BHP Mobile Wallet

Security Assessment

Initial Report: December 30th, 2020

First Revision: January 8th, 2021

For :
BHP Network

Confidentiality Statement

All information contained in this document is provided in confidence for the sole purpose of adjudication of the document and shall not be published or disclosed wholly or in part to any other party without CertiK's prior permission in writing and shall be held in safe custody. These obligations shall not apply to information that is published or becomes known legitimately from some source other than CertiK.
All transactions are subject to the appropriate CertiK Standard Terms and Conditions. Certain information given in connection with this proposal is marked “In Commercial Confidence”. That information is communicated in confidence, and disclosure of it to any person other than with CertiK’s consent will be a breach of confidence actionable on the part of CertiK.

Disclaimer

This document is provided for information purposes only. CertiK accepts no responsibility for any errors or omissions that it may contain.
This document is provided without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In no event shall CertiK be liable for any claim, damages or other liability (either direct or indirect or consequential), whether in an action of contract, tort or otherwise, arising from, out of or in connection with this document or the contents thereof.
This document represents our budgetary price proposal for the solution further described in this herein and is provided for information and evaluation purposes only and is not currently a formal offer capable of acceptance.

Overview

Scope

At the start of the engagement, CertiK worked with BHP Network to identify the target and set the limits on the scope of the test. A White Box type of testing approach was done where CertiK performed the test with the source code available from the public bhp-wallet GitHub repository.

Application Name	BHP mobile wallet
Codebase	GitHub Repository
Commits	507267efa3d788ce4c72821e4efeb3e534d9df16
Updated Commits	8d2c8ac4229d21b618264c38f4dff8ca854c179a

Audit Summary

Delivery Date	Dec. 30, 2020
Method of Audit	Static Analysis, Dynamic Testing
Consultants Engaged	2
Timeline	Dec. 22, 2020 - Dec. 30, 2020

Vulnerability Summary

Total Issues	9
Total Meduim	3
Total Low	3
Total Informational	3

Executive Summary

BHP Network engaged CertiK to perform an application penetration test for their mobile wallet. The main objective of the engagement is to test the overall resiliency of the application to various real-world attacks against the application’s controls and functions, and thereby be able to identify its weaknesses and provide recommendations to fix and improve its overall security posture.

CertiK started the test on December 22, 2020 and completed on December 30, 2020.

After a thorough review of the updated application, CertiK believes that the BHP application is currently at a low risk level. Given the severity of the vulnerabilities on the application, it is unlikely that the application will be directly compromised.

BHP has worked diligently to remediate security vulnerabilities discovered by CertiK, significantly increasing the overall security posture of their application. Note that because the mobile application is built with the "uni-app(https://uniapp.dcloud.io/)" framework, some findings can't be directly addressed. In the long term, we recommend the application switch to native apps to take advantage of native APIs to further secure the application.

Findings

ID	Title	Severity	Vulnerability Class	Status
BHP-01	Unnecessary App permissions	Medium	Security Misconfiguration	Fixed
BHP-02	Custom keyboards not disabled in BHP wallet	Medium	Information Disclosure	Partially Fixed
BHP-03	Information leakage via screenshot backgrounding	Low	Information Disclosure	Open
BHP-04	No root or jailbreak detection on Android/iOS application	Informational	Reverse Engineering and Code Tampering	Open
BHP-05	Ineffective and problematic "confirm password" implmentation	Low	Logic Flaws	Fixed
BHP-06	Insecure minimum Android SDK version	Informational	Security Misconfiguration	Fixed
BHP-07	Bad cryptography implementation	Medium	Insufficient Cryptography	Fixed
BHP-08	Mnemonic	Informational	Information	Open

	display allow screenshot		Disclosure	
BHP-09	Uses of insecure protocol	Low	Security Misconfiguration	Open

BHP-01: Unnecessary App permissions

Severity: Medium

Description

When analyzing the AndroidManifest.xml file in the Android APK, we found that the application requests access to an extensive amount of unnecessary permissions than it requires. Specifically, the application requests the following unnecessary permissions:

In AndroidManifest.xml

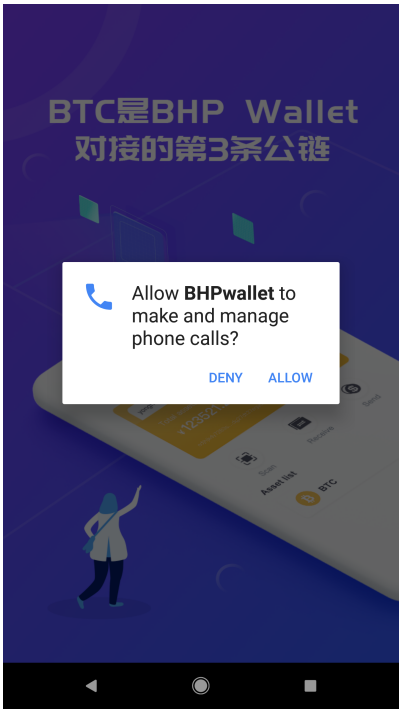
```
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"
/>
<uses-permission android:name="android.permission.READ_LOGS" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Impact

The "android:minSdkVersion=19" in AndroidManifest.xml indicates the minimal SDK version to run the application is 19. Android grants permissions to the application during installation on devices with SDK version lower than 23. If a user installs the application on such a device, the application will grant more permission than it should. Furthermore, when the BHP application is launched for the first time, the application requires users to allow the wallet to make and manage phone calls, as shown in the "Evidence" section. However, we found no features in the application that would require the "CALL_PHONE" permission.

Requesting permissions that are not needed by the application could negatively impact the application's user experience and may degrade the application's public reputation.

Evidence



Recommendation

Permission requests protect sensitive information available from a device and should only be used when access to information is necessary for the functioning of the application.

We recommend the following when dealing with Android permissions:

- Only require the permissions necessary for the application to work.
- While making a permissions request, be clear about what permission the application is accessing and why the application needs that permission.
- Provide continuous indications while accessing sensitive capabilities. Make it clear to the user when the application is collecting data and avoid the perception that the application is collecting data surreptitiously.

For more information about Android application permission and its best practices, please visit "https://developer.android.com/training/permissions/usage-notes"

BHP-02: Custom keyboards not disabled in BHP wallet

Severity: Medium

Description:

The BHP client does not disable custom keyboards. Since a long time ago, users have been able to install custom keyboards that can be used in any app, replacing the system's default keyboard. Custom keyboards can, and very frequently do, log and exfiltrate the data users enter.

The custom keyboard is allowed to use in the mnemonic input field in the BHP application. Malicious third-party keyboards can log and exfiltrate users' mnemonic. The attacker would then have total control over the victim's wallet account.

Exploit Scenario

Bob installed a custom keyboard that logs and exfiltrates data he enters on his device. Bob enters his wallet mnemonic with the custom keyboard, and his mnemonic can be collected by the third party keyboard.

Recommendation

We have no information regarding how the app artifact(APK and iPA) is built, as there is no build file in the Github repository. We are not able to provide specific recommendations to mitigate this issue. In general, third-party keyboards should be disabled in sensitive data input fields to prevent leakage of sensitive data entered by the user.

Reference:
https://developer.apple.com/documentation/uikit/uiapplication/extensionpointidentifier

BHP-03: Information leakage via screenshot backgrounding

Severity: Low

Intoduction

On mobile devices, a screenshot of the current activity is taken when an application goes into the background and displayed for aesthetic purposes when the app returns to the foreground. This feature may pose a security risk. Sensitive data may be exposed if the user background the application while sensitive data is displayed. A malicious application that is running on the device and able to continuously capture the screen may also expose data.

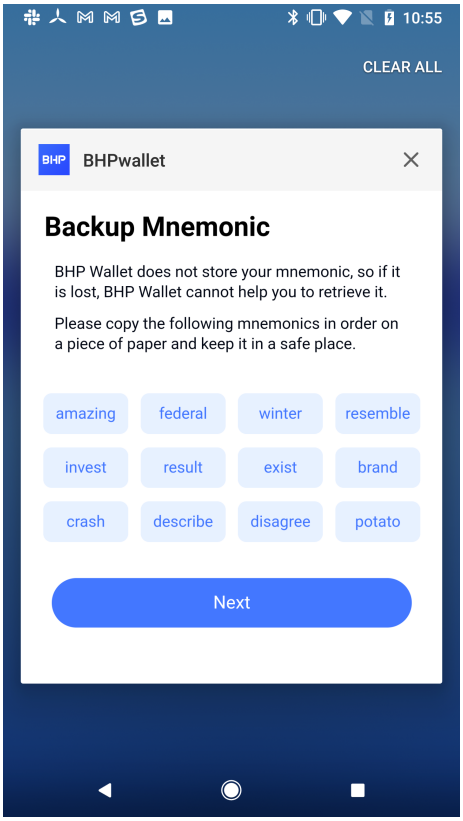
Description

The BHP application does not use an overlay to hide or obscure sensitive information such as wallet balance or the wallet mnemonic on the screen before moving to the background. Such information may be included in the auto-generated screenshot.

Impact

An attacker with physical access to the unlocked device or a malicious third-party app with access to the auto-generated screenshot of the BHP application can retrieve sensitive information included in the screenshot.

Evidence



Recommendation

It's recommended for the BHP application to add an overlay to hide or obscure the application screen before moving to the background.

For iOS, add an overlay screen before the application goes into the background, and remove the screen when the application goes into the foreground.

For Android, in addition to adding an overlay, this can be done by setting the FLAG_SECURE option. The FLAG_SECURE flag can prevent sensitive information included in the auto-generated screenshot.

For more information about the FLAG_SECURE flag, please see https://developer.android.com/reference/android/view/Display#FLAG_SECURE and https://stackoverflow.com/questions/9822076/how-do-i-prevent-android-taking-a-screenshot-when-my-app-goes-to-the-background

BHP-04: Ineffective and problematic "confirm password" implmentation

Severity: Low

Description:

The BHP wallet requires a user to set up a password while creating the wallet. The user needs to type the same password in both "Password" and "Repeat password" fields to pass the check. Users are not allowed to finish the wallet creation if two passwords are not the same. We found that the implementation is ineffective and problematic.

1. If users modify the value in the "Repeat password" field and then modify the value in the "Password" field, the validation will fail even if two passwords are the same.
2. If users enter the same password in the "Repeat password" field and the value in the "Password" field, then modify the value in the "Password" field, the application would let the user pass the check. However, the value in the "Password" field and "Repeat password" is different.

Location

Create wallet interface

Impact

The ineffective and problematic implementation can confuse users and allow users to create an unintended password. This would negatively influence the application's user experience.

Step to Reproduce

1. Click "Create Wallet" in BHP wallet and enter a valid value such as "111111aa" in both the "Password" and the "Repeat Password" field.
2. Change the password in the "Password" field to another password such as "22222aaa" and keep the "Repeat Password" field with "111111aa".
3. Click "Create wallet", and notice the application creates a wallet with the password "22222aaa".

Recommendation:

The application should perform the validation after users modify the value in either the "Password" field or the "Repeat Password" field. Make sure the users enter the same password in both fields.

BHP-05: No root or jailbreak detection on Android/iOS application

Severity: Informational

Description

No root or jailbreak detection is implemented in the application. This allows an attacker to modify the app on a rooted Android or jailbroken iOS device, which means the attacker can potentially induce behaviors that otherwise would not occur. Examples of the impact include accessing sensitive application data, overwriting critical functions, and an overall wider attack surface.

Impact

A malicious application with root permission can access and modify data belongs to the BHP application. Combined with the "unnecessary app permissions" issue above, a malicious third-party application can take advantage of the BHP app permissions and perform more privileged actions.

Step to Reproduce

Install and run the BHP wallet on a rooted Android or jailbroken iOS device.

Recommendation

Implement root and jailbreak detection at the beginning of the runtime of your application. Shut down the application or at least display a warning message when a user attempts to use the wallet on a rooted or jailbroken device.

Some methods to check for a rooted/jailbroken device are listed below:

Android

- Check for existing su binaries, such as:
    - /system/bin/su
    - /system/xbin/su
    - /sbin/su
    - /system/su
    - /system/bin/.ext/.su
- Attempt to use the su command directly and compare the current user ID before and after to see if the user was successfully upgraded to root.

- Utilize SafetyNet by Google. This was developed to try and detect device modifications and will prevent the application from running if the device fails the checks.

iOS

- Check for the existence of common files or directories that are associated with a jailbroken device. A list of some of the potential files to check can be seen here:
    - /Applications/Cydia.app
    - /Applications/FakeCarrier.app
    - /Applications/Icy.app
    - /usr/bin/sshd
- Attempt to write to a file in a location that is outside of the application's virtual sandbox. An example would be to try and write to the /private directory. This attempt would fail on a non-jailbroken device because the application does not have permission to access anything outside of its virtual sandbox, but if the application succeeds in writing to that location, then it can be deduced that it has root permissions, which means that the device is jailbroken.

BHP-06: Insecure minimum Android SDK version

Severity: Informational

Description

The `android:minSdkVersion` declared in AndroidManifest.xml is set to 19, which is Android 4.4. Older versions of Android SDK may contain security vulnerabilities that remain unpatched.

For example, Android versions below Android 5.1 (API level 22) have multiple vulnerabilities in SQLite that can cause another application or service to execute arbitrary SQL queries. Successful exploitation could result in arbitrary code execution in the context of the target application.

Location

AndroidManifest.xml

Impact

The application may be vulnerable to remote code execution if the application runs on Android version below Android 5.1 (API level 22) and the device has a malicious application installed. Furthermore, many other security vulnerabilities are already discovered and remain unpatched in an old version of the Android SDK. An attacker can take advantage of known vulnerabilities to compromise users' wallets.

Recommendation

Set `android:minSdkVersion` to at least 23, which is Android 6.0. According to Android Studio, about 85% of Android devices are Android 6.0 or newer. This can reduce security risks while not sacrificing the compatibility of the application.

BHP-07: Bad cryptography implementation

Severity: Medium

Description

The BHP application stores encrypted private keys and mnemonics in the device. The application implements encryption in the following way:
1. Generate the sha256 hash of the user password.
2. Use the AES-ECB algorithm to encrypt the hash from step 1 with the user password as the encryption key.
3. Use the AES-CBC algorithm to encrypt the private key and mnemonics with the ciphertext from step 2 as the encryption key and plaintext password as IV(initialization vector).

The encryption key derivation mechanism and the use of IV are confusing and improper.

Impact

A securely generated encryption key should be able to against dictionary attack and brute force attack to some degree. Performing the extra AES-ECB encryption does not extend the cost of time to brute force the password. If the attacker gets access to the encrypted wallet data, it's trivially to brute-force the password and decrypts the data.

Recommendation

We recommend using a key derivation function such as Scrypt and Argon2id to generate the encryption key. Password-based key derivation algorithms are designed to be slow, memory-hard and therefore will frustrate attempts at parallelization and brute-force password cracking.

The IV used in the AES-CBC encryption algorithms should be a unique and randomly generated string. The IV can be appended at the front of the ciphertext and store in the devices. When performing the decryption, simply obtain the IV from the beginning of the stored data and use it in the decryption.

For more information about Key derivation function, please see:
https://en.wikipedia.org/wiki/Key_derivation_function

BHP-08: Mnemonic display allow screenshot

Severity: Informational

Description

The mnemonic phrase is a list of words that can be used to recover a wallet account. Obtaining the mnemonic can potentially allow the attacker to gain full control of the wallet. The application warns users not to take screenshot before displaying wallet private keys or mnemonics. However, on an Android device, screenshot is still allowed while displaying mnemonics or private key.

Impact

Third-party apps with "READ_EXTERNAL_STORAGE" permission on an Android device or apps with all photo access on an iPhone can read screenshots on the device. Third-party apps can retrieve the mnemonic if the mnemonic is included in a screenshot taken by the user. Since the application displays warning to the user about not taking screenshots before showing private keys or mnemonics, we set the serverity of this issue to informational.

Step to Reproduce

1. Open the BHP wallet and create a new wallet.
2. Set up a name and password for the wallet, click "backup" to naviagte to "Backup Mnemonic" interface.
3. Take a screenshot including mnemonics.

Recommendation

Android

Screen capture can be prevented by setting the FLAG_SECURE option. The FLAG_SECURE flag can prevent users and malicious third-party apps from recording the mnemonic screens and taking screenshots of sensitive information.

For more information about the FLAG_SECURE flag, please see “https://developer.android.com/reference/android/view/Display#FLAG_SECURE”

BHP-09: Uses of insecure protocol

Severity: Low

Description

The communication protocol uses between the BHP mobile application and the server running on `rpc.bhpnet.io` and `47.244.216.20` is HTTP instead of HTTPS. Communications between the application and its server should be protected with TLS (Transport Layer Security).

Location

https://github.com/bhpnet/bhp-wallet/blob/507267efa3d788ce4c72821e4efeb3e534d9df16/src/api/api.js

```
export function getAccounts(address) {
    return axios.get("http://rpc.bhpnet.io/auth/accounts/" + address,
    { params: {} })
}

export function broadcastTrading(json) {
    return axios.post("http://rpc.bhpnet.io/txs", json)
}

export function sendBTC(stx) {
    return axios.post("http://47.244.216.20:16969/api/sendrawtransaction",
    {
        tx: stx
      })
}
```

Impact

An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. Furthermore, an attacker able to modify traffic could use the application as a platform for attacks against its users.

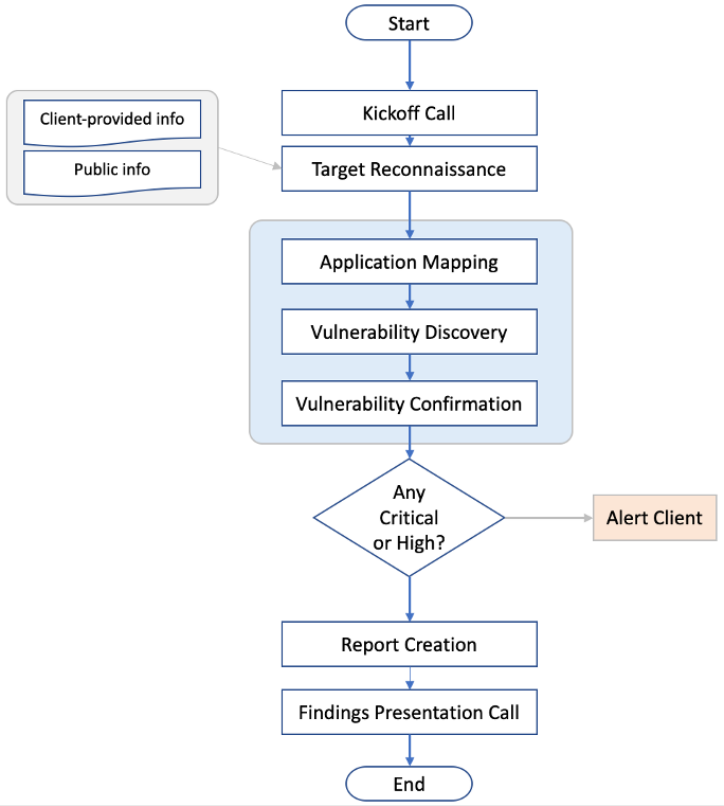
Recommendation

Applications should use transport-level encryption (SSL/TLS) to protect all communications between the client and the server.

Appendix – Methodology

CertiK uses a comprehensive penetration testing methodology which adheres to industry best practices and standards in security assessments including from OWASP (Open Web Application Security Project), NIST, PTES (Penetration Testing Execution Standard).

Below is a flowchart of our assessment process:



Coverage and Prioritization

As many components as possible will be tested manually. Priority is generally based on three factors: critical security controls, sensitive data, and the likelihood of vulnerability.

Critical security controls will always receive the top priority in the test. If a vulnerability is discovered in the critical security control, the entire application is likely to be compromised, resulting in a critical-risk to the business. For most applications, critical controls will include the login page, but it could also include major workflows such as the checkout function in an online store.

The Second priority is given to application components that handle sensitive data. This is dependent on business priorities, but common examples include payment card data, financial data, or authentication credentials.

Final priority includes areas of the application that are most likely to be vulnerable. This is based on CertiK' experience with similar applications developed using the same technology or with other applications that fit the same business role. For example, large applications will often have older sections that are less likely to utilize modern security techniques.

Reconnaissance

CertiK gathers information about the target application from various sources depending on the type of test being performed. CertiK obtains whatever information that is possible and appropriate from the client during scoping and supplements it with relevant information that can be gathered from public sources. This helps provide a better overall picture and understanding of the target.

Application Mapping

CertiK examines the application, reviewing its contents, and mapping out all its functionalities and components. CertiK makes use of different tools and techniques to traverse the entire application and document all input areas and processes. Automated tools are used to scan the application and it is then manually examined for all its parameters and functionalities. With this, CertiK creates and widens the overall attack surface of the target application.

Vulnerability Discovery

Using the information that is gathered, CertiK comes up with various attack vectors to test against the application. CertiK uses a combination of automated tools and manual techniques to identify vulnerabilities and weaknesses. Industry-recognized testing tools will be used, including Burp Suite, Nikto, Metasploit, and Kali. Furthermore, any controls in place that would inhibit the successful exploitation of a particular system will be noted.

Vulnerability Confirmation

After discovering vulnerabilities in the application, CertiK validates the vulnerabilities and assesses its overall impact. To validate, CertiK performs a Proof-of-Concept of an attack on the vulnerability, simulating real world scenarios to prove the risk and overall impact of the vulnerability.

Through CertiK' knowledge and experience on attacks and exploitation techniques, CertiK is able to process all weaknesses and examine how they can be combined to compromise the application. CertiK may use different attack chains, leveraging different weaknesses to escalate and gain a more significant compromise.

To minimize any potential negative impact, vulnerability exploitation was only attempted when it would not adversely affect production applications and systems, and then only to confirm the presence of a specific vulnerability. Any attack with the potential to cause system downtime or seriously impact business continuity was not performed. Vulnerabilities were never exploited to delete or modify data; only read-level access was attempted. If it appeared possible to modify data, this was noted in the list of vulnerabilities below.

Immediate escalation of High or Critical Findings

If critical or high findings are found whereby application elements are compromised, client's key security contacts will be notified immediately.

Vulnerability Classes

Insecure Data Storage	• Sensitive Data Store in Plant Text • Use of Public Storage • Logging sensitive data
Information Disclosure	• Directory Indexing • Verbose Error Messages • HTML CommentsDefault Content
Account Policy	• Default / Weak Passwords • Unlimited Login Attempts • Password Reset • Insufficient Session Expiration
Session Management	• Session Identifier PredictionSession Hijacking • Cross-Site Request Forgery
Injection	• SQL Injection • Cross-Site Scripting • HTML InjectionXML Injection • OS Command Injection
Broken Access Control	• Authentication Bypass • Authorization Bypass • Privilege Escalation • Insecure Inter-Process Communication(intents, sockets)
Application Resource Handling	• Path Traversal • Predictable Object Identifiers • XML External Entity Expansion • Local & Remote File Inclusion
Logic Flaws	• Abuse of FunctionalityWorkflow Bypass
Insufficient Cryptography	• Weak Hashing Algorithms • Weak Encryption Algorithms • Hard Coded Cryptographic Key

Security Misconfiguration	• Missing Security Headers • Debugging Enabled
Reverse Engineering and Code Tampering	• Lack of Root Detection • Lack of Tampering Detection • Lack of Code Obfuscation

Risk Assessment

The following risk levels categorize the risk level of issues presented in the report:

Risk Level	CVSS Score	Impact	Exploitability
Critical	9.0-10.0	Root-level or full-system compromise, large-scale data breach	Trivial and straightforward
High	7.0-8.9	Elevated privilege access, significant data loss or downtime	Easy, vulnerability details or exploit code are publicly available, but may need additional attack vectors (e.g., social engineering)
Medium	4.0-6.9	Limited access but can still cause loss of tangible assets, which may violate, harm, or impede the org's mission, reputation, or interests.	Difficult, requires a skilled attacker, needs additional attack vectors, attacker must reside on the same network, requires user privileges
Low	0.1-3.9	Very little impact on an org’s business	Extremely difficult, requires local or physical system access
Informational	0.0	Discloses information that may be of interest to an attacker.	Not exploitable but rather is a weakness that may be useful to an attacker should a higher risk issue be

found that allows for
a system exploit