

Mobile Application Penetration Testing Report

Anahat Solutions

Version: 1.1

Issue Date: 06-11-2023



Chembakam, Infopark Koratty, Thrissur, Kerala,
info@cereiv.com | cereiv.com

Document Information

Document Title	Mobile Application Penetration Testing Report
Document Classification	Confidential
Document Type	Audit Report

Version History

Version	Issued Date	Author	Reviewed & Approved By
V1.1	06-Nov-2023	Vishnu AR	Renjith TC

Disclosure Statement:

This document contains sensitive information about the computer security environment, practices, current vulnerabilities, and weaknesses for the client security infrastructure as well as proprietary tools and methodologies from CEREIV. The reproduction or distribution of this document must be approved by the client or CEREIV. This document is subject to the terms and conditions of a non-disclosure agreement between CEREIV and the Client.

Disclaimer And Limitations

Any outcome of the services performed is limited to a point-in-time examination of the environments tested. CEREIV does not constitute any form of representation, warranty, or guarantee that the systems are 100% secure from every form of attack. While CEREIV's methodology includes both automated and manual testing to identify and attempt exploitation of the most common security issues, testing was limited to an agreed-upon timeframe. It is possible that not every vulnerability identified by our scanning platform was tested during this engagement.

Denial of service issues

- CEREIV did not test vulnerabilities that would intentionally lead to denial-of-service issues, to prevent operational disruptions to the Client environment.

Social Engineering

- Social Engineering attacks were not in scope for this assessment.

Client-Side Attacks

- Client-side attacks were not in scope for this assessment.

Industry Term Definitions

Several information security terms and acronyms are in use throughout this document. A glossary of terms can be found at the end of this document and can be used as a reference.

Table Of Contents

1	EXECUTIVE SUMMARY	6
1.1	PROJECT OVERVIEW	6
1.2	SUMMARY OF SCOPE & ATTACK SCENARIOS	6
1.3	CONCLUSION	7
1.4	GRAPHICAL REPRESENTATION.....	7
1.5	SUMMARY OF FINDINGS	8
1.5.1	Mobile Application Penetration Test – Summary of Findings	8
	Lack of Rate Limiting.....	8
1.6	RECOMMENDATIONS	9
2	OBJECTIVES & SCOPE	10
2.1	OBJECTIVES	10
2.2	SCOPE.....	10
3	APPLICATION PENETRATION TEST DETAILS	11
3.1	METHODOLOGY - APPLICATION PENETRATION TEST	11
3.2	REFERENCES.....	13
3.3	OWASP TOP 10 SUMMARY.....	14
4	DETAILED FINDINGS	15
4.1	CRITICAL RISK VULNERABILITIES.....	15
4.2	HIGH RISK VULNERABILITIES.....	16
4.3	MEDIUM RISK VULNERABILITIES	17
4.3.1	Insecure Heap Memory Management.....	17
4.3.2	Insecure Data Storage.....	21
4.4	LOW RISK VULNERABILITIES	23
4.4.1	Lack of Rate Limiting	23
4.4.2	Root Detection Bypass.....	31
4.4.3	SSL Pinning Bypass	36
4.4.4	Improper Input Validation	40
4.4.5	Lack of Code Obfuscation	43
4.4.6	Improper Error Handling.....	45
4.4.7	Use of Insecure Block Cipher Mode.....	48

4.5	INFO RISK VULNERABILITIES	50
4.5.1	Copy/Paste Buffer Caching	50
4.5.2	Insufficient Cryptography	54
5	PENETRATION TESTING PROCESS	56
5.1	APPROACH	56
5.2	PENETRATION TESTING METHODOLOGY CHART	56
5.3	TOOLS.....	57
5.4	RISK RATING SCALE	58
6	GLOSSARY OF TERMS	59
7	APPENDIX A	62
8	APPENDIX B	62
9	APPENDIX C	62
10	APPENDIX D.....	68
11	APPENDIX E	73

1 EXECUTIVE SUMMARY

1.1 PROJECT OVERVIEW

CEREIV was engaged during the period of 30/10/2023 through 06/11/2023 to perform security testing for the Anahat Solutions. The security testing included penetration tests against the defined client environment to proactively discover flaws, weaknesses and vulnerabilities. Testing for this project was done in accordance with Information Security Best Practices. The objective of this service was to identify and safely exploit vulnerabilities that could lead to critical infrastructure service interruption, destruction of facilities or compromise of sensitive systems and data. By providing details on successful attack scenarios and specific remediation guidance, CEREIV intends to help Anahat Solutions protect its business-critical networks and data.

New attack techniques are developed on regular basis that would affect the security of all applications. Regular security assessments need to be carried out in order to cover such new attacks and vulnerabilities.

1.2 SUMMARY OF SCOPE & ATTACK SCENARIOS

The scope of the penetration testing includes a mobile application attack scenario.

- The Mobile application penetration test included one (1) iOS application And one (1) Android application. Credentials were provided to CEREIV by Anahat Solutions.
- The penetration test simulated an attack by an external attacker, and application testing was performed with and without the use of provided credentials.

1.3 CONCLUSION

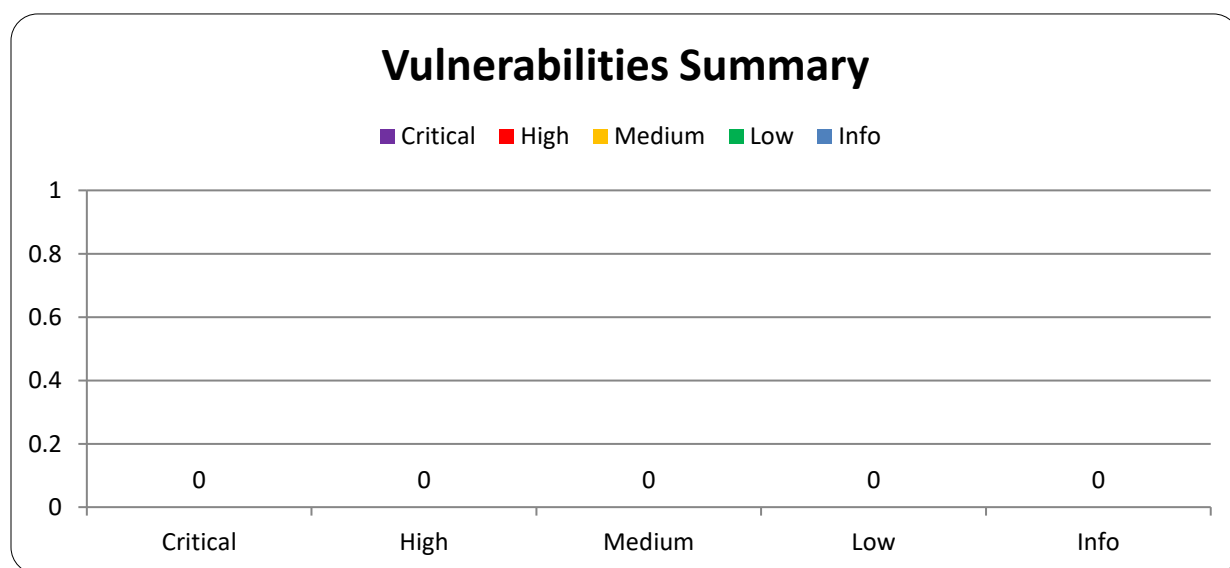
The overall risk of attack is **Low**. The overall risk indicates the level of remediation required for the issues that exist in the environment. These issues can impact the overall security posture and put Anahat Solutions assets at risk of compromise. The table in section 1.5 contains individual risk rankings for each finding, that comprises the overall risk ranking.

During the assessment a total of 0 vulnerabilities were observed. The distribution of vulnerabilities among various risk levels and categories is mentioned in detail as below.

The assessment revealed areas where applications need improvement. The detailed description and mitigation of observed vulnerabilities along with applicable proof of concept is mentioned in Section 4 of this report.

1.4 GRAPHICAL REPRESENTATION

The following graph summarizes the distribution of the risks identified by vulnerability rating.



1.5 SUMMARY OF FINDINGS

The table below summarizes the risk rankings for each of the attack scenarios tested as part of this project. In addition, general recommendations have been provided to assist in remediation activities.

1.5.1 Mobile Application Penetration Test – Summary of Findings

#	Vulnerability	Severity	Status	Synopsis
1.	Insecure Heap Memory Management	Medium	Closed	All variables stored by the application in unencrypted memory can potentially be retrieved by an unauthorized user, with privileged access to the device.
2.	Insecure Data Storage	Medium	Closed	The application stores sensitive information in the local storage without any encryption.
3.	Lack of Rate Limiting	Low	Closed	The application does not prevent frequency of requests to the server.
4.	Root Detection Bypass	Low	Closed	The root detection implemented on the application can be bypassed.
5.	SSL Pinning Bypass	Low	Closed	The SSL pinning implemented on the application can be bypassed.
6.	Improper Input Validation	Low	Closed	Application fails to validate user supplied data.
7.	Lack of Code Obfuscation	Low	Closed	The application lacks proper code obfuscation.
8.	Improper Error Handling	Low	Closed	Application discloses internal path information through error message.
9.	Use of Insecure Block Cipher Mode	Low	Closed	Cipher Block Chaining mode (CBC) is vulnerable to multiple attack types.
10.	Copy/Paste Buffer Caching	Info	Closed	The application allows the device to store pii such as username in the clipboard.
11.	Insufficient Cryptography	Info	Closed	The application uses insecure and/or

				deprecated cryptographic configurations.
--	--	--	--	--

1.6 RECOMMENDATIONS

Based upon the findings summarized above (and detailed later in this report), CEREIV provides the following remediation guidance:

- **Critical Level Recommendations**
No Critical level vulnerabilities were identified.
- **High Level Recommendations**
No High level vulnerabilities were identified.
- **Medium Level Recommendations**
No Medium level vulnerabilities were identified.
- **Low Level Recommendations**
No Low level vulnerabilities were identified.
- **Info Level Recommendations**
No Info level vulnerabilities were identified.

It is CEREIV's opinion that taking the corrective actions outlined in this report to remediate the identified vulnerabilities will improve Anahat Solutions overall information security program and will assist in mitigating risk to the confidentiality, integrity, and availability of Anahat Solutions data and network resources.

2 OBJECTIVES & SCOPE

2.1 OBJECTIVES

CEREIV conducted an application penetration test for Anahat Solutions, in accordance with Information Security Best Practices. The objective of this service was to attempt compromise of systems, applications, or data within the tested environment.

2.2 SCOPE

The following table provides a synopsis of target systems that were within the scope of this engagement.

Health-e		
iOS App	Package Name	com.health.e.app
	Version (Build)	5.0
Android App	Package Name	com.health.e.app
	Version (Build)	3.0.0
User Accounts Tested	1. 9745037780	
Exposure	Internet	

3 APPLICATION PENETRATION TEST DETAILS

In this section, CEREIV provides detailed information for the Mobile application penetration test that was completed for Anahat Solutions. Included are references to external documents which may contain raw scan data, exploit attack narratives (if appropriate) and details regarding each finding.

3.1 METHODOLOGY - APPLICATION PENETRATION TEST

CEREIV follows a penetration testing methodology that aligns with industry best practice. With a combination of comprehensive threat modelling and practical tests, we identify potential security threats that may affect data confidentiality, data integrity and the availability of the application. The mobile application security assessment is split into four categories: Device, Network, Backend, and Reverse Engineering.

Device

Analysts install the application on real devices or emulators with the target OS (iOS or Android) and conduct an analysis of the device for specific application or data storage vulnerabilities. The test cases for device testing are as follows:

- Identifying sensitive data stored on device (in plaintext or reversible hashing/encoding).
- Investigating Keychain/Keystore artifacts.
- Database structure and content evaluation.
- Biometric authentication.
- Sensitive data retained in memory.
- Data encryption.

Network

Analysts operate all aspects of the application as a user would and attempt to detect vulnerabilities via network communications. The test cases for network testing are as follows:

- Identifying sensitive data sent over the network (in plaintext or reversible hashing/encoding).
- Evaluating login/logout process.
- Evaluating session management techniques.
- Evaluating the security of the certificate exchange for HTTPS communications.
- Evaluation of Multi-factor authentication implementation.

Backend

Analysts conduct reconnaissance and limited exploitation of backend services that the mobile application interacts with. The test cases for backend testing are as follows:

- Evaluating server cipher negotiation strength.
- Evaluating API authorization and rate limiting implementation.
- Evaluating session management techniques.
- Investigating user/token discovery and enumeration efficacy

Reverse Engineering

Analysts take the role of an attacker with limited knowledge of the application and attempt to reverse engineer the application to discover how the application works, determine if any sensitive data can be found in reversed binary source code, and attempt to manipulate the application. The test cases for reverse engineering are as follows:

- Source code obfuscation techniques.
- Anti-debug/anti-tamper techniques.
- Investigation of hard-coded secrets or other sensitive information.
- Check for outdated/vulnerable third-party libraries.
- Evaluate weak cryptography implementations.
- Biometric authentication bypass techniques.
- Certificate pinning bypass techniques.

CEREIV attempted to exploit each vulnerability discovered during the test and penetration test report was prepared by including details of identified vulnerabilities.

3.2 REFERENCES

As part of the penetration testing process, CEREIV used one or more automated scanning tools. The raw output from these tools is included as one or more appendices to this report. For this project, the following external reference documents contain the raw vulnerability findings and detailed remediation guidance for each:

Appendix A – Excel Tracker

- Anahat Solutions_Mobile_App_VAPT_Report_06112023_V1.1.xlsx

Appendix B – Raw Scan Reports

- Raw_Scan_Reports

Note: The above external documents should be used for reference and remediation information only as it may contain false-positive findings not vetted by CEREIV. All vetted and confirmed

vulnerability information is fully described in this report.

3.3 OWASP TOP 10 SUMMARY

CEREIV focuses on the OWASP Top 10 list for mobile application vulnerabilities. The table below summarizes the overall findings for each type of vulnerability.

OWASP TOP 10	
Category	Discovered
M1: Improper Platform Usage	NO
M2: Insecure Data Storage	NO
M3: Insecure Communication	NO
M4: Insecure Authentication	NO
M5: Insufficient Cryptography	NO
M6: Insecure Authorization	NO
M7: Client Code Quality	NO
M8: Code Tampering	NO
M9: Reverse Engineering	NO
M10: Extraneous Functionality	NO

4 DETAILED FINDINGS

This section of the report provides details regarding vulnerabilities that were exploited during the project timeframe and vulnerabilities that was not exploitable but represent significant threats to the environment. Additionally, an attack narrative with step-by-step screenshots will provide evidence of our attacks and help the client to recreate our findings. Remediation guidance is also provided, in line with our findings.

4.1 CRITICAL RISK VULNERABILITIES

No Critical level vulnerabilities were identified.

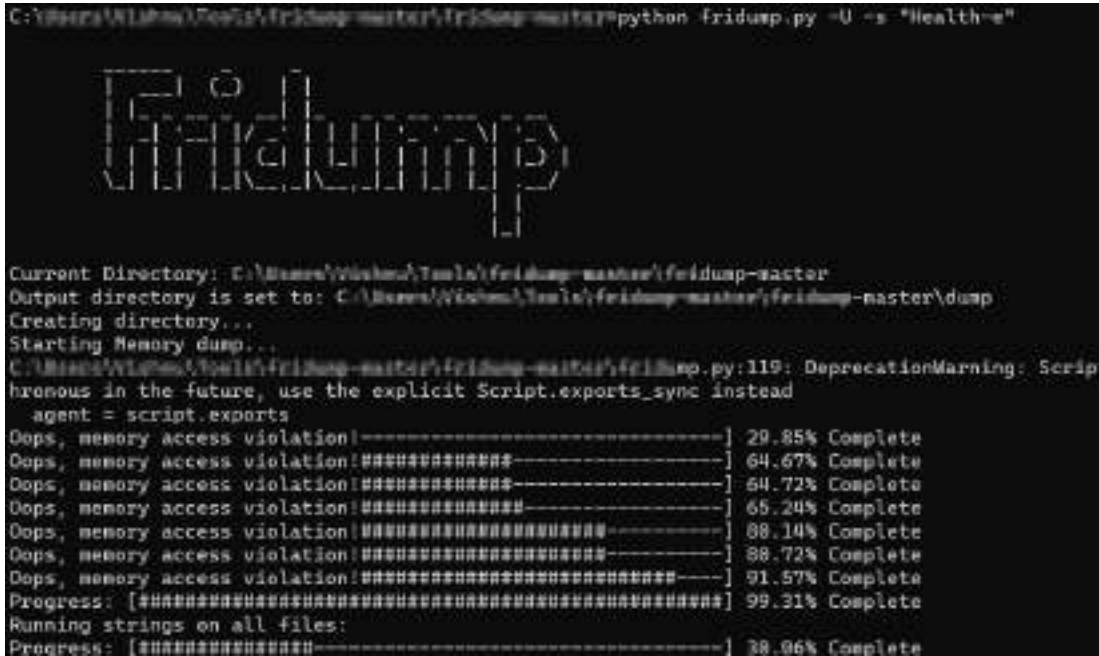
4.2 HIGH RISK VULNERABILITIES

No High level vulnerabilities were identified.

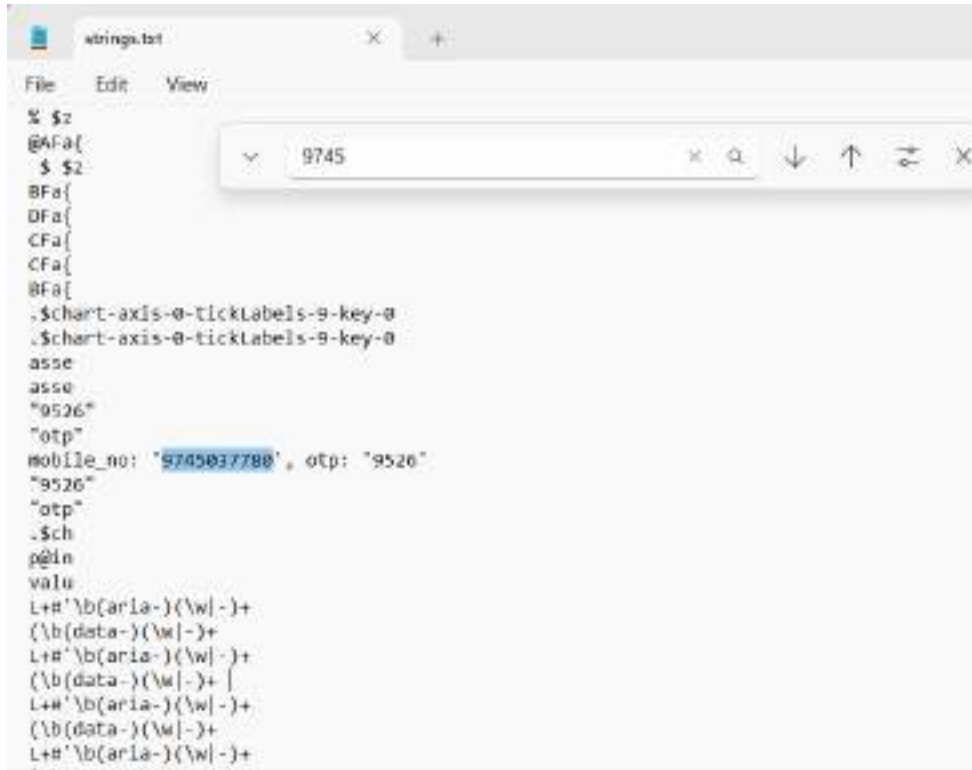
4.3 MEDIUM RISK VULNERABILITIES

4.3.1 Insecure Heap Memory Management

Severity	Medium	Status	Closed
OWASP Category	M2: Insecure Data Storage	CVSS	5.4
Assets Affected			
<ul style="list-style-type: none"> Android Application 			
Synopsis			
All variables stored by the application in unencrypted memory can potentially be retrieved by an unauthorized user, with privileged access to the device.			
Description			
Any data that is required by the application needs to be in memory at some point in time. However, the recommendation is to keep that data in memory only for the time strictly necessary and, when no longer needed, to remove it out of the memory. For applications handling highly sensitive information this is important because an attacker may be able to access the application's memory contents via a memory dump, a crash dump or through physical access and, if the data is still in memory, then it may be compromised.			
Recommendations			
<ul style="list-style-type: none"> Do not store sensitive data, such as passwords or encryption keys, in memory in plain-text, even for a short period of time. Prefer to use specialized classes that store encrypted data in memory to ensure it cannot be trivially retrieved from memory. When required to use sensitive data in its raw form, temporarily store it in mutable data types, such as byte arrays, to reduce readability from memory, and then promptly zeroize the memory locations, to reduce exposure duration of this data while in memory. In Java, do not store passwords in immutable strings - prefer using an encrypted memory object, such as SealedObject. 			

Tools Used
<ul style="list-style-type: none"> Frida
References
<ul style="list-style-type: none"> https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage
Testing steps and Evidence
<p><u>Android Application Proof of Concept:</u></p> <p>1: Run the Fridump script (https://github.com/Nightbringer21/fridump) for dumping data stored by the app in the heap memory.</p> <p>Command: <code>python fridump.py -U -s "Health-e"</code></p>  <p style="text-align: center;">Fridump script dumping memory</p> <p>2: By executing the previous step, the command will create a folder in the current directory called dump and a file named 'strings.txt', which contains all the memory strings.</p> <p>Open the 'strings.txt' file and it is observed that user-sensitive data are not cleared or</p>

replaced from memory.

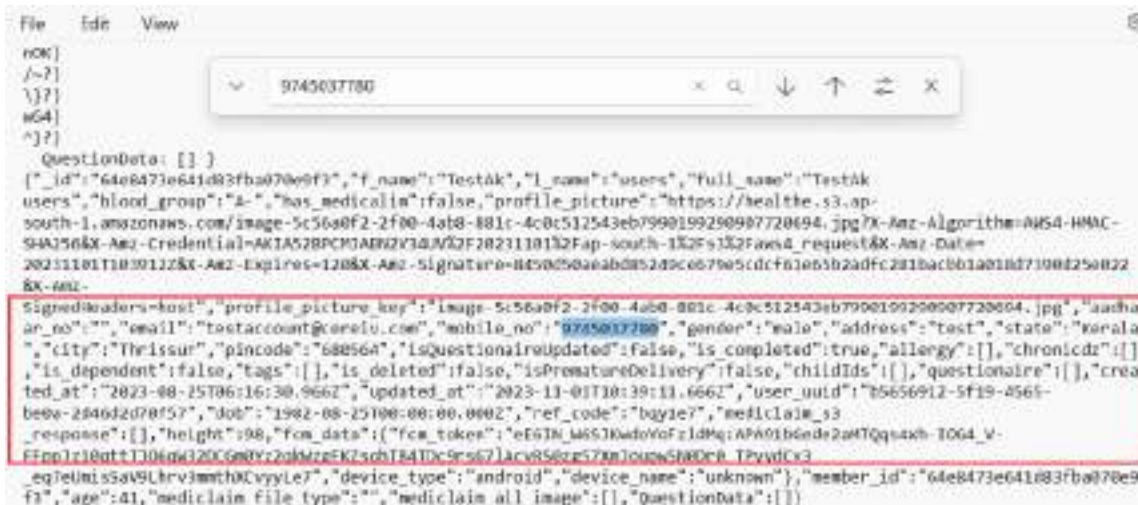


```

File Edit View
% $2
@AFa{
$ $2
BFa{
DFa{
CFa{
CFa{
BFa{
.$chart-axis-0-tickLabels-9-key-0
.$chart-axis-0-tickLabels-9-key-0
asse
asse
"9526"
"otp"
mobile_no: "9745037780", otp: "9526"
"9526"
"otp"
.$sch
p@in
valu
L+#'\b(aria-)(\w|-)+
(\b(data-)(\w|-)+
L+#'\b(aria-)(\w|-)+
(\b(data-)(\w|-)+ |
L+#'\b(aria-)(\w|-)+
(\b(data-)(\w|-)+
L+#'\b(aria-)(\w|-)+

```

User sensitive data found in memory dump.



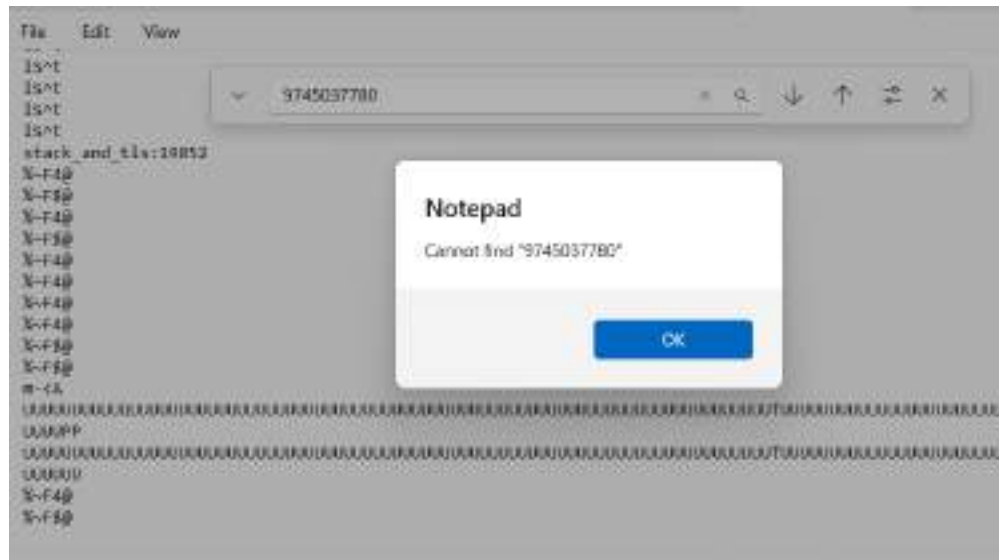
```

File Edit View
nOK}
/~?}
\}?)
mG4}
^?)
QuestionData: [] }
{"_id":"64e8473e641e83fba070e9f3","f_name":"TestAk","l_name":"users","full_name":"TestAk
users","blood_group":"A-","has_medicalim":false,"profile_picture":"https://healthe.s3.ap-
south-1.amazonaws.com/image-5c56a0f2-2f00-4ab8-881c-4c0c512543eb7990199290007720094.jpg?X-Amz-Algorithm=AWS4-HMAC-
SHA256&X-Amz-Credential=AKIA528PCMJA862V3QV32F2023110132Fap-south-1%2F%32Faws4_request&X-Amz-Date=
20231101T030112&X-Amz-Expires=120&X-Amz-Signature=8450050a0ab0852d9ce079e5cdc61065b2adfc281bacb01a01bd71908f250822
&X-Amz-
SignedHeaders=host","profile_picture_key":"image-5c56a0f2-2f00-4ab8-881c-4c0c512543eb7990199290007720094.jpg","aasha
ar_no":"","email":"testaccount@cereiv.com","mobile_no":"9745037780","gender":"male","address":"test","state":"kerala
","city":"Thiruvananthapuram","pincode":"688564","isQuestionnaireUpdated":false,"is_completed":true,"allergy":[],"chronicdr":[],
"is_dependent":false,"tags":[],"is_deleted":false,"isPrematureDelivery":false,"childIds":[],"questionnaire":[],"crea
ted_at":"2023-08-25T06:16:30.966Z","updated_at":"2023-11-01T10:39:11.666Z","user_uid":"b5656912-5f19-4565-
be0e-2846d2d70f57","dot":"1982-08-25T00:00:00.000Z","ref_code":"bgyie7","mediclaim_s3
_response":[],"height":108,"fcm_data":{"fcm_token":"eE6IN_M86JKwdoVofzIdMq:APA01b6nde2aMTQquaxh-1064_V-
FFnplz10ott1106qW320C6m0Y:2dkWzEzEK2sobT84Tdc9s67lacv856zE57XnTousc5H8Dre-TPyvdCv3
eqJeUm5sAV9Lhrv3mth0CvyyLe7","device_type":"android","device_name":"unknown"},"member_id":"64e8473e641e83fba070e9
f3","age":41,"mediclaim_file_type":"","mediclaim_all_image":[],"QuestionData":[]}

```

User sensitive data found in memory dump.

Revalidation Test Evidence



User sensitive data not found in memory dump.

4.3.2 Insecure Data Storage

Severity	Medium	Status	Closed
OWASP Category	M2: Insecure Data Storage	CVSS	5.2
Resource Affected / URL /Parameters			
<ul style="list-style-type: none"> Android application 			
Synopsis			
The application stores sensitive information in the local storage without any encryption.			
Description			
The application is storing sensitive information in device local storage. If an attacker physically attains the mobile device, the attacker can extract the app's sensitive information via mobile malware, modified apps, or forensic tools. Attacker can hook the application to tools that allow the attacker to see all third-party application directories that often contain stored personally identifiable information (PII) or other sensitive information assets.			
Recommendations			
<ul style="list-style-type: none"> Encrypt all sensitive data stored in the local storage. When required to use sensitive data in its raw form, temporarily store it in mutable data types, such as byte arrays, to reduce readability from memory, and then promptly zeroize the memory locations, to reduce exposure duration of this data while in memory. Prefer to use specialized classes that store encrypted data in memory to ensure it cannot be trivially retrieved from memory. 			
Tools Used			

- Not applicable

References

- <https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#checking-memory-for-sensitive-data-mstg-storage-10>

Testing steps and Evidence

- 1) Use the command to get into the app.
- 2) Access the root directory of the rooted Android phone, navigate to `/data/data/com.healthe.app/databases`, and then execute `'cat clevertap'`.

It is seen that the application stores sensitive data in the cache folder.

[illegible]

Insecure Data Storage

Revalidation Test Evidence

It is seen that the application does not stores sensitive data in the cache folder.

```

$H41/data/data/com.health.app/databases # ls
com.amplitude.api          com.google.android.datatransport.events      google_app_measurement_local.db
com.amplitude.api-journal  com.google.android.datatransport.events-journal  google_app_measurement_local.db-journal
$H41/data/data/com.health.app/databases #

```

No Insecure Data Storage

4.4 LOW RISK VULNERABILITIES

4.4.1 Lack of Rate Limiting

Severity	Low	Status	Closed
OWASP Category	A04:2021-Insecure Design	CVSS Score	3.5
Resource Affected / URL /Parameters			
<ul style="list-style-type: none"> https://staging2-api.health-e.in/users/v1/health_record/add [Application Wide] 			
Synopsis			
The application does not prevent frequency of requests to the server.			
Description			
Rate limiting refers to preventing the frequency of an operation from exceeding some constraint. In large-scale systems, rate limiting is commonly used to protect underlying services and resources. Lack of rate limiting allows an attacker to send multiple requests to the backend server.			
Recommendations			
<ul style="list-style-type: none"> Implement rate limiting to all POST request Implement CAPTCHA mechanism 			
Tools Used			
<ul style="list-style-type: none"> Burp Suite Browser 			
References			

- <https://www.wallarm.com/what/lack-of-resources-rate-limiting>
- <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa4-lack-of-resources-and-rate-limiting.md>
- <https://cloud.google.com/architecture/rate-limiting-strategies-techniques>

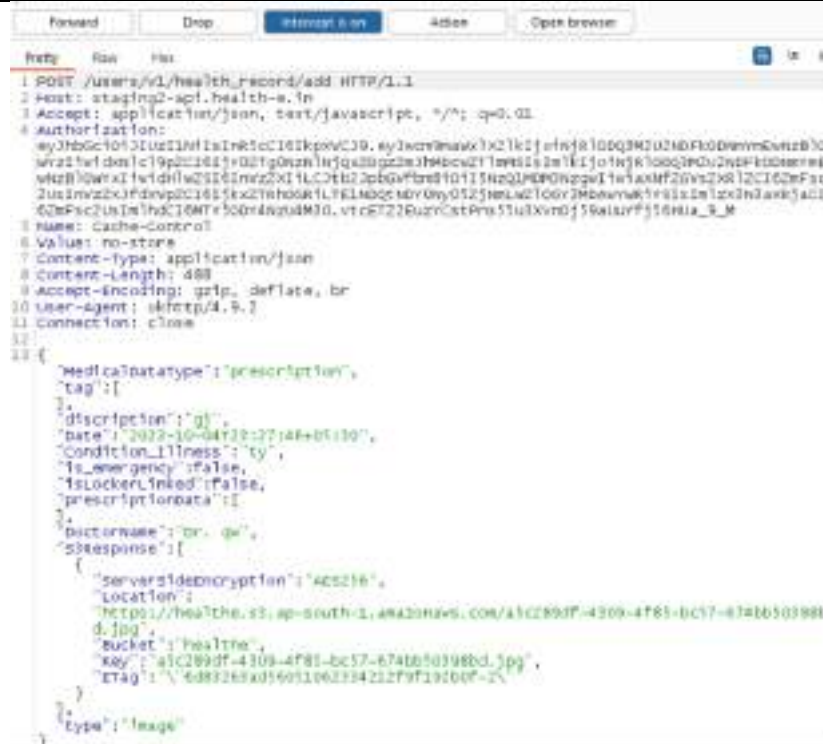
Testing steps and Evidence

1: Navigate to upload prescription section then fill the form shown in below screenshot.



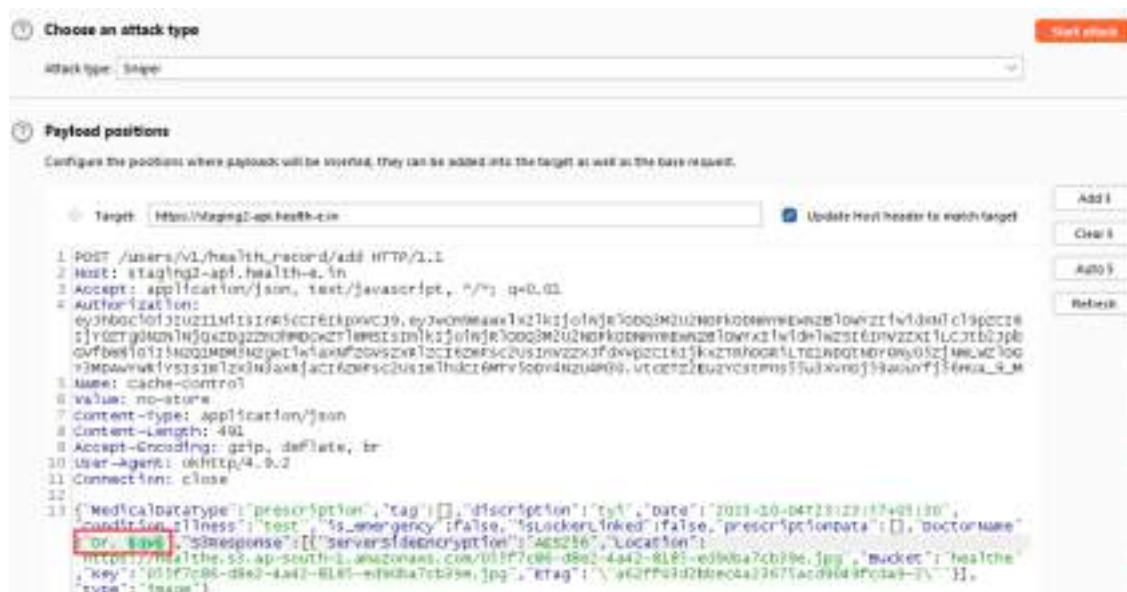
Upload prescription

2: intercept the above request



Upload prescription (Request in Burp View)

2: Forward the captured request to intruder and add payload position on “DoctorName” (Or any other parameter)



Payload Configuration

3: Configure payload sets and payload options as shown below and start the attack.

Positions
Payloads
Resource pool
Settings

Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type and each payload type can be customized in different ways.

Payload set: 1
Payload count: 56

Payload type: Numbers
Request count: 56

Payload settings [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: ☒ Sequential ☐ Random

From: 00

To: 55

Step: 1

How many:

Payload Configuration

Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
40						
52	51	200			1137	
53	52	200			1137	
54	53	200			1137	
55	54	200			1137	
56	55	200			1137	

Request
Response

Pretty
Raw
Hex
Binary

```

1 Content-Type: application/json; charset=utf-8
2 Content-Length: 99
3 Connection: keep-alive
4 Access-Control-Allow-Origin: *
5 Content-Security-Policy: default-src 'self';base-uri 'self';font-src 'self' https: data:;form-act'
  'self' data:;object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self' https:
6 Cross-Origin-Embedder-Policy: require-corp
7 Cross-Origin-Opener-Policy: same-origin
8 Cross-Origin-Resource-Policy: same-origin
9 Origin-Agent-Cluster: ?1
10 Referrer-Policy: no-referrer
11 Strict-Transport-Security: max-age=31552000; includeSubdomains
12 X-Content-Type-Options: nosniff
13 X-DNS-Prefetch-Control: off
14 X-Download-Options: noopen
15 X-Frame-Options: SAMEORIGIN
16 X-Permitted-Cross-Domain-Policies: none
17 X-XSS-Protection: 0
18 ETag: W/"63-07HaBmkggw0Sxd0ET93RfPsfw"
19 X-Frame-Options: SAMEORIGIN
20 X-Content-Type-Options: nosniff
21 X-XSS-Protection: 1; mode=block
22
23 {
24   "success":true,
25   "data":{
26     "id":"637faw258be0287caa4b11?"
27   },
28   "message":"Your document has been saved!"
29 }

```

Successfully saved document.

3: Refresh the application



Successfully saved document.

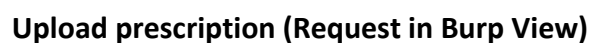
Note: This vulnerability occurred because of a lack of rate limiting, and it is present throughout the application.

Revalidation Test Evidence

1: Navigate to upload prescription section then fill the form shown in below screenshot.



2: intercept the above request



2: Forward the captured request to intruder and add payload position on "DoctorName" (Or any other parameter)

[illegible]

Payload Configuration

3: Configure payload sets and payload options as shown below and start the attack.

Positions

Payloads

Resource pool

Settings

?

Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the different ways.

Payload set:

1

Payload count: 69

Payload type:

Numbers

Request count: 69

?

Payload settings [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type:

☒ Sequential
 ☐ Random

From:

222

To:

290

Step:

1

How many:

Number format

Base:

☒ Decimal
 ☐ Hex
 ☐ Octal
 ☐ Binary

Payload Configuration

V Files Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
86	288	429			1005	
87	288	429			1005	
88	287	429			1005	
89	288	429			1005	
90	289	429			1005	
91	290	429			1005	

Request

Response

Reply

Raw

Hex

Binary

```


1 HTTP/1.1 429 Too Many Requests
2 server: nginx
3 Date: Mon, 06 Nov 2023 06:21:00 GMT
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 114
6 Connection: keep-alive
7 Access-Control-Allow-Origin: *
8 Content-Security-Policy: default-src 'self';base-uri 'self';font-src 'self' https: data:image/svg+xml 'self';object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self' https: 'unsafe-inline'
9 Cross-Origin-Embedder-Policy: require-corp
10 Cross-Origin-Opener-Policy: same-origin
11 Cross-Origin-Resource-Policy: same-origin
12 Origin-Agent-Cluster: ?1
13 Referrer-Policy: no-referrer
14 Strict-Transport-Security: max-age=15552000; IncludeSubdomains
15 X-Content-Type-Options: nosniff
16 X-DNS-Prefetch-Control: off
17 X-Download-Options: noopen
18 X-Frame-Options: SAMEORIGIN
19 X-Permitted-Cross-Domain-Policies: none
20 X-XSS-Protection: 0
21 ETag: W/"72-6ghvefubkb30p+qPx1uq3icw7w"
22
23 {
24   "message": "You have uploaded too many record in day so your account is freeze for 1 day",
25   "data": [],
26   "status": false
27 }

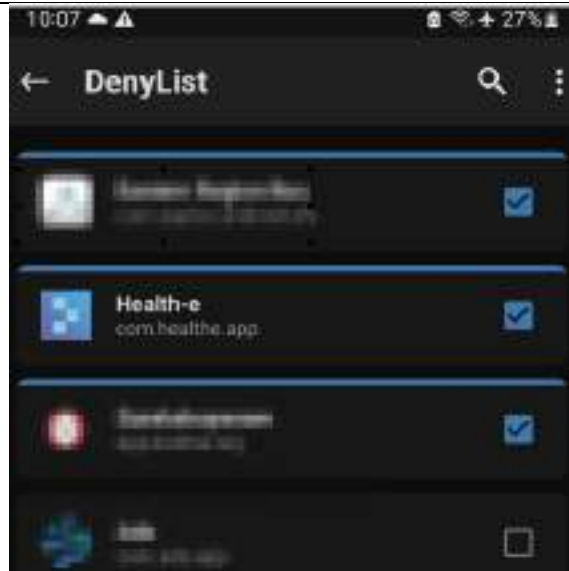
```

Rate Limit

4.4.2 Root Detection Bypass

Severity	Low	Status	Closed
OWASP Category	M7: Client Code Quality	CVSS	2.1
Resource Affected / URL /Parameters			
<ul style="list-style-type: none"> Android application 			
Synopsis			
The root detection implemented on the application can be bypassed.			
Description			
<p>Rooting is the process of allowing users of the Android mobile operating system to attain privileged control (known as root access) over various Android subsystems. It gives the ability to alter or replace system applications and settings, run specialized applications that require administrator-level permissions or perform other operations that are otherwise inaccessible to a normal Android user.</p> <p>Rooting or jailbreaking a device impacts the security of applications in two ways:</p> <ul style="list-style-type: none"> It could allow malicious applications or attackers to perform actions as a root user which compromises the security of other applications running on the phone. Attackers can perform static and dynamic analysis of an application which helps find more vulnerabilities. 			
Recommendations			
<ul style="list-style-type: none"> Implement multiple root detection features within the Android application code and prevent its bypass. Refer Appendix E – Root Detection Methods. 			

Tools Used
<ul style="list-style-type: none">• Magisk
References
<ul style="list-style-type: none">• https://owasp.org/www-project-mobile-top-10/2016-risks/m7-client-code-quality• https://www.indusface.com/learning/how-to-implement-root-detection-in-android-applications/• https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/jailbreak-detection-methods/
Testing steps and Evidence
<p>1: Open the build on a rooted android device and notice that access to the application is denied.</p>  <p>Application prevents the usage on a rooted device</p> <p>2: Enable Denylist for Health-e in Magisk</p>



Enable Denylist



Root detection bypassed

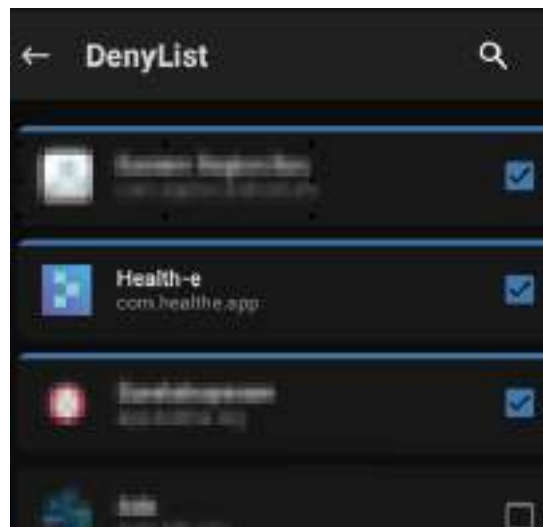
Revalidation Test Evidence

1: Open the build on a rooted android device and notice that access to the application is denied.



Application prevents the usage on a rooted device

2: Enable Denylist for Health-e in Magisk



Enable Denylist



Application prevents the usage on a rooted device.

4.4.3 SSL Pinning Bypass

Severity	Low	Status	Closed
OWASP Category	M3: Insecure Communication	CVSS	2.7
Assets Affected			
<ul style="list-style-type: none"> Android application 			
Synopsis			
The SSL pinning implemented on the application can be bypassed.			
Description			
<p>SSL certificate pinning prevents attackers from intercepting the data transmitted between the application and server. Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key. If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the host name. When a certificate is invalid or malicious, it might allow an attacker to spoof a trusted entity by interfering in the communication path between the host and client. The application might connect to a malicious host while believing it is a trusted host, or the software might be deceived into accepting spoofed data that appears to originate from a trusted host. Different methods for implementing SSL pinning in mobile apps are Certificate Pinning, Public Key Pinning, and Hash Pinning. Hash pinning is the most secure but also the most complex, while certificate pinning is the least secure but the easiest to implement. Public key pinning offers a balance between security and complexity.</p>			
Recommendations			
<ul style="list-style-type: none"> Implement hook/tamper detection. SSL pinning can be implemented in the following ways: Certificate pinning, public key pinning, hash pinning. 			

- Implement root detection to prevent SSL pinning bypass techniques.

Tools Used

- Objection Tool
- Frida

References

- <https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05g-testing-network-communication>
- <https://www.guardsquare.com/blog/how-to-securely-implement-tls-certificate-checking-in-android-apps>
- <https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication>

Testing steps and Evidence

1: Launch Burp Suite on a system that is network-accessible from the rooted device. Start the Burp proxy listener and configure the proxy in the WiFi settings of the device. Download the CA certificate from Burp Suite and install it as a trusted system certificate on the Android device.



The screenshot shows the 'Proxy listeners' tab in Burp Suite. It contains a table with two listeners configured. The first listener is on interface 127.0.0.1:8080 with per-host certificates and default TLS protocols. The second listener is on interface 192.168.70.82:8083, also with per-host certificates and default TLS protocols. Both have 'Support HTTP' checked. Below the table, there is a note about CA certificates and buttons for 'Import / export CA certificate' and 'Regenerate CA certificate'.

	Running	Interface	Invisible	Redirect	Certificate	TLS Protocols	Support HTTP...
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	127.0.0.1:8080	<input type="checkbox"/>	<input type="checkbox"/>	Per-host	Default	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	192.168.70.82:8083	<input type="checkbox"/>	<input type="checkbox"/>	Per-host	Default	<input checked="" type="checkbox"/>

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for use in other tools or another installation of Burp.

Burp Proxy is setting to specific address.

2: Use the command to get the application process ID. Command: `-frida-ps -Ua`

```
PS C:\Users\Bishnu> frida-ps -Ua
```

PID	Name	Identifier
11743	Game Launcher	com.samsung.android.game.ganeshome
9749	Google	com.google.android.googlequicksearch
26998	Google Play Store	com.android.vending
28211	Health-e	com.health.e.app
9788	Messages	com.google.android.apps.messaging
18836	Settings	com.android.settings

Frida command

3: After getting the application Process ID, use Objection tool to bypass SSL Pinning.

```

PS C:\Users\Uliana> objection -g 28211 explore
Using USB device 'SM E146B'
Agent injected and responds ok!

  _ _ _ _ _
 | _ | _ | _ | _ | _ | _ | _ | _ |
 | . | . | - | _ | _ | . | _ |
 | _ | _ | _ | _ | _ | _ | _ |
   | _ | (object)inject(ion) v1.11.0

Runtime Mobile Exploration
by: @leonjza from @sensepost

[tab] for command suggestions
com.healthcare.on (samsung: 13) [usb] # android sslpinning disable
(agent) Custom TrustManager ready, overriding SSLContext.init()
(agent) Found okhttp3.CertificatePinner, overriding CertificatePinner.check()
(agent) Found okhttp3.CertificatePinner, overriding CertificatePinner.check$
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustMan
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustMan
(agent) Registering job 138177. Type: android-sslpinning-disable
com.healthcare.on (samsung: 13) [usb] #

```

SSL Pinning Bypassed using Objection.

4: Initiate HTTPS request from mobile app and intercept using Burp Proxy.



The Burp captures the network traffic.

Revalidation Test Evidence

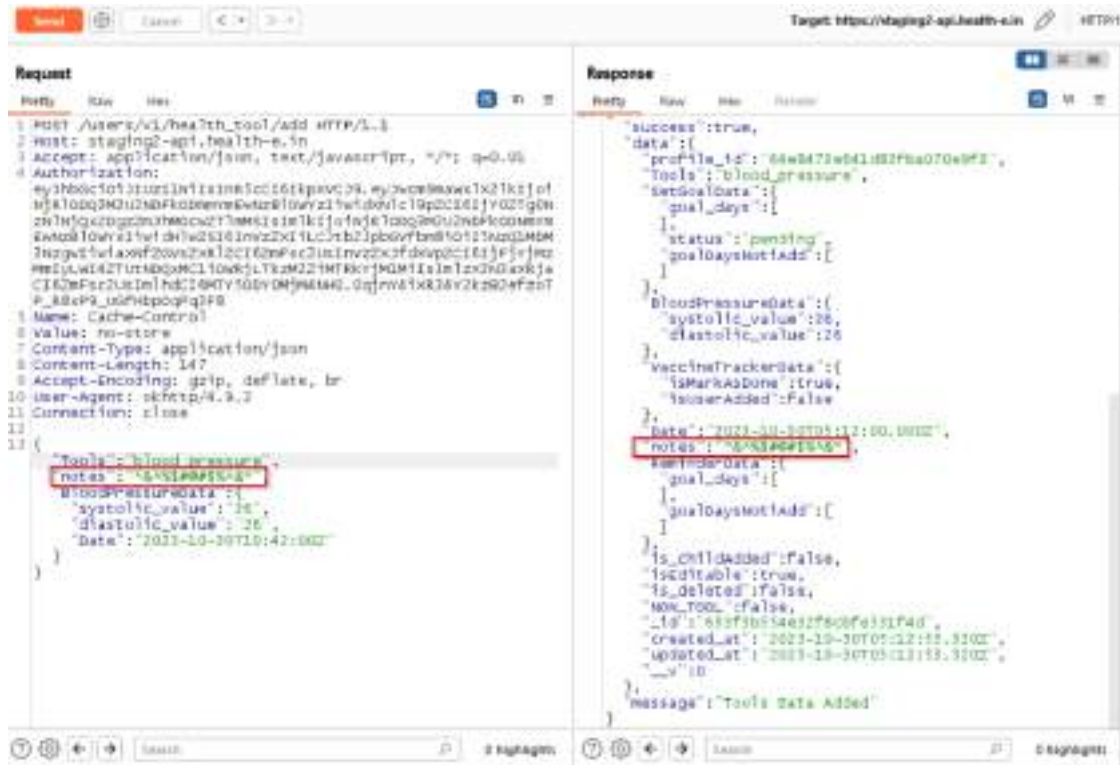
Note: It has been determined that the application is unable to run on rooted devices due to the successful implementation of Root/Jailbreak detection mechanisms, which prevented the interception of network traffic. As such, this issue will be marked as closed.

4.4.4 Improper Input Validation

Severity	Low	Status	Closed
OWASP Category	A04:2021-Insecure Design	CVSS	2.9
Assets Affected			
<ul style="list-style-type: none"> https://staging2-api.health-e.in/users/v1/health_tool/add [Application Wide] 			
Synopsis			
Application fails to validate user supplied data.			
Description			
<p>When an application does not validate input properly, an attacker can craft the input in a form that is not expected by the rest of the application. This application does not validate the input data provided in the parameters.</p> <p>This weakness leads to attacks in applications such as, locale/Unicode attacks, file system attacks and buffer overflows. Writing outside the bounds of a block of allocated memory can corrupt data or crash the program.</p>			
Recommendations			
<ul style="list-style-type: none"> It is recommended to implement server-side validation for all the user inputs. Limit the size of data that user can provide. 			
Tools Used			
<ul style="list-style-type: none"> Burp Suite Browser 			
References			
<ul style="list-style-type: none"> https://www.owasp.org/index.php/Improper_Data_Validation 			

Testing steps and Evidence

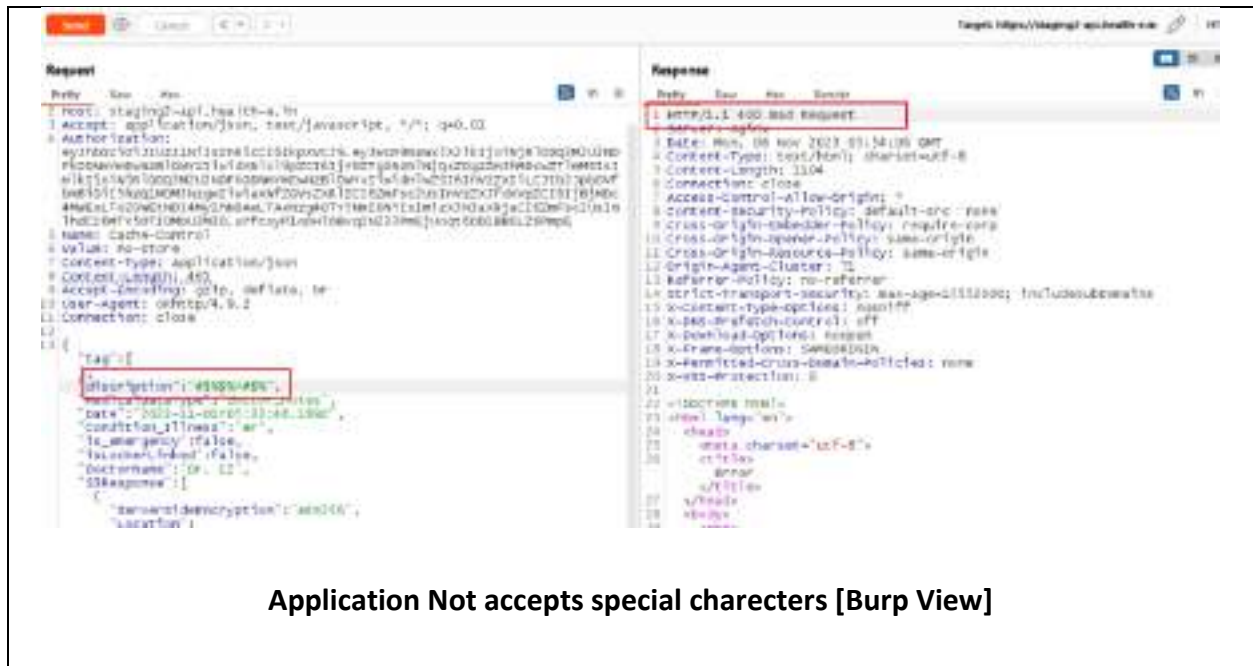
The below given POC illustrates that the input characters are accepted by the application.



Application accepts special charecters [Burp View]

Revalidation Test Evidence

The below given POC illustrates that the special characters are not accepted by the application.



Request

```

POST /api/has HTTP/1.1
Host: 192.168.1.100:8080
Accept: application/json, text/javascript, */*; q=0.01
Accept-Encoding: gzip, deflate, br
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0
Content-Type: application/json
Content-Length: 1104
Connection: close

{"data": {"condition_allowed": "ar", "is_emergency": false, "is_memo_included": false, "document": "01_02_03", "SSRResponse": [{"document": "01_02_03", "document": "01_02_03"}]}}
```

Response

```

HTTP/1.1 400 Bad Request
Content-Type: text/html; charset=utf-8
Content-Length: 1104
Connection: close
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: *
Access-Control-Allow-Methods: *
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff
X-Download-Options: noopen
X-Frame-Options: DENY
X-Permitted-Cross-Domain-Policies: none
X-XSS-Protection: 0

{"data": {"condition_allowed": "ar", "is_emergency": false, "is_memo_included": false, "document": "01_02_03", "SSRResponse": [{"document": "01_02_03", "document": "01_02_03"}]}}
```

Application Not accepts special charactes [Burp View]

4.4.5 Lack of Code Obfuscation

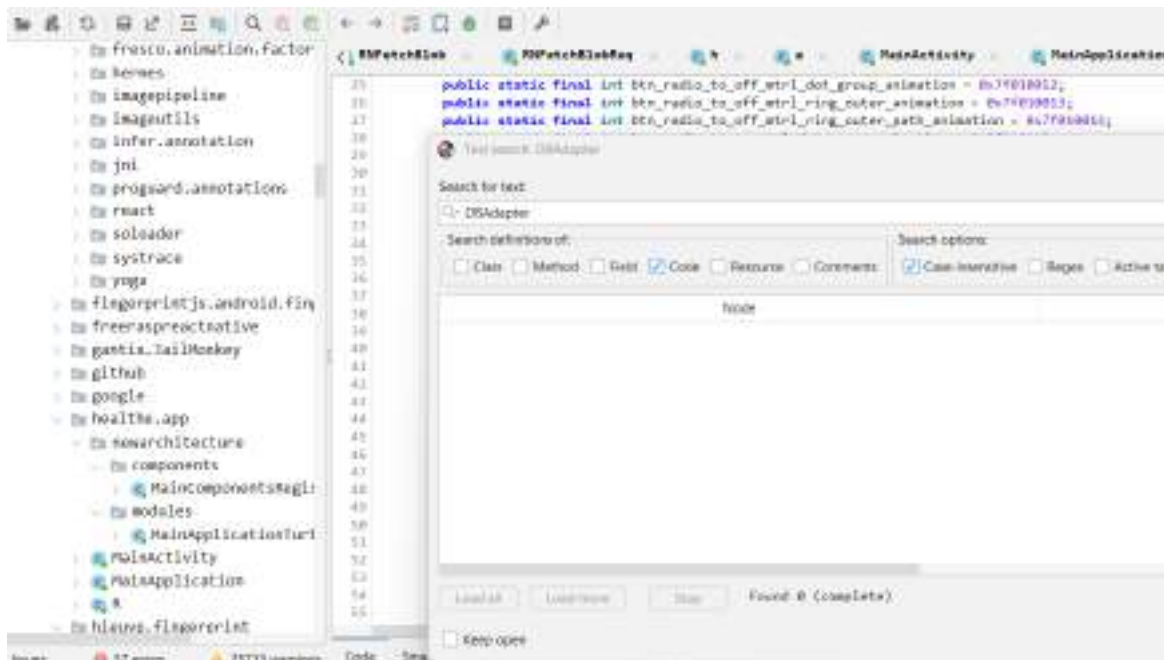
Severity	Low	Status	Closed
OWASP Category	M9: Reverse Engineering	CVSS	2.1
Assets Affected			
<ul style="list-style-type: none"> Android Application 			
Synopsis			
The application lacks proper code obfuscation.			
Description			
<p>Lack of adequate code and resource protection could lead to the reverse engineering of the business logic of the application. These countermeasures are not only needed in order to protect the intellectual property of the code, but they can also help to prevent the application from being cloned, for example, in order to add malicious functionality or to replace legitimate functionality with malicious code.</p> <p>Typical protection techniques are divided into three categories: code obfuscation, anti-debugging and anti-tampering, respectively used to make code analysis harder, avoid debug tracing and prevent code alteration.</p>			
Recommendations			
<ul style="list-style-type: none"> Recommended to consider the use of obfuscators for the source code during the build process of the application. 			
Tools Used			
<ul style="list-style-type: none"> Jadx 			
References			
<ul style="list-style-type: none"> https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering https://github.com/Guardsquare/proguard 			
Testing steps and Evidence			
The below screenshot illustrates lack of code obfuscation			



Lack of code obfuscation

Revalidation Test Evidence

The below screenshot illustrates No lack of code obfuscation



No Lack of code obfuscation

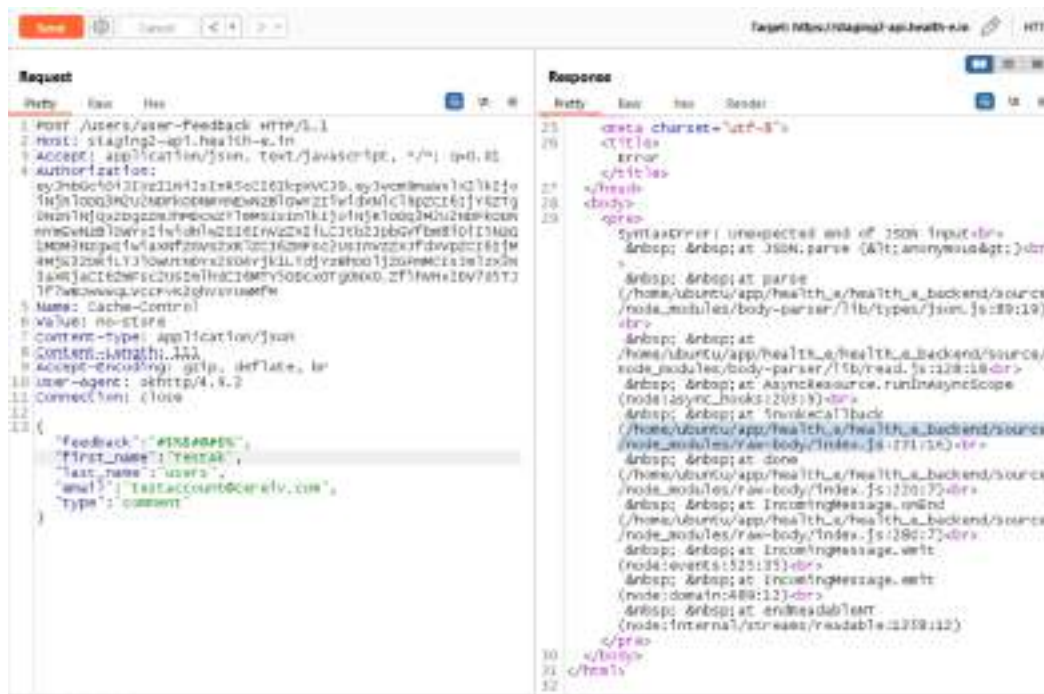
4.4.6 Improper Error Handling

Severity	Low	Status	Closed
OWASP Category	A05:2021-Security Misconfiguration	CVSS	3.8
Resource Affected / URL /Parameters			
<ul style="list-style-type: none"> https://staging2-api.health-e.in/users/user-feedback 			
Synopsis			
Application discloses internal path information through error message.			
Description			
Improper handling of errors can introduce a variety of security problems for a web site. The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user. These messages reveal implementation details that should never be revealed. Such details can provide hackers important clues on potential flaws in the site and are also disturbing to normal users.			
Recommendations			
<ul style="list-style-type: none"> Implement custom error messages for all error case types. 			
Tools Used			
<ul style="list-style-type: none"> Browser 			
References			
<ul style="list-style-type: none"> https://cwe.mitre.org/data/definitions/544.html 			

- <https://www.veracode.com/security/error-handling-flaws-information-and-how-fix-tutorial>

Testing steps and Evidence

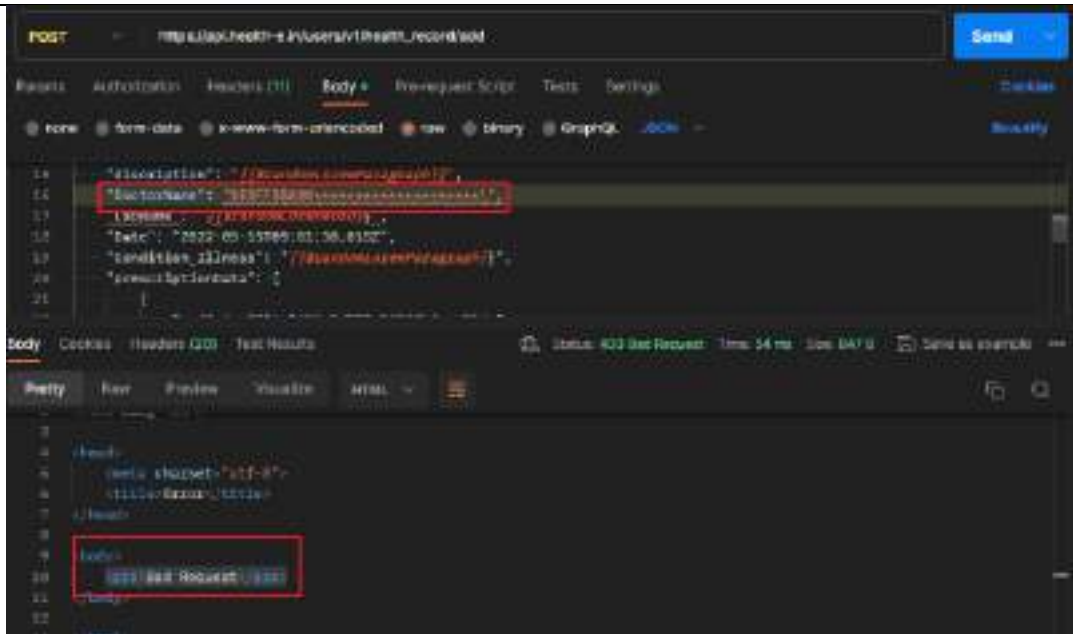
1: Add special characters in the feedback form and see the response.



Improper Error Handling (Burp View)

Revalidation Test Evidence

1: Add special characters and see the response.



Improper Error Handling Fixed (Production server)

4.4.7 Use of Insecure Block Cipher Mode

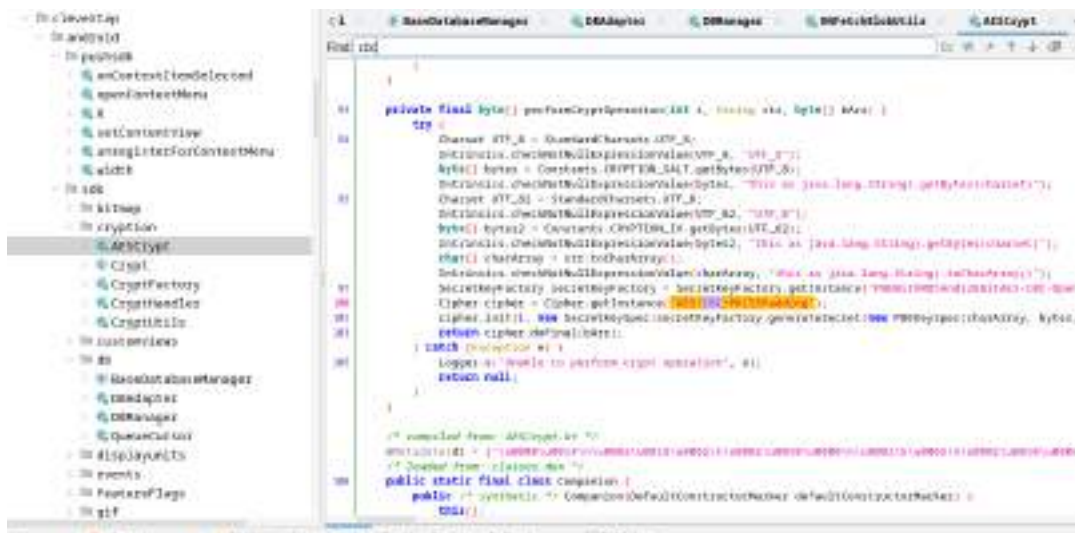
Severity	Low	Status	Closed
OWASP Category	M5: Insufficient Cryptography	CVSS	2.0
Resource Affected / URL /Parameters			
<ul style="list-style-type: none"> Android Application 			
Synopsis			
Cipher Block Chaining mode (CBC) is vulnerable to multiple attack types.			
Description			
<p>In CBC the mode, every encryption of the same plaintext should result in a different ciphertext. The CBC mode does this with an initialization vector. The vector has the same size as the block that is encrypted. Cipher Block Chaining mode(CBC) is vulnerable to multiple attack types: Chosen Plaintext Attack(CPA) — Attacks with a set of chosen plaintexts and to obtain respective ciphertext, Chosen Ciphertext Attack(CCA) — Attacks with a set of chosen ciphertexts to obtain respective plaintexts, and Padding oracle attacks. Both the AES-CBC and AES-GCM are able to secure valuable data with a good implementation. but to prevent complex CBC attacks such as Chosen Plaintext Attack(CPA) and Chosen Ciphertext Attack(CCA) it is necessary to use Authenticated Encryption. So the best option is for that is GCM. AES-GCM is written in parallel which means throughput is significantly higher than AES-CBC by lowering encryption overheads.</p>			
Recommendations			
<ul style="list-style-type: none"> Use AES with GCM mode. 			
Tools Used			
<ul style="list-style-type: none"> JADX 			

References

- <https://isuruka.medium.com/selecting-the-best-aes-block-cipher-mode-aes-gcm-vs-aes-cbc-ee3ebae173c>
- <https://github.com/MobSF/owasp-mstg/blob/master/Document/0x04g-Testing-Cryptography.md#identifying-insecure-and-or-deprecated-cryptographic-algorithms-mstg-crypto-4>

Testing steps and Evidence

1: Decompile the apk file and inspect below file using JADX. The application uses AES with CBC mode for the encryption process.



Usage of CBC Mode

Revalidation Test Evidence

Note: Decompile the apk file and inspect file using JADX. The application does not use AES with CBC mode for the encryption process.

4.5 INFO RISK VULNERABILITIES

4.5.1 Copy/Paste Buffer Caching

Severity	Info	Status	Closed
OWASP Category	M7: Client Code Quality	CVSS	0
Assets Affected			
<ul style="list-style-type: none"> Android application 			
Synopsis			
The application allows the device to store pii such as username in the clipboard.			
Description			
<p>Sensitive data may be stored, recoverable, or could be modified from the clipboard in clear text, regardless of whether the source of the data was initially encrypted. If it is in plaintext at the moment the user copies it, it will be in plaintext when other applications access the clipboard.</p> <p>For example, it follows stricter rules, this means that applications cannot read or write the clipboard, and the only way to use it is by user interaction, doing long-taps to raise the clipboard menu.</p>			
Recommendations			
<ul style="list-style-type: none"> Disable copy/paste function for sensitive fields in the application. Clear the clipboard after taking the contents, to avoid other apps read them and leak what the user is doing. 			
Tools Used			
<ul style="list-style-type: none"> Frida Objection 			

References

- <https://owasp.org/www-project-mobile-top-10/2014-risks/m4-unintended-data-leakage>
- <https://books.nowsecure.com/secure-mobile-development/en/caching-logging/be-aware-of-copy-paste.html>

Testing steps and Evidence

- 1: Start Frida server on the android device and open the application.
- 2: Start the Objection tool on the system by attaching the target application to it.
- 3: Execute the Objection module: *android clipboard monitor*.
- 4: Notice the copied data being captured by the Objection module.



Input information in application and copy to clipboard

```
PS C:\Users\Wishnu> objection -g 12611 explore
Using USB device 'SM E140B'
Agent injected and responds ckl
```

```
(object)inject(ion) v1.11.0
```

Runtime Mobile Exploration
by: @leonjza from @sensepost

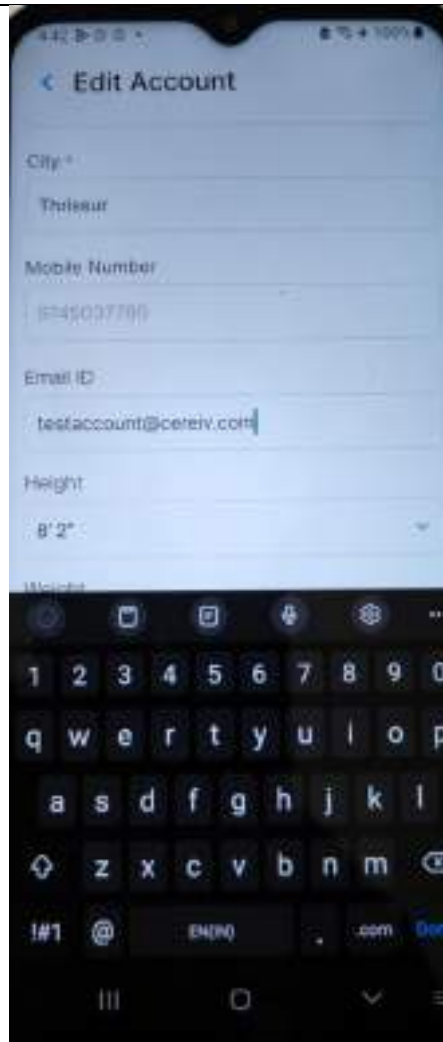
```
[tab] for command suggestions
```

```
com.healthbe.app on (samsung: 13) [usb] # android clipboard monitor
(agent) Warning: This module is still broken. A pull request fixing it would be awesome!
com.healthbe.app on (samsung: 13) [usb] # (agent) [pasteboard-monitor] Data: testaccount@coreiv.com
```

Objection captures user data from the clipboard cache

Revalidation Test Evidence

Note that there is no option to copy the content.



Data cannot be copied into the clip board

4.5.2 Insufficient Cryptography

Severity	Info	Status	Closed
OWASP Category	M5: Insufficient Cryptography	CVSS	0
Assets Affected			
<ul style="list-style-type: none"> Android Application 			
Synopsis			
The application uses insecure and/or deprecated cryptographic configurations.			
Description			
<p>When assessing a mobile app, make sure that it does not use cryptographic algorithms and protocols that have significant known weaknesses or are otherwise insufficient for modern security requirements. Algorithms that were considered secure in the past may become insecure over time; therefore, it's important to periodically check current best practices and adjust configurations accordingly. Verify that cryptographic algorithms are up to date and in line with industry standards. Vulnerable algorithms include outdated block ciphers (such as DES and 3DES), stream ciphers (such as RC4), hash functions (such as MD5 and SHA1), and broken random number generators (such as Dual_EC_DRBG and SHA1PRNG). Note that even algorithms that are certified (for example, by NIST) can become insecure over time. A certification does not replace periodic verification of an algorithm's soundness. Algorithms with known weaknesses should be replaced with more secure alternatives. Additionally, algorithms used for encryption must be standardized and open to verification. Encrypting data using any unknown, or proprietary algorithms may expose the application to different cryptographic attacks which may result in recovery of the plaintext.</p>			
Recommendations			
<ul style="list-style-type: none"> Verify that cryptographic algorithms are up to date and in-line with industry standards. Use AES with CBC/GCM mode. Avoid weaker padding or block operation implementations. Use cryptographically secure RNGs. 			
Tools Used			

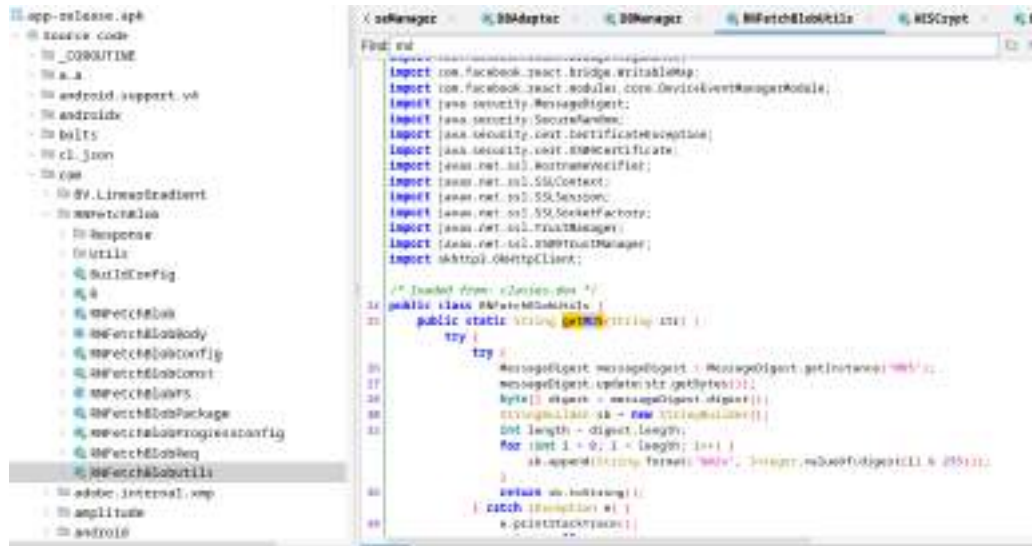
JADX

References

<https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography>

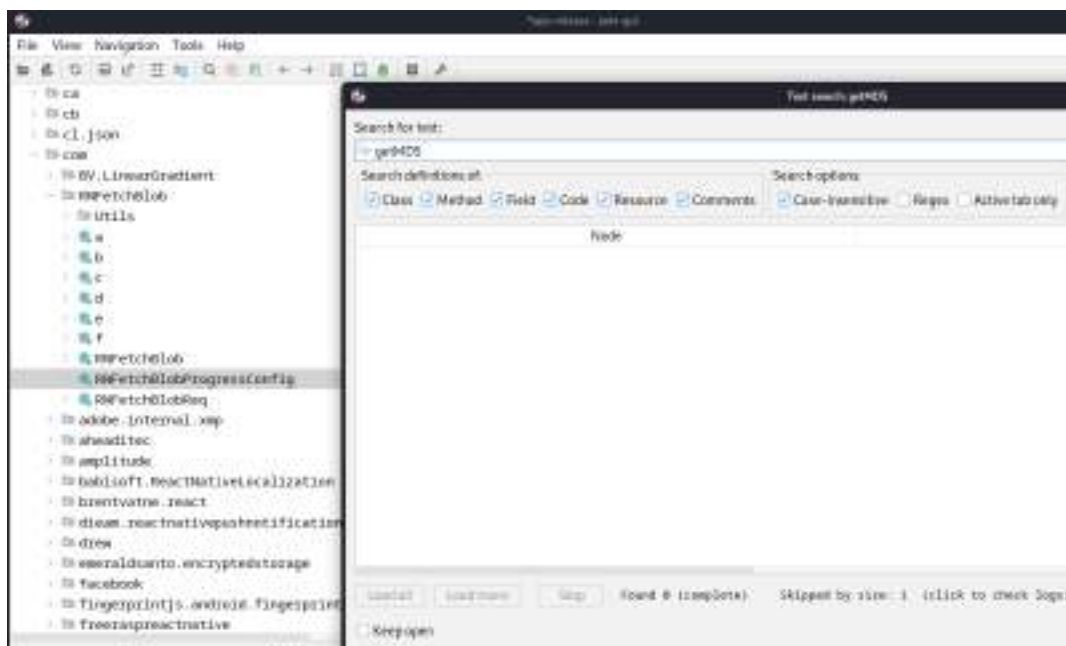
Testing steps and Evidence

MD5 is a weak hash known to have hash collisions.



Usage of MD5

Revalidation Test Evidence

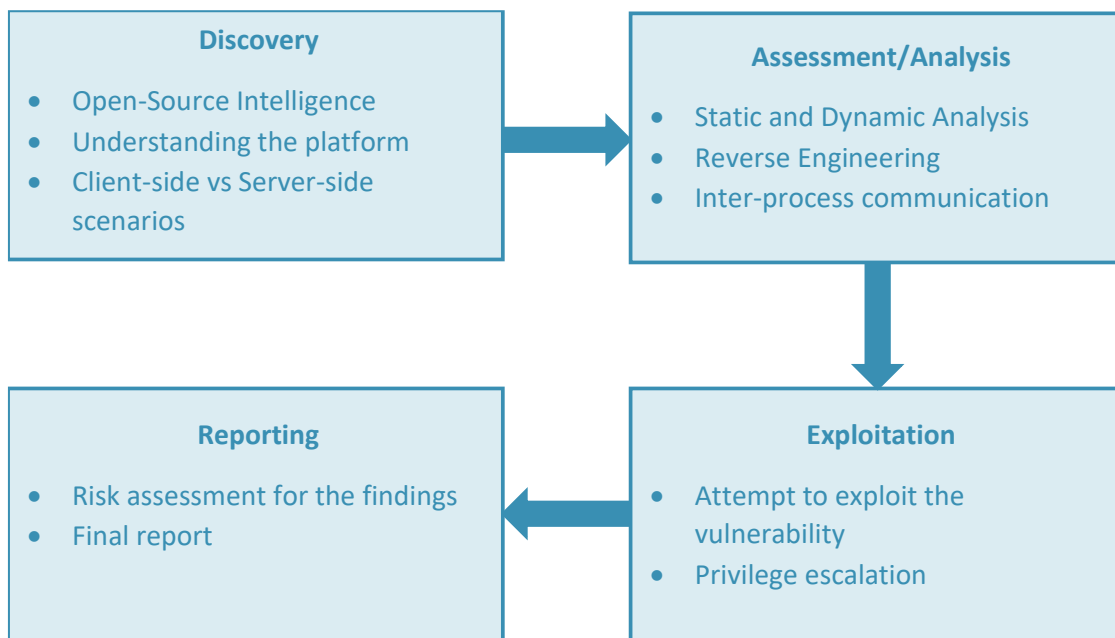


No Usage of MD5

5 PENETRATION TESTING PROCESS

5.1 APPROACH

In addition to the process described below, every penetration test is approached with varying amounts of prior knowledge about the environment. These approaches can be black-box, white-box or grey-box (defined in the Glossary of Terms).



5.2 PENETRATION TESTING METHODOLOGY CHART

The chart above is a visual representation of our overall penetration testing process. This process is applied for mobile application penetration testing.

5.3 TOOLS

CEREIV utilized a series of automated tools along with manual exploitation methods to identify security vulnerabilities and perform tests to actively exploit them in a non-harmful manner.

The following tools were used in this penetration test:

Tool Name	Description
Nessus	Vulnerability assessment tool to identify security issues.
Burp Professional	Application assessment tool to spider and identify application security issues.
MobSF	Android/iOS security assessment framework to perform static and dynamic analysis.
Frida	Dynamic instrumentation toolkit used to perform function hooking.
Objection	Runtime mobile exploration toolkit, powered by Frida.
Kali Linux	Open-source security testing distribution tool used to identify and exploit security issues.
NMAP	("Network Mapper") is a free and open-source utility for network exploration or security auditing.
Custom Scripts	Any python, perl, bash or other programming scripts that are developed to solve a problem.

5.4 RISK RATING SCALE

The significance of each finding below is defined with a Severity Rating of Critical, High, Medium, or Low to simplify reporting, analysis, and remediation planning. The table below illustrates the general methodology followed for identifying the Severity Rating of each finding.

Rating	CVSS Score
None/Info	0.0
Low	0.1 – 3.9
Medium	4.0 – 6.9
High	7.0 – 8.9
Critical	9.0 – 10.0

Often, an attacker may leverage a combination of vulnerabilities to exploit and gain remote unauthorized access to applications and data. If the penetration tester was able to chain vulnerabilities in this manner it will be noted in the narrative description of the exploitation process provided later in this report, and it may alter the severity rating of a specific individual vulnerability accordingly.

6 GLOSSARY OF TERMS

Air Gap: Air gap is a network security technique that physically separates a secure network from other, higher risk networks. This technique is often in use in industrial control networks.

Black-Box Testing: This approach simulates a malicious attacker with very limited information about the network or application being tested. In black box testing, more time is spent on reconnaissance and discovery through publicly accessible information. In addition, CEREIV's source IP addresses will not be whitelisted.

Breached Air Gap: This describes a network air gap that has been compromised in some way. This could be by way of a "bridge" that physically connects the SCADA network to a non-industrial control system network. In the worst case, the bridged network would have internet access, thus putting SCADA assets at higher risk. Not only is "bridging" an issue but a breach could occur by other means like planted rogue access points, or by personnel carrying malware on removable media to the SCADA network, exposing it and essentially "jumping the air gap."

Client-Side Attack: A Client-side attack is the act of using an email with a link, document, or other object that a user might be tempted to click on to compromise the system. This can also include exploiting client-side applications like a web browser or local application. Client-side exploits are often paired with Social Engineering attacks to increase effectiveness.

Denial of Service (DoS) attack: A DOS attack is an attack with the purpose of making a machine or network resource unavailable to its intended users by consuming resources like bandwidth or CPU cycles.

Grey-Box Testing: This approach involves limited information from the environment from the client. This is usually obtained through interviews with the client. This approach allows CEREIV to target specific systems in scope to produce better results rather than spending resources on discovery and reconnaissance. This approach helps simulate an attacker with knowledge of the systems as well as an attacker without any knowledge of the systems. Grey box testing can include whitelisting, but it is not required.

OWASP: OWASP stands for Open Web Application Security Project. The Open Web Application Security Project is an open source project which includes corporations, educational organizations, and individuals from around the world. Their goal is to create freely available articles, methodologies, documentation, tools, and technologies related to web application security. <https://www.owasp.org>

OWASP Top 10 Project: The goal of the Top 10 Project is to raise awareness about application security by identifying some of the most critical risks facing organizations. The Top 10 Project is referenced by many standards, books, tools, and organizations, including MITRE, PCI DSS, DISA, FTC, and many more. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Penetration Test: The intent of a penetration test is to simulate a real-world attack with a goal of identifying how far an attacker would be able to infiltrate an environment. This allows a company to gain a better understanding of their potential exposure and develop a strategy to defend against attacks. A penetration test differs from a vulnerability assessment, as a penetration test is an active process that includes exploiting identified vulnerabilities.

Pivoting: Pivoting is using access obtained on a system to reach previously unreachable systems. This is often accomplished by using collected credentials or using the compromised system's network connections to reach a network location previously unreachable by the attacking system.

Social Engineering: Social Engineering is the use of coercion on people to obtain a desired piece of information or access. Phishing and pre-text calling are two examples of social engineering.

Phishing: 'Phishing' is a technique used to gain personal information for purposes of identity theft, using fraudulent e-mail messages that appear to come from a legitimate source. These authentic-looking messages are designed to fool recipients into divulging sensitive personal information such as account numbers and passwords, credit card numbers and social security numbers.

Pretext Calling: Pretext phone calling is a method of social engineering where an attacker calls employee on the phone and attempts to gather sensitive information by tricking the employee into thinking the attacker is a legitimate company representative who can be trusted.

Vulnerability Assessment: A vulnerability assessment is the detection and compilation of weaknesses that are associated with versions of operating systems, applications, and their corresponding configuration. This includes the susceptibility of in-scope systems to detected attack vectors and the ease and opportunities available to a threat agent to mount a specific attack. Vulnerability assessments are conducted using a vulnerability scanner to detect network-based vulnerabilities but does not include active exploitation of any system.

Whitelisting: Whitelisting is the process of creating exceptions in security devices like firewalls or intrusion protection systems which might otherwise drop packets if the observed network activity appears to be malicious.

White-Box Testing: To simulate an internal attack from an authenticated user such as a disgruntled employee or customer, a white-box testing approach can be utilized for testing. Hence, CEREIV can be provided user credentials to gain access and perform in-depth testing activities to identify issues an authenticated user could identify and exploit

7 APPENDIX A

- Anahat Solutions_Mobile_App_VAPT_Report_061112023_V1.1

The external reference document contains the excel tracker that can be used for remediation guidance.

8 APPENDIX B

- Raw_Scan_Reports

The result of automated scans performed on vulnerability scanners are shared separately. It should be used for reference and remediation information only as it may contain false-positive findings not vetted by CEREIV.

9 APPENDIX C

SSL Pinning can be effectively implemented in the following ways:

- **Certificate Pinning**

Certificate Pinning is the easiest method to perform. The certificate can be stored in the application and when the certificate expires, the application must be updated with the new certificate. At runtime, when the server's certificate is retrieved in the callback, it is compared with the certificate embedded within the app. If it matches, the host is trusted else it will throw an SSL certificate error.

The drawback of Certificate Pinning is that each time the server rotates its certificate, the application needs to be updated. So, if the server rotates its certificate on a frequent basis, then the application would need to be updated frequently as well.

- **Public Key Pinning**

Public key pinning is more flexible to achieve but a little trickier as it requires some extra steps which are necessary to extract the public key from a certificate. In this approach, a keypair is generated, the private key is put in the server and the public key in the application. And just like in

certificate pinning, the extracted public key is checked with its embedded copy of the public key. If it matches, the host is trusted else it will throw an SSL certificate error. By using public key pinning, the frequent application updates can be prevented as the public key can remain same for longer periods.

The drawbacks of Public key Pinning is that it's harder to work with keys since it involves the process of extracting the key from the certificate and the key is static and may violate key rotation policies.

- **Hash Pinning**

The hash of the Public Key of the server's certificate can be pinned and matched with the hash of the certificate's public key received during a network request. This technique is more complex than others. After having the certificate, it can be hashed with any secure hashing algorithm.

The advantages of this technique from others are the following:

- This gives anonymity to a certificate or public key.
- A digested certificate fingerprint is often available as a native API for many libraries, so it's convenient to use.
- An organization might want to supply a reserve identity in case the primary identity is compromised.

The key should be hashed with any secure hashing algorithm. After calculating the hash, it can be encoded with Base64 encoding, to make it easier to store, and read.

Risks of improper SSL Pinning Implementation

Implementing SSL Pinning which can be bypassed by an attacker could lead to the following attacks:

- **MiTM Attack**

MiTM attack refers to some sort of malicious actor sitting in the middle of communications between the client (user/app) and the server. In this position it is possible to intercept, read and modify the traffic to and from the two end devices affecting the users' confidentiality, and the integrity of the data. There are numerous ways that MitM can occur, this is not just limited to smart devices, however in the context of mobile, one of the biggest threats are fake Android certificates or iOS device profiles.

- **Identification and exploitation of Server-Side Vulnerabilities**

After successful pinning bypass, the intercepted requests could help an attacker to understand the API request flow and to find any existing vulnerabilities or new vulnerabilities at the server side.

Common Techniques used to bypass SSL Verification and Certificate Pinning

There are multiple techniques to bypass Android SSL Verification and Certificate Pinning implementations, items mentioned below are some commonly used and known techniques.

Android

- **Adding a custom CA to the trusted certificate store**

System-installed certificates can be managed on the Android device in the Settings -> Security -> Certificates -> 'System'-section, whereas the user trusted certificates are managed in the 'User'-section there. When using user trusted certificates, Android will force the user of the Android device to implement additional safety measures: the use of a PIN-code, a pattern-lock or a password to unlock the device are mandatory when user-supplied certificates are used.

If the application targets Android versions later than 6.0, it will not trust the user-added CA store. To get around this, the application's manifest can be edited and forced to target Android 6.0. The targeted API level is specified in the 'platformBuildVersionCode' attribute of the 'manifest' element in the AndroidManifest.xml file. The value of platformBuildVersionCode' can be changed to 23 and when the application is repackaged with this updated manifest, it will trust the user-added CA store.

Alternatively, if running on a specific platform version is required, we can define specific trust anchors in the '/res/xml/network_security_config.xml' configuration file of the APK.

Installing CAcert certificates as 'user trusted'-certificates is very easy. Installing new certificates as 'system trusted'- certificates requires more work (and requires root access), but it has the advantage of avoiding the Android lock screen requirement.

- **Overwriting a packaged CA cert with a custom CA cert**

Developers can take additional steps to restrict the set of CAs trusted by the application. If a custom certificate chain is being distributed with an application, extracting the APK and overwriting the provided CA with our custom CA should be enough to cause our intercepting certificate to be trusted. In some cases, additional verification of the trust chain may be happening, so this method may yield mixed results.

Opening the APK with a tool such as APK Studio makes the presence of certificates bundled with the deployed application obvious. Overwriting the provided CA with the custom CA should trick the application into accepting the custom certificate.

- **Using Frida and Objection**

Frida is a framework that allows to tamper with an application's code at runtime. Typically, Frida will run on the operating system as a stand-alone program – but that requires rooting a device. To avoid that, Frida Gadget can be injected into the target APK. Frida Gadget contains most of the functionality of Frida but encapsulated in a dynamic library that gets loaded by the target app at runtime, allowing to instrument and modify the target app's code.

To load Frida Gadget, we need to extract the APK, insert the dynamic library, edit some smali code so our dynamic library is the first thing that gets called at application startup, then re-package the APK and install it.

Objection automates this entire process and requires only the target APK to be provided on the command line. The syntax is “`objection patchapk -s test_app.apk`”. After the objection-altered APK has been installed on the target device, running the app should result in a pause at the application startup screen. At this point, we can connect to a Frida server that should be listening on the device.

References:

<https://frida.re/>

<https://codeshare.frida.re/>

<https://github.com/sensepost/objection>

- **Using Xposed Framework Modules**

Xposed is a framework that allows users to easily apply add-ons (called Modules) to the ROM. Rather than flashing a new ROM to get a specific feature, you can use Xposed to add individual features to whatever ROM you're using, or even just the stock ROM. The Xposed framework requires root privilege, but Frida gains access to the full suite of Frida functionality without rooting a device.

SSLUnpinning and TrustMeAlready are Xposed Modules to bypass SSL certificate validation by making several hooks in SSL classes for specific apps and allows us to intercept application traffic.

References:

<https://www.xda-developers.com/xposed-framework-hub/>

https://github.com/ac-pm/SSLUnpinning_Xposed

<https://github.com/ViRb3/TrustMeAlready>

- **Reversing custom certificate code**

Developers might choose to provide their own SSL libraries instead of relying on the system libraries to handle the SSL certificate validation. If this is the case, it would be required to extract the APK and convert the smali back to Java so that the code responsible for handling the certificate validation can be identified.

iOS

- **Installing your own CA**

Installing your CA is relatively easy inside of iOS. The first step is to get the CA onto the device. This could be done through opening an email attachment or downloading the certificate. First off, configure the mobile device and web proxy to be able to intercept web traffic. Specifically, for Burp Suite, simply browse to <http://burp> and click on "CA Certificate". Clicking on install prompts a warning that the certificate that is going to be installed will be added to the list of trusted certificates. To verify that the certificate is installed, go to Settings > General > Profile. In iOS 10.3 and later, the installed certificate must be manually trusted by going to Settings > General > About > Certificate Trust Settings and enable trust for that certificate.

- **Installing Software to iOS Device**

If the application server is using some sort of TLS chain validation or SSL certificate pinning. The simplest method to bypass SSL certificate pinning is to install an SSL Pinning bypassing software. The tools listed below are easy to setup and get running.

- SLKillSwitch 2
<https://github.com/nabla-c0d3/ssl-kill-switch2>
- SSLBypass
<https://github.com/evilpenguin/SSLBypass>
- Burp Mobile Assistant
<https://portswigger.net/support/configuring-burp-suite-mobile-assistant>

Installation instructions are listed on each of the webpages. However, with these methods, a jailbroken iOS device is required.

- **Using Objection and Frida**

Frida is a framework that allows tampering an application's code at runtime. Specifically, interfering with the logic behind certificate validation. This is limited to using jailbroken devices. However, Frida Gadget, which has the full arsenal of the framework can be used which avoids the need for a jailbroken device. Objection is a wrapper for this framework.

Prerequisites for modifying a binary to load the FridaGadget.dylib:

- A valid provisioning profile
- A code-signing certificate from an Apple Developer account.

Steps to follow for injecting Frida Gadget:

- Extract the IPA file,
- Modify the binary to load the FridaGadget.dylib, code-sign the binary and dylib
- Repackage the updated IPA file.

Objection can be used to perform all this work. This can be done with the simple command below where -s is the IPA file and -c is the code-signing certificate.

```
objection patchipa -s test.ipa -c 0[code-signing_cert]C
```

The newly generated ipa can then be deployed to the iOS device. There is a handy tool called ios-deploy which can work with non-jailbroken iOS devices. Other options include running a debugger, deploying app over USB, etc.

Once the modified ipa is deployed to the device, open the application and connect to it via Objection, which can be achieved by using the command 'objection explore'

After successful hook, run the built-in command that bypasses the certificate validation. i.e. "ios sslpinning disable "

- **Using disassemblers to modify IPA file**

Disassemblers can be used to modify the IPA file to bypass any certificate validation. Some of the more common disassemblers are Hopper and IDA. Once the binary has been loaded into the application, following the logic behind what functions are called when the mobile application attempts to make an SSL connection with the application server can point in the right direction of where the certificate pinning is taking place. Modifying the IPA will most likely break the signed application and it cannot be installed on an iOS device. Resigning the IPA file will allow to install the mobile app.

10 APPENDIX D

Below is a list of some of the more popular methods of detecting jailbroken iOS devices.

- **File-based Checks**

Check for files and directories typically associated with jailbreaks. The most popular files that jailbreak detection is based on are listed below:

- /Library/MobileSubstrate/DynamicLibraries/!ABypass2.dylib
- /Library/MobileSubstrate/DynamicLibraries/SSLBypass.dylib
- /Library/MobileSubstrate/DynamicLibraries/AppSyncUnified-installId.dylib
- /Applications/vnodebypass.app
- /usr/lib/substitute-loader.dylib
- /usr/lib/frida/frida-agent.dylib
- /usr/libexec/cydia

- /usr/libexec/filza
- zzzzzLiberty.dylib
- /usr/lib/libcycrypt.dylib
- libsparkapplist.dylib
- SSLKILLSwitch2.dylib
- zeroshadow.dylib
- SubstrateInserterd.ylib
- UnSub.dylib
- Liberty.dylib
- /private/var/stash
- /private/var/lib/apt
- /private/var/tmp/cydia.log
- /private/var/lib/cydia
- /private/var/mobile/Library/SBSettings/Themes
- /Library/MobileSubstrate/MobileSubstrate.dylib
- /Library/MobileSubstrate/DynamicLibraries/Veency.plist
- /Library/MobileSubstrate/DynamicLibraries/LiveClock.plist
- /System/Library/LaunchDaemons/com.ikey.bbot.plist
- /System/Library/LaunchDaemons/com.saurik.Cydia.Startup.plist
- /var/cache/apt
- /var/lib/apt
- /var/lib/cydia
- /var/log/syslog
- /var/tmp/cydia.log
- /bin/bash
- /bin/sh
- /usr/sbin/sshd
- /usr/libexec/ssh-keysign
- /usr/sbin/sshd

- /usr/bin/sshd
- /usr/libexec/sftp-server
- /etc/ssh/sshd_config
- /etc/apt
- /Applications/Cydia.app
- /Applications/RockApp.app
- /Applications/Icy.app
- /Applications/WinterBoard.app
- /Applications/SBSettings.app
- /Applications/MxTube.app
- /Applications/IntelliScreen.app
- /Applications/FakeCarrier.app
- /Applications/blackra1n.app
- **Existence of symbolic links**

Some directories are originally located in the small system partition, however, this partition is overwritten during the jailbreak process. Therefore the data must be relocated to the larger data partition. Because the old file location must remain valid, symbolic links are created. The following list contains files/directories which would be symbolic links on a jailbroken device. An application could check for these symbolic links, and, if they exist, detect a jailbreak.

- /Library/Ringtones
- /Library/Wallpaper
- /usr/arm-apple-darwin9
- /usr/include
- /usr/libexec
- /usr/share
- /Applications
- **Checking File Permissions**

Another way to check for jailbreaking mechanisms is to try to write to a location that's outside the application's sandbox. This can be done by having the application attempt to create a file in, for example, the /private directory. If the file is created successfully, the device has been jailbroken.

- **Checking Protocol Handlers**

Protocol handlers can be checked by attempting to open a Cydia URL. The Cydia app store, which practically every jailbreaking tool installs by default, installs the cydia:// protocol handler.

- **Calling System APIs**

Calling the system function with a "NULL" argument on a non-jailbroken device will return "0"; doing the same thing on a jailbroken device will return "1". This difference is due to the function's checking for access to /bin/sh on jailbroken devices only.

Some API calls provided by iOS behave differently if run on jailbroken devices. Detecting a jailbroken device based on API calls can be both effective and difficult for a malicious individual to recognize and bypass.

fork()

The sandbox denies process forking on non-jailbroken devices. By checking the returned pid on fork(), an app can detect if it has successfully forked. If the fork is successful, the app can deduce that it is running on a jailbroken device.

system()

Calling the system() function with a NULL argument on a non-jailbroken device will return 0. Doing the same on a jailbroken device will return 1. This is because the function will check whether /bin/sh exists, and it only exists on jailbroken devices.

dyld functions

This detection method starts with calling functions like _dyld_image_count() and _dyld_get_image_name() to see what dylibs are currently loaded. This method is very difficult to

dynamically patch due to the fact that the patches themselves are part of dylibs. Attackers have a difficult time bypassing this detection method.

- **Anti-Tampering**

Some mobile OSs may provide native frameworks to detect tampering and to check the identity and integrity of applications. Additionally, anti-tampering may search for an installed instrumentation framework and known debugging tools (such as the Frida instrumentation framework). Finally, anti-tampering can use various ways to 'bind' the application to some cryptographic secrets and some identification values (PUF variables, identifiers provided by the hardware or cryptographic hashes).

Anti-tampering Swift Library: <https://github.com/securing/IOSSecuritySuite>

- **Implement Hook Detection:**

Anti-hooking measures should be put in place to try stop things like Frida, Cydia Substrate, Liberty, Liberty Lite, SSLKillSwitch2, SSLBypass etc.

- **Implement Name obfuscation:**

To bypass SSL pinning, the attacker first needs to find out which method to hook. By using a tool to obfuscate Swift and Objective-C metadata in their iOS app, developers can make it much more difficult for the attacker to determine which methods to hook.

Name obfuscation will also throw off all automated tools that look for a known method name. An obfuscator can rename methods in a different way in each single build of the application, forcing an attacker to search the actual name in each new version.

It is important to note that name obfuscation only protects against tools that bypass SSL checks implemented in the code of applications or in libraries included in the application. Tools that work by hooking system frameworks will not be deterred by it.

- **Implement Control Flow Obfuscation**

Control flow obfuscation will alter the structure of a mobile app's code. It is an essential step to increase the complexity of the program logic, so that even decompilers will not be able to parse it. Control flow obfuscation modifies the logical structure of the code to make it less predictable and traceable.

- **Implement Arithmetic Obfuscation**

Arithmetic obfuscation will effectively harden all arithmetic calculations, by replacing them with mathematical equivalents, but much harder to understand computations.

- **Implement Encryption**

Encryption ensures the code of the application and the data it contains cannot be accessed while the application is at rest. The encrypted code is decrypted on-the-fly when the application is executed guaranteeing that it functions as intended. To be effective, the encryption must be applied in various layers. Essential encryption techniques include string encryption, class encryption, asset encryption and resource encryption.

11 APPENDIX E

The goal of comprehensive root detection is to make running the application on a rooted device more difficult. Detecting rooted devices alone are not sufficient, but implementing various checks scattered throughout the app can improve the effectiveness of overall implementation and improve security of Android apps.

- **File existence checks**

Check for files typically found on rooted devices, such as package files of common rooting apps and their associated files and directories.

Detection code also often looks for binaries that are usually installed once a device has been rooted. These searches include checking for busybox and attempting to open the su binary at different locations. Checking whether su is on the PATH also works.

- **Executing su and other commands**

Another way of determining whether su exists is attempting to execute it through the `Runtime.getRuntime.exec` method. An `IOException` will be thrown if su is not on the PATH. The same method can be used to check for other programs often found on rooted devices, such as busybox and the symbolic links that typically point to it.

- **Checking running processes**

Supersu-by far the most popular rooting tool-runs an authentication daemon named daemonsu, so the presence of this process is another sign of a rooted device. Running processes can be enumerated with the ActivityManager. getRunningAppProcesses and manager.getRunningServices APIs, the ps command, and browsing through the /proc directory.

- **Checking installed app packages**

The Android package manager can be used to obtain a list of installed packages. Check for the package names belonging to popular rooting tools. For e.g.:

- com.thirdparty.superuser
- eu.chainfire.supersu
- com.noshufou.android.su
- com.koushikdutta.superuser
- com.zachspeng.temprootremovejb
- com.ramandroid.appquarantine
- com.topjohnwu.magisk

- **Checking for writable partitions and system directories**

Unusual permissions on system directories may indicate a customized or rooted device. Although the system and data directories are normally mounted read-only, you will sometimes find them mounted read-write when the device is rooted. Look for these filesystems mounted with the “rw” flag or try to create a file in the data directories.

- **Checking for custom Android builds**

Check for files typically found on rooted devices, such as package files of common rooting apps and their associated Checking for signs of test builds and custom ROMs is also helpful. One way to do this is to check the BUILD tag for test-keys, which normally indicate a custom Android image. Missing Google Over-The-Air (OTA) certificates is another sign of a custom ROM.

- **Implement Emulator Detection:**

Many security researchers and penetration testers use virtual devices for testing the security of Android applications. Having emulator detection gives one layer of additional protection to your application against runtime manipulation. The two techniques we can leverage in Android emulator detection are the use of runtime execution, and Java reflection.

Android Anti-Emulator is one such GitHub repo that uses various ways of detecting an emulated Android environment and determines if the application is running on a virtual device or not.

Link of source code: <https://github.com/strazzere/anti-emulator>

- **Google SafetyNet Attestation API**

SafetyNet is an Android API that creates profiles of devices according to software and hardware information and then compares it to a list of whitelisted device models, which have cleared Android compatibility testing.

When this API is called, SafetyNet will download a package containing device validation code which is provided by Google. Then, the code is dynamically executed on the device to check the integrity of the device.

Link of documentation: <https://developer.android.com/training/safetynet/attestation#java>

Link of reference implementation: <https://github.com/googlesamples/android-play-safetynet>

- **Implement Root Detection Libraries**

With the invention of RootCloak, RootCloak Plus, “system-less” root, Magisk Hide, Frida root bypass scripts, bypassing root detection checks has become easier. Hence, there is no single check that detects all types of rooting methods. Hence, implementing multiple checks will ensure a higher detection rate.

Root Inspector

Most detection apps simply try to run su or perform basic checks. Root Inspector uses multiple methods of root detection. There are 15 root checks via SDK (Java) and 13 checks via NDK (Native Code). This can be an effective detection mechanism against RootCloak or RootCloak Plus as well.

Source Code: <https://github.com/devadvance/rootinspector>

RootBeer

RootBeer library is an open-source project that checks if the Android device is rooted or not. RootBeer Fresh is based on the original RootBeer project but implements some new and different techniques to detect rooted devices including basic checks to detect Magisk.

Source Code: <https://github.com/KimChangYoun/rootbeerFresh>

Implementation: <https://play.google.com/store/apps/details?id=com.kimchangyoun.rootbeerFresh.sample>

- **Implement Hook Detection:**

Anti-hooking measures should be put in place to try stop things like Frida, Xposed Framework, Magisk, Edxposed etc.

DetectFrida

DetectFrida is one GitHub project that used three 3 ways to detecting frida hooking:

- Detect through named pipes used by Frida
- Detect through frida specific named thread
- Compare text section in memory with text section in disk

Source Code: <https://github.com/darvincisec/DetectFrida>

Reference Implementation with obfuscated APK:

<https://github.com/darvincisec/DetectFrida/blob/master/obfuscated-app-release.apk>

AntiFrida

AntiFrida is another GitHub repo for detecting Frida instrumentation within a process. AntiFrida performs a scan for all local TCP ports, sends a D-Bus message to each port to identify frida-server. It also scans text sections to search for a specific string found inside frida-gadget*.so or frida-agent*.so files.

Source Code: <https://github.com/b-mueller/frida-detection-demo>

MagiskDetector

Magisk Detector performs various checks to detect Magisk. It also checks if Magisk Hide is enabled or not, resulting in an effective detection mechanism.

Source Code: <https://github.com/vvb2060/MagiskDetector>

DetectMagiskHide

DetectMagiskHide is another GitHub repo for detecting the presence of Magisk and to check if Magisk Hide is enabled on the application or not. Compared to DetectMagiskHide, MagiskDetector has more effective checks.

Source Code: <https://github.com/darvincisec/DetectMagiskHide>

- **Implement Name obfuscation:**

To bypass SSL pinning, the attacker first needs to find out which method to hook. By using a tool to obfuscate metadata in the android app, developers can make it much more difficult for the attacker to determine which methods to hook.

Name obfuscation will also throw off all automated tools that look for a known method name. An obfuscator can rename methods in a different way in each single build of the application, forcing an attacker to search the actual name in each new version.

- **Implement Control Flow Obfuscation**

Control flow obfuscation will alter the structure of a mobile app's code. It is an essential step to increase the complexity of the program logic, so that even decompilers will not be able to parse it. Control flow obfuscation modifies the logical structure of the code to make it less predictable and traceable.

- **Implement Arithmetic Obfuscation**

Arithmetic obfuscation will effectively harden all arithmetic calculations, by replacing them with mathematical equivalents, but much harder to understand computations.

- **Implement Encryption**

Encryption ensures the code of the application and the data it contains cannot be accessed while the application is at rest. The encrypted code is decrypted on-the-fly when the application is

executed guaranteeing that it functions as intended. To be effective, the encryption must be applied in various layers. Essential encryption techniques include string encryption, class encryption, asset encryption and resource encryption.