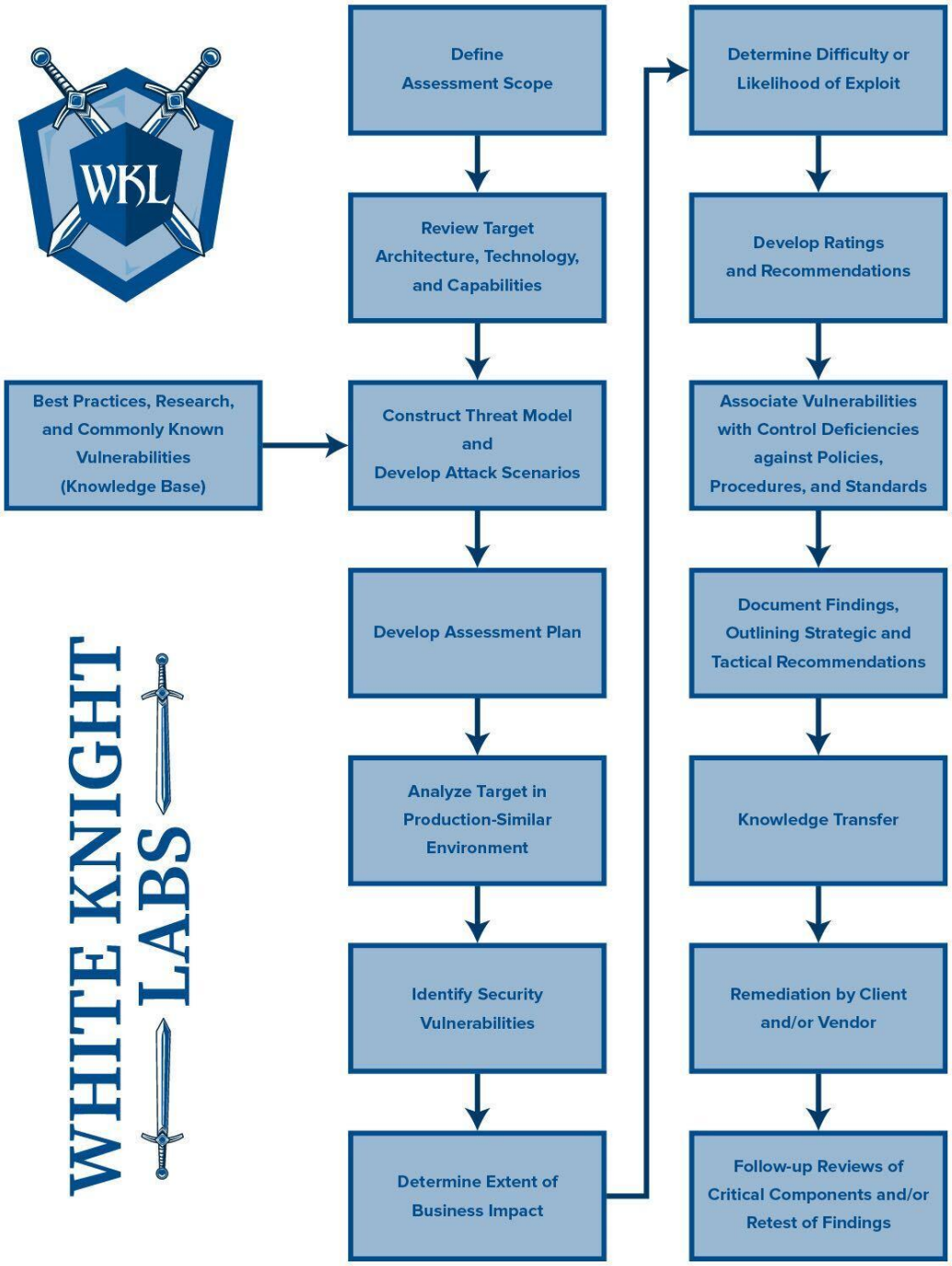
The findings from the security testing are categorized in the table provided, with in-depth analysis available in the Findings section of this report. By addressing the vulnerabilities listed, [client] can continue to improve its defense mechanisms and maintain a strong security framework.

Risk	Vulnerability
Medium	Sensitive Data in Logs
Medium	Insecure Storage of MFA "Remember Me"
Low	Hard-Coded Cryptographic Key
Low	TLS Configuration Weaknesses
Low	Lack of Anti-Instrumentation/Jailbreak Detection
Low	Lack of Certificate Pinning

Application Testing Methodology

WKL defines an application security assessment as an assessment designed to highlight potential security vulnerabilities within an application based upon a defined threat model. An application assessment is intended to identify design failures and unsafe coding practices. Security-critical issues are commonly encountered in the following areas: authentication, authorization, session management, data validation, use of cryptography, error handling, information leakage, and other language-specific issues. During the assessment, WKL assigned business risk ratings based on our current understanding of the application.

WKL utilizes a comprehensive assessment methodology, providing results with the utmost accuracy and ensuring representational coverage of risks facing an application or information system. This assessment methodology is based upon an understanding of the business use cases, and the types of data stored, processed, or transmitted by a given system or system component. Once these elements decompose, potential risks affecting their interaction are evaluated by the assessment team as illustrated by the following process flow:


Define Assessment Scope
Review Target Architecture, Technology, and Capabilities
Best Practices, Research, and Commonly Known Vulnerabilities (Knowledge Base)
Construct Threat Model and Develop Attack Scenarios
Develop Assessment Plan
Analyze Target in Production-Similar Environment
Identify Security Vulnerabilities
Determine Extent of Business Impact
Determine Difficulty or Likelihood of Exploit
Develop Ratings and Recommendations
Associate Vulnerabilities with Control Deficiencies against Policies, Procedures, and Standards
Document Findings, Outlining Strategic and Tactical Recommendations
Knowledge Transfer
Remediation by Client and/or Vendor
Follow-up Reviews of Critical Components and/or Retest of Findings

Application Penetration Assessments

The assessment team relies primarily on manual penetration testing to ensure coverage across the OWASP Top 10 vulnerability classes, as well as assessing other risks resulting from choices in technology, application logic, and integration between application and system components or application use cases.

The WKL approach and methodology is not limited to the OWASP Top 10 vulnerability classes. Instead, it allows the assessment team to adapt testing based upon the risks most likely to affect the client using the threat model and attack plan defined during the threat modeling phase of the engagement. The following OWASP Top 10 vulnerability classes are included in each application penetration assessment:

- Broken Access Control
- Cryptographic Failures
- Injection
- Insecure Design
- Security Misconfiguration
- Vulnerable and Outdated Components
- Identification and Authentication Failures
- Software and Data Integrity Failures
- Security Logging and Monitoring Failures
- Server-Side Request Forgery

The inclusion of manual penetration testing executed during the assessment provides greater coverage of classes of vulnerabilities that often go undetected by automated vulnerability assessment tools and dynamic web application security scanners. These classes include authentication, authorization, session management, cryptographic weaknesses, and application business logic. Lastly, careful manual execution of the test cases allows the application security team to identify and closely coordinate test cases that may be more likely to impact system and service availability, thereby minimizing potential impact to production systems.

Common Attack Vectors Considered

During initial preparation for an application security assessment, common attack vectors are specified to ensure consistent focus and a comprehensive approach. These provide the structure for the engagement team's tasks and are reflected in the final reporting. Some potential attack vectors considered in web-based applications include:

ATTACK VECTORS		
CATEGORY	**TYPICAL VULNERABILITIES**	**AREAS OF INVESTIGATION**
Data Validation	Failure to test the validity of user-supplied data against known parameters, including but not limited to length, character composition, or conformance to a pre-determined syntax.	• SQL injection • Cross-site scripting • Form field manipulation • Canonicalization • Buffer overflows • Format string attacks • Shell meta-character injection • Reliance on client-side security or behavior • Miscellaneous input validation issues
Session Management	Failure to use strong, unpredictable session identifiers and to maintain server-stored state such that each request can be uniquely identified and attributed to a certain user.	• General observations • Static session identifiers • Easily predictable identifiers • Insufficient length • Known algorithms • Miscellaneous session management issues
Access Controls	Failure to verify the authenticity of a user and enforce appropriate restrictions on certain data or functionality.	• Authentication bypass • Authorization bypass • Inconsistent use of access control • State manipulation • Miscellaneous access control issues
Cryptography	Failure to use strong encryption. This implies using a cryptographically proven algorithm along with a key that is sufficiently random and unpredictable.	• Proprietary or home-grown encryption • Insecure cipher mode • Poor key selection • Insufficient key length • Inappropriate key reuse • Miscellaneous cryptography issues
Third-Party Components	Vulnerabilities in supporting architecture that can be remotely exploited to compromise the server or gather useful information.	• Publicly disclosed vulnerabilities • Team proprietary vulnerabilities • Configuration errors • Default content

Application Testing Findings

Finding: Medium – Sensitive Data in Logs

Description

The iOS application logs sensitive HTTP response data in logs. WKL observed that all API responses are logged in decrypted format. This bypasses the protection offered by application layer encryption and does not need a jailbroken environment to be exploited. Sensitive data, such as [specific data], were found to be exposed. Attackers with physical access to a device may be able to obtain the sensitive data in logs without needing credentials.

Impact

Attackers with physical access to devices may be able to obtain sensitive decrypted data from application logs.

Evidence

The following screenshot shows access and refresh tokens used to authenticate to the service logged in decrypted format in the console logs.

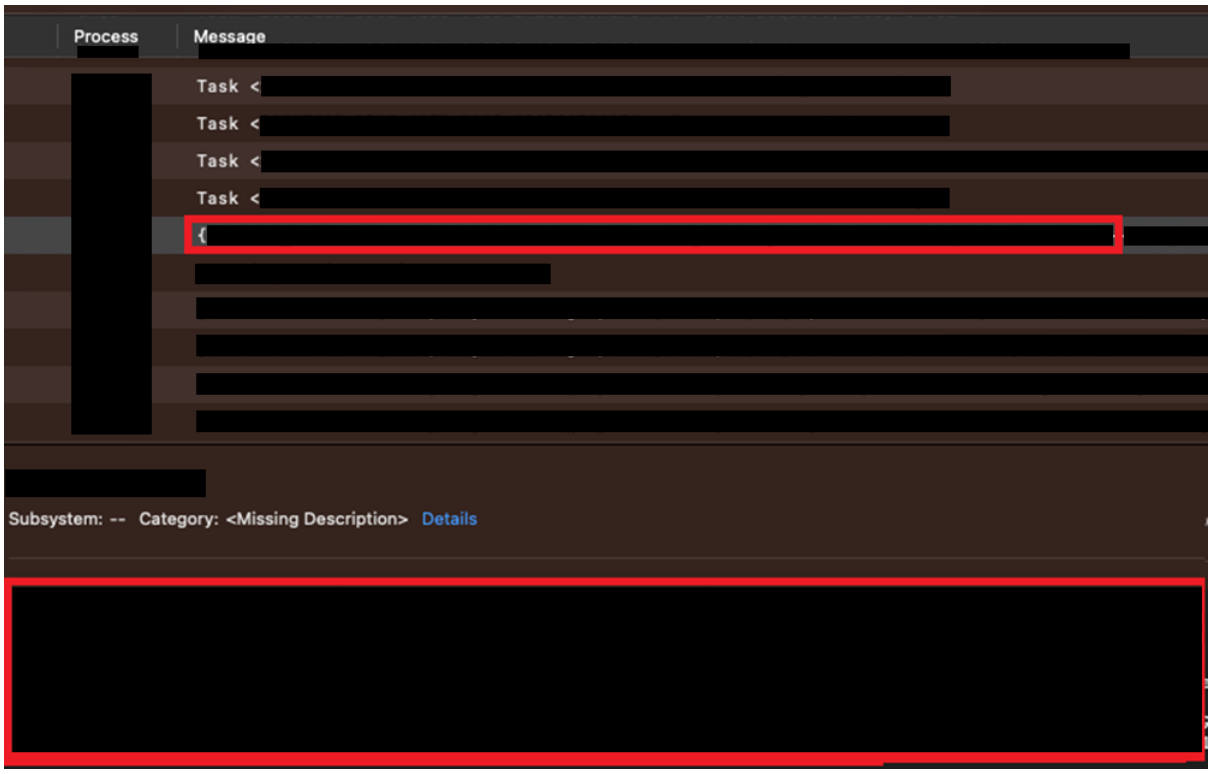


Figure 1: Console output with access and refresh tokens

URL Locations

- URL

Recommendations

Do not log sensitive data. WKL recommends the client review the source code for areas that make use of a specific function. It may be useful for testing purposes to log requests and responses, but this functionality as well as debug functionality needs to be removed in production applications.

Debug log functionality can be controlled at a higher level for test builds, using the debug preprocessor. This flag can then be disabled in production, to prevent logging. For more information, please see the following Apple developer blog article and reference URL:

- https://developer.apple.com/library/archive/technotes/tn2347/_index.html
- https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Finding: Medium – Insecure Storage of MFA "Remember Me"

Description

A multi-factor authentication (MFA) "remember me" token is stored on the device in an insecure fashion. When the user logs into the application, they are required to enter an MFA token via SMS, email, or an authenticator app. To prevent needing to enter an MFA token on every login, the application allows the user to select if they are using a shared device or a personal device. The personal device option returns a special token (cookie) that is stored on the device and submitted along with each login request to validate the MFA step. A device ID is needed in conjunction with this MFA cookie. The device ID is also stored in an insecure location on the device, in a plaintext SQLite database. Effectively, if an attacker recovered this value, they could bypass MFA if they had the user's credentials.

Impact

Attackers with physical access to the device may be able to recover the plaintext data through forensic techniques or application backups. The data is easily available on jailbroken devices.

Evidence

The following output shows the deviceID recovered from a SQLite database in the device's app data folder.

```
 $  sqlite3 [redacted]
SQLite version [redacted]
Enter ".help" for usage hints.
sqlite> .tables
[TableName]  Device           User
sqlite> select * from [TableName];
[redacted]
[redacted]
sqlite> select * from Device;
iOS Device (iPhone9,1)|15.8|[redacted]
```

The following Objection output shows the MFA "remember me" cookie values stored in the plaintext specific location within the application on iOS:

```
$ [redacted] on (iPhone: 15.8) [usb] # ios [redacted] get
{
  3DTouchEnabled = false;
   "4.2.1" = 1;
```

```
    AKLastIDMSEnvironment = 0;
    [redacted]FontSize = 2;
    AddingEmojiKeybordHandled = 1;
    AdditionalTransactionDownload = 30;
    AppleLanguages =     (
        en
    );
    AppleLanguagesDidMigrate = 19H370;
    ApplePasscodeKeyboards =     (
        "en_US",
        emoji


    );
    CookieId = "[redacted]";
    INNextFreshmintRefreshDateKey = "[date]";
    INNextHearbeatDate = "[date]";
    InitialTransactionDownload = 90;
    LoginTouchIDEnabled = false;
    NSAllowsDefaultLineBreakStrategy = 1;
    NSInterfaceStyle = macintosh;
    NSLanguages =     (
        en
    );
    PKKeychainVersionKey = #;
    PKLogNotificationServiceResponsesKey = #;
    PrivacyTimeoutDatetime = "[date]";
    QuickViewEnabled = false;
    RequirePin = false;
    TouchIDEnabled = false;
    "com.apple.content-rating.AppRating" = 1000;
    "com.apple.content-rating.ExplicitBooksAllowed" = 1;
    "com.apple.content-rating.ExplicitMusicPodcastsAllowed" = 0;
    "com.apple.content-rating.MovieRating" = 1000;
    "com.apple.content-rating.TVShowRating" = 1000;
}
```

The following decrypted HTTP request shows an example of the MFA "remember me" cookie use. Also, take note of the device ID.

```
POST /api/[redacted]/V3/MultiFactorLogin HTTP/2
Host: [redacted]
Accept: */*
Content-Type: application/json
Accept-Encoding: [redacted]
User-Agent: [redacted]
Content-Length: 461
Accept-Language: en-US,en;q=0.9
AES: Killer
{'PayloadId' : '[redacted]', 'UserName' : '[redacted]' , 'Password' : '[redacted]' , 'Model' :
'iOS Device (iPhone9,1)' , 'OS' : '15.8' , 'InstallationId' : '[redacted]' , 'Version' :
'4.2.1' , 'UserAgent': '' , 'CookieId' : '[redacted]'}
```

The resulting HTTP response shows an example of the response, indicating that login was successful.

```
HTTP/2 200 OK
Cache-Control: no-cache,no-cache, no-store, must-revalidate, pre-check=0, post-check=0, max-age=0, s-maxage=0
Pragma: no-cache,no-cache
Content-Length: 1511
Content-Type: text/plain; charset=utf-8
Expires: -1,0
Content-Security-Policy: frame-ancestors 'self'
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
Strict-Transport-Security: max-age=31536000; includeSubDomains, preload
Date: [redacted]
AES: Killer
{"access_token":"[redacted]"}
```

URL Locations

- URL

Recommendations

For any sensitive or moderately sensitive data, ensure that it is stored in the Keychain. The Keychain provides entitlements for each application so that no other application can access each other’s keys, except on jailbroken devices. The iOS Keychain provides strong encrypted storage that uses keys derived by both the user’s device password/passcode and hardware-based keys that cannot be extracted by attackers. Consider storing both the device ID and MFA "remember me" cookie in the Keychain.

Finding: Low – Hard-Coded Cryptographic Key

Description

The iOS application makes use of a hard-coded cryptographic key. Normally the exposure of cryptographic material is considered a high severity problem when it can be used to obtain sensitive data or manipulate critical application functionality. However, in this case, the severity is considered low due to the consequences of exploitation. [Additional, specific finding details]. Although the HTTP traffic can be intercepted and decrypted, the TLS layer is offering confidentiality, integrity, and authenticity. Therefore, MITM attacks are largely prevented, making the consequences of breaking application-layer encryption a low severity. However, it should be noted that although the consequences are relatively low, the exposure is high. The key material and algorithm are the same for all application installs, which means that, if obtained, this protection is compromised for every device and every user who installs it from the app store.

Impact

A hard-coded cryptographic key can be obtained by attackers and used to decrypt or encrypt HTTP payloads at the application layer. Since the key is the same for all application installs, an attacker can compromise this protection for any and all users of the client mobile application.

Evidence

Hard-coding a key makes it possible for attackers to extract the key by performing dynamic instrumentation. The key is also readily available to any internal user with access to the internal source code. Reverse engineering the application to extract the data from compiled code is more difficult but possible once the app store encryption is stripped by repackaging the decrypted binary from memory. The following examples show the key extracted from the source and from instrumentation.

The following code snippet shows the hard-coded key located at file path.

```
 NSString *dataToBase64Id1 = @"[redacted]";
NSString *dataToBase64Id2 = @"[redacted]";
+(NSString*) encrypt:(NSString*)plaintext
{
   @try {
       NSData *objEncryptData = [NSData dataWithData:[plaintext
dataUsingEncoding:NSUTF8StringEncoding]];
       objEncryptData = [objEncryptData AES256EncryptWithKey:dataToBase64Id1
iv:dataToBase64Id2];


       NSString *encrypted = [objEncryptData base64EncodedStringWithOptions:0];
       objEncryptData = nil;
```

```
        return encrypted;
    }
    @catch (NSException *exception) {
        NSLog( @"Class: %@", TAG);
        NSLog( @"Name: %@", exception.name);
        NSLog( @"Error: %@", exception.reason);
        [LogError logError:[[exception.name stringByAppendingString:@" : "]
stringByAppendingString:exception.reason] class:TAG];
        return @"";
    }
    @finally {


    }
}
+(NSString*) decrypt:(NSString*)encrypted
{
    @try {


        NSData *decodedData = [[NSData alloc] initWithBase64EncodedString:encrypted options:0];


        NSData *objDecryptData = [decodedData AES256DecryptWithKey:dataToBase64Id1
iv:dataToBase64Id2];
        NSString *decrypted = [[NSString alloc] initWithData:objDecryptData
encoding:NSUTF8StringEncoding];


        objDecryptData = nil;
        return  decrypted;
    }
    @catch (NSException *exception) {
        NSLog( @"Class: %@", TAG);
        NSLog( @"Name: %@", exception.name);
        NSLog( @"Error: %@", exception.reason);
        [LogError logError:[[exception.name stringByAppendingString:@" : "]
stringByAppendingString:exception.reason] class:TAG];
        return @"";
    }
    @finally {
```

The following text shows the hard-coded key and decrypted HTTP data returned from dynamic instrumentation.

```
agent) Registering job 607982. Type: ios-crypto-monitor
[redacted] on (iPhone: 15.8) [usb] # (agent) [redacted] [CCCryptorUpdate] (
 dataIn : [redacted]
 dataOut : [redacted]
)
(agent) [redacted] [CCCryptorFinal] (
 dataOut : [redacted]
)
(agent) [redacted] [CCCrypt] (
 op : [redacted]
```

```
 alg : [redacted]
 options : [redacted]
 keyLength : 32
 key : [redacted]
 iv : [redacted]
 dataIn : [redacted]
 dataOut : [redacted]
)
(agent) [redacted] [CCCryptorUpdate] (
 dataIn : {'PayloadId' : '[redacted]' , 'UserName' : '[redacted]' , 'Password' : '[redacted]' ,
'Model' : 'iOS Device (iPhone9,1)' , 'OS' : '15.8' , 'InstallationId' : '[redacted]' ,
'Version' : '4.2.1' , 'UserAgent': '' , 'CookieId' : '[redacted]'}
 dataOut : [redacted]
)
(agent) [redacted] [CCCryptorFinal] (
 dataOut : [redacted])
```

URL Locations

- **Source Code:** location
- **API host:** location

Recommendations

Do not hard code cryptographic key material. Hard-coded keys are the same for all instances of an installation and affect all users of an application if extracted. Extraction of key material requires advanced techniques but is a very feasible task. Obfuscation is not a solution, because obfuscation is ultimately a form of encoding that can be defeated, especially through runtime manipulation.

If this protection is to be kept, consider implementing an asymmetric key exchange algorithm such as Elliptic Curve Diffie-Hellman (ECDH). Data can either be sent using pure asymmetric key cryptography, or the key-exchange can be used to establish a symmetric key, which is unique and securely randomly generated per-session. For more information, please visit the following URL:

- https://cryptobook.nakov.com/asymmetric-key-ciphers/ecdh-key-exchange

Finding: Low – TLS Configuration Weaknesses

Description

WKL identified TLS configuration weaknesses on the target server. These weaknesses include outdated TLS versions TLS1.0 and TLS1.1 and the support for medium strength cipher suites. These vulnerabilities can expose the server to potential security risks and compromise the confidentiality and integrity of data transmitted over the network.

TLS 1.2 or higher is recommended, although TLS 1.3 is faster and more secure than TLS 1.2. One of the changes that makes TLS 1.3 faster is an update to the way a TLS handshake works: TLS handshakes in TLS 1.3 only require one round trip (or back-and-forth communication) instead of two, shortening the process by a few milliseconds. And in cases when the client has connected to a website before, the TLS handshake will have zero round trips. This makes HTTPS connections faster, cutting down on latency and improving the overall user experience.

Many of the major vulnerabilities in TLS 1.2 had to do with older cryptographic algorithms that were still supported. TLS 1.3 drops support for these vulnerable cryptographic algorithms and, as a result, it is less vulnerable to cyber attacks.

Impact

Attackers on shared networks, who are also in the man-in-the-middle position, may be able to downgrade (or initially establish) a connection to a weak or vulnerable TLS version or cipher suite. This can allow attackers to decrypt otherwise encrypted data. The opportunity for this attack is lessened since, although the server supports vulnerable versions, the iOS client will negotiate TLS 1.2 or higher.

Evidence

To validate the SSL/TLS configuration weaknesses, the sslyze tool was used with the following command.

```
sslyze [redacted]
```

The output of the sslyze command revealed the vulnerabilities and weaknesses in the TLS configuration. The output below shows the command execution and the resulting vulnerabilities.

```
CHECKING CONNECTIVITY TO SERVER(S)
----------------------------------
  [redacted] => [redacted]
```

```
SCAN RESULTS FOR [redacted] - [redacted]
------------------------------------------------------------
* Certificates Information:
      Hostname sent for SNI:             [redacted]
      Number of certificates detected:   1

    Certificate #0 ( _RSAPublicKey )
      SHA1 Fingerprint:                  [redacted]
      Common Name:                       * [redacted]
      Issuer:                            [redacted]
      Serial Number:                     [redacted]
      Not Before:                        [redacted]
      Not After:                         [redacted]
      Public Key Algorithm:              _RSAPublicKey
      Signature Algorithm:               sha256
      Key Size:                          [redacted]
      Exponent:                          [redacted]
      SubjAltName - DNS Names:           ['* [redacted], '[redacted]']
    Certificate #0 - Trust
      Hostname Validation:               OK - Certificate matches server hostname
      Android CA Store (13.0.0_r9):      OK - Certificate is trusted
      Apple CA Store (iOS 16, iPadOS 16, macOS 13, tvOS 16, and watchOS 9):OK - Certificate is
trusted
      Java CA Store (jdk-13.0.2):        OK - Certificate is trusted
      Mozilla CA Store (2022-12-11):     OK - Certificate is trusted
      Windows CA Store (2023-02-19):     OK - Certificate is trusted
      Symantec 2018 Deprecation:         OK - Not a Symantec-issued certificate
      Received Chain:                    * [redacted] --> Entrust Certification Authority - L1K
      Verified Chain:                    * [redacted] --> Entrust Certification Authority - L1K
--> Entrust Root Certification Authority - G2
      Received Chain Contains Anchor:    OK - Anchor certificate not sent
      Received Chain Order:              OK - Order is valid
      Verified Chain contains SHA1:      OK - No SHA1-signed certificate in the verified
certificate chain
    Certificate #0 - Extensions
      OCSP Must-Staple:                  NOT SUPPORTED - Extension not found
      Certificate Transparency:          OK - 3 SCTs included
    Certificate #0 - OCSP Stapling
      OCSP Response Status:              SUCCESSFUL
      Validation w/ Mozilla Store:       OK - Response is trusted
      Responder Name:                    CN=Entrust Validation Authority,O=Entrust\, Inc.,C=US
      Cert Status:                       GOOD
      Cert Serial Number:                [redacted]
      This Update:                       [redacted]
      Next Update:                       [redacted]
* SSL 2.0 Cipher Suites:
    Attempted to connect using 7 cipher suites; the server rejected all cipher suites.
* SSL 3.0 Cipher Suites:
    Attempted to connect using 80 cipher suites; the server rejected all cipher suites.
* TLS 1.0 Cipher Suites:
    Attempted to connect using 80 cipher suites.
    The server accepted the following 4 cipher suites:
       TLS_RSA_WITH_AES_256_CBC_SHA                    256
       TLS_RSA_WITH_AES_128_CBC_SHA                    128
       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA              256       ECDH: secp384r1 (384 bits)
       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA              128       ECDH: prime256v1 (256 bits)
    The group of cipher suites supported by the server has the following properties:
      Forward Secrecy                 OK - Supported
```

```
      Legacy RC4 Algorithm              OK - Not Supported

* TLS 1.1 Cipher Suites:
    Attempted to connect using 80 cipher suites.
    The server accepted the following 4 cipher suites:
       TLS_RSA_WITH_AES_256_CBC_SHA                    256
       TLS_RSA_WITH_AES_128_CBC_SHA                    128
       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA              256       ECDH: secp384r1 (384 bits)
       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA              128       ECDH: prime256v1 (256 bits)
    The group of cipher suites supported by the server has the following properties:
      Forward Secrecy                   OK - Supported
      Legacy RC4 Algorithm              OK - Not Supported

* TLS 1.2 Cipher Suites:
    Attempted to connect using 156 cipher suites.
    The server accepted the following 14 cipher suites:
       TLS_RSA_WITH_AES_256_GCM_SHA384                 256
       TLS_RSA_WITH_AES_256_CBC_SHA256                 256
       TLS_RSA_WITH_AES_256_CBC_SHA                    256
       TLS_RSA_WITH_AES_128_GCM_SHA256                 128
       TLS_RSA_WITH_AES_128_CBC_SHA256                 128
       TLS_RSA_WITH_AES_128_CBC_SHA                    128
       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384           256       ECDH: secp384r1 (384 bits)
       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384           256       ECDH: secp384r1 (384 bits)
       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA              256       ECDH: secp384r1 (384 bits)
       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256           128       ECDH: prime256v1 (256 bits)
       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256           128       ECDH: prime256v1 (256 bits)
       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA              128       ECDH: prime256v1 (256 bits)
       TLS_DHE_RSA_WITH_AES_256_GCM_SHA384             256       DH (2048 bits)
       TLS_DHE_RSA_WITH_AES_128_GCM_SHA256             128       DH (2048 bits)
    The group of cipher suites supported by the server has the following properties:
      Forward Secrecy                   OK - Supported
      Legacy RC4 Algorithm              OK - Not Supported

* TLS 1.3 Cipher Suites:
    Attempted to connect using 5 cipher suites; the server rejected all cipher suites.
* Deflate Compression:
                                        OK - Compression disabled
* OpenSSL CCS Injection:
                                        OK - Not vulnerable to OpenSSL CCS injection
* OpenSSL Heartbleed:
                                        OK - Not vulnerable to Heartbleed
* ROBOT Attack:
                                        OK - Not vulnerable.
* Session Renegotiation:
      Client Renegotiation DoS Attack:  OK - Not vulnerable
      Secure Renegotiation:             OK - Supported
* Elliptic Curve Key Exchange:
      Supported curves:                 X25519, prime256v1, secp384r1
      Rejected curves:                  X448, prime192v1, secp160k1, secp160r1, secp160r2,
secp192k1, secp224k1, secp224r1, secp256k1, secp521r1, sect163k1, sect163r1, sect163r2,
sect193r1, sect193r2, sect233k1, sect233r1, sect239k1, sect283k1, sect283r1, sect409k1,
sect409r1, sect571k1, sect571r1
SCANS COMPLETED IN 38.736273 S
---------------------------
COMPLIANCE AGAINST MOZILLA TLS CONFIGURATION
-------------------------------------------
```

```
   Checking results against Mozilla's "MozillaTlsConfigurationEnum.INTERMEDIATE" configuration.
See https://ssl-config.mozilla.org/ for more details.
    [redacted]: FAILED - Not compliant.
        * maximum_certificate_lifespan: Certificate life span is 396 days, should be less than
366.
        * tls_versions: TLS versions {'TLSv1.1', 'TLSv1'} are supported, but should be rejected.
        * ciphers: Cipher suites {'TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA',
'TLS_RSA_WITH_AES_256_CBC_SHA', 'TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256',
'TLS_RSA_WITH_AES_128_GCM_SHA256', 'TLS_RSA_WITH_AES_256_CBC_SHA256',
'TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384', 'TLS_RSA_WITH_AES_256_GCM_SHA384',
'TLS_RSA_WITH_AES_128_CBC_SHA256', 'TLS_RSA_WITH_AES_128_CBC_SHA',
'TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA'} are supported, but should be rejected.
```

The output clearly indicates the presence of TLS configuration weaknesses, including the support for outdated TLS versions and medium strength cipher suites.

Recommendations

To mitigate the SSL/TLS configuration weaknesses, the following recommendations should be implemented:

1. **Update TLS Version:** Disable support for outdated TLS versions TLS v1.0 and TLS v1.1 and enforce the use of newer and more secure TLS versions (e.g., TLS v1.2 or TLS v1.3).
2. **Strong Cipher Suites:** Review and update the server's cipher suite configuration to only allow strong cryptographic algorithms and eliminate the support for medium strength cipher suites.

Regularly monitor and update the SSL/TLS configuration to stay up to date with best practices and industry standards. By implementing these recommendations, the server can enhance its SSL/TLS security and protect sensitive data transmitted over the network. For more information, please visit the following URLs:

- https://ssl-config.mozilla.org/
- https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/01-Testing_for_Weak_SSL_TLS_Ciphers_Insufficient_Transport_Layer_Protection
- https://www.cloudflare.com/learning/ssl/why-use-tls-1.3/

Finding: Low – Lack of Anti-Instrumentation/Jailbreak Detection

Description

The application fails to prevent itself from performing integrity checks, running in hostile environments, or attempting to instrument the process at runtime. Typically, iOS applications do not need to implement these protections in a standard iOS environment, as they are protected by app container segregation, SELinux file system permissions, and cryptographic signing and verification through trusted sources via the app store. However, on a jailbroken device, these protections all break down as root access can be granted to any process. As a defense-in-depth strategy, it is best to detect hostile environments, along with changes to the application integrity and attempts to manipulate the application through dylib injection (a common way of instrumentation of the running process).

Impact

If the application is run in a hostile environment, any sensitive data stored on the device, held in memory, or sent across a network can be captured by attackers.

Evidence

The screenshot below shows the client app running alongside the Sileo repository application and Palera1n jailbreak app on a jailbroken iOS 15.8 device.

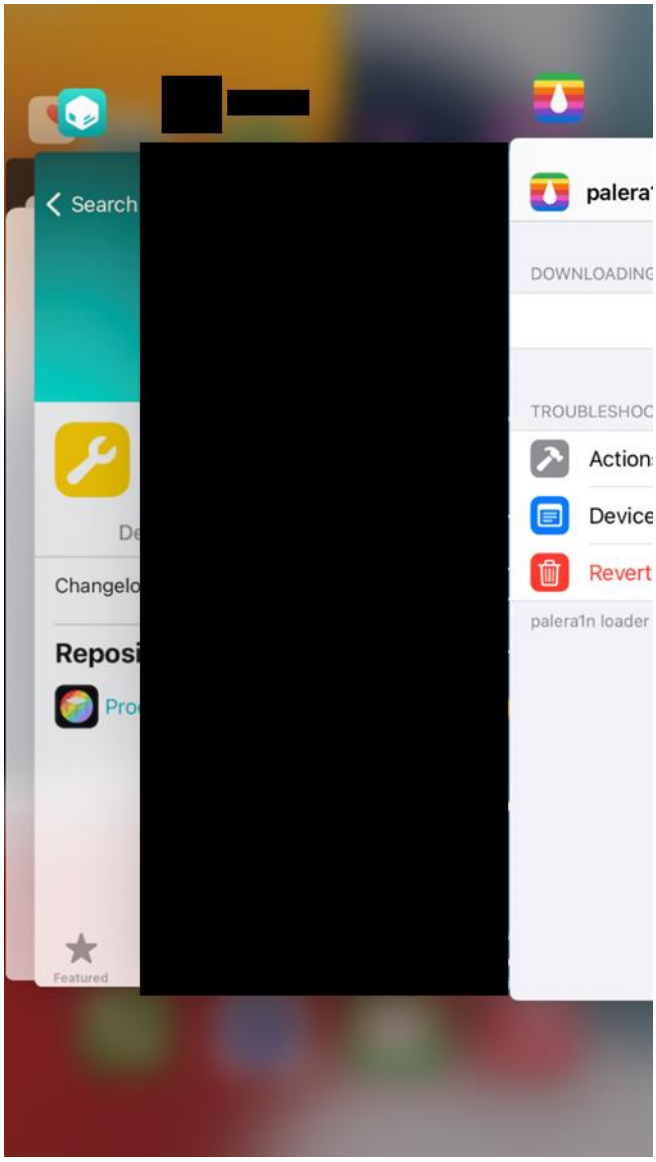


Figure 2: [Client] app running alongside Sileo and Palera1n

URL Locations

- URL

Recommendations

WKL recommends enabling integrity, instrumentation, and jailbreak protections. These can be performed in a number of different ways. For instance, an application can detect the presence of a jailbroken environment on iOS by attempting to escalate to a root shell, attempting a method not allowed in jailed environments such as “fork()”, or detecting the presence of files and apps that indicate a jailbroken environment. Please see the following guide for more information:

- https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md

Finding: Low – Lack of Certificate Pinning

Description

The application fails to perform certificate pinning. Certificate pinning is an additional layer of protection, on top of TLS certificate validation, that ensures the authenticity of the server. In an instance where a certificate authority has been compromised, a malicious certificate could be issued by a trusted authority, allowing attackers to silently perform man-in-the-middle (MITM) attacks on otherwise encrypted connections. This protection is useful as a defense-in-depth strategy.

Impact

If a certificate authority (CA) is compromised, attackers may be able to perform man-in-the-middle attacks without the end user being notified of a potential malicious connection. When attackers compromise CAs they can issue certificates that are signed by a trusted CA and pass certificate validation without needing to install on a client.

Evidence

The following HTTP request shows data intercepted and decrypted via an intercepting proxy.

```
 POST /api/[redacted]/[redacted] HTTP/2
Host: [redacted]
Accept: */*
Content-Type: application/json
Accept-Encoding: gzip, deflate, br
Authorization: Bearer [redacted]
User-Agent: [redacted]
Accept-Language: en-US,en;q=0.9
Content-Length: 199
AES: Killer
{'PayloadId' : '[redacted]' , 'TrackingId' : '[redacted]' , 'Key' : '[redacted]' , 'Version' :
'[redacted]' , 'DeviceId' : '[redacted]' , 'CreateCookie' : 'True'}
HTTP/2 200 OK
Cache-Control: no-cache,no-cache, no-store, must-revalidate, pre-check=0, post-check=0, max-age=0,
s-maxage=0
Pragma: no-cache,no-cache
Content-Length: 291
Content-Type: text/plain; charset=utf-8
Expires: -1,0
Content-Security-Policy: frame-ancestors 'self'
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
Strict-Transport-Security: max-age=31536000; includeSubDomains, preload
Date: [date]
AES: Killer
```

```
{"CallId":"[redacted]","status_message":"","status_code":0,"cookie_id":"[redacted]","security_stamp
":"[redacted]","session_id": [redacted]}
```

The following HTTP response shows data intercepted and decrypted via an intercepting proxy.

```
HTTP/2 200 OK
Cache-Control: no-cache,no-cache, no-store, must-revalidate, pre-check=0, post-check=0, max-age=0,
s-maxage=0
Pragma: no-cache,no-cache
Content-Length: 291
Content-Type: text/plain; charset=utf-8
Expires: -1,0
Content-Security-Policy: frame-ancestors 'self'
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
Strict-Transport-Security: max-age=31536000; includeSubDomains, preload
Date: [date]
AES: Killer
{"CallId":"[redacted]","status_message":"","status_code":0,"cookie_id":"[redacted]","security_stamp
":"[redacted]","session_id": [redacted]}
```

URL Locations

- URL

Recommendations

WKL recommends implementing certificate pinning. Certificate pinning can be accomplished in several ways. For instance, the fingerprint (Sha256 hash) of a certificate can usually be used to uniquely identify it. It's important to not only validate the authenticity of the certificate but also the fingerprint. This makes certificate pinning an additional step to TLS certificate validation. For more information, please visit the following guide for concepts and implementation:

- https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning

Appendix A: Artifacts

This appendix details the artifacts that may have been generated or utilized during the mobile application penetration testing process. To maintain the security and integrity of the environment, it may be necessary for the following actions to be taken post-assessment.

Test User Accounts:

- **Users:**
  - [test user]
  - [test user]
  - [test user]
- **Action Recommended**: Deactivate and delete the test accounts for WKL.

API environment Tested:

- **Environment:** [environment]
- **Action Recommended**: Verify that no test artifacts remain and that endpoints are secured post-testing.
- **Action Recommended**: Review server logs for API endpoints to confirm no unintentional changes or sensitive data was logged.

By addressing the listed artifacts, the web application's security stance is further solidified. Should any additional artifacts or concerns have been identified during the assessment, please refer to the detailed assessment report for comprehensive guidance and recommended remediation actions.

Appendix B: Risk Profile

During the mobile application assessment, two medium and four low vulnerabilities were identified that could pose a risk to users and [client]'s security. These findings serve to provide valuable insights into the security posture of the application. It is recommended that a further analysis be conducted to determine the severity of the detected vulnerabilities and the potential impact of a compromise. This should include an evaluation of the likelihood of exploitation and the potential repercussions to better inform risk management strategies and remediation prioritization.

Upon completion of the technical segment of the assessment, consultants at White Knight Labs calculated the "Risk Score." The subsequent chart explains how White Knight Labs assigns these Risk Score levels. The definitions are influenced by the Penetration Testing Execution Standards (PTES) Information Security Risk Rating Scale. White Knight Labs employs the industry-standard risk calculation method, multiplying the potential impact by the likelihood associated with each finding, considering various criteria. The scoring is also based on the engineers' professional opinion and the impact of the issues presented.

Rating	Likelihood	Impact
Critical	Almost Certain to Occur: Probability greater than 90%	Severe: Catastrophic financial loss, long-term reputational damage, potential legal consequences, potential loss of life
High	Likely to Occur: Probability between 60% and 90%	Major: Significant financial loss, substantial disruption to operations, potential legal scrutiny
Medium	Possible but Not Likely: Probability between 30% and 60%	Moderate: Noticeable financial loss, temporary disruption to some functions, possible customer dissatisfaction
Low	Unlikely to Occur: Probability less than 30%	Minor: Minimal financial or operational impact, easily recoverable, limited customer or stakeholder concern

Below are descriptions of each vulnerability classification level:

Critical Risk Findings: These represent vulnerabilities that grant remote attackers root or administrator capabilities. With this degree of vulnerability, the entire host could be compromised. Critical risk findings include vulnerabilities that allow remote attackers full read and write access to the file system, as well as the ability to remotely execute commands as a root or administrator user. The existence of backdoors or malicious code also falls under this category.

High-Risk Findings: These vulnerabilities grant attackers limited privileges, not extending to remote administrator or root user access. High-risk findings may enable attackers to partially access file systems, such as having full read access without corresponding write permissions. Any vulnerabilities that reveal sensitive data, like session details or personal information (e.g., PII or credit card data), are also considered High-risk.

Medium Risk Findings: These vulnerabilities allow attackers to access specific information on the host, including security configurations. Such exposures could lead to potential misuse by attackers. Medium risk findings might encompass partial file content disclosure, access to particular host files, directory browsing, exposure of filtering protocols and security measures, susceptibility to DoS attacks, or unauthorized exploitation of system or application functions.

Low Risk Findings: These findings reveal information that could facilitate more targeted attacks. Examples include directory structures, account names, network addresses, or internal data about other systems.

Informational Findings: These do not necessarily constitute vulnerabilities but include information that the application owner should review and analyze. This category highlights details that may not pose an immediate threat but warrant attention for comprehensive security awareness.

By categorizing these findings, White Knight Labs provides an organized and clear assessment of the risk landscape, based on the professional opinions of our engineers and the impact of the identified issues.