Sejoy Blood Glucose Monitoring System Penetration Testing Report

TÜV SÜD Products Testing (Shanghai) Co., Ltd
No. 151 Heng Tong Road
Shanghai 200 070
P.R. China

Telephone: +86 21 6141 0123
Telefax: +86 21 6140 8600
www.tuv-sud.cn

Project Name	Sejoy Blood Glucose Monitoring System Penetration Testing
Project Number	721684991
Customer Information	**Sejoy Biomedical Co., Ltd.** Area C, Building 2, No.365, Wuzhou Road, Yuhang Economic Development Zone, 311100 Hangzhou City Zhejiang, P. R. China
Project Manager	Evan Yuan 
Quality Control Reviewer	Renjie Wei 
Duration of Testing	10.18.2023-11.22.2023

Revision History

No	Date	Version	Description	Author
1	11.15.2023	0.1	Draft	Evan Yuan
2	11.24.2023	0.2	Add Retest Detail	Evan Yuan
3	11.30.2023	0.3	Quality Control Review	Renjie Wei
4	12.04.2023	1.0	Release	Evan Yuan
5				

TABLE OF CONTENTS

1 PENETRATION TEST METHODOLOGY

1.1 Standards-Based Testing and Reporting

Our penetration test plans are performed according to internally developed guidelines by TÜV SÜD penetration test experts. Our test cases, where possible, are grounded and compliant in publicly available standards published by organizations such as BSI, OWASP, OSSTMM, NIST, PCI and SWIFT and our experience as a cyber security technical assessment team[1]. As an extension to these test cases, additional test cases may be identified based on penetration test expert and the attack surface of the target of evaluation.

Our report is designed to support the activities set forth by CREST in their 2017 guidelines, "A Guide for Running an Effective Penetration Testing Programmer"[2]. It provides both a summary and detailed view of the penetration test and its results. For each vulnerability observed during testing, a description of the vulnerability will be provided along with a generic vulnerability score and suggestions for remediation.

1.2 CVSS: Scoring Vulnerabilities

The overarching goal of a penetration test is to identify the vulnerabilities in a target of evaluation. To assist in the prioritization of vulnerability remediation, TÜV SÜD utilizes the Common Vulnerability Scoring System (CVSS). CVSS assists in the assessment of a vulnerability's severity by providing a standard set of characteristics by which the vulnerability is scored. These scores are then used to calculate an overall severity score from 0.1-10.0; 0.1 being lowest and 10 being highest.

TÜV SÜD utilizes CVSS version 3.1 for scoring. In this version, there are three types of scores that can be calculated: a base score, a temporal score and an environmental score. For purposes of reporting in this document, the CVSS base score will be provided. The base score assesses the following characteristics:

Characteristics	Description
Attack Vector	Assesses whether an adversary can mount an attack from a remote network, a local network or if an adversary must be logged on to the target of evaluation or physically connected.
Attack Complexity	Assesses the complexity of an attack depending on how many of the attack variables are within the control of the adversary.
Privileges Required	Assesses the level of access that an attacker needs to mount a successful attack.
User Interaction	Assesses the extent to which actions of the victim are required for an attack to be successful.
Scope	Assess whether the impact of an attack is limited to the target of evaluation or if the attack has an impact on other systems as well.
Confidentiality	Assesses the negative impact that an attack can have on the target of evaluation's confidentiality.
Integrity	Assesses the negative impact that an attack can have on the target of evaluation's integrity.
Availability	Assesses the negative impact that an attack can have on the target of evaluation's availability.

As indicated above, the assessment of these characteristics results in a severity score which ranges from 0.1-10.0. This score can be further broken down into the following rating levels:

Range	Rating	Description
9.0 - 10.0	**Critical**	These types of vulnerabilities should be reviewed immediately for impact to the business. This rating usually indicates that an exploit exists that could easily be used to severely impact confidentiality, integrity and/or availability.
7.0 – 8.9	**High**	These types of vulnerabilities need to be assessed in the short term for impact to the business. A score in this range indicates that a vulnerability could be exploited with low to medium complexity and could have a moderate or high impact on confidentiality, integrity and/or availability.
4.0 – 6.9	**Medium**	These vulnerabilities should also be evaluated for impact to the business, but the base score shows that these types of vulnerabilities may be only exploitable with increased effort or have little impact to confidentiality, integrity and/or availability.
0.1 – 3.9	**Low**	These vulnerabilities should also be evaluated, but from evaluating the base characteristics, the exploitation of these vulnerabilities is likely to result in little negative impact on confidentiality, integrity and/or availability.

[1] See: https://www.isecom.org/OSSTMM.3.pdf
http://www.pentest-standard.org/index.php/Main_Page
[2] See: https://www.crest-approved.org/wp-content/uploads/2022/04/CREST-Penetration-Testing-Guide-1.pdf

The CVSS score provided in this report is meant to assist with the prioritization of vulnerability resolution. This score, however, does not take into consideration the context of the business. For some business IT contexts some lower-scored vulnerabilities could have serious business impact. Hence, all the reported vulnerabilities should be taken into consideration.

1.3 CWE: Vulnerabilities

Common Weakness Enumeration (CWE) is a set of standards that classify common software and hardware defects and vulnerabilities. The goal of this standard is to summarize and sort out different software defect mechanisms, and unify software security vulnerabilities (including architecture, Design or code implementation level) description language and measurement caliber, and used as a baseline for identifying, repairing and preventing these defects.

1.4 How to Use This Document

The vulnerabilities reported in this document provide a view of the target of evaluation's security posture *at the time of testing*. This timeframe, "at the time of testing", is important to highlight because the report cannot address future changes to the target of evaluation, changes in the systems that support the target of evaluation and emerging, publicly disclosed exploits that could have an impact on the target of evaluation.

The goal of this document is to provide input to help identify and prioritize the vulnerabilities that were observed at the time of testing and to provide some guidance as to how the vulnerabilities might be mitigated. This document, ideally, should be used as a tool within a broader ISMS or penetration testing program which helps in the prioritization of the overall security needs of the organization.

The report is broken up into five major sections: methodology, executive summary, retest, technical detailed findings and appendix. The executive summary will provide a high-level overview of the vulnerabilities observed during the penetration test.

The technical detailed findings will provide the details of the vulnerabilities observed during the penetration test. Each vulnerability will include the following descriptors:

Descriptor	Content
Vulnerability ID	This is the unique ID for this test. It is comprised of the initials of the test type and an incrementing number. For example, if the penetration test type is "Internal Infrastructure Penetration Test", and it is the first vulnerability that has been reported for that test, the ID will look like: IIP-1. These IDs can be used to uniquely identify vulnerabilities in the case that a retest is scheduled.
Description	This provides an overview of the observed vulnerability including how it could be useful to an adversary. It also contains the finding's impact on the security posture.
Impacted System(s)	This provides a list of the systems that are relevant to the vulnerability.
Status	This field will contain either "Verified" or "Identified". If this value is "Verified", then the tester exploited this vulnerability during the penetration test. If it is "Identified", then evidence of the vulnerability was found, but it was not exploited during testing. There are many reasons why a tester may not be able to exploit a vulnerability during testing. Examples include threat of system instability after exploit, lack of time during testing and/or inability to find a vector by which a vulnerability could be exploited.
CVSSv3 Base Score	This provides the overall severity score for a vulnerability including the individual assessments for attack vector, attack complexity, privileges required, user interaction, scope, confidentiality, integrity and availability.
Remediation	This provides suggestions on how to mitigate the vulnerability.
References	This provides links to CWEs and other known resources to learn more about the vulnerability and how to mitigate the vulnerability.

The appendix will contain information about the testing environment and further details gathered during testing that do not fit within the four chapters. This information is necessary to have a complete picture of the penetration test, but it is in the appendix to make accessing the testing results more user-friendly.

2 EXECUTIVE SUMMARY

2.1 Analysis Overview

The objective of a medical device is to determine security vulnerabilities in tested devices that can be exploited by external entities. The tests were carried out assuming the identity of an attacker or user with malicious intent.

Overall, the product has a high level of safety, can avoid the vast majority of common safety risks. However, some problems were found during testing, please refer to Section 2.4

For details concerning the assessment methodology, please refer to Section 1

For details concerning the testing environment, please refer to Section 4.1

2.2 Test Scope

The scope of this penetration test includes security analysis of the hardware.

Penetration Test Scope	
Evaluation Target(s)	Blood Glucose Monitoring System: BG-710 / BG-710b
Used IP Addresses During the Test	N/A
Testing Timeframe	10.18.2023-11.22.2023

The main software details are as follows:	
Software Name	/
Software Version	V01

The scope of this penetration test includes security analysis of the PC software：

Penetration Test Scope	
Evaluation Target(s)	GLUCOJOY-B
Version	V 1.25

The details of the retest version are as follows:	
App Name	GLUCOJOY-B
Retest Version	V 1.29

The scope of this penetration test includes security analysis of the mobile application (Android and iOS)：

Penetration Test Scope	
Evaluation Target(s)	GLUCOJOY
Version	V 1.2.1

The details of the retest version are as follows:	
App Name	GLUCOJOY
Retest Version	V 1.2.2

Based on the security control/configuration requirements of medical devices themselves, the scope of penetration testing is as follows:

Category Types	Description(Purpose of the category)	Applicability
Information Gathering	Application Architecture and Identifying the Languages and Frameworks Used. Network Communication Between the Client and the Server.	√
Weak encryption checks	Check the algorithms related to application encryption.	√
Signature check	Verify the signature information of the application.	√
Session Management	Test session management flaws.	√
Testing for Error Handling	Test error handling and error information leakage.	√
Component security testing	List components and check the security problems.	√
Identifying DLL Hijacking Vulnerability	Checks whether the tested client application is vulnerable to DLL hijacking vulnerabilities.	√
Reverse Engineering	Decompile the application and detect core elements such as the structure, configuration, logic, and sensitive data of the source code, etc. The process of decompiling APK. • Hard-coded credentials	√

	• API Keys • Hidden functions	
Files Analysis	Applications usually store information in local files and the registry. Sensitive information that we might look for in a client pentest includes: • Usernames • Passwords • Connection Strings • API keys	√
Memory Analysis	Sensitive information leaked by performing memory analysis while the client application is running includes: • Usernames • Passwords • Connection Strings • Hard-coded credentials	√
Communication Security	Check whether there is an insecure communication method or a communication protocol with vulnerabilities. Check ATS Policy Security for iOS Apps. Check TLS/SSL protocols and cipher suites. Check Bluetooth protocols security (Not Applicable for BG-710)	√
Broken Authentication & Broken Access Control	Identified authorization bypass vulnerability could allow authenticated attackers to access other users' confidential information.	√
Host Service Security Check	Check whether the host opens dangerous services and there are exploitable vulnerabilities.	√
Data Injection Test	Construct malicious code or files to attack the target to check if it can cause unexpected results.	√
Disassemble hardware	Disassemble and test the hardware, and disassemble the key chips and circuits by thermal welding.	√
Circuit analysis	Analyze the overall layout of the board to identify critical components and circuit paths.	√
User Input/Output interface Identify	Analyze the user I/O interfaces on the device, including external interfaces such as USB, RJ45, and COM.	√
Key chip pin identify	Identify the model and FFID of key chips such as CPU and EEPROM, and mark the pins.	√
ROM chip content extract	Extract chip content from memory chips such as Flash.	√
Debug interfaces identify	Check the commonly used pins, printing pins and reserved solder joints for debugging such as UART/JTAG/SPI/I2C/SWIM, etc., and perform waveform analysis and protocol identification.	√

2.3 Test Classes

The test classes are defined as the mode of exploitation and technique used by an attacker, for example,

- **Man in the middle attack (MITM)** – in this type of attack an adversary can intercept the traffic and either sniff (passive) or manipulate data in transit (active) to cause unexpected behavior of the device/software. This can result in denial of situation (DoS) to complete compromise of any sensitive information or the device software.
- **Replay Attack** – in this type of attack an adversary can resend a manipulated/altered data previously captured by passive monitoring. This altered data can have an unintended behavior of the device. Eg: an unintentional device restart.
- **Known Vulnerabilities** – These tests look to exploit any known vulnerabilities that have been known and published in the public.eg: known CVEs for BLE and WiFi implementation.
- **Cracking** – This involves recovering any sensitive data from captured encrypted data by a process of cracking. This could be possible due to a possible flaw in the implementation of the encryption algorithm or a possible device misconfiguration.
- **Malware scanning** – To scan and identify and remove malware from target system.
- **Malformed input testing / Fuzzing testing** – To send randomized inputs through the USART proprietary protocol

to find test cases that cause anomalous behavior.

The various test classes are subjective to the type of testing performed eg MED-DEV-XX will include attack vectors application to physical access to the device, MED-WRL-XX will include attack vectors related to wireless access used by the device, if any.

2.4 Summary of Technical Findings

A comprehensive penetration testing of the medical software components product was conducted. In summary, 5 high-level, 3 medium-level, and 1 low-level risk vulnerabilities were observed for the application. Based on the highest observed risk rating, the exposure level at the time of testing period is considered **HIGH**.

These vulnerabilities could lead to compromise of the confidentiality, integrity and availability of the data held within the device. The security posture of the application observed during this assessment is shown in the table below.

After the customer's evaluation and rectification of the target product, we retest and verify it. Finally, all vulnerabilities have been fixed, and the residual risk was considered **LOW**.

The details of each finding are summarized below.

Vulnerability ID	Name	Impacted System(s)	CVSSv3.1 Score	Retest
MED-PC-001	Database plaintext storage	GLUCOJOY-B	7.4	Fixed
MED-PC-002	Insecure third-party components	GLUCOJOY-B	7.4	Fixed
MED-PC-003	Inadequate Database Encryption Handling	GLUCOJOY-B	6.2	Fixed
MED-PC-004	Passwords stored locally in plaintext	GLUCOJOY-B	7.3	Fixed
MED-PC-005	Bypass of Login Authentication	GLUCOJOY-B	7.3	Fixed
MED-APP-006	Insecure Cleartext Traffic Configuration	GLUCOJOY (Android)	7.4	Fixed
MED-APP-007	Unencrypted Password Transmission	GLUCOJOY (Android)	5.3	Fixed
MED-APP-008	Weak Passwords	GLUCOJOY (Android)	5.3	Fixed
MED-APP-009	No Root detection	GLUCOJOY (Android)	2.9	Fixed

Table 1: Technical Findings Table of device

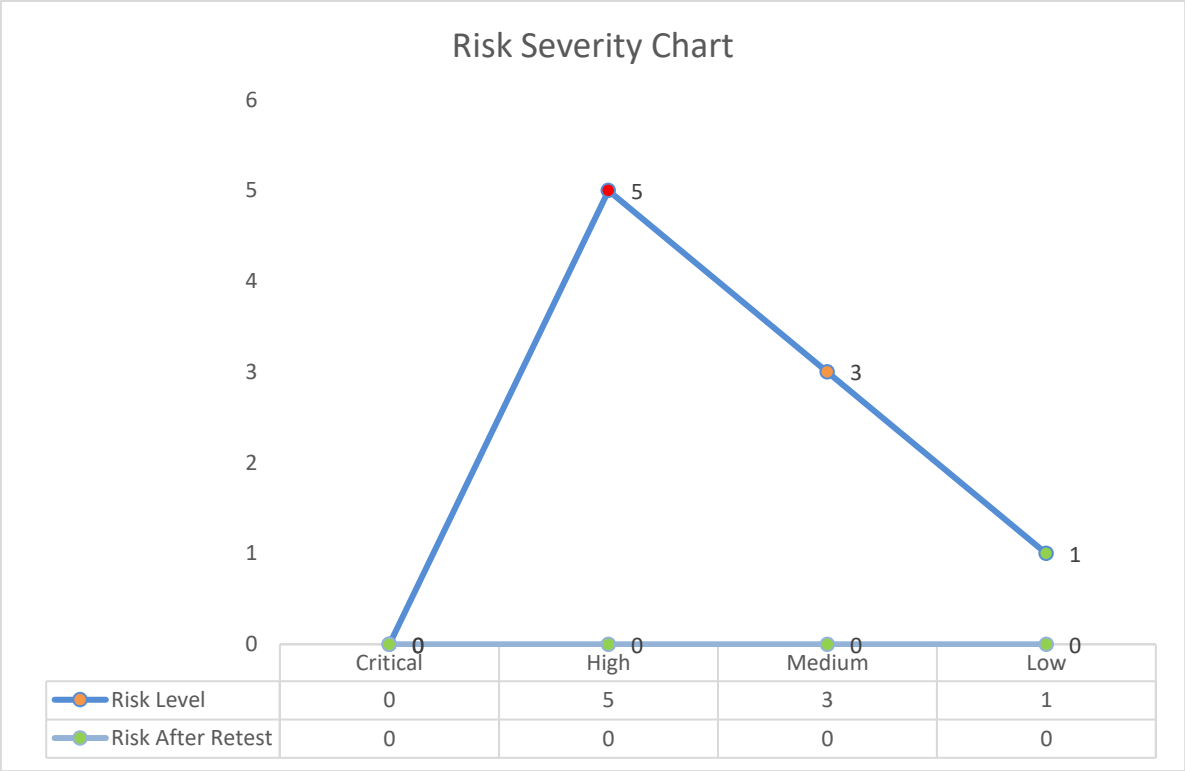


Figure 1: Technical Findings Risk Severity Chart

2.5 Recommendations

During the assessment, we found that the system has some common vulnerabilities, such as Database plaintext storage, weak password, and no root detection.

In addition, the software is compiled without proper obfuscation, causes the code to be easily decompiled. This allows attackers to bypass authentication, query database information, and obtain business source code.

In summary, it is recommended to fix high and medium vulnerabilities as soon as possible, especially pay attention to the data storage security and code compilation to avoid security risks, etc. And lastly conduct safety awareness training regarding development security if necessary.

After retesting and verification, the above problems have been fixed by the developers, which greatly alleviates the security risk of this Blood Glucose Monitoring System.

3 TECHNICAL DETAILED FINDINGS

The following pages contain the detailed technical findings.

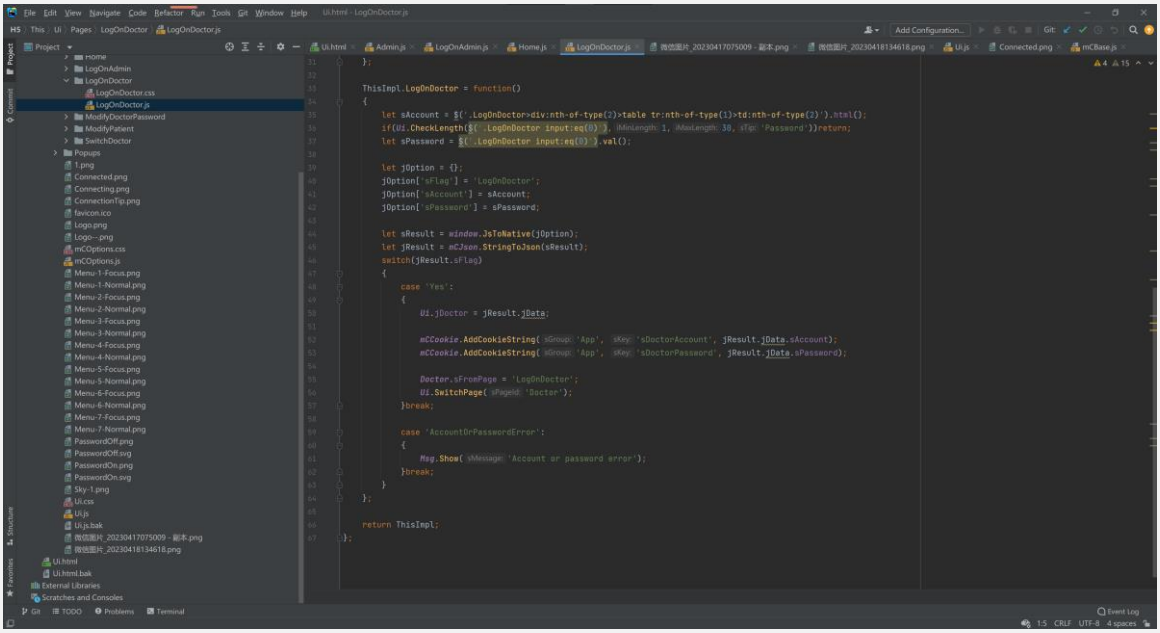
3.1 Windows Application technical findings

MED-PC-001 Business Source Code Exposure

Description	We found the JavaScript code is situated in the H5/This/Ui/Pages directory and can be directly viewed and modified.
Impacted System(s)	GLUCOJOY-B
Status	Verified
CVSSv3 Base Score	**CVSSv3.1 Base Score Vector** **Attack Vector**: Local; **Scope**: Unchanged **Attack Complexity**: High; **Confidentiality**: High **Privileges Required**: None; **Integrity**: High **User Interaction**: None; **Availability**: High **Vector String:** CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H **Base Score:** 7.4 (High)

Details of the Finding

The source code pertaining to H5 is located in the H5 directory within the installation path. The primary JavaScript code is situated in the H5/This/Ui/Pages directory and can be directly viewed and modified. This exposes a vulnerability wherein attackers can gain direct access to and inspect the source code, thereby obtaining sensitive information such as program logic and functionality.



Remediation	To obfuscate JavaScript code, can use tools like UglifyJS, Terser, or JavaScript obfuscators available online.
Reference(s)	N/A
Retest	After the remediation, the JavaScript code has been obfuscated.

```
let LogOnAdmin = new (function () {
  let _0x528158 = {};
  _0x528158["\u0049\u006e\u0069\u0074"] = function () {
    $(
      "gmi )2(epyt-fo-htn:rt elbat>)2(epyt-fo-htn:vid>nimdAnOgoL."
        .split("")
        .reverse()
        .join("")
    )["\u0063\u006c\u0069\u0063\u006b"](function () {
      if (
        $(")0(qe:tupni nimdAnOgoL.".split("").reverse().join(""))[
          "\u0061\u0074\u0074\u0072"
        ]("epyt".split("").reverse().join("")) ==
        "drowssap".split("").reverse().join("")
      ) {
        $(")0(qe:tupni nimdAnOgoL.".split("").reverse().join(""))[
          "\u0061\u0074\u0074\u0072"
        ](
          "epyt".split("").reverse().join(""),
          "txet".split("").reverse().join("")
        );
        $(".LogOnAdmin\x20div:nth-of-type(2)>table\x20tr:nth-of-type(2)\x20img")[
          "\u0061\u0074\u0074\u0072"
        ](
          "crs".split("").reverse().join(""),
          "\u0054\u0068\u0069\u0073\u002f\u0055\u0069\u002f\u0050\u0061\u0073\u0073\u0077\u006f\u0072\u0064\u004f\u006e\u002e\u0070\u006e\u0067"
        );
      } else {
        $(".LogOnAdmin\x20input:eq(0)")["attr"](
          "epyt".split("").reverse().join(""),
          "\u0070\u0061\u0073\u0073\u0077\u006f\u0072\u0064"
        );
        $(".LogOnAdmin>div:nth-of-type(2)>table\x20tr:nth-of-type(2)\x20img")[
          "attr"
        ]("crs".split("").reverse().join(""), "This/Ui/PasswordOff.png");
      }
    });
    $(
      "\u002e\u004c\u006f\u0067\u004f\u006e\u0041\u0064\u006d\u0069\u006e\u003e\u0064\u0069\u0076\u003a\u006e\u0074\u0068\u002d\u006f\u0066\u002d\u0074\u0
```

MED-PC-002 Decompilation Vulnerability

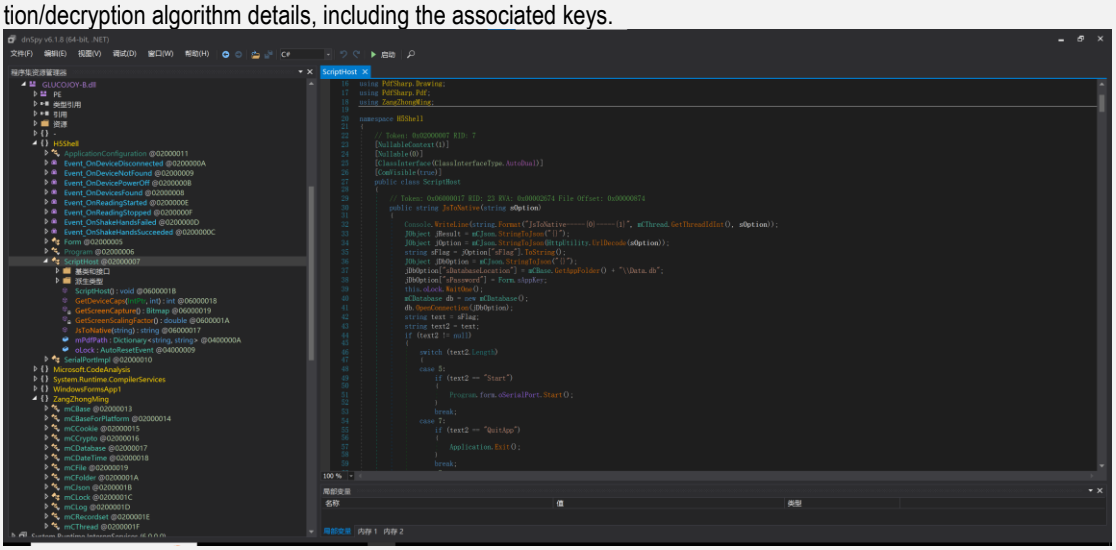
Description	The core program GLUCOJOY-B.dll, compiled using C#, lacks code obfuscation.
Impacted System(s)	GLUCOJOY-B
Status	Verified

CVSSv3 Base Score

CVSSv3.1 Base Score Vector			
Attack Vector	Local	**Scope**	Unchanged
Attack Complexity	High	**Confidentiality**	High
Privileges Required	None	**Integrity**	High
User Interaction	None	**Availability**	High
Vector String:	**CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H**		
Base Score:	**7.4 (High)**		

Details of the Finding

The core program GLUCOJOY-B.dll, compiled using C#, lacks code obfuscation. Direct decompilation reveals original variable names, method names, class names, etc., making it highly readable. Tools such as dnspy can easily decompile it, enabling straightforward analysis of program logic. This poses a risk of exposing database keys and encryption/decryption algorithm details, including the associated keys.



Reverse analysis of GLUCOJOY-B.dll reveals the encryption and decryption logic, as well as key information. The en-

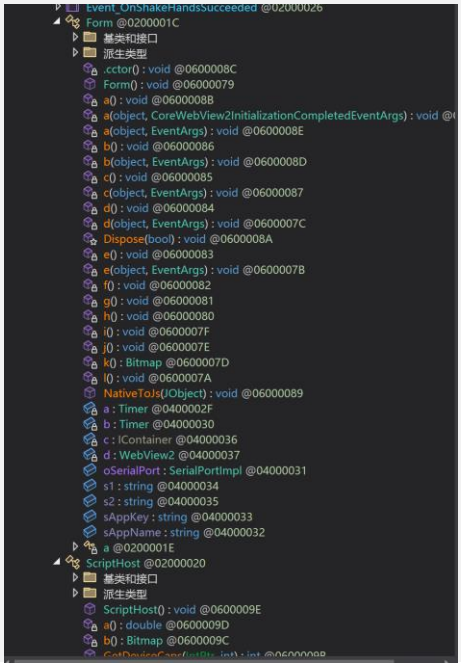
cryption method employed is AES with CBC mode.

```
    }

    // Token: 0x04000027 RID: 39
    private static readonly byte[] Key = Encoding.UTF8.GetBytes("3A98F12B4C678D9E");

    // Token: 0x04000028 RID: 40
    private static readonly byte[] IV = Encoding.UTF8.GetBytes("792AFD341B6C987D");
}
```

```
namespace ZangZhongMing
{
    // Token: 0x02000016 RID: 22
    [NullableContext(1)]
    [Nullable(0)]
    internal class mCCrypto
    {
        // Token: 0x06000075 RID: 117 RVA: 0x00007ED4 File Offset: 0x000060D4
        public static string Encrypt(string plainText)
        {
            string result;
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.Key = mCCrypto.Key;
                aesAlg.IV = mCCrypto.IV;
                ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
                using (MemoryStream msEncrypt = new MemoryStream())
                {
                    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
                    {
                        using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                        {
                            swEncrypt.Write(plainText);
                        }
                        byte[] encryptedBytes = msEncrypt.ToArray();
                        result = Convert.ToBase64String(encryptedBytes);
                    }
                }
            }
            return result;
        }
```

Remediation	Enhance the security of your software by implementing robust encryption techniques, such as code obfuscation and secure key management, to safeguard against reverse engineering and unauthorized access to critical information.
Reference(s)	N/A
Retest	After remediation, obfuscation measures have been applied to the function names associated with the aforementioned functions. 

MED-PC-003 Inadequate Database Encryption Handling

Description	We found the 'Birthday' and 'Note' fields in the database are unencrypted. The 'Password' field is processed using reversible AES encryption.

Impacted System(s)	GLUCOJOY-B
Status	Verified

CVSSv3 Base Score

CVSSv3.1 Base Score Vector			
Attack Vector	Local	**Scope**	Unchanged
Attack Complexity	High	**Confidentiality**	High
Privileges Required	None	**Integrity**	Low
User Interaction	None	**Availability**	Low
Vector String:	**CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:L/A:L**		
Base Score:	**6.2 (Medium)**		

Details of the Finding

The 'Birthday' and 'Note' fields in the database lack encryption, posing a potential risk of information disclosure.

r	sBirthday	sPhone	sEmail	sNote	jS
	过滤	过滤	过滤	过滤	过滤
					{"aTimeValues'
	2023-10-01	JaGma/···	izXuOtCUFZBcdeILht···	text	{"aTimeValues'
	2023-10-19	2ZG/···	KaM5M8XqoM8JDsf17k···	cccccc	{"aTimeValues'

```
            }
            else if (text2 == "LogOnAdmin")
            {
                mCRecordset rs3 = new mCRecordset(db);
                string sSql2 = string.Format("select * from admin_data where sAccount='{0:S}' and sPassword='{1:S}'", mCCrypto.Encrypt((string)jOption["sAccount"]), mCCrypto.Encrypt
                  ((string)jOption["sPassword"]));
                rs3.Query(sSql2);
                bool flag2 = !rs3.bEmpty;
                if (flag2)
                {
                    JObject jData = new JObject();
                    jData["sAccount"] = rs3.GetString("sAccount");
                    jData["sPassword"] = rs3.GetString("sPassword");
                    jData["sDateTime"] = rs3.GetString("sDateTime");
                    jResult["jData"] = jData;
                    jResult["sFlag"] = "Yes";
                }
                else
                {
                    jResult["sFlag"] = "AccountOrPasswordError";
// Token: 0x06000075 RID: 117 RVA: 0x00007ED4 File Offset: 0x000060D4
public static string Encrypt(string plainText)
{
    string text;
    using (Aes aesAlg = Aes.Create())
    {
        aesAlg.Key = mCCrypto.Key;
        aesAlg.IV = mCCrypto.IV;
        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
        using (MemoryStream msEncrypt = new MemoryStream())
        {
            using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
            {
                using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                {
                    swEncrypt.Write(plainText);
                }
                byte[] encryptedBytes = msEncrypt.ToArray();
                text = Convert.ToBase64String(encryptedBytes);
            }
        }
    }
    return text;
}
```

The password can be decrypted using the AES decryption algorithm.

```
key = b'3A98F12B4C678D9E'
iv = b'792AFD341B6C987D'

cipher = AES.new(key, AES.MODE_CBC, iv=iv)

print(cipher.decrypt(base64.decodebytes(b'gvj/5IxwWB9uE9mTYYnWnQ==')))
```

```
main
D:\work\python\pythonProject\venv\Scripts\python.exe D:\work\python\pythonProject\main.py
b'glucoJ01\x08\x08\x08\x08\x08\x08\x08\x08'

Process finished with exit code 0
```

The 'Password' field is processed using reversible AES encryption, which may lead to the exposure and reconstruction of user passwords.

Remediation	All user information in the database is encrypted using cryptographic algorithms. And recommended to employ hash-based encryption for secure storage of user passwords, mitigating the risk of compromise and enhancing overall security.
Reference(s)	N/A
Retest	After remediation, encryption has been applied to the user information fields in the database, and the password field is now processed using SHA hashing.

der	sBirthday	sPhone	sEmail	sNote
	过滤	过滤	过滤	过滤
	PIZ7F1B3TFayOi1o42YWhg==	JaGma/…	931sDo6FCAbm/KOHUQIBkA==	PndijwngEAJC9hkdqIjTBg==

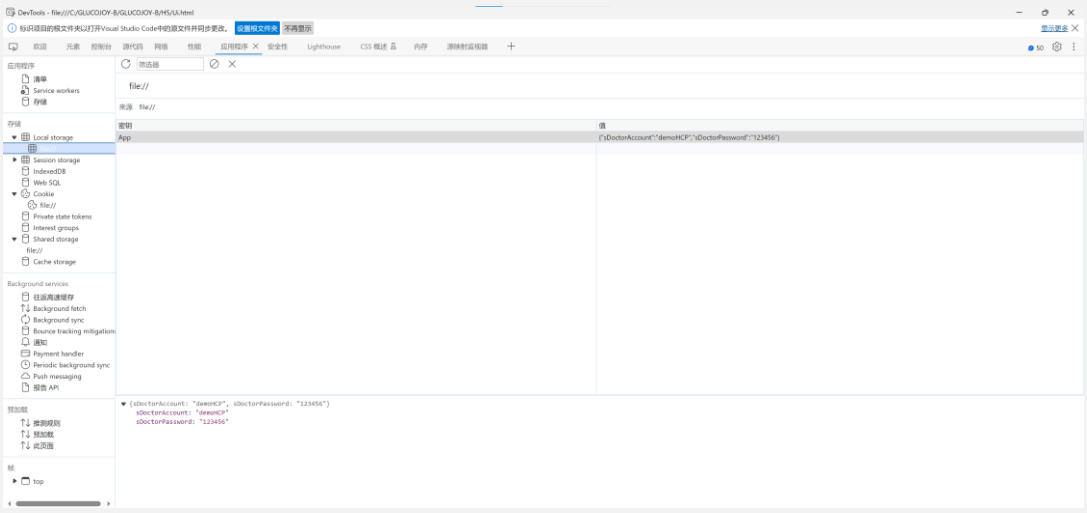
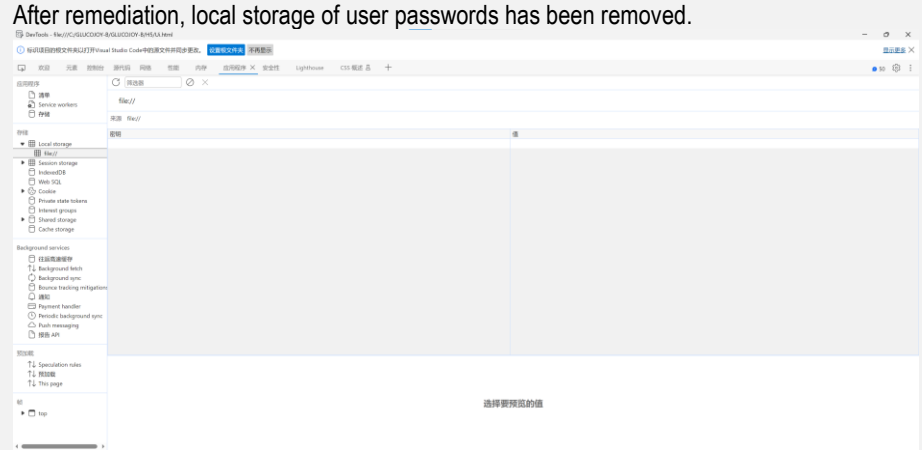
```
else if (text3 == "LogOnAdmin")
{
    q q4 = new q(k);
    string text7 = string.Format("select * from admin_data where sAccount='{0:S}' and sPassword='{1:S}'", j.e((string)jobject2["sAccount"]), j.e((string)jobject2["sPassword"]));
    q4.g(text7);
    bool flag4 = !q4.c;
    if (flag4)
    {
        JObject jobject5 = new JObject();
        jobject5["sAccount"] = q4.b("sAccount");
        jobject5["sPassword"] = q4.b("sPassword");
        jobject5["sDateTime"] = q4.b("sDateTime");
        jobject["jData"] = jobject5;
        jobject["sFlag"] = "Yes";
        Form.s2 = "Yes";
    }
    else
    {
```

```
// Token: 0x06000028 RID: 40 RVA: 0x00002C24 File Offset: 0x00000E24
public static string g(string A_0)
{
    string text;
    using (SHA256 sha = SHA256.Create())
    {
        byte[] array = sha.ComputeHash(Encoding.UTF8.GetBytes(A_0));
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < array.Length; i++)
        {
            stringBuilder.Append(array[i].ToString("x2"));
        }
        text = stringBuilder.ToString();
    }
    return text;
}

// Token: 0x0400000D RID: 13
```

MED-PC-004 Passwords stored locally in plaintext

Description	WebView locally stores usernames and passwords in plaintext.
Impacted System(s)	GLUCOJOY-B

Status	Verified
CVSSv3 Base Score	**CVSSv3.1 Base Score Vector** **Attack Vector**: Local; **Scope**: Unchanged **Attack Complexity**: Low; **Confidentiality**: High **Privileges Required**: None; **Integrity**: Low **User Interaction**: None; **Availability**: Low **Vector String:** CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L **Base Score:** 7.3 (High)
Details of the Finding	PlainText storage of credentials in WebView poses a significant security risk, compromising confidentiality and potentially leading to unauthorized access and abuse of user accounts. 
Remediation	Encrypting or removing the local storage of passwords.
Reference(s)	N/A
Retest	After remediation, local storage of user passwords has been removed. 

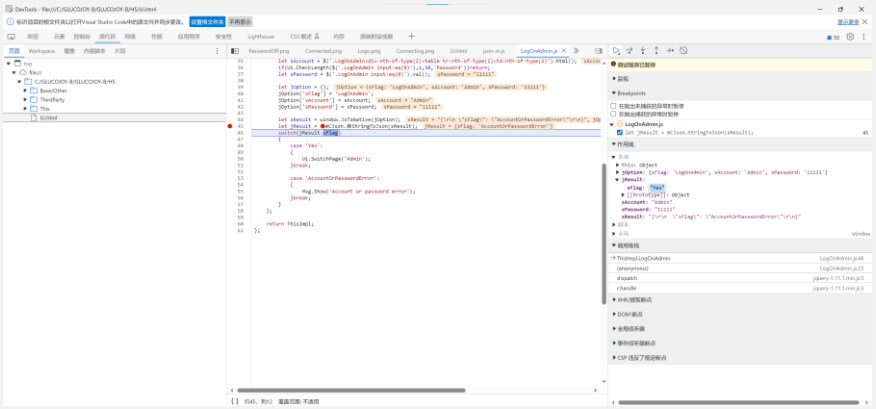
MED-PC-005 Bypass of Login Authentication

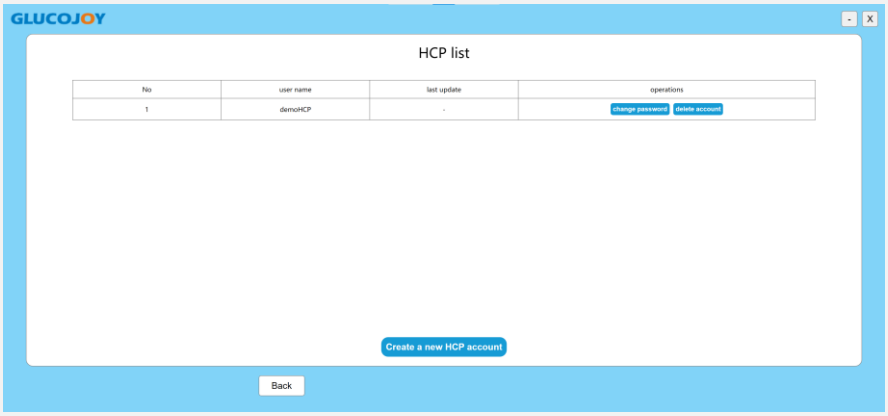
Description	authentication can be circumvented by modifying the 'jResult' and 'Flag' values to 'yes' after entering any password.
Impacted System(s)	GLUCOJOY-B
Status	Verified

CVSSv3 Base Score	**CVSSv3.1 Base Score Vector**			
	Attack Vector	Local	**Scope**	Unchanged
	Attack Complexity	Low	**Confidentiality**	High
	Privileges Required	None	**Integrity**	Low
	User Interaction	None	**Availability**	Low
	Vector String:	**CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L**		
	Base Score:	**7.3 (High)**		

Details of the Finding

Identity Bypass Vulnerability: Admin authentication can be circumvented by modifying the 'jResult' and 'Flag' values to 'yes' after entering any password. Similar identity verification bypass exists for Doctor authentication.





Remediation	Implement enhanced authentication mechanisms and controls to address the identified authentication bypass vulnerability.
Reference(s)	N/A
Retest	After remediation, validation checks have been added to the form. Following the steps outlined above no longer reproduces the issue; instead, the system now redirects to the login main page.

```
else if (text3 == "LogOnAdmin")
{
    q q4 = new q(k);
    string text7 = string.Format("select * from admin_data where sAccount='{0:S}' and sPassword='{1:S}'", j.e((string)jobject2["sAccount"]
        ["sPassword"]));
    q4.g(text7);
    bool flag4 = !q4.c;
    if (flag4)
    {
        JObject jobject5 = new JObject();
        jobject5["sAccount"] = q4.b("sAccount");
        jobject5["sPassword"] = q4.b("sPassword");
        jobject5["sDateTime"] = q4.b("sDateTime");
        jobject["jData"] = jobject5;
        jobject["sFlag"] = "Yes";
        Form.s2 = "Yes";
    }
    else
    {
        jobject["sFlag"] = "AccountOrPasswordError";
        Form.s2 = "No";
    }
    q4.b();
```

3.2 Mobile Application technical findings

MED-APP-006 Insecure Cleartext Traffic Configuration

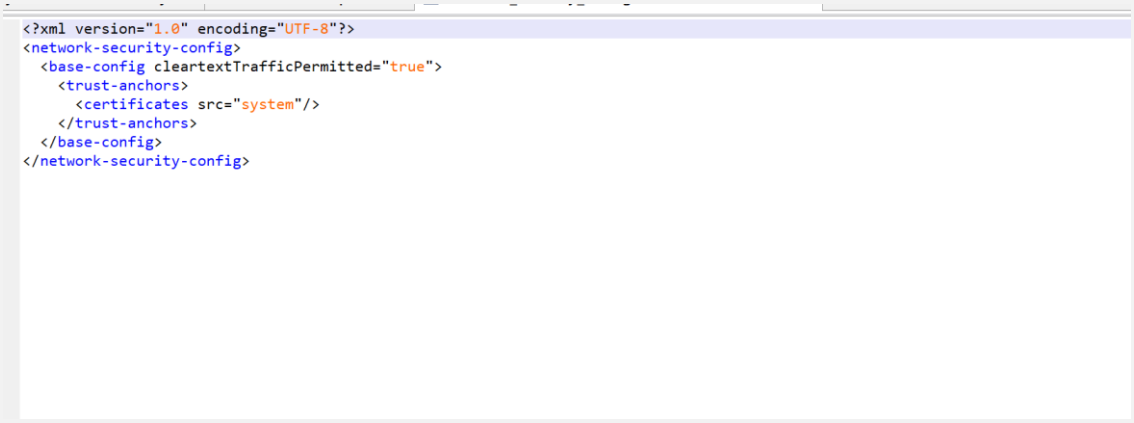
Description	The foundational configuration is insecurely set to permit plaintext traffic for all domains: <base-config cleartextTrafficPermitted="true" />.
Impacted System(s)	GLUCOJOY (Android)
Status	Verified

CVSSv3 Base Score

CVSSv3.1 Base Score Vector			
Attack Vector	Network	**Scope**	Unchanged
Attack Complexity	High	**Confidentiality**	High
Privileges Required	None	**Integrity**	High
User Interaction	None	**Availability**	None
Vector String:	**CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N**		
Base Score:	**7.4 (High)**		

Details of the Finding

The configuration <base-config cleartextTrafficPermitted="true" /> poses a significant security risk as it allows insecure, plaintext traffic for all domains. This could lead to potential data interception, unauthorized access, and compromise of sensitive information during transmission. It is essential to secure the communication by disallowing cleartext traffic to mitigate these risks.



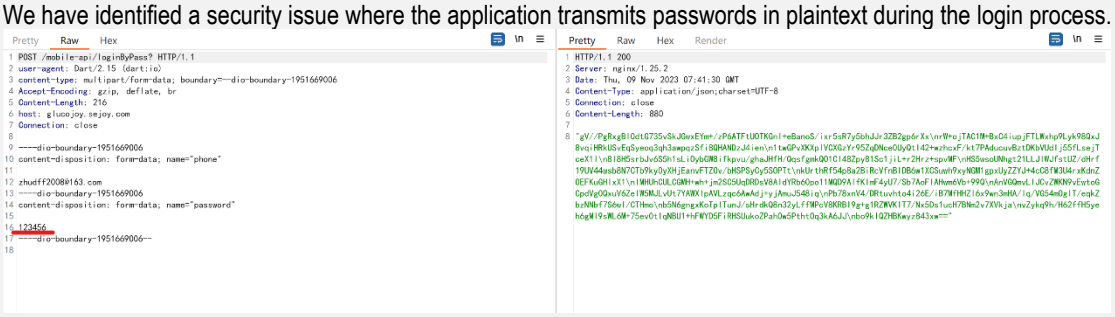
```
<?xml version="1.0" encoding="UTF-8"?>
<network-security-config>
  <base-config cleartextTrafficPermitted="true">
    <trust-anchors>
      <certificates src="system"/>
    </trust-anchors>
  </base-config>
</network-security-config>
```

Remediation	Disable cleartext traffic for enhanced security by configuring the <base-config> to disallow plaintext transmission, reducing the risk of data interception and unauthorized access.
Reference(s)	N/A
Retest	Following the remediation, the app has deactivated this field for enhanced security measures.

```
<?xml version="1.0" encoding="UTF-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="false">
    <domain includeSubdomains="true">sejoy.com</domain>
  </domain-config>
</network-security-config>
```

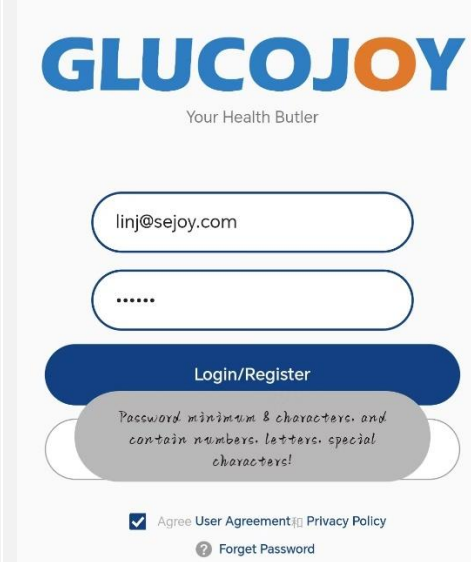
MED-APP-007 Unencrypted Password Transmission

Description	The transmission of passwords in plaintext represents a critical security vulnerability. It is imperative to ensure that passwords are consistently conveyed in an encrypted format, irrespective of the implementation of HTTPS.
Impacted System(s)	GLUCOJOY (Android)

Status	Verified
CVSSv3 Base Score	**CVSSv3.1 Base Score Vector** **Attack Vector**: Network; **Scope**: Unchanged **Attack Complexity**: Low; **Confidentiality**: Low **Privileges Required**: None; **Integrity**: None **User Interaction**: None; **Availability**: None **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N **Base Score:** 5.3 (Medium)
Details of the Finding	We have identified a security issue where the application transmits passwords in plaintext during the login process. 
Remediation	Passwords undergo supplementary encryption during the transmission process for enhanced security.
Reference(s)	N/A
Retest	Passwords now undergo additional encryption during transmission.

MED-APP-008 Weak Passwords

Description	The application lacks password complexity requirements for user passwords, allowing users to set weak passwords.
Impacted System(s)	GLUCOJOY (Android)
Status	Verified
CVSSv3 Base Score	**CVSSv3.1 Base Score Vector** **Attack Vector**: Network; **Scope**: Unchanged **Attack Complexity**: Low; **Confidentiality**: Low **Privileges Required**: None; **Integrity**: None **User Interaction**: None; **Availability**: None **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N **Base Score:** 5.3 (Medium)
Details of the Finding	Our analysis identified insufficient password complexity requirements enforced by the platform, allowing users to create low-entropy passwords. This finding indicates inadequate password policies that fall short of industry best practices for password strength. The lack of proper controls to mandate password complexity heightens the risk of brute force attacks and credential stuffing, leading to account takeovers.
Remediation	Implement password complexity requirements including minimum length of 8 characters, use of uppercase, lowercase, numbers and symbols.
Reference(s)	N/A
Retest	Post remediation, the application enforces stringent requirements for the complexity of user passwords, mandating a minimum length of 8 characters, inclusive of numbers, letters, and special characters.



MED-APP-009 No Root detection

Description	We have identified that the app permits execution within a root environment.
Impacted System(s)	GLUCOJOY (Android)
Status	Verified

CVSSv3 Base Score

CVSSv3.1 Base Score Vector			
Attack Vector	Local	**Scope**	Unchanged
Attack Complexity	High	**Confidentiality**	None
Privileges Required	None	**Integrity**	Low
User Interaction	None	**Availability**	None
Vector String:	**CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N**		
Base Score:	**2.9 (Low)**		

Details of the Finding

The necessity of root detection in the context of network security lies in mitigating potential risks associated with rooted or jailbroken devices. Rooted devices grant users elevated privileges, allowing them to bypass built-in security mechanisms. Consequently, apps running on rooted devices may face an increased risk of unauthorized access, data breaches, and malicious activities.

Remediation	Implement robust root detection mechanisms to enhance application security by preventing unauthorized access and potential exploits on rooted or jailbroken devices.
Reference(s)	N/A
Retest	After the remediation, the app has implemented root detection, causing it to terminate if the device is rooted.

4 APPENDIX

4.1 Testing Environment

4.1.1 Provided Credentials

User	Role(s)
demoHCP	pc software normal user
Zh*******08@163.com	mobile application normal user
l**j@sejoy.com	mobile application normal user

4.1.2 Conditions and Limitations

It is noted that the test was performed against a test environment.

Testers work under limitations of time and scope that may not apply to the activities of a malicious user or an attacker. They also work at a specific point in time and the threat profiles are dynamically evolving. It is therefore possible that vulnerabilities exist, or will arise, that were not identified during the test.

4.1.3 Tools Used

Name	Description	Version
Dependency Check	Open source code scanning Tool	8.0.2
Fiddler Classic	network debugging tool that logs all HTTP(S) network traffic.	5.0.0
Wireshark	Network Packet Analysis Software.	4.0.5
burpsuite_pro	Application Security Testing Software.	2022.2.2
Dnspy	.NET Programming Tool	6.4.1.0
DB Browser for SQLite	Database tool	3.12.2.0
Bluestack	Android emulator	5.11.1