



Veracode Detailed Report  
**Application Security Report**  
**As of 16 Mar 2025**

Prepared for:	AIA Company Limited
Prepared on:	March 16, 2025
Application:	AIA HK - Agency Corner
Sandbox:	Integrator Sandbox
Industry:	Not Specified
Business Criticality:	BC4 (High)
Required Analysis:	Static
Type(s) of Analysis Conducted:	Static
Scope of Static Scan:	2 of 2 Modules Analyzed

#### Inside This Report

Executive Summary	1
Summary of Flaws by Severity	1
Action Items	1
Flaw Types by Category	3
Policy Summary	5
Findings & Recommendations	6
Methodology	

*While every precaution has been taken in the preparation of this document, Veracode, Inc. assumes no responsibility for errors, omissions, or for damages resulting from the use of the information herein. The Veracode platform uses static and/or dynamic analysis techniques to discover potentially exploitable flaws. Due to the nature of software security testing, the lack of discoverable flaws does not mean the software is 100% secure.*



## Veracode Detailed Report Application Security Report As of 16 Mar 2025

**Veracode Level: VL2**

Rated: Mar 16, 2025

Application: AIA HK - Agency Corner  
Target Level: None

Business Criticality: High  
Published Rating: B

### Scans Included in Report

Static Scan	Dynamic Scan	Manual Penetration Test
app-uat-debug.apk_16 Mar 2025 Static Score: 76 Completed: 3/16/25	Not Included in Report	Not Included in Report

### Executive Summary

This report contains a summary of the security flaws identified in the application using manual penetration testing, automated static and/or automated dynamic security analysis techniques. This is useful for understanding the overall security quality of an individual application or for comparisons between applications.

#### Application Business Criticality: BC4 (High)

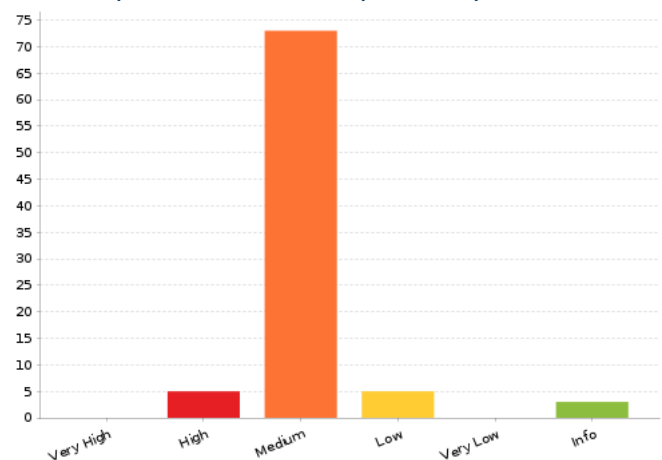
Impacts: Operational Risk (Medium), Financial Loss (Medium)

An application's business criticality is determined by business risk factors such as: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations. The Veracode Level and required assessment techniques are selected based on the policy assigned to the application.

#### Analyses Performed vs. Required

	Any	Static	Dynamic	Manual Penetration Test
Performed:		●	○	○
Required:	○	●	○	○

#### Summary of Flaws Found by Severity



### Action Items:

Veracode recommends the following approaches ranging from the most basic to the strong security measures that a vendor can undertake to increase the overall security level of the application.

#### Flaws To Fix For Minimum Score

- A rule in your policy requires a minimum score of 80. Fix 5 High flaws and 2 Medium flaws to reach a score of 80.

#### Flaw Severities

- Medium severity flaws and above must be fixed for policy compliance.

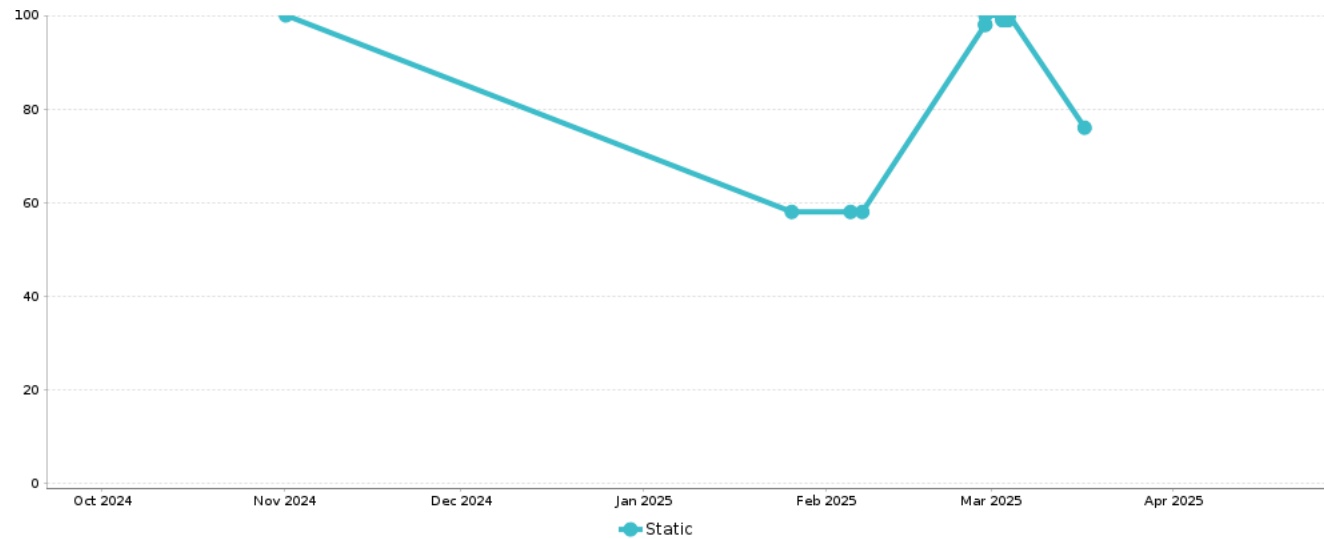


#### Longer Timeframe (6 - 12 months)

- Certify that software engineers have been trained on application security principles and practices.



Application Trend Data



Scope of Static Scan

The following modules were included in the static scan because the scan submitter selected them as entry points, which are modules that accept external data.

Engine Version: 20250312184913

The following modules were included in the application scan:

Module Name	Compiler	Operating Environment	Engine Version
app-uat-debug.apk	Android	Android	20250312184913
Javascript files within app-uat-debug.apk	JAVASCRIPT_5_1	JavaScript	20250312184913

**File Differences Between Scans**

The uploaded modules for this scan do not match the modules you uploaded for the previous scan. This disparity can affect the scan results even if Veracode did not find flaws in the files with differences. See appendix for more details.

Flaw Types by Severity and Category

Static Scan Security Quality Score = 76 (-24) from prior scan			
Very High	0		
High	5 (+5)		
Credentials Management	3 (+3)		
Cryptographic Issues	2 (+2)		
Medium	73 (+73)		
Authorization Issues	1 (+1)		
CRLF Injection	11 (+11)		
Cross-Site Scripting (XSS)	4 (+4)		



Static Scan Security Quality Score = 76 (-24) from prior scan			
Cryptographic Issues	33	(+33)	
Dangerous Functions	3	(+3)	
Deployment Configuration	5	(+5)	
Directory Traversal	3	(+3)	
Information Leakage	1	(+1)	
Insufficient Input Validation	9	(+9)	
Server Configuration	2	(+2)	
Time and State	1	(+1)	
<b>Low</b>	<b>5</b>	<b>(+5)</b>	
Information Leakage	5	(+5)	
<b>Very Low</b>	<b>0</b>		
<b>Informational</b>	<b>3</b>	<b>(+3)</b>	
Code Quality	2	(+2)	
Potential Backdoor	1	(+1)	
<b>Total</b>	<b>86</b>	<b>(+86)</b>	



## Policy Evaluation

Policy Name: AIA AppSecurity

Revision: 9

Policy Status: Not Assessed

Description: Policy defined to achieve identification and remediation of all Very High, High and Medium severity issues.

### Rules

Rule type	Requirement	Findings	Status
<b>Min Analysis Score</b>	80	76	Did not pass
<b>Max Severity</b>	Medium	Flaws found: 78	Did not pass

### Software Composition Analysis Rules

Rule type	Requirement	Findings	Status
<b>Disallow Vulnerabilities by Severity</b>	Medium and Above Not Allowed	1 Component	Did not pass
<b>Disallow Component Blocklist</b>	Prevent an application from passing policy if blocklisted components are detected	0 Blocklisted	Passed



## Findings & Recommendations

### Detailed Flaws by Severity

#### Very High (0 flaws)

No flaws of this type were found

#### High (5 flaws)

 **Fix Required by Policy**

#### → Credentials Management(3 flaws)

##### Description

Improper management of credentials, such as usernames and passwords, may compromise system security. In particular, storing passwords in plaintext or hard-coding passwords directly into application code are design issues that cannot be easily remedied. Not only does embedding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software. If a hard-coded password is compromised in a commercial product, all deployed instances may be vulnerable to attack, putting customers at risk.

One variation on hard-coding plaintext passwords is to hard-code a constant string which is the result of a cryptographic one-way hash. For example, instead of storing the word "secret," the application stores an MD5 hash of the word. This is a common mechanism for obscuring hard-coded passwords from casual viewing but does not significantly reduce risk. However, using cryptographic hashes for data stored outside the application code can be an effective practice.

##### Recommendations

Avoid storing passwords in easily accessible locations, and never store any type of sensitive data in plaintext. Avoid using hard-coded usernames, passwords, or hash constants whenever possible, particularly in relation to security-critical components. Store passwords out-of-band from the application code. Follow best practices for protecting credentials stored in alternate locations such as configuration or properties files.

#### Associated Flaws by CWE ID:




#### → Use of Hard-coded Credentials (CWE ID 798)(3 flaws)

##### Description

The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

*Effort to Fix:* 4 - Simple design error. Requires redesign and up to 5 days to fix.

##### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	 530119	23	-	Javascript files within app-uat-debug.apk	/assets/www/js/function.js 4147	3/30/25
NEW	 530190	31	-	app-uat-debug.apk	com/.../config/ProductConfig.java 1	3/30/25
NEW	 530191	31	-	app-uat-debug.apk	com/.../config/ProductConfig.java 1	3/30/25

## → Cryptographic Issues(2 flaws)

### Description

Applications commonly use cryptography to implement authentication mechanisms and to ensure the confidentiality and integrity of sensitive data, both in transit and at rest. The proper and accurate implementation of cryptography is extremely critical to its efficacy. Configuration or coding mistakes as well as incorrect assumptions may negate a large degree of the protection it affords, leaving the crypto implementation vulnerable to attack.

Common cryptographic mistakes include, but are not limited to, selecting weak keys or weak cipher modes, unintentionally exposing sensitive cryptographic data, using predictable entropy sources, and mismanaging or hard-coding keys.

Developers often make the dangerous assumption that they can improve security by designing their own cryptographic algorithm; however, one of the basic tenets of cryptography is that any cipher whose effectiveness is reliant on the secrecy of the algorithm is fundamentally flawed.

### Recommendations

Select the appropriate type of cryptography for the intended purpose. Avoid proprietary encryption algorithms as they typically rely on "security through obscurity" rather than sound mathematics. Select key sizes appropriate for the data being protected; for high assurance applications, 256-bit symmetric keys and 2048-bit asymmetric keys are sufficient. Follow best practices for key storage, and ensure that plaintext data and key material are not inadvertently exposed.

### Associated Flaws by CWE ID:

## → Use of Hard-coded Cryptographic Key (CWE ID 321)(2 flaws)

### Description



A method uses a hard-coded cryptographic key that may compromise system security in a way that cannot be easily remedied. The use of a hard-coded key significantly increases the possibility that encrypted data may be recovered. Moreover, the key cannot be changed without patching the software. If a hard-coded key is compromised in a commercial product, all deployed instances may be vulnerable to attack.

*Effort to Fix:* 4 - Simple design error. Requires redesign and up to 5 days to fix.

### Recommendations

Store encryption keys out-of-band from the application code. Follow best practices for protecting keys stored in locations such as configuration or properties files.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	 530120	11	-	Javascript files within app-uat-debug.apk	/assets/www/js/chatBot.js 1	3/30/25
NEW	 530117	23	-	Javascript files within app-uat-debug.apk	/assets/www/js/function.js 93	3/30/25





## Medium (73 flaws)

**Fix Required by Policy** **Authorization Issues(1 flaw)****Description**

Authorization is the process or method by which an application determines whether a user, service, or application has the necessary permissions to perform a requested action. Web applications often restrict access to specific content or functionality based on the user's role or privilege level. If authorization is not implemented properly, an attacker can manipulate a web site to gain access to data or functionality that should be protected.

Authorization should not be confused with authentication. Authentication is the process of verifying a user's identity, while authorization enforces what that user is permitted to do after they have successfully authenticated to the system.

**Recommendations**

Be sure that authorization is properly enforced at the server side for every action. Centralize authorization routines when possible. Follow the principle of least privilege when designing security controls.

**Associated Flaws by CWE ID:** **User Interface (UI) Misrepresentation of Critical Information (CWE ID 451)(1 flaw)****Description**

The application must define the Manifest.permission.HIDE\_OVERLAY\_WINDOWS permission and call Window#setHideOverlayWindows(true) to prevent 3rd party applications from being able to draw over it. Ability to obscure an application view partially or entirely may lead to the end-user being tricked into executing unwanted or dangerous actions.

*Effort to Fix:* 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

**Recommendations**

Ensure that the application sets the Manifest.permission.HIDE\_OVERLAY\_WINDOWS permission in AndroidManifest.xml as well as invokes Window#setHideOverlayWindows(true) early during initialization. This will protect end-user from malicious applications that may be attempting to deceive the end-user.

**Instances found via Static Scan**

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530150	39	-	app-uat-debug.apk	com/aia/SplashScreen.java 1	6/14/25

**CRLF Injection(11 flaws)****Description**

The acronym CRLF stands for "Carriage Return, Line Feed" and refers to the sequence of characters used to denote the end of a line of text. CRLF injection vulnerabilities occur when data enters an application from an untrusted source and is not properly validated before being used. For example, if an attacker is able to inject a CRLF into a log file, he could append falsified log entries, thereby misleading administrators or cover traces of the attack. If an attacker is able to inject CRLFs into an HTTP response header, he can use this ability to carry out other attacks such as cache poisoning. CRLF vulnerabilities primarily affect data integrity.



## Recommendations

Apply robust input filtering for all user-supplied data, using centralized data validation routines when possible. Use output filters to sanitize all output derived from user-supplied input, replacing non-alphanumeric characters with their HTML entity equivalents.

### Associated Flaws by CWE ID:

#### → Improper Output Neutralization for Logs (CWE ID 117)(11 flaws)

#### Description

A function call could result in a log forging attack. Writing untrusted data into a log file allows an attacker to forge log entries or inject malicious content into log files. Corrupted log files can be used to cover an attacker's tracks or as a delivery mechanism for an attack on a log viewing or processing utility. For example, if a web administrator uses a browser-based utility to review logs, a cross-site scripting attack might be possible.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

#### Recommendations

Avoid directly embedding user input in log files when possible. Sanitize untrusted data used to construct log entries by using a safe logging mechanism such as the OWASP ESAPI Logger, which will automatically remove unexpected carriage returns and line feeds and can be configured to use HTML entity encoding for non-alphanumeric data. Only write custom blacklisting code when absolutely necessary. Always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

#### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530132	7	-	app-uat-debug.apk	me/.../cordova/AppPreferences.java 48	6/14/25
NEW	530160	14	-	app-uat-debug.apk	org/.../auth/DefaultLogger.kt 105	6/14/25
NEW	530161	14	-	app-uat-debug.apk	org/.../auth/DefaultLogger.kt 109	6/14/25
NEW	530162	14	-	app-uat-debug.apk	org/.../auth/DefaultLogger.kt 113	6/14/25
NEW	530163	14	-	app-uat-debug.apk	org/.../auth/DefaultLogger.kt 116	6/14/25
NEW	530121	17	-	Javascript files within app-uat-debug.apk	/.../www/enroll_details.html 280	6/14/25
NEW	530189	22	-	app-uat-debug.apk	com/aia/fr/FRUtils.java 53	6/14/25
NEW	530116	24	-	Javascript files within app-uat-debug.apk	/assets/www/js/keyStore.js 16	6/14/25
NEW	530126	27	-	Javascript files within app-uat-debug.apk	/.../www/ops_inbox_details.html 198	6/14/25
NEW	530201	37	-	app-uat-debug.apk	com/.../aia/SmartWebActivity.java 294	6/14/25
NEW	530192	42	-	app-uat-debug.apk	com/aia/WebActivity.java 264	6/14/25

## → Cross-Site Scripting (XSS)(4 flaws)

### Description

Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed occur whenever a web application uses untrusted data in the output it generates without validating or encoding it. XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise sensitive information, with new attack vectors being discovered on a regular basis. XSS is also commonly referred to as HTML injection.

XSS vulnerabilities can be either persistent or transient (often referred to as stored and reflected, respectively). In a persistent XSS vulnerability, the injected code is stored by the application, for example within a blog comment or message board. The attack occurs whenever a victim views the page containing the malicious script. In a transient XSS vulnerability, the injected code is included directly in the HTTP request. These attacks are often carried out via malicious URLs sent via email or another website and requires the victim to browse to that link. The consequence of an XSS attack to a victim is the same regardless of whether it is persistent or transient; however, persistent XSS vulnerabilities are likely to affect a greater number of victims due to its delivery mechanism.

### Recommendations

Several techniques can be used to prevent XSS attacks. These techniques complement each other and address security at different points in the application. Using multiple techniques provides defense-in-depth and minimizes the likelihood of a XSS vulnerability.

- \* Use output filtering to sanitize all output generated from user-supplied input, selecting the appropriate method of encoding based on the use case of the untrusted data. For example, if the data is being written to the body of an HTML page, use HTML entity encoding. However, if the data is being used to construct generated Javascript or if it is consumed by client-side methods that may interpret it as code (a common technique in Web 2.0 applications), additional restrictions may be necessary beyond simple HTML encoding.
- \* Validate user-supplied input using positive filters (white lists) to ensure that it conforms to the expected format, using centralized data validation routines when possible.
- \* Do not permit users to include HTML content in posts, notes, or other data that will be displayed by the application. If users are permitted to include HTML tags, then carefully limit access to specific elements or attributes, and use strict validation filters to prevent abuse.

### Associated Flaws by CWE ID:

## → Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (CWE ID 80)(4 flaws)

### Description

This call contains a cross-site scripting (XSS) flaw. The application populates the HTTP response with untrusted input, allowing an attacker to embed malicious content, such as Javascript code, which will be executed in the context of the victim's browser. XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise confidential information, with new attack vectors being discovered on a regular basis.

*Effort to Fix:* 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

### Recommendations



Use contextual escaping on all untrusted data before using it to construct any portion of an HTTP response. The escaping method should be chosen based on the specific use case of the untrusted data, otherwise it may not protect fully against the attack. For example, if the data is being written to the body of an HTML page, use HTML entity escaping; if the data is being written to an attribute, use attribute escaping; etc. When a web framework provides built-in support for automatic XSS escaping, do not disable it. Both the OWASP Java Encoder library for Java and the Microsoft AntiXSS library provide contextual escaping methods. For more details on contextual escaping, see [https://cheatsheetseries.owasp.org/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html). In addition, as a best practice, always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530122	40	-	Javascript files within app-uat-debug.apk	/assets/www/survey.html 309	6/14/25
NEW	530123	40	-	Javascript files within app-uat-debug.apk	/assets/www/survey.html 441	6/14/25
NEW	530124	40	-	Javascript files within app-uat-debug.apk	/assets/www/survey.html 525	6/14/25
NEW	530125	40	-	Javascript files within app-uat-debug.apk	/assets/www/survey.html 587	6/14/25

## → Cryptographic Issues(33 flaws)

### Description

Applications commonly use cryptography to implement authentication mechanisms and to ensure the confidentiality and integrity of sensitive data, both in transit and at rest. The proper and accurate implementation of cryptography is extremely critical to its efficacy. Configuration or coding mistakes as well as incorrect assumptions may negate a large degree of the protection it affords, leaving the crypto implementation vulnerable to attack.

Common cryptographic mistakes include, but are not limited to, selecting weak keys or weak cipher modes, unintentionally exposing sensitive cryptographic data, using predictable entropy sources, and mismanaging or hard-coding keys.

Developers often make the dangerous assumption that they can improve security by designing their own cryptographic algorithm; however, one of the basic tenets of cryptography is that any cipher whose effectiveness is reliant on the secrecy of the algorithm is fundamentally flawed.

### Recommendations

Select the appropriate type of cryptography for the intended purpose. Avoid proprietary encryption algorithms as they typically rely on "security through obscurity" rather than sound mathematics. Select key sizes appropriate for the data being protected; for high assurance applications, 256-bit symmetric keys and 2048-bit asymmetric keys are sufficient. Follow best practices for key storage, and ensure that plaintext data and key material are not inadvertently exposed.

### Associated Flaws by CWE ID:



## → Cleartext Storage of Sensitive Information in Memory (CWE ID 316)(2 flaws)

### Description

The application reads and/or stores sensitive information (such as passwords) unencrypted in memory, leaving it susceptible to compromise or erroneous exposure. An attacker with access to the system running the application may be able to obtain access to this sensitive data by examining core dumps and swap files, or by attaching to the running process with a debugger and searching mapped memory pages. Unless memory is explicitly overwritten, the sensitive information will persist until it is garbage collected and reallocated for other purposes.

*Effort to Fix:* 4 - Simple design error. Requires redesign and up to 5 days to fix.

### Recommendations

Try to avoid storing sensitive data in plaintext. When possible, always clear sensitive data after use by explicitly zeroing out the memory. In languages that do not provide a mechanism for zeroing out memory, such as Java or C#, focus on minimizing the risk rather than eliminating it. Try to avoid using immutable types when handling sensitive information (for example, use a character array rather than a String). Keep the time window in which sensitive information is present in memory as short as possible to minimize the likelihood of it being swapped to disk.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530130	28	-	app-uat-debug.apk	.../PasswordDialogPlugin.java 309	6/14/25
NEW	530131	28	-	app-uat-debug.apk	.../PasswordDialogPlugin.java 313	6/14/25

## → Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) (CWE ID 338)(1 flaw)

### Description

The product uses a Pseudo-Random Number Generator (PRNG) in a security context, but the PRNG is not cryptographically strong.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

### Recommendations

If this random number is used where security is a concern, such as generating a session identifier or cryptographic key, use a trusted cryptographic random number generator instead.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530183	4	-	app-uat-debug.apk	com/.../aia/network/AESUtil.java 57	6/14/25



## → Use of Password Hash With Insufficient Computational Effort (CWE ID 916)(5 flaws)

### Description

The software generates a hash for a password, but it uses a scheme that does not provide a sufficient level of computational effort that would make password cracking attacks infeasible or expensive.

*Effort to Fix:* 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

### Recommendations

Use an iteration count of at least 200,000 for any password-based key derivation functions.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530151	3	-	app-uat-debug.apk	com/.../AES256Concrete.java 14	6/14/25
NEW	530184	4	-	app-uat-debug.apk	com/.../aia/network/AESUtil.java 78	6/14/25
NEW	530139	16	-	app-uat-debug.apk	com/.../utils/EncryptUtil.java 1	6/14/25
NEW	530135	41	-	app-uat-debug.apk	com/.../utils/ToolsUtils.java 60	6/14/25
NEW	530137	41	-	app-uat-debug.apk	com/.../utils/ToolsUtils.java 239	6/14/25

## → Use of RSA Algorithm without OAEP (CWE ID 780)(5 flaws)

### Description

The software uses the RSA algorithm but does not incorporate Optimal Asymmetric Encryption Padding (OAEP), which might weaken the encryption.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

### Recommendations

Use OAEP padding scheme when using RSA algorithm for encryption/decryption.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530166	8	-	app-uat-debug.apk	.../AsymmetricEncryptor.java 54	6/14/25
NEW	530167	8	-	app-uat-debug.apk	.../AsymmetricEncryptor.java 75	6/14/25
NEW	530154	18	-	app-uat-debug.apk	com/.../handle/FingerHandle.java 1	6/14/25
NEW	530155	18	-	app-uat-debug.apk	com/.../handle/FingerHandle.java 19	6/14/25
NEW	530156	18	-	app-uat-debug.apk	com/.../handle/FingerHandle.java 993	6/14/25



## → Use of a Broken or Risky Cryptographic Algorithm (CWE ID 327)(20 flaws)

### Description

The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the disclosure of sensitive information.

*Effort to Fix:* 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530187	1	-	app-uat-debug.apk	.../AbstractSymmetricEncryptor.java 91	6/14/25
NEW	530153	2	-	app-uat-debug.apk	.../AbstractYESSafeCommon.java 2	6/14/25
NEW	530146	3	-	app-uat-debug.apk	com/.../AES256Concrete.java 1	6/14/25
NEW	530147	3	-	app-uat-debug.apk	com/.../AES256Concrete.java 9	6/14/25
NEW	530152	3	-	app-uat-debug.apk	com/.../AES256Concrete.java 14	6/14/25
NEW	530181	4	-	app-uat-debug.apk	com/.../aia/network/AESUtil.java 41	6/14/25
NEW	530182	4	-	app-uat-debug.apk	com/.../aia/network/AESUtil.java 49	6/14/25
NEW	530185	4	-	app-uat-debug.apk	com/.../aia/network/AESUtil.java 78	6/14/25
NEW	530186	4	-	app-uat-debug.apk	com/.../aia/network/AESUtil.java 92	6/14/25
NEW	530133	10	-	app-uat-debug.apk	helper/Calculator.java 73	6/14/25
NEW	530164	13	-	app-uat-debug.apk	org/.../android/auth/CryptoKey.kt 62	6/14/25
NEW	530165	13	-	app-uat-debug.apk	org/.../android/auth/CryptoKey.kt 64	6/14/25
NEW	530159	15	-	app-uat-debug.apk	org/.../auth/DeviceIdentifier.java 64	6/14/25
NEW	530140	16	-	app-uat-debug.apk	com/.../utils/EncryptUtil.java 7	6/14/25
NEW	530144	16	-	app-uat-debug.apk	com/.../utils/EncryptUtil.java 13	6/14/25
NEW	530176	19	-	app-uat-debug.apk	com/.../FingerprintAuth.java 513	6/14/25
NEW	530145	26	-	app-uat-debug.apk	com/.../vccard/algorithm/OCRA.java 1	6/14/25
NEW	530138	34	-	app-uat-debug.apk	com/.../utils/SecureStorage.java 4	6/14/25
NEW	530134	41	-	app-uat-debug.apk	com/.../utils/ToolsUtils.java 1	6/14/25
NEW	530136	41	-	app-uat-debug.apk	com/.../utils/ToolsUtils.java 208	6/14/25



## → Dangerous Functions(3 flaws)

### Description

Certain functions behave in dangerous ways regardless of how they are used. Functions in this category were often implemented without taking security concerns into account. For example, the `gets()` function is unsafe because it does not perform bounds checking on the size of its input, nor does it provide a mechanism for specifying a maximum size.

### Recommendations

Most inherently dangerous functions were implemented before application security was a prevalent concern and have since been replaced by safer alternatives. In rare cases where no safe alternative exists, wrap the dangerous function with an appropriate conditional such as a boundary check before calling it.

### Associated Flaws by CWE ID:




## → Use of Potentially Dangerous Function (CWE ID 676)(3 flaws)

### Description

The program invokes a potentially dangerous function that could introduce a vulnerability if it is used incorrectly, but the function can also be used safely.

*Effort to Fix:* 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	 530174	9	-	app-uat-debug.apk	.../BaseFingerprintUiHelper.java 14	6/14/25
NEW	 530157	20	-	app-uat-debug.apk	.../FingerprintUiHelper.java 8	6/14/25
NEW	 530175	21	-	app-uat-debug.apk	.../FingerprintUiHelper.java 90	6/14/25

## → Deployment Configuration(5 flaws)

### Description

A deployment descriptor is a component in J2EE applications that describes how a web application should be deployed. It directs a deployment tool to deploy a module or application with specific container options and describes specific configuration requirements that a deployer must resolve. Deployment descriptors are part of the packaged application and are usually formatted as XML files. Failing to properly lock down the `web.xml` file or other files that define components and operating parameters for the web application can reduce the security of the application.

### Recommendations

Remove all unnecessary functionality from deployment descriptors, including servlets, servlet mappings, extension mappings, etc. Only elements that are required for the application to function should remain.

### Associated Flaws by CWE ID:





## → Improper Export of Android Application Components (CWE ID 926)(5 flaws)

### Description

Various issues related to configuration in AndroidManifest.xml, please see Triage Flaws in the Veracode Platform for more details.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530180	5	-	app-uat-debug.apk	.../AIAFcmListenerService.java 1	6/14/25
NEW	530179	6	-	app-uat-debug.apk	.../AIAInstanceIdListenerService.java 1	6/14/25
NEW	530129	35	-	app-uat-debug.apk	.../ShareChooserPendingIntent.java 1	6/14/25
NEW	530142	36	-	app-uat-debug.apk	com/.../aia/ShutDownReceiver.java 1	6/14/25
NEW	530200	39	-	app-uat-debug.apk	com/aia/SplashScreen.java 1	6/14/25

## → Directory Traversal(3 flaws)

### Description

Allowing user input to control paths used in filesystem operations may enable an attacker to access or modify otherwise protected system resources that would normally be inaccessible to end users. In some cases, the user-provided input may be passed directly to the filesystem operation, or it may be concatenated to one or more fixed strings to construct a fully-qualified path.

When an application improperly cleanses special character sequences in user-supplied filenames, a path traversal (or directory traversal) vulnerability may occur. For example, an attacker could specify a filename such as "../../../etc/passwd", which resolves to a file outside of the intended directory that the attacker would not normally be authorized to view.

### Recommendations

Assume all user-supplied input is malicious. Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters and ensure that the end result is not dangerous.

### Associated Flaws by CWE ID:

## → External Control of File Name or Path (CWE ID 73)(3 flaws)

### Description

This call contains a path manipulation flaw. The argument to the function is a filename constructed using untrusted input. If an attacker is allowed to specify all or part of the filename, it may be possible to gain unauthorized access to files on the server, including those outside the webroot, that would be normally be inaccessible to end users. The level of exposure depends on the effectiveness of input validation routines, if any.




*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.



## Recommendations

Validate all untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	 530188	38	-	app-uat-debug.apk	nl/.../plugins/SocialSharing.java 169	6/14/25
NEW	 530127	38	-	app-uat-debug.apk	nl/.../plugins/SocialSharing.java 285	6/14/25
NEW	 530128	38	-	app-uat-debug.apk	nl/.../plugins/SocialSharing.java 766	6/14/25

## → Information Leakage(1 flaw)

### Description

An information leak is the intentional or unintentional disclosure of information that is either regarded as sensitive within the product's own functionality or provides information about the product or its environment that could be useful in an attack. Information leakage issues are commonly overlooked because they cannot be used to directly exploit the application. However, information leaks should be viewed as building blocks that an attacker uses to carry out other, more complicated attacks.

There are many different types of problems that involve information leaks, with severities that can range widely depending on the type of information leaked and the context of the information with respect to the application. Common sources of information leakage include, but are not limited to:

- \* Source code disclosure
- \* Browsable directories
- \* Log files or backup files in web-accessible directories
- \* Unfiltered backend error messages
- \* Exception stack traces
- \* Server version information
- \* Transmission of uninitialized memory containing sensitive data

### Recommendations

Configure applications and servers to return generic error messages and to suppress stack traces from being displayed to end users. Ensure that errors generated by the application do not provide insight into specific backend issues.

Remove all backup files, binary archives, alternate versions of files, and test files from web-accessible directories of production servers. The only files that should be present in the application's web document root are files required by the application. Ensure that deployment procedures include the removal of these file types by an administrator. Keep web and application servers fully patched to minimize exposure to publicly-disclosed information leakage vulnerabilities.

### Associated Flaws by CWE ID:

## → Improper Restriction of XML External Entity Reference (CWE ID 611)(1 flaw)

### Description


The product processes an XML document that can contain XML entities with URLs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output. By default, the XML entity resolver will attempt to resolve and retrieve external references. If attacker-controlled XML can be submitted to one of these functions, then the attacker could gain access to information about an internal network, local filesystem, or other sensitive data. This is known as an XML eXternal Entity (XXE) attack.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

### Recommendations

Configure the XML parser to disable external entity resolution.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	 530158	30	-	app-uat-debug.apk	org/.../auth/PolicyAdvice.kt 89	6/14/25

## → Insufficient Input Validation(9 flaws)

### Description

Weaknesses in this category are related to an absent or incorrect protection mechanism that fails to properly validate input that can affect the control flow or data flow of a program.

### Recommendations

Validate input from untrusted sources before it is used. The untrusted data sources may include HTTP requests, file systems, databases, and any external systems that provide data to the application. In the case of HTTP requests, validate all parts of the request, including headers, form fields, cookies, and URL components that are used to transfer information from the browser to the server side application.

Duplicate any client-side checks on the server side. This should be simple to implement in terms of time and difficulty, and will greatly reduce the likelihood of insecure parameter values being used in the application.

### Associated Flaws by CWE ID:

## → URL Redirection to Untrusted Site ('Open Redirect') (CWE ID 601)(9 flaws)

### Description

A web application accepts a untrusted input that specifies a link to an external site, and uses that link to generate a redirect. This enables phishing attacks.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

### Recommendations

Always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible. Check the supplied URL against a whitelist of approved URLs or domains before redirecting.



## Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530118	23	-	Javascript files within app-uat-debug.apk	/assets/www/js/function.js 2038	6/14/25
NEW	530141	37	-	app-uat-debug.apk	com/.../aia/SmartWebActivity.java 303	6/14/25
NEW	530193	42	-	app-uat-debug.apk	com/aia/WebActivity.java 275	6/14/25
NEW	530194	42	-	app-uat-debug.apk	com/aia/WebActivity.java 278	6/14/25
NEW	530195	42	-	app-uat-debug.apk	com/aia/WebActivity.java 288	6/14/25
NEW	530196	42	-	app-uat-debug.apk	com/aia/WebActivity.java 290	6/14/25
NEW	530197	42	-	app-uat-debug.apk	com/aia/WebActivity.java 295	6/14/25
NEW	530198	42	-	app-uat-debug.apk	com/aia/WebActivity.java 305	6/14/25
NEW	530199	42	-	app-uat-debug.apk	com/aia/WebActivity.java 308	6/14/25

## → Server Configuration(2 flaws)

### Description

The application's supporting infrastructure, including web servers and application servers, can impact the security of the deployed application. Failing to lock down a server, for example, can result in information leaks via error pages, stack traces, or unnecessary files left in a web-accessible directory. Even though these servers are not part of the application codebase, they create insecurities in the environment which contribute to overall risk.

### Recommendations

Remove all extraneous files, including demonstration applications and sample code, from production systems. Configure production servers with the minimum set of services required for the application to function, and ensure that information leaks do not occur via server-generated error pages.

Audit any third party dependencies or services that are deployed by default to ensure that they do not compromise the security of the application being supported.

### Associated Flaws by CWE ID:

## → Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade') (CWE ID 757)(2 flaws)

### Description

A protocol or its implementation supports interaction between multiple actors and allows those actors to negotiate which algorithm should be used as a protection mechanism such as encryption or authentication, but it does not select the strongest algorithm that is available to both parties.

*Effort to Fix:* 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.



## Recommendations

Do not support SSLv2 or weak SSL/TLS ciphers (i.e. 56-bit key length or less, or other inherent weaknesses).

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530143	25	-	app-uat-debug.apk	com/.../aia/MainApplication.java 141	6/14/25
NEW	530178	33	-	app-uat-debug.apk	com/.../core/ScriptApiData.java 4753	6/14/25

## → Time and State(1 flaw)

### Description

Time and State flaws are related to unexpected interactions between threads, processes, time, and information. These interactions happen through shared state: semaphores, variables, the filesystem, and basically anything that can store information. Vulnerabilities occur when there is a discrepancy between the programmer's assumption of how a program executes and what happens in reality.

State issues result from improper management or invalid assumptions about system state, such as assuming mutable objects are immutable. Though these conditions are less commonly exploited by attackers, state issues can lead to unpredictable or undefined application behavior.

### Recommendations

Limit the interleaving of operations on resources from multiple processes. Use locking mechanisms to protect resources effectively. Follow best practices with respect to mutable objects and internal references. Pay close attention to asynchronous actions in processes and make copious use of sanity checks in systems that may be subject to synchronization errors.

### Associated Flaws by CWE ID:

## → Insecure Temporary File (CWE ID 377)(1 flaw)

### Description

Creating and using insecure temporary files can leave application and system data vulnerable to attack. In particular, file names created by the tmpnam family of functions can be easily guessed by an attacker. If an attacker can predict the filename and create a malicious collision, he may be able to manipulate the behavior of the application.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

### Recommendations

Ensure that unpredictable names are used for temporary files and that files are created in a secure directory with appropriate permissions. Using mkstemp() is a reasonably safe way to create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user, combined with a series of randomly generated characters. Note that mkstemp() is safe if only the descriptor is used and the returned filename is not used in a subsequent function call with extra privileges. Using mkstemp() does not completely eliminate race conditions but does provide better protection than other methods.



## Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	 530177	12	-	app-uat-debug.apk	.../ChromeClientImageUploadHelper.java 111	6/14/25

## Low (5 flaws)

## → Information Leakage(5 flaws)

## Description

An information leak is the intentional or unintentional disclosure of information that is either regarded as sensitive within the product's own functionality or provides information about the product or its environment that could be useful in an attack. Information leakage issues are commonly overlooked because they cannot be used to directly exploit the application. However, information leaks should be viewed as building blocks that an attacker uses to carry out other, more complicated attacks.

There are many different types of problems that involve information leaks, with severities that can range widely depending on the type of information leaked and the context of the information with respect to the application. Common sources of information leakage include, but are not limited to:

- \* Source code disclosure
- \* Browsable directories
- \* Log files or backup files in web-accessible directories
- \* Unfiltered backend error messages
- \* Exception stack traces
- \* Server version information
- \* Transmission of uninitialized memory containing sensitive data

## Recommendations

Configure applications and servers to return generic error messages and to suppress stack traces from being displayed to end users. Ensure that errors generated by the application do not provide insight into specific backend issues.

Remove all backup files, binary archives, alternate versions of files, and test files from web-accessible directories of production servers. The only files that should be present in the application's web document root are files required by the application. Ensure that deployment procedures include the removal of these file types by an administrator. Keep web and application servers fully patched to minimize exposure to publicly-disclosed information leakage vulnerabilities.

## Associated Flaws by CWE ID:

## → Generation of Error Message Containing Sensitive Information (CWE ID 209)(2 flaws)

## Description

The software generates an error message that includes sensitive information about its environment, users, or associated data. The sensitive information may be valuable information on its own (such as a password), or it may be useful for launching other, more deadly attacks. If an attack fails, an attacker may use error information provided by the server to launch another more focused attack. For example, file locations disclosed by an exception stack trace may be leveraged by an attacker to exploit a path traversal issue elsewhere in the application.

*Effort to Fix:* 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.



## Recommendations

Ensure that only generic error messages are returned to the end user that do not reveal any additional details.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530168	37	-	app-uat-debug.apk	com/.../aia/SmartWebActivity.java 830	
NEW	530169	37	-	app-uat-debug.apk	com/.../aia/SmartWebActivity.java 848	

## → Insertion of Sensitive Information Into Sent Data (CWE ID 201)(3 flaws)

### Description

Sensitive information may be exposed as a result of outbound network connections made by the application.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

### Recommendations

Ensure that the transfer of sensitive data is intended and that it does not violate application security policy or user expectations.

### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530148	29	-	app-uat-debug.apk	helper/.../PhotoActivity.java 121	
NEW	530149	29	-	app-uat-debug.apk	helper/.../PhotoActivity.java 242	
NEW	530170	37	-	app-uat-debug.apk	com/.../aia/SmartWebActivity.java 848	

## Very Low (0 flaws)

No flaws of this type were found

## Info (3 flaws)

### → Code Quality(2 flaws)

### Description

Code quality issues stem from failure to follow good coding practices and can lead to unpredictable behavior. These may include but are not limited to:

- \* Neglecting to remove debug code or dead code
- \* Improper resource management, such as using a pointer after it has been freed
- \* Using the incorrect operator to compare objects
- \* Failing to follow an API or framework specification
- \* Using a language feature or API in an unintended manner



While code quality flaws are generally less severe than other categories and usually are not directly exploitable, they may serve as indicators that developers are not following practices that increase the reliability and security of an application. For an attacker, code quality issues may provide an opportunity to stress the application in unexpected ways.

## Recommendations

The wide variance of code quality issues makes it impractical to generalize how these issues should be addressed. Refer to individual categories for specific recommendations.

### Associated Flaws by CWE ID:

#### → Improper Resource Shutdown or Release (CWE ID 404)(2 flaws)

##### Description

The application fails to release (or incorrectly releases) a system resource before it is made available for re-use. This condition often occurs with resources such as database connections or file handles. Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, it may be possible to launch a denial of service attack by depleting the resource pool.

*Effort to Fix:* 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

##### Recommendations

When a resource is created or allocated, the developer is responsible for properly releasing the resource as well as accounting for all potential paths of expiration or invalidation. Ensure that all code paths properly release resources.

##### Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530173	3	-	app-uat-debug.apk	com/.../AES256Concrete.java 17	
NEW	530172	32	-	app-uat-debug.apk	com/.../utils/RootDetector.java 4	

#### → Potential Backdoor(1 flaw)

##### Description

Application backdoors are modifications to programs that are designed to bypass security mechanisms or inject malicious functionality. Backdoors are often inserted by rogue developers with legitimate access to the source code or distribution binaries. Backdoors can take many forms, such as hard-coded credentials, "easter egg" style functionality, rootkits, or time bombs, among others.

##### Recommendations

Investigate all potential backdoors thoroughly to ensure there is no undesirable functionality. If the backdoors are real, eliminate them, and initiate a broader effort to inspect the entire codebase for malicious code. This may require a detailed review of all code, as it is possible to hide a serious attack in only one or two lines of code. These lines may be located almost anywhere in an application and may have been intentionally obfuscated by the attacker.

### Associated Flaws by CWE ID:





→ Reliance on Security Through Obscurity (CWE ID 656)(1 flaw)

Description

The strength of a security mechanism depends heavily on its obscurity, such that knowledge of its algorithms or key data is sufficient to allow the mechanism to be compromised.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	530171	41	-	app-uat-debug.apk	com/.../utils/ToolsUtils.java 73	



## About Veracode's Methodology

The Veracode platform uses static and dynamic analysis (for web applications) to identify software security flaws in your applications. Using both static and dynamic analysis helps reduce false negatives and detect a broader range of security flaws. Veracode static analysis models the application into an intermediate representation, which is then analyzed for security flaws using a set of automated security tests. Dynamic analysis uses an automated penetration testing technique to detect security flaws at runtime. Once the automated process is complete, a security technician verifies the output to ensure the lowest false positive rates in the industry. The end result is an accurate list of security flaws for the classes of automated scans applied to the application.

## Veracode Rating System Using Multiple Analysis Techniques

Higher assurance applications require more comprehensive analysis to accurately score their security quality. Because each analysis technique (automated static, automated dynamic, manual penetration testing or manual review) has differing false negative (FN) rates for different types of security flaws, any single analysis technique or even combination of techniques is bound to produce a certain level of false negatives. Some false negatives are acceptable for lower business critical applications, so a less expensive analysis using only one or two analysis techniques is acceptable. At higher business criticality the FN rate should be close to zero, so multiple analysis techniques are recommended.

## Application Security Policies

The Veracode platform allows an organization to define and enforce a uniform application security policy across all applications in its portfolio. The elements of an application security policy include the target Veracode Level for the application; types of flaws that should not be in the application (which may be defined by flaw severity, flaw category, CWE, or a common standard including OWASP, CWE/SANS Top 25, or PCI); minimum Veracode security score; required scan types and frequencies; and grace period within which any policy-relevant flaws should be fixed.

### Policy constraints

Policies have three main constraints that can be applied: rules, required scans, and remediation grace periods.

### Evaluating applications against a policy

When an application is evaluated against a policy, it can receive one of four assessments:

**Not assessed** The application has not yet had a scan published

**Passed** The application has passed all the aspects of the policy, including rules, required scans, and grace period.

**Did not pass** The application has not completed all required scans; has not achieved the target Veracode Level; or has one or more policy relevant flaws that have exceeded the grace period to fix.

**Conditional pass** The application has one or more policy relevant flaws that have not yet exceeded the grace period to fix.

## Understand Veracode Levels

The Veracode Level (VL) achieved by an application is determined by type of testing performed on the application, and the severity and types of flaws detected. A minimum security score (defined below) is also required for each level.

There are five Veracode Levels denoted as VL1, VL2, VL3, VL4, and VL5. VL1 is the lowest level and is achieved by demonstrating that security testing, automated static or dynamic, is utilized during the SDLC. VL5 is the highest level and is achieved by performing automated and manual testing and removing all significant flaws. The Veracode Levels VL2, VL3, and VL4 form a continuum of increasing software assurance between VL1 and VL5.

For IT staff operating applications, Veracode Levels can be used to set application security policies. For deployment scenarios of different business criticality, differing VLs should be made requirements. For example, the policy for applications that handle credit card transactions, and therefore have PCI compliance requirements, should be VL5. A medium business criticality internal application could have a policy requiring VL3.

Software developers can decide which VL they want to achieve based on the requirements of their customers. Developers of software that is mission critical to most of their customers will want to achieve VL5. Developers of general purpose business software may want



to achieve VL3 or VL4. Once the software has achieved a Veracode Level it can be communicated to customers through a Veracode Report or through the Veracode Directory on the Veracode web site.

### Criteria for achieving Veracode Levels

The following table defines the details to achieve each Veracode Level. The criteria for all columns: Flaw Severities Not Allowed, Flaw Categories not Allowed, Testing Required, and Minimum Score.

\*Dynamic is only an option for web applications.

Veracode Level	Flaw Severities Not Allowed	Testing Required*	Minimum Score
VL5	V.High, High, Medium	Static AND Manual	90
VL4	V.High, High, Medium	Static	80
VL3	V.High, High	Static	70
VL2	V.High	Static OR Dynamic OR Manual	60
VL1		Static OR Dynamic OR Manual	

When multiple testing techniques are used it is likely that not all testing will be performed on the exact same build. If that is the case the latest test results from a particular technique will be used to calculate the current Veracode Level. After 6 months test results will be deemed out of date and will no longer be used to calculate the current Veracode Level.

### Business Criticality

The foundation of the Veracode rating system is the concept that more critical applications require higher security quality scores to be acceptable risks. Less business critical applications can tolerate lower security quality. The business criticality is dictated by the typical deployed environment and the value of data used by the application. Factors that determine business criticality are: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations.

US. Govt. OMB Memorandum M-04-04; NIST FIPS Pub. 199

Business Criticality Description

Very High	Mission critical for business/safety of life and limb on the line
High	Exploitation causes serious brand damage and financial loss with long term business impact
Medium	Applications connected to the internet that process financial or private customer information
Low	Typically internal applications with non-critical business impact
Very Low	Applications with no material business impact

### Business Criticality Definitions

**Very High (BC5)** This is typically an application where the safety of life or limb is dependent on the system; it is mission critical the application maintain 100% availability for the long term viability of the project or business. Examples are control software for industrial, transportation or medical equipment or critical business systems such as financial trading systems.

**High (BC4)** This is typically an important multi-user business application reachable from the internet and is critical that the application maintain high availability to accomplish its mission. Exploitation of high criticality applications cause serious brand damage and business/financial loss and could lead to long term business impact.

**Medium (BC3)** This is typically a multi-user application connected to the internet or any system that processes financial or private customer information. Exploitation of medium criticality applications typically result in material business impact resulting



in some financial loss, brand damage or business liability. An example is a financial services company's internal 401K management system.

**Low (BC2)** This is typically an internal only application that requires low levels of application security such as authentication to protect access to non-critical business information and prevent IT disruptions. Exploitation of low criticality applications may lead to minor levels of inconvenience, distress or IT disruption. An example internal system is a conference room reservation or business card order system.

**Very Low (BC1)** Applications that have no material business impact should its confidentiality, data integrity and availability be affected. Code security analysis is not required for applications at this business criticality, and security spending should be directed to other higher criticality applications.

## Scoring Methodology

The Veracode scoring system, Security Quality Score, is built on the foundation of two industry standards, the Common Weakness Enumeration (CWE) and Common Vulnerability Scoring System (CVSS). CWE provides the dictionary of security flaws and CVSS provides the foundation for computing severity, based on the potential Confidentiality, Integrity and Availability impact of a flaw if exploited.

The Security Quality Score is a single score from 0 to 100, where 0 is the most insecure application and 100 is an application with no detectable security flaws. The score calculation includes non-linear factors so that, for instance, a single Severity 5 flaw is weighted more heavily than five Severity 1 flaws, and so that each additional flaw at a given severity contributes progressively less to the score.

Veracode assigns a severity level to each flaw type based on three foundational application security requirements — Confidentiality, Integrity and Availability. Each of the severity levels reflects the potential business impact if a security breach occurs across one or more of these security dimensions.

### Confidentiality Impact

According to CVSS, this metric measures the impact on confidentiality if an exploit should occur using the vulnerability on the target system. At the weakness level, the scope of the Confidentiality in this model is within an application and is measured at three levels of impact -None, Partial and Complete.

### Integrity Impact

This metric measures the potential impact on integrity of the application being analyzed. Integrity refers to the trustworthiness and guaranteed veracity of information within the application. Integrity measures are meant to protect data from unauthorized modification. When the integrity of a system is sound, it is fully proof from unauthorized modification of its contents.

### Availability Impact

This metric measures the potential impact on availability if a successful exploit of the vulnerability is carried out on a target application. Availability refers to the accessibility of information resources. Almost exclusive to this domain are denial-of-service vulnerabilities. Attacks that compromise authentication and authorization for application access, application memory, and administrative privileges are examples of impact on the availability of an application.

## Security Quality Score Calculation

The overall Security Quality Score is computed by aggregating impact levels of all weaknesses within an application and representing the score on a 100 point scale. This score does not predict vulnerability potential as much as it enumerates the security weaknesses and their impact levels within the application code.

The Raw Score formula puts weights on each flaw based on its impact level. These weights are exponential and determined by empirical analysis by Veracode's application security experts with validation from industry experts. The score is normalized to a scale of 0 to 100, where a score of 100 is an application with 0 detected flaws using the analysis technique for the application's business criticality.

## Understand Severity, Exploitability, and Remediation Effort

Severity and exploitability are two different measures of the seriousness of a flaw. Severity is defined in terms of the potential impact to confidentiality, integrity, and availability of the application as defined in the CVSS, and exploitability is defined in terms of the likelihood



or ease with which a flaw can be exploited. A high severity flaw with a high likelihood of being exploited by an attacker is potentially more dangerous than a high severity flaw with a low likelihood of being exploited.

Remediation effort, also called Complexity of Fix, is a measure of the likely effort required to fix a flaw. Together with severity, the remediation effort is used to give Fix First guidance to the developer.

## Veracode Flaw Severities

Veracode flaw severities are defined as follows:

Severity	Description
Very High	The offending line or lines of code is a very serious weakness and is an easy target for an attacker. The code should be modified immediately to avoid potential attacks.
High	The offending line or lines of code have significant weakness, and the code should be modified immediately to avoid potential attacks.
Medium	A weakness of average severity. These should be fixed in high assurance software. A fix for this weakness should be considered after fixing the very high and high for medium assurance software.
Low	This is a low priority weakness that will have a small impact on the security of the software. Fixing should be consideration for high assurance software. Medium and low assurance software can ignore these flaws.
Very Low	Minor problems that some high assurance software may want to be aware of. These flaws can be safely ignored in medium and low assurance software.
Informational	Issues that have no impact on the security quality of the application but which may be of interest to the reviewer.

### Informational findings

Informational severity findings are items observed in the analysis of the application that have no impact on the security quality of the application but may be interesting to the reviewer for other reasons. These findings may include code quality issues, API usage, and other factors.

Informational severity findings have no impact on the security quality score of the application and are not included in the summary tables of flaws for the application.

## Exploitability

Each flaw instance in a static scan may receive an exploitability rating. The rating is an indication of the intrinsic likelihood that the flaw may be exploited by an attacker. Veracode recommends that the exploitability rating be used to prioritize flaw remediation within a particular group of flaws with the same severity and difficulty of fix classification.

The possible exploitability ratings include:

Exploitability	Description
V. Unlikely	Very unlikely to be exploited
Unlikely	Unlikely to be exploited



Exploitability	Description
Neutral	Neither likely nor unlikely to be exploited.
Likely	Likely to be exploited
V. Likely	Very likely to be exploited

Note: All reported flaws found via dynamic scans are assumed to be exploitable, because the dynamic scan actually executes the attack in question and verifies that it is valid.

## Effort/Complexity of Fix

Each flaw instance receives an effort/complexity of fix rating based on the classification of the flaw. The effort/complexity of fix rating is given on a scale of 1 to 5, as follows:

Effort/Complexity of Fix	Description
5	Complex design error. Requires significant redesign.
4	Simple design error. Requires redesign and up to 5 days to fix.
3	Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.
2	Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.
1	Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

## Flaw Types by Severity Level

The flaw types by severity level table provides a summary of flaws found in the application by Severity and Category. The table puts the Security Quality Score into context by showing the specific breakout of flaws by severity, used to compute the score as described above. If multiple analysis techniques are used, the table includes a breakout of all flaws by category and severity for each analysis type performed.

## Flaws by Severity

The flaws by severity chart shows the distribution of flaws by severity. An application can get a mediocre security rating by having a few high risk flaws or many medium risk flaws.

## Flaws in Common Modules

The flaws in common modules listing shows a summary of flaws in shared dependency modules in this application. A shared dependency is a dependency that is used by more than one analyzed module. Each module is listed with the number of executables that consume it as a dependency and a summary of the impact on the application's security score of the flaws found in the dependency.

The score impact represents the amount that the application score would increase if all the flaws in the shared dependency module were fixed. This information can be used to focus remediation efforts on common modules with a higher impact on the application security score.

Only common modules that were uploaded with debug information are included in the Flaws in Common Modules listing.



## Action Items

The Action Items section of the report provides guidance on the steps required to bring the application to a state where it passes its assigned policy. These steps may include fixing or mitigating flaws or performing additional scans. The section also includes best practice recommendations to improve the security quality of the application.

## Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) is an industry standard classification of types of software weaknesses, or flaws, that can lead to security problems. CWE is widely used to provide a standard taxonomy of software errors. Every flaw in a Veracode report is classified according to a standard CWE identifier.

More guidance and background about the CWE is available at <http://cwe.mitre.org/data/index.html>.

## About Manual Assessments

The Veracode platform can include the results from a manual assessment (usually a penetration test or code review) as part of a report. These results differ from the results of automated scans in several important ways, including objectives, attack vectors, and common attack patterns.

A manual penetration assessment is conducted to observe the application code in a run-time environment and to simulate real-world attack scenarios. Manual testing is able to identify design flaws, evaluate environmental conditions, compound multiple lower risk flaws into higher risk vulnerabilities, and determine if identified flaws affect the confidentiality, integrity, or availability of the application.

### Objectives

The stated objectives of a manual penetration assessment are:

- Perform testing, using proprietary and/or public tools, to determine whether it is possible for an attacker to:
- Circumvent authentication and authorization mechanisms
- Escalate application user privileges
- Hijack accounts belonging to other users
- Violate access controls placed by the site administrator
- Alter data or data presentation
- Corrupt application and data integrity, functionality and performance
- Circumvent application business logic
- Circumvent application session management
- Break or analyze use of cryptography within user accessible components
- Determine possible extent access or impact to the system by attempting to exploit vulnerabilities
- Score vulnerabilities using the Common Vulnerability Scoring System (CVSS)
- Provide tactical recommendations to address security issues of immediate consequence

Provide strategic recommendations to enhance security by leveraging industry best practices

### Attack vectors

In order to achieve the stated objectives, the following tests are performed as part of the manual penetration assessment, when applicable to the platforms and technologies in use:

- Cross Site Scripting (XSS)
- SQL Injection
- Command Injection
- Cross Site Request Forgery (CSRF)
- Authentication/Authorization Bypass
- Session Management testing, e.g. token analysis, session expiration, and logout effectiveness
- Account Management testing, e.g. password strength, password reset, account lockout, etc.
- Directory Traversal
- Response Splitting
- Stack/Heap Overflows
- Format String Attacks



- Cookie Analysis
- Server Side Includes Injection
- Remote File Inclusion
- LDAP Injection
- XPATH Injection
- Internationalization attacks
- Denial of Service testing at the application layer only
- AJAX Endpoint Analysis
- Web Services Endpoint Analysis
- HTTP Method Analysis
- SSL Certificate and Cipher Strength Analysis
- Forced Browsing

### CAPEC Attack Pattern Classification

The following attack pattern classifications are used to group similar application flaws discovered during manual penetration testing. Attack patterns describe the general methods employed to access and exploit the specific weaknesses that exist within an application. CAPEC (Common Attack Pattern Enumeration and Classification) is an effort led by Cigital, Inc. and is sponsored by the United States Department of Homeland Security's National Cyber Security Division.

### Abuse of Functionality

Exploitation of business logic errors or misappropriation of programmatic resources. Application functions are developed to specifications with particular intentions, and these types of attacks serve to undermine those intentions.

Examples:

- Exploiting password recovery mechanisms
- Accessing unpublished or test APIs
- Cache poisoning

### Spoofing

Impersonation of entities or trusted resources. A successful attack will present itself to a verifying entity with an acceptable level of authenticity.

Examples:

- Man in the middle attacks
- Checksum spoofing
- Phishing attacks

### Probabilistic Techniques

Using predictive capabilities or exhaustive search techniques in order to derive or manipulate sensitive information. Attacks capitalize on the availability of computing resources or the lack of entropy within targeted components.

Examples:

- Password brute forcing
- Cryptanalysis
- Manipulation of authentication tokens

### Exploitation of Authentication

Circumventing authentication requirements to access protected resources. Design or implementation flaws may allow authentication checks to be ignored, delegated, or bypassed.

Examples:

- Cross-site request forgery
- Reuse of session identifiers
- Flawed authentication protocol





## Resource Depletion

Affecting the availability of application components or resources through symmetric or asymmetric consumption. Unrestricted access to computationally expensive functions or implementation flaws that affect the stability of the application can be targeted by an attacker in order to cause denial of service conditions.

Examples:

- Flooding attacks
- Unlimited file upload size
- Memory leaks

## Exploitation of Privilege/Trust

Undermining the application's trust model in order to gain access to protected resources or gain additional levels of access as defined by the application. Applications that implicitly extend trust to resources or entities outside of their direct control are susceptible to attack.

Examples:

- Insufficient access control lists
- Circumvention of client side protections
- Manipulation of role identification information

## Injection

Inserting unexpected inputs to manipulate control flow or alter normal business processing. Applications must contain sufficient data validation checks in order to sanitize tainted data and prevent malicious, external control over internal processing.

Examples:

- SQL Injection
- Cross-site scripting
- XML Injection

## Data Structure Attacks

Supplying unexpected or excessive data that results in more data being written to a buffer than it is capable of holding. Successful attacks of this class can result in arbitrary command execution or denial of service conditions.

Examples:

- Buffer overflow
- Integer overflow
- Format string overflow

## Data Leakage Attacks

Recovering information exposed by the application that may itself be confidential or may be useful to an attacker in discovering or exploiting other weaknesses. A successful attack may be conducted passive observation or active interception methods. This attack pattern often manifests itself in the form of applications that expose sensitive information within error messages.

Examples:

- Sniffing clear-text communication protocols
- Stack traces returned to end users
- Sensitive information in HTML comments

## Resource Manipulation

Manipulating application dependencies or accessed resources in order to undermine security controls and gain unauthorized access to protected resources. Applications may use tainted data when constructing paths to local resources or when constructing processing environments.



Examples:

- Carriage Return Line Feed log file injection
- File retrieval via path manipulation
- User specification of configuration files

### Time and State Attacks

Undermining state condition assumptions made by the application or capitalizing on time delays between security checks and performed operations. An application that does not enforce a required processing sequence or does not handle concurrency adequately will be susceptible to these attack patterns.

Examples:

- Bypassing intermediate form processing steps
- Time-of-check and time-of-use race conditions
- Deadlock triggering to cause a denial of service

## Terms of Use

Use and distribution of this report are governed by the agreement between Veracode and its customer. In particular, this report and the results in the report cannot be used publicly in connection with Veracode's name without written permission.



Appendix A: Changes from Last Scan

Latest Scan		Prior Scan	
<b>Static Scan</b>			
Scan Name:	app-uat-debug.apk_16 Mar 2025 Static	Scan Name:	20250304-app-frintegrator-experience-api-3rd
Completed:	3/16/25	Completed:	3/4/25
Score:	76	Score:	100

Changes in scope of scan (static)

New Modules

Module Name	Compiler	Operating Environment	Engine Version
app-uat-debug.apk	Android	Android	20250312184913
Javascript files within app-uat-debug.apk	JAVASCRIPT_5_1	JavaScript	20250312184913

Removed modules

Module Name	Compiler	Operating Environment	Engine Version
app-frintegrator-experience-api-1.0.0.jar	JAVAC_17	Java J2SE 17	20250225181258



## Appendix B: Referenced Source Files

Id	Filename	Path
1	AbstractSymmetricEncryptor.java	org/forgerock/android/auth/
2	AbstractYESSafeCommon.java	com/isprint/library/
3	AES256Concrete.java	com/isprint/vccard/algorithm/
4	AESUtil.java	com/aia/network/
5	AIAFcmListenerService.java	com/aia/network/core/
6	AIAInstanceIDListenerService.java	com/aia/network/core/
7	AppPreferences.java	me/apla/cordova/
8	AsymmetricEncryptor.java	org/forgerock/android/auth/
9	BaseFingerprintUiHelper.java	com/isprint/fingerprint/
10	Calculator.java	helper/
11	chatBot.js	/assets/www/js/
12	ChromeClientImageUploadHelper.java	com/aia/webview/
13	CryptoKey.kt	org/forgerock/android/auth/
14	DefaultLogger.kt	org/forgerock/android/auth/
15	DeviceIdentifier.java	org/forgerock/android/auth/
16	EncryptUtil.java	com/isprint/vccard/utils/
17	enroll_details.html	/assets/www/
18	FingerHandle.java	com/isprint/handle/
19	FingerprintAuth.java	com/cordova/plugin/android/fingerprintauth/
20	FingerprintUiHelper.java	com/isprint/fingerprint/
21	FingerprintUiHelper.java	com/cordova/plugin/android/fingerprintauth/
22	FRUtils.java	com/aia/fr/
23	function.js	/assets/www/js/
24	keyStore.js	/assets/www/js/
25	MainApplication.java	com/aia/
26	OCRA.java	com/isprint/vccard/algorithm/
27	ops_inbox_details.html	/assets/www/
28	PasswordDialogPlugin.java	net/justin_credible/cordova/
29	PhotoActivity.java	helper/photo/
30	PolicyAdvice.kt	org/forgerock/android/auth/
31	ProductConfig.java	com/aia/config/
32	RootDetector.java	com/isprint/vccard/utils/
33	ScriptApiData.java	com/aia/network/core/
34	SecureStorage.java	com/isprint/vccard/utils/
35	ShareChooserPendingIntent.java	nl/xservices/plugins/



Id	Filename	Path
	a	
36	ShutDownReceiver.java	com/aia/
37	SmartWebActivity.java	com/aia/
38	SocialSharing.java	nl/xservices/plugins/
39	SplashScreen.java	com/aia/
40	survey.html	/assets/www/
41	ToolsUtils.java	com/isprint/vccard/utils/
42	WebActivity.java	com/aia/