CONFIDENTIAL

(Customer)

Mobile Application Penetration Testing Report

10.07.2021, Version 1.1

—

Security For Everyone, LLC
Sepapaja 6, Tallinn 15551,
Estonia

https://securityforeveryone.com

ABOUT US

We are a team that has been working on cybersecurity in the industry for a long time. In 2021, we created securityforeveyone.com to help our customers to minimize their cybersecurity risks.

Mission: We want to make cybersecurity understandable, affordable, and manageable for everyone.

Vision: To manage cybersecurity processes correctly, automate processes, offer feasible solutions, and prioritize simplicity while doing all these.

TABLE OF CONTENT

CONFIDENTIALITY STATEMENT

This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires the consent of (Customer) or Security For Everyone.

DISCLAIMERS

The information contained in this report covers the results of penetration testing carried out until the finish date. Due to the changing conditions in the test environment after or during the penetration testing, Security For Everyone is not responsible.

Security for everyone is not responsible for incorrect assessment or consequences of using this report. In addition, the necessary hardening and controls should be done correctly in all systems by using the supplied recommendations in this report. Since the tests are carried out in a specific time interval, the vulnerabilities published after the test are not included in this report.

The penetration test was made within the terms of the contract.

REPORT HISTORY

Version	Date	Description
1.0	06.07.2021	Initial version
1.1	10.07.2021	Information about updates.

1. Executive Summary

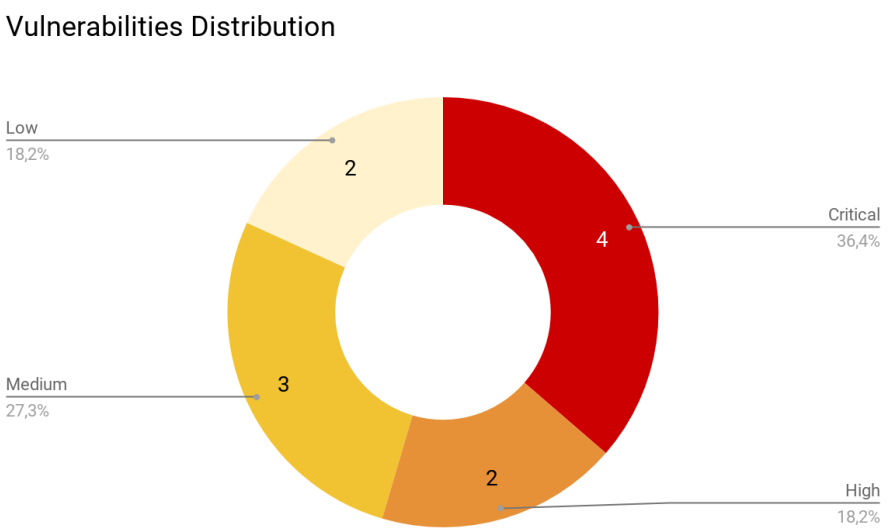
The primary objective of the performed test is making a security assessment for your assets. Penetration tests were carried out by simulating real-life attack scenarios without damaging the systems. Some critical vulnerabilities have been discovered during the penetration test. If attackers use these vulnerabilities, they can carry out a range of malicious acts.

According to the worst-case scenario, your systems can be seized by malicious people. In addition, third parties may obtain critical information such as customer data.

During the security tests, four critical, two high, three medium, and two low-level findings were detected. These findings were due to weak password usage, missing patches, misconfiguration, and improper input validation. Key recommendations are listed below:

- **Setting a strong password policy:** Weak passwords should not be allowed. Passwords should be changed periodically. Two-factor authentication should be used if possible.
- **Making regular updates:** It should regularly update the related systems.
- **Input validation:** All data received from the user must be properly filtered and used.

Vulnerabilities distribution is given below:



Detailed information about the findings, solution suggestions, screenshots and recommended information sources are shared later in the report.

2. Test Scope and Duration

Penetration tests have been performed only to the following scope.

Applications	**Related Information**
securityforeveryone.apk	Given creds: username / pass****
securityforeveryone.ipa	Given creds: username / pass****

Table 1: Test Scope

The list of S4E staff working together from the beginning to the end of the test is given below. This team carried out tasks related to scoping, instant reporting of critical vulnerabilities, sharing information during the tests, and transmitting the report.

Personel	**Contact**	**Description**
Andrew	andrew@mail.mail	Penetration Tester
Cooper	cooper@mail.mail	Penetration Tester and Report Writer

Table 2: Responsible people

The tests started on 06.07.2021 and ended on 09.07.2021. This report was transmitted on 10.07.2021 via email using the PGP key in a password protected compressed file format.

Password information was shared with Bob.

3. Methodology

The test methodology is a 5-step process that starts with determining the scope of the test and finishes with preparing a report for the customer. The steps in this methodology can be used to understand the penetration testing process. According to the agreement, some steps may not be included in the test.

In our penetration test, we use payloads and software that we developed, along with other automated vulnerability scanning tools. However, we do not add any outputs of automated tools in the report without manual control.

3.1. Scoping

In this step, the type and scope of the test are decided. The type of test can be a white-box, gray-box, or black-box. If complete information is obtained about the systems to be tested, this is called white-box tests. It is called gray-box tests only if certain information is received. If no information is received, black-box tests are performed.

3.2. Information Gathering

This step is the most important step of the penetration test. Information about the target systems is collected. The information gathering phase is both an automated and a manual process. This step includes but is not limited to:

- Framework and Library versions
- Application information
- Registered email addresses
- Social media accounts (related to domains and emails)
- Search engine queries
- API Detection
- Paths / files detection
- Document analysis
- Source Code analysis
- Threat intelligence information (API domain history, API IP information, leak passwords, etc)
- Application mapping

3.3. Vulnerability Assessment

Target systems are scanned by vulnerability scanning tools, and comparison is made with up-to-date vulnerability databases. Also, the information obtained in step 1 is researched on the internet for known vulnerabilities. Application or system reviews are performed. It is checked whether there are any logical or coding errors. If available, test accounts are included in the process at this stage. Vulnerability checks (automatic and manual) are performed for applications. This step includes but is not limited to:

- Network vulnerability scanning tools
- Application logic and data analysis
- Application vulnerability scanning tools
- Open source researchers for vulnerabilities
- Authorization tests
- Special payloads
- Brute force

3.4. Exploitation

This step is the second most crucial step after gathering information. Automatic tools outputs are evaluated at this step. Attacks are made like a real hacker, which means vulnerabilities are exploited without damaging the systems. This step includes but is not limited to:

- Exploitation of vulnerabilities
- Web application attacks
- Password cracking
- Custom malware development
- Lateral movement
- Social engineering attack

3.5. Reporting

All findings are collected and analyzed. Evidence of vulnerabilities, data obtained from screenshots are matched with findings. A detailed report is created by adding solution suggestions and external sources. Risk levels of vulnerabilities were calculated according to the Common Vulnerability Scoring System (CVSS) system. For detailed information, you can visit the 'https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator' link. The risk table is shared below.

Risk Level	CVSS Score
Critical	9.0-10.0
High	7.0-8.9
Medium	4.0-6.9
Low	0.1-3.9
Info	0

4. Findings

4.1. Overview

Applications	Critical	High	Medium	Low	Info	#
securityforeveryone.apk	1	1	2	0	1	5
securityforeveryone.ipa	2	0	0	1	1	4

Table 3: Findings Overview

4.2. Findings Card

4.2.1. SQL Injection	Critical

Description: In dynamic pages in web applications, the information provided to the user is mostly served from databases. Structured Query Language (SQL) is a programming language created to manage data in relational databases. Values received from the user in web applications can be included in SQL queries. These queries are named as dynamic SQL queries. The attacks made by changing the dynamic SQL queries running in the target system are called SQL Injection attacks. In SQL Injection attacks, many malicious actions can be performed, such as obtaining passwords, deleting data, retrieving data, reading a file, running a command on the operating system.

Request:

```
GET /login.php?id=1' or '1'='1 HTTP/1.1
Host: securityforeveryone.com
```

Response: Successful login

Screenshots: (example)

Solutions: WAF settings can be used as a quick solution to reduce risk. But it does not solve this vulnerability precisely. Because WAFs can be bypassed. Therefore, filtering should be done in the application layer. Any user input must be filtered before using in the SQL queries. The attacks made by changing the dynamic SQL queries running in the target system are called SQL Injection attacks.

Additional Sources:

- https://owasp.org/www-community/attacks/SQL_Injection
- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md

4.2.2. Incorrect Permissions	High

Description: App permissions must be appropriate for the purpose of the app. In addition, the principle of least privilege should be used in practice against possible exploitation. Permissions allow accessing controlled functionality such as the camera or GPS are requested in the program. Permissions can be implicitly granted to an app without the user's consent. An app with too many permissions may perform unintended functions outside the scope of the app's intended functionality. Additionally, the permissions are vulnerable to hijacking by another app.

Request: N/A

Response: N/A

Screenshots: (example)

Solutions: Permissions that the application does not need should not be used. The policy of least privilege must be applied in the application.

Additional Sources:

- https://en.wikipedia.org/wiki/Principle_of_least_privilege
- https://developer.android.com/training/permissions/usage-notes
- https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/accessing-user-data/

4.2.3. Insecure Communication	High

Description: External communication allows information to leave the current device to be transmitted to another device. Insecure communications allow apps to gather unintended information and inject new information. Insecure communication (data network, Wi-Fi, Bluetooth, NFC, etc.) leaves information open to disclosure or man-in-the-middle attacks.

Request: N/A

Response: N/A

Screenshots: (example)

Solutions:

- Apply SSL/TLS to transport channels that the mobile app will use to transmit sensitive information, session tokens, or other sensitive data to a backend API or web service.
- Use certificates signed by a trusted CA provider.
- Never allow self-signed certificates, and consider certificate pinning for security conscious applications.
- Always require SSL chain verification.

Additional Sources:

- https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

4.2.4. Insecure Direct Object References	High

Description: Insecure direct object references (IDOR) are a type of access control vulnerability that arises when an application uses user-supplied input to access objects directly. Applications frequently use the actual name or key of an object when generating web pages. Applications don't always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. An attacker who detects this vulnerability can compromise all data by changing a parameter value that refers to an object accessible to an authorized system user.

Request:

```
GET /login.php?id=1
Host: securityforeveryone.com
```

Response: N/A

Screenshots: (example)

Solutions:

- Use per user or session indirect object references. This prevents attackers from directly targeting unauthorized resources.
- Check access. Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.
- In the system architecture, the principle of least privilege should be used, and access to data should be limited on the basis of the authorization.

Additional Sources:

- https://wiki.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References

4.2.5. Incorrect/Missing Obfuscation	Medium

Description: Code Obfuscation is the process of modifying an executable so that it is no longer useful to a hacker but remains fully functional. While the process may modify actual method instructions or metadata, it does not alter the output of the program. As a result of wrong or missing code obfuscation, it will be easier for the attacker to access the code during reverse engineering, so this vulnerability needs to be fixed.

Request: N/A

Response: N/A

Screenshots: (example)

Solutions:

- Narrow down what methods/code segments to obfuscate
- Obfuscate string tables as well as methods
- In the system architecture, the principle of least privilege should be used and access to data should be limited on the basis of the authorization.
- Format data, rename identifiers and remove code comments.

Additional Sources:

- https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering
- https://en.wikipedia.org/wiki/Obfuscation_(software)

4.2.6. Example	Info

Description: ***The vulnerabilities found in this report are for illustrative purposes. For more, please contact with https://securityforeeveryone.com***

Request: N/A

Response: N/A

Screenshots: (example)

Solutions: Contact Us

Additional Sources:

- https://securityforeeveryone.com

5. Remediation Status

Title	Asset	Level	Status
SQL Injection	securityforeeveryone.com	Critical	Active (07/07/2021)
Incorrect Permissions	securityforeeveryone.apk	High	Active (07/07/2021)
Insecure Communication	securityforeeveryone.ipa	High	Active (07/07/2021)
Insecure Direct Object References	securityforeeveryone.com	High	Active (07/07/2021)
Incorrect/Missing Obfuscation	securityforeeveryone.apk	Medium	Active (07/07/2021)