PathCheck - GAEN-Mobile App - September 2020 Penetration Test Report

TARGET(S)

""

TEST PERIOD

Sep 29, 2020 → Oct 13, 2020

STATUS

Final

TEST PERFORMED BY

 Jana — Lead

 Arun S — Pentester

Contents

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

Executive Summary

A gray box penetration test of the PathCheck - GAEN-Mobile App mobile application was conducted in order to assess its risk posture and identify security issues that could negatively affect PathCheck's data, systems, or reputation. The scope of the assessment covered PathCheck gaen-mobile External iOS Beta version 1378 and PathCheck gaen-mobile Google Alpha Closed 1.0.3. The pentest was conducted by 2 pentester(s) between Sep 29, 2020 and Oct 13, 2020.

This penetration test was a manual assessment of the security of the application's functionality, business logic, and vulnerabilities such as those catalogued in the OWASP Top 10. The assessment also included a review of security controls and requirements listed in the OWASP Mobile Application Security Verification Standard (MASVS). The pentesters leveraged tools to facilitate their work; however, the majority of the assessment involves manual analysis.

The pentesters identified 0 High, 0 Medium, and 2 Low risk vulnerabilities.

Overall, the pentesters found that the application exhibits a good security posture during the assessment. The application is well built, with many security best practices in place. No major issues were discovered during the test because of this. One of the reported issues involves the lack of SSL Pinning allowing attackers to potentially launch a Man-In-The-Middle attack. The other issue related to the lack of root detection mechanism which could allow an attacker with local access to the deice to steal data or interact with the application in an unintended fashion. Both findings are considered Low severity.

Significant findings include:

- Lack of SSL Pinning
- Lack of Root Detection Mechanism

Specific recommendations are provided for each finding. As a whole, the recommendations indicate gaps that can be addressed by configuring SSL pinning as well as implementing checks which can detect if the mobile application is running on a rooted/ jailbroken device.

Scope of Work

Coverage

This penetration test was a manual assessment of the security of the app's functionality, business logic, and vulnerabilities such as those cataloged in the OWASP Top 10. The assessment also included a review of security controls and requirements listed in the OWASP Mobile Application Security Verification Standard (MASVS). The researchers conduct manual analysis assisted by tools.

The mobile application did not authenticate users beyond the device where it was installed and did not send any personally identifiable information (PII) to the server. Therefore, testing related to security controls across roles and permissions did not apply. However, testing related to data security and privacy has been performed, including testing that data is stored securely on the device as well as handled security over the network.

The following is a brief summary of the main tests performed on the mobile application:

- Checks of native components and the source code
- Exported interface tests for Android applications
- Reverse engineering of the mobile application
- OWASP Mobile Top 10 testing

While the server was out of scope for testing, the following is a brief summary of the review performed on the API to make sure that user data was secure over the network:

- Authenticated endpoint testing for missing access control issues
- Input validation tests

Below is the summary of methodologies used to assess security at mentioned endpoints:

Inventory of API endpoints:

We inventoried the calls made by the app during all user activities. We then proceeded to analyze requests and responses to observe the underlying technology and any possible vulnerabilities. We observed that no PII is being sent to the server.

Manual and automated fuzzing of API endpoints:

We proceeded to reverse engineer the endpoints and perform modified calls using manual and automated methods. We attempted the following:

- Parameter manipulation (adding / modifying parameters to perform new functions, horizontal privilege escalation, etc.)
- Code injection (SQL injection attempts, Template injection, Cross Site Scripting attacks, etc.).
- Conducted testing of data storage for sensitive information disclosure.
- Tested the apps to ensure cryptographic best practices were adhered to.
- Performed analysis of local authentication mechanisms used by the applications.
- Tested network communication security controls such as certificate pinning, the use of TLS for application to API communication, and the security provider that is present on the mobile device. (this was deemed out of scope)
- Conducted an analysis of the different platform’s (iOS and Android)

interaction with the application. These tests focused on the use of intents, IPC, broadcast receivers, services, URL schemes, file transfer mechanisms, etc.
- Worked to identify code quality and build setting deficiencies such as application signing, debug options, symbols that weren't properly stripped, verbose logging, exception handling, and any issues stemming from the use of native libraries.
- Noted the effectiveness of code obfuscation techniques used to limit analysis efforts, and anti-reverse engineering mechanisms such as jailbreak or root detection.

Test Cases that successfully thwarted exploitation

During testing many positive controls were observed the be in place within the application. A brief overview of these includes:

- The JWT token used for verification was secured against common attacks like algorithm manipulation and brute force. The token signature was properly validated.
- The application does not store any sensitive data locally or send sensitive data to the server.
- The verification code is not reusable.
- The application does not expose any sensitive information in logs

Target description

- PathCheck gaen-mobile External iOS Beta version 1378 tested on non-jailbroken iOS 14.0.1, iPhone 11 Pro
- PathCheck gaen-mobile Google Alpha Closed 1.0.3 tested on Android v 10 Pixel 2

Assumptions/Constraints

- Server was out of scope for testing.
- We were not able to test on a jailbroken iOS device and relied heavily on static analysis and code review.

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

Methodology

The test was done according to penetration testing best practices. The flow from start to finish is listed below.

Pre Engagement

- Scoping
- Customer
- Documentation
- Information
- Discovery

Penetration Testing

- Tool assisted assessment
- Manual assessment
- Exploitation
- Risk analysis
- Reporting

Post Engagement

- Prioritized remediation
- Best practice support
- Re-testing

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

Risk Factors

Each finding is assigned two factors to measure its risk. Factors are measured on a scale of 1 (very low) through 5 (very high).

Impact

This indicates the finding's effect on technical and business operations. It covers aspects such as the confidentiality, integrity, and availability of data or systems; and financial or reputational loss.

Likelihood

This indicates the finding's potential for exploitation. It takes into account aspects such as skill level required of an attacker and relative ease of exploitation.

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

Criticality Definitions

Findings are grouped into three criticality levels based on their risk as calculated by their business impact and likelihood of occurrence, `risk = impact * likelihood`. This follows the OWASP Risk Rating Methodology.

High

Vulnerabilities with a high or greater business impact and high or greater likelihood are considered High severity. Risk score minimum 16.

Medium

Vulnerabilities with a medium business impact and likelihood are considered Medium severity. This also includes vulnerabilities that have either very high business impact combined with a low likelihood or have a low business impact combined with a very high likelihood. Risk score between 5 and 15.

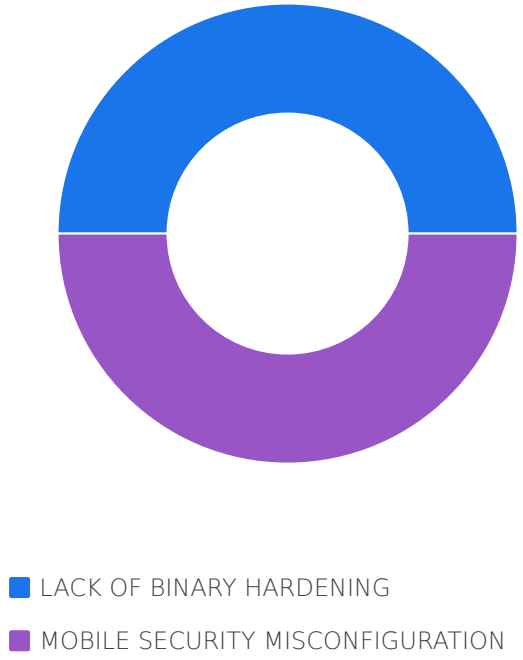
Low

Vulnerabilities that have either a very low business impact, maximum high likelihood, or very low likelihood, maximum high business impact, are considered Low severity. Also, vulnerabilities where both business impact and likelihood are low are considered Low severity. Risk score 1 through 4.

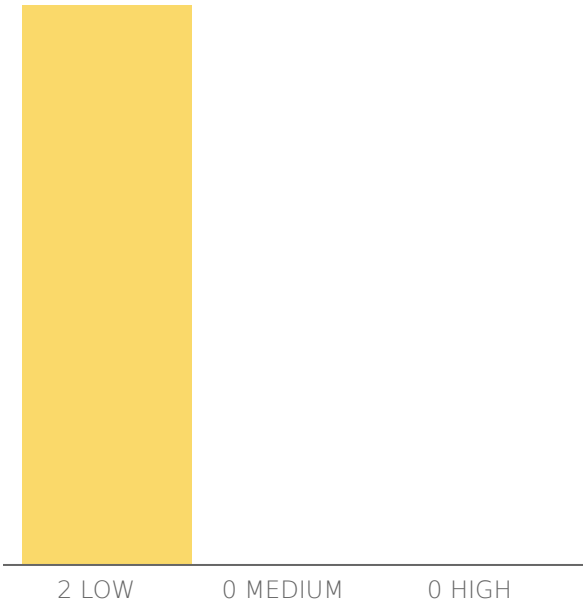
Summary of Findings

The following charts group discovered vulnerabilities by OWASP vulnerability type, and by overall estimated severity.

BY VULNERABILITY TYPE



BY CRITICALITY



PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA • cobalt.io

Analysis

During the test there was a strong focus on the local handling of data, as well as the static analysis leveraging the source code provided for the assessment. In general, the pentesters found the application to be well secured, with some issues related to security configurations and root detection. No critical security issues have been identified during the assessment.
As a result of this test, the following findings were detailed:

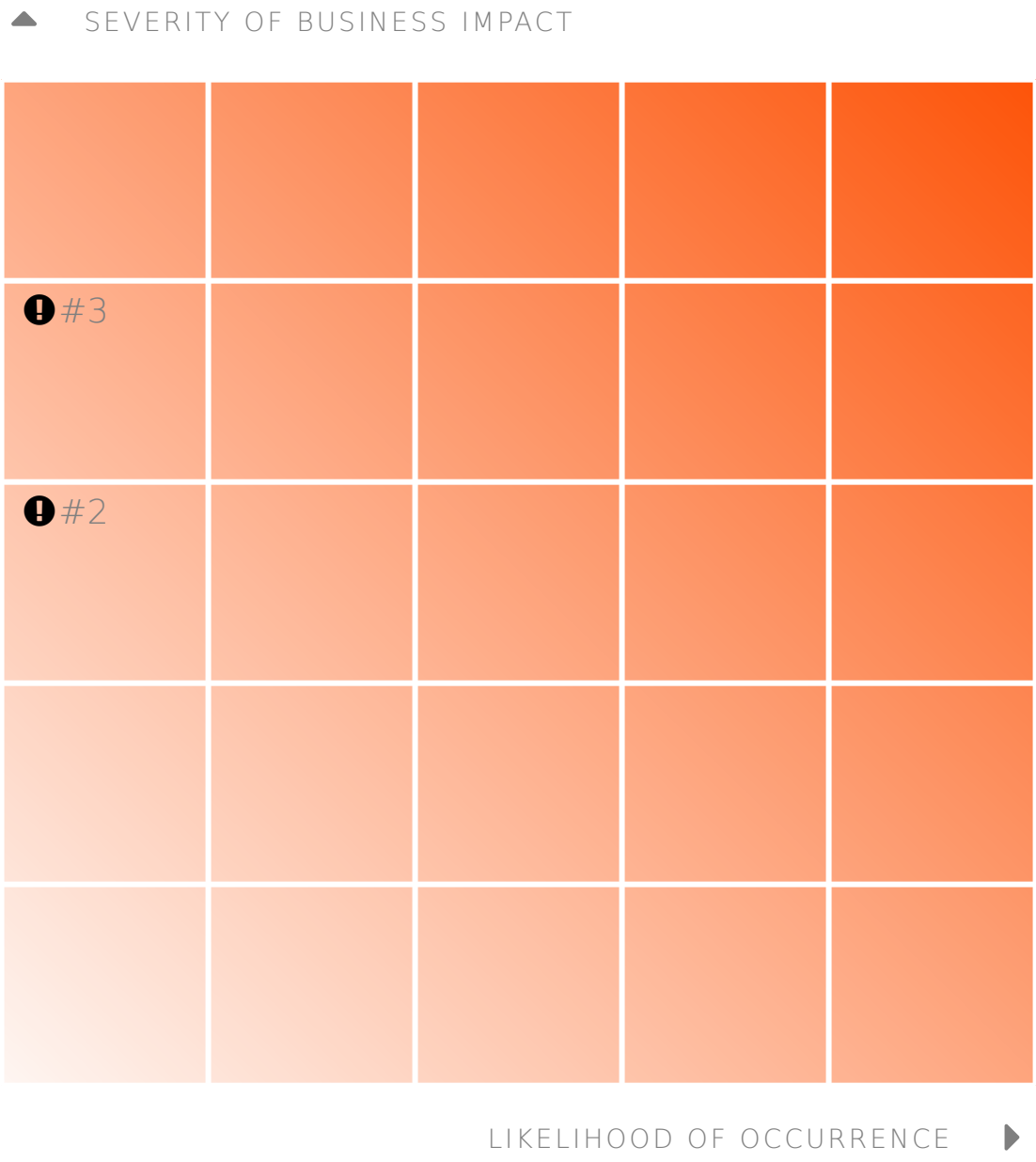
- #PT4556_2 - Mobile application does not implement jailbreak or root detection - A malicious user or application running on a rooted/ jailbroken device can potentially access and modify any data that the application utilizes.
- #PT4556_3 - Mobile application does not implement SSL pinning. This could allow an attacker to man-in-the-middle the connection between the mobile applications and back-end web services.

Open Ports and Services

This has not been reviewed as the server was out of scope of the assessment.

General Risk Profile

The chart below summarizes vulnerabilities according to business impact and likelihood, increasing to the top right.



PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

Recommendations

Based on the findings the following remediation is suggested:

1. Implement certificate pinning on the app (#PT4556_3)
2. Implement root detection and inform the user who runs the application that the device is rooted (#PT4556_2)

Post-Test Remediation

As of the conclusion of this document, the following mitigations have been implemented for the identified vulnerabilities.

FINDING	TYPE	CRITICALITY	STATE	RESOLVED
#PT4556_2	Lack of Binary Hardening	Low	Pending Fix	
#PT4556_3	Mobile Security Misconfiguration	Low	Pending Fix	

Terms

Please note that it is impossible to test networks, information systems and people for every potential security vulnerability. This report does not form a guarantee that your assets are secure from all threats. The tests performed and their resulting issues are only from the point of view of Cobalt Labs. Cobalt Labs is unable to ensure or guarantee that your assets are completely safe from every form of attack. With the ever-changing environment of information technology, tests performed will exclude vulnerabilities in software or systems that are unknown at the time of the penetration test.

Appendix A - Finding Details

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

(Android) Lack of Root Detection Mechanism #PT4556 2

Arun S. (hehacks)

Pending Fix Low · Lack of Binary Hardening · Oct 05, 2020

VULNERABILITY TYPE

Lack of Binary Hardening > Lack of Jailbreak Detection

DESCRIPTION

It was observed that the PathCheck gaen-mobile android application, does not adequately detect if the device is rooted. As a result, the data which is stored via the PathCheck gaen-mobile android application can be exposed when the device is rooted.

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

PROOF OF CONCEPT

To reproduce this issue, follow the steps below:

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

1. Install the PathCheck gaen-mobile application on a rooted Android device.
2. Open the application and observe that the application doesn't prompt for any warnings on rooted device.

CRITICALITY

The risk exists that an attacker with local access to the device might be able to steal data or to interact with the application by using the actual user privileges.
This issue represents a threat with Low severity.

SUGGESTED FIX

To fix this vulnerability is recommended to add a root detection when the application starts. From here two approaches can be followed:

- Inform the user that the device which runs the application is rooted and close the application.
- Inform the user that the device which runs the application is rooted and ask them to accept the risk of running it on a rooted device.

(Android+iOS) Lack of SSL Pinning

#PT4556 3

Arun S.
(hehacks)

Pending Fix Low · Mobile Security Misconfiguration · Oct 08, 2020

VULNERABILITY TYPE

Mobile Security Misconfiguration > SSL Certificate Pinning > Absent

DESCRIPTION

PathCheck gaen-mobile Android & iOS app has not implemented SSL pinning when establishing a trusted connection between mobile applications and back-end web services. Without SSL pinning enforced, an attacker could man-in-the-middle the connection between the mobile applications and back-end web services. This would allow the attacker to sniff user verification code, tokens and so on.

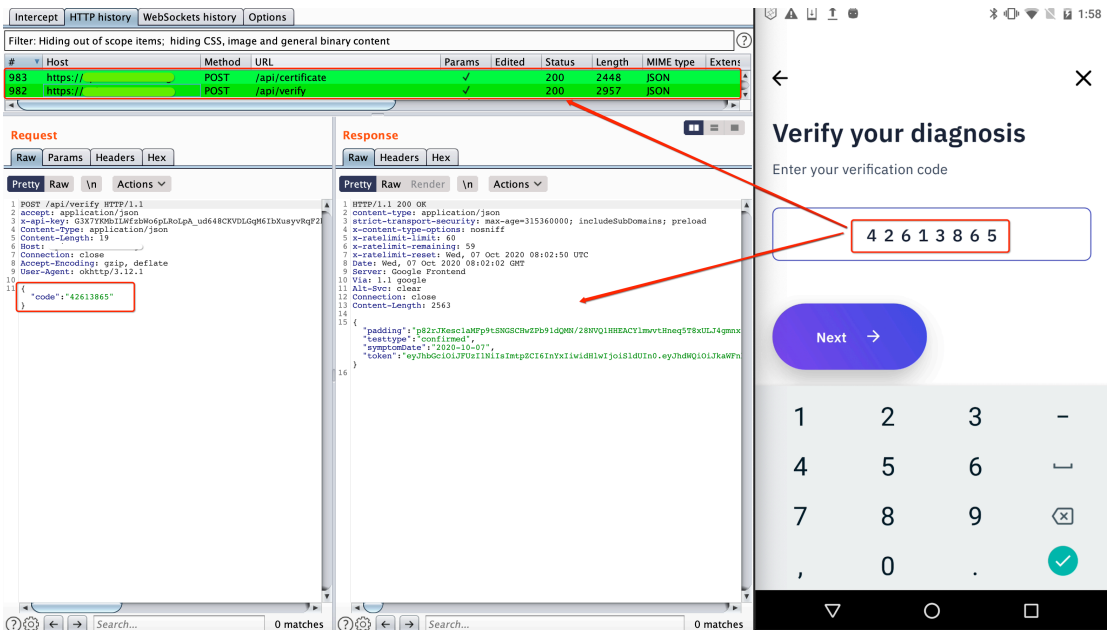
Certificate pinning is the process of associating the backend server with a particular X.509 certificate or public key instead of accepting any certificate signed by a trusted certificate authority. After storing ("pinning") the server certificate or public key, the mobile app will subsequently connect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

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

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

PROOF OF CONCEPT

1. Setup any proxy listener tools like BurpSuite to listen on port 8080 and enabled the proxy option to listen on '**All Interfaces**'.
2. On Android/iOS device, go in WiFi setting and set the host system's IP as a proxy system and install the BURP CA certificate onto the Android/iOS device.
3. Start navigating through the application and enter verification code. Now you can observe the sensitive application traffic in burp suite proxy tool.



PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io

CRITICALITY

Attackers create Fake WiFi Access Point where the victim connects and installed a fake certificate. Later, the attacker can launch MITM to view all HTTPS traffic in plain text including all sensitive data like verification code etc.,
This issue represents a threat with Low severity.

SUGGESTED FIX

1. Establish a HTTP Public Key Pinning (HPKP) policy that is communicated to the client application and/or support HPKP in the client application if applicable.
2. Since Android N, the preferred way for implementing pinning is by leveraging Android's Network Security Configuration feature, which lets apps customize their network security settings in a safe, declarative configuration file without modifying app code.
3. To enable pinning, the **<pin-set>** configuration setting can be used.
4. If devices running a version of Android that is earlier than N need to be supported, a backport of the Network Security Configuration pinning functionality is available via the **TrustKit Android library.**
5. Avoid implementing pinning validation from scratch, as implementation mistakes are extremely likely and usually lead to severe vulnerabilities.
6. The Android documentation provides an example of how SSL validation can be customized within the app's code (in order to implement pinning) in the Unknown CA implementation document.

7. For iOS you can use tools like iOS SecuritySuite or iXGuard to implement SSL pinning and Jailbreak detection protections.

References:

Android:

- https://developer.android.com/training/articles/security-config.html
- https://developer.android.com/training/articles/security-config.html#CertificatePinning
- https://github.com/datatheorem/TrustKit-Android
- https://developer.android.com/training/articles/security-ssl.html#UnknownCa

iOS:

- https://www.nowsecure.com/blog/2017/06/15/certificate-pinning-for-android-and-ios-mobile-man-in-the-middle-attack-prevention/
- https://www.raywenderlich.com/1484288-preventing-man-in-the-middle-attacks-in-ios-with-ssl-pinning.
- https://github.com/securing/IOSSecuritySuite
- https://www.guardsquare.com/en/products/ixguard

PDF created Oct 23, 2020 21:09 UTC
Cobalt Web App 2020
by Cobalt Labs • San Francisco, CA •
cobalt.io