VERACODE

Veracode Detailed Report

Application Security Report
As of 27 Feb 2013

Prepared for:	Radio Free Asia
Prepared on:	March 11, 2013
Application:	Guardian - Gibberbot

Industry:	Not Specified
Business Criticality:	BC4 (High)
Required Analysis:	Static
Type(s) of Analysis Conducted:	Static
Scope of Static Scan:	1 of 1 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	

65 Network Drive, Burlington, MA 01803

Tel.+1.339.674.2500 **Fax.**+1.339.674.2502 **URL:**http://www.veracode.com

Veracode Detailed Report

Application Security Report As of 27 Feb 2013

Veracode Level: VL2

Rated: Feb 27, 2013

Application:	Guardian - Gibberbot	Business Criticality:	High
Target Level:	VL4	Published Rating:	C

Scans Included in Report

Static Scan	Dynamic Scan	Manual Scan
27 Feb 2013 Static Score: 68 Completed: 2/27/13	Not Included in Report	Not Included in Report

Executive Summary

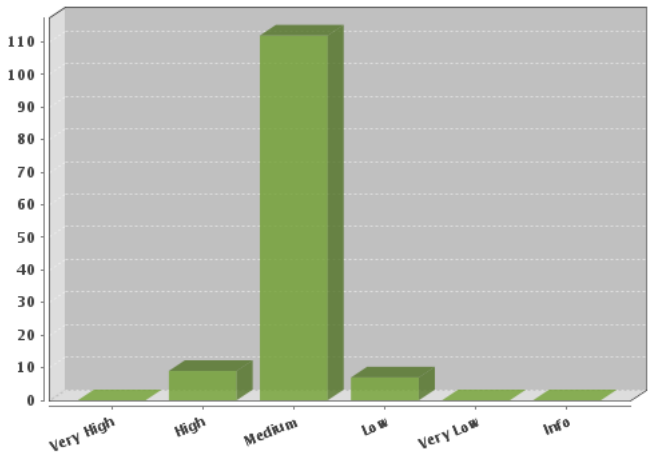
This report contains a summary of the security flaws identified in the application using automated static, automated dynamic and/or manual security analysis techniques. This is useful for understanding the overall security quality of an individual application or for comparisons between applications.

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.

Summary of Flaws Found by Severity



Analyses Performed vs. Required

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

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.

Required Analysis

→ Your policy requires periodic Static Scan. Your next analysis must be completed by 5/27/13. Please submit your application for Static Scan by the deadline and remediate the required detected flaws to conform to your assigned policy.

Flaws To Fix By Expires Date

A grace period is specified for any flaw that violates the rules contained in your policy. These include CWE, Rollup Category, Issue Severity, Industry Standards as well as any flaws the prevent an application from achieving a minimum Veracode Level and/or score. To maintain policy compliance you must fix these flaws and resubmit your application for scanning before the grace period expires. The detailed flaw listing will badge the flaws that must be fixed and show the fix by date as well.

→ The grace period has expired [2/20/13] for 118 flaws that were found in your Static Scan.

→ The grace period has expired [2/27/13] for 3 flaws that were found in your Static Scan.

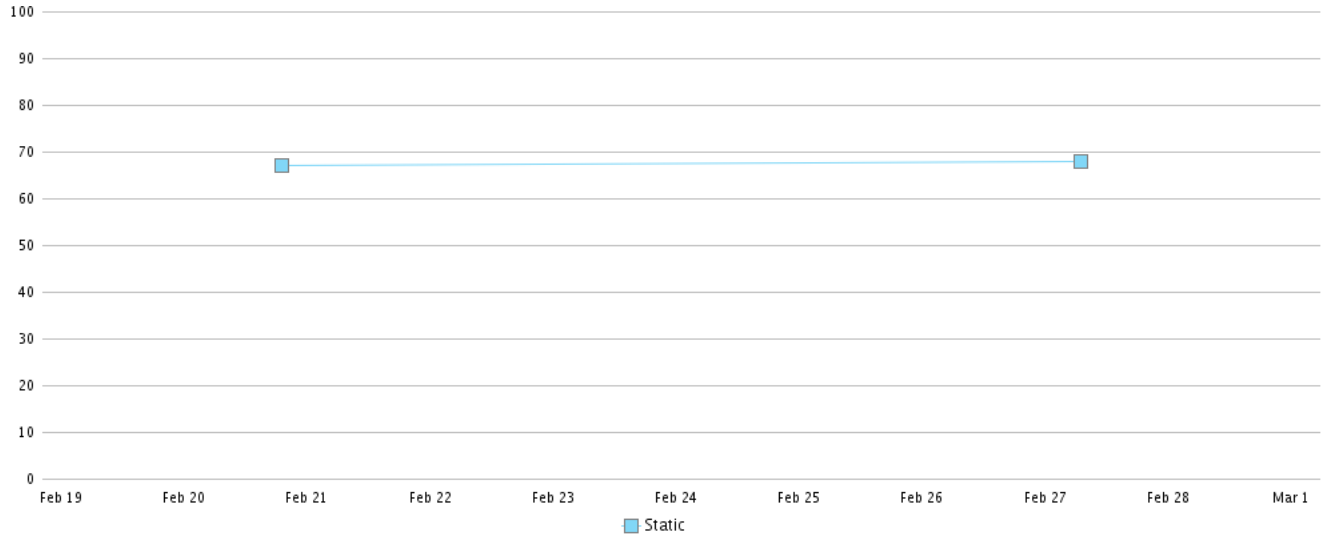
Flaws To Fix For Minimum Score

- Your current policy requires a minimum score. In order to achieve the score, you must fix all of the flaws that violate your current policy plus additional flaws. You must fix 9 High flaws and 42 Medium flaws to increase the application Static Scan Security Quality Score to 80.
- Your Static Scan was due on 2/27/13 for follow-up analysis to satisfy the grace period on your minimum score rule and your application is no longer compliant with your policy. Submit application for follow-up Static Scan once flaws have been remediated in order to regain compliance with your policy.

Longer Timeframe (6 - 12 months)

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

Application Ratings Trend



Scope of Static Scan

It is important to note that this application may include additional modules which were not included in this analysis. We recommend that you contact the vendor to determine whether all modules have been included.

The following modules were included in the application scan:

Module Name	Compiler	Operating Environment	Engine Version
Gibberbot-debug.apk	Android	Android	56824

Flaw Types by Severity and Category

	Static Scan Security Quality Score = 68 (+1) from prior scan			
Very High	**0**			
High	**9**			
SQL Injection	9			
Medium	**112**	**(-4)**		
CRLF Injection	74	(-4)		
Code Quality	8			
Credentials Management	6			
Cryptographic Issues	20			
Insufficient Input Validation	3			
Time and State	1			
Low	**7**			
Code Quality	6			

Static Scan Security Quality Score = 68 (+1) from prior scan			
Information Leakage	1		
Very Low	**0**		
Informational	**0**		
Total	**128** (-4)		

Policy Control

Policy Name: Veracode Recommended High

Revision: 1

Policy Status: Did Not Pass

Description

Veracode provides default policies to make it easier for organizations to begin measuring their applications against policies. Veracode Recommended Policies are available for customers as an option when they are ready to move beyond the initial bar set by the Veracode Transitional Policies. The policies are based on the Veracode Level definitions.

Rules

Rule type	Requirement	Findings	Status
Minimum Veracode Level	VL4	VL2	Did not pass
(VL4) Min Analysis Score	80	68	Did not pass
(VL4) Max Severity	Medium	Flaws found: 121	Did not pass

Scan Requirements

Scan Type	Frequency	Last performed	Status
Static	Quarterly	2/27/13	Passed

Remediation

Flaw Severity	Grace Period	Flaws Exceeding	Status
Very High	0 days	0	Passed
High	0 days	9	Did not pass
Medium	0 days	112	Did not pass
Low	0 days	0	Passed
Very Low	0 days	0	Passed
Informational	0 days	0	Passed

Type	Grace Period	Exceeding	Status
Min Analysis Score	0 days	1	Did not pass

Findings & Recommendations

Detailed Flaws by Severity

Very High (0 flaws)

No flaws of this type were found

High (9 flaws)

Fix Required by Policy

SQL Injection(9 flaws)

Fix Required by Policy

Description

SQL injection vulnerabilities occur when data enters an application from an untrusted source and is used to dynamically construct a SQL query. This allows an attacker to manipulate database queries in order to access, modify, or delete arbitrary data. Depending on the platform, database type, and configuration, it may also be possible to execute administrative operations on the database, access the filesystem, or execute arbitrary system commands. SQL injection attacks can also be used to subvert authentication and authorization schemes, which would enable an attacker to gain privileged access to restricted portions of the application.

Recommendations

Several techniques can be used to prevent SQL injection 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 SQL injection vulnerability.

* Use parameterized prepared statements rather than dynamically constructing SQL queries. This will prevent the database from interpreting the contents of bind variables as part of the query and is the most effective defense against SQL injection.
* 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.
* Normalize all user-supplied data before applying filters or regular expressions, or submitting the data to a database. This means that all URL-encoded (%xx), HTML-encoded (&#xx;), or other encoding schemes should be reduced to the internal character representation expected by the application. This prevents attackers from using alternate encoding schemes to bypass filters.
* When using database abstraction libraries such as Hibernate, do not assume that all methods exposed by the API will automatically prevent SQL injection attacks. Most libraries contain methods that pass arbitrary queries to the database in an unsafe manner.

Associated Flaws by CWE ID:

Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (CWE ID 89)(9 flaws)

Description

This database query contains a SQL injection flaw. The function call constructs a dynamic SQL query using a variable derived from user-supplied input. An attacker could exploit this flaw to execute arbitrary SQL queries against the database.

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

Recommendations

Avoid dynamically constructing SQL queries. Instead, use parameterized prepared statements to prevent the database from interpreting the contents of bind variables as part of the query. Always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Instances found via Static Scan

	Module #	Class #	Module	Location	Fix By	Flaw Id
	2	-	Gibberbot-debug.apk	info/.../app/AccountActivity.java 193	2/20/13	66
	6	-	Gibberbot-debug.apk	.../BlockedContactsActivity.java 124	2/20/13	127
	14	-	Gibberbot-debug.apk	.../ContactPresenceActivity.java 110	2/20/13	128
NEW	15	-	Gibberbot-debug.apk	.../ContactsPickerActivity.java 112	2/27/13	145
NEW	15	-	Gibberbot-debug.apk	.../ContactsPickerActivity.java 138	2/27/13	150
	15	-	Gibberbot-debug.apk	.../ContactsPickerActivity.java 152	2/20/13	129
	35	-	Gibberbot-debug.apk	info/.../app/im/app/ImApp.java 485	2/20/13	113
	36	-	Gibberbot-debug.apk	info/.../im/provider/Imps.java 1531	2/20/13	10
	64	-	Gibberbot-debug.apk	info/.../app/SignoutActivity.java 54	2/20/13	44

Medium (112 flaws)

Fix Required by Policy

CRLF Injection(74 flaws)

Fix Required by Policy

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)(74 flaws)

Description

A function call could result in a log forging attack. Writing unsanitized user-supplied 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 user-supplied 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 user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
1	-	Gibberbot-debug.apk	.../AbstractMessageParser.java 184	2/20/13	59
2	-	Gibberbot-debug.apk	info/.../app/AccountActivity.java 229	2/20/13	45
4	-	Gibberbot-debug.apk	de/.../smack/AndroidDebugger.java 89	2/20/13	37
4	-	Gibberbot-debug.apk	de/.../smack/AndroidDebugger.java 155	2/20/13	139
5	-	Gibberbot-debug.apk	net/.../AuthContextImpl.java 416	2/20/13	79
5	-	Gibberbot-debug.apk	net/.../AuthContextImpl.java 470	2/20/13	65
5	-	Gibberbot-debug.apk	net/.../AuthContextImpl.java 569	2/20/13	125
5	-	Gibberbot-debug.apk	net/.../AuthContextImpl.java 610	2/20/13	137
7	-	Gibberbot-debug.apk	com/.../jbosh/BodyParserSAX.java 93	2/20/13	47
7	-	Gibberbot-debug.apk	com/.../jbosh/BodyParserSAX.java 94	2/20/13	91
7	-	Gibberbot-debug.apk	com/.../jbosh/BodyParserSAX.java 95	2/20/13	126
7	-	Gibberbot-debug.apk	com/.../jbosh/BodyParserSAX.java 119	2/20/13	23
7	-	Gibberbot-debug.apk	com/.../jbosh/BodyParserSAX.java 142	2/20/13	122
7	-	Gibberbot-debug.apk	com/.../jbosh/BodyParserSAX.java 147	2/20/13	42
8	-	Gibberbot-debug.apk	com/.../BodyParserXmlPull.java 72	2/20/13	62
8	-	Gibberbot-debug.apk	com/.../BodyParserXmlPull.java 88	2/20/13	12
8	-	Gibberbot-debug.apk	com/.../BodyParserXmlPull.java 89	2/20/13	32
8	-	Gibberbot-debug.apk	com/.../BodyParserXmlPull.java 90	2/20/13	34
8	-	Gibberbot-debug.apk	com/.../BodyParserXmlPull.java 116	2/20/13	124
10	-	Gibberbot-debug.apk	.../ChatListenerAdapter.java 74	2/20/13	24
12	-	Gibberbot-debug.apk	.../ChatSessionListenerAdapter.java 40	2/20/13	90
13	-	Gibberbot-debug.apk	.../io/ChunkedInputStream.java 137	2/20/13	33
13	-	Gibberbot-debug.apk	.../io/ChunkedInputStream.java 172	2/20/13	14
13	-	Gibberbot-debug.apk	.../io/ChunkedInputStream.java 234	2/20/13	35
13	-	Gibberbot-debug.apk	.../io/ChunkedInputStream.java 246	2/20/13	88
16	-	Gibberbot-debug.apk	.../ContentLengthInputStream.java 137	2/20/13	103
16	-	Gibberbot-debug.apk	.../ContentLengthInputStream.java 164	2/20/13	97
17	-	Gibberbot-debug.apk	.../DefaultClientConnection.java 249	2/20/13	43
17	-	Gibberbot-debug.apk	.../DefaultClientConnection.java 252	2/20/13	96
17	-	Gibberbot-debug.apk	.../DefaultClientConnection.java 255	2/20/13	21
17	-	Gibberbot-debug.apk	.../DefaultClientConnection.java 268	2/20/13	78
18	-	Gibberbot-debug.apk	.../DefaultRedirectHandler.java 112	2/20/13	115
19	-	Gibberbot-debug.apk	.../DefaultRedirectStrategy.java 113	2/20/13	73

	Module #	Class #	Module	Location	Fix By	Flaw Id
	20	-	Gibberbot-debug.apk	.../DefaultRequestDirector.java 1056	2/20/13	36
	21	-	Gibberbot-debug.apk	.../DefaultResponseParser.java 98	2/20/13	48
NEW	21	-	Gibberbot-debug.apk	.../DefaultResponseParser.java 113	2/27/13	149
	33	-	Gibberbot-debug.apk	org/.../io/HttpRequestParser.java 90	2/20/13	133
	34	-	Gibberbot-debug.apk	.../io/HttpResponseParser.java 90	2/20/13	98
	37	-	Gibberbot-debug.apk	info/.../ImpsProvider.java 354	2/20/13	40
	37	-	Gibberbot-debug.apk	info/.../ImpsProvider.java 371	2/20/13	76
	37	-	Gibberbot-debug.apk	info/.../ImpsProvider.java 390	2/20/13	50
	37	-	Gibberbot-debug.apk	info/.../ImpsProvider.java 410	2/20/13	64
	37	-	Gibberbot-debug.apk	info/.../ImpsProvider.java 432	2/20/13	105
	37	-	Gibberbot-debug.apk	info/.../ImpsProvider.java 448	2/20/13	106
	37	-	Gibberbot-debug.apk	info/.../ImpsProvider.java 466	2/20/13	102
	38	-	Gibberbot-debug.apk	info/.../app/ImUrlActivity.java 64	2/20/13	53
	42	-	Gibberbot-debug.apk	.../LoggingSessionInputBuffer.java 82	2/20/13	38
	42	-	Gibberbot-debug.apk	.../LoggingSessionInputBuffer.java 84	2/20/13	71
	42	-	Gibberbot-debug.apk	.../LoggingSessionInputBuffer.java 90	2/20/13	112
	42	-	Gibberbot-debug.apk	.../LoggingSessionInputBuffer.java 115	2/20/13	28
	42	-	Gibberbot-debug.apk	.../LoggingSessionInputBuffer.java 120	2/20/13	51
	44	-	Gibberbot-debug.apk	de/.../ssl/MemorizingActivity.java 55	2/20/13	68
	44	-	Gibberbot-debug.apk	de/.../ssl/MemorizingActivity.java 56	2/20/13	49
	44	-	Gibberbot-debug.apk	de/.../ssl/MemorizingActivity.java 67	2/20/13	138
	47	-	Gibberbot-debug.apk	org/.../auth/NegotiateScheme.java 250	2/20/13	54
	48	-	Gibberbot-debug.apk	.../NetworkConnectivityListener.java 68	2/20/13	94
	51	-	Gibberbot-debug.apk	info/.../otr/OtrDebugLogger.java 13	2/20/13	140
	52	-	Gibberbot-debug.apk	net/.../OtrEngineImplTest.java 41	2/20/13	134
	60	-	Gibberbot-debug.apk	.../ResponseProcessCookies.java 133	2/20/13	99
	62	-	Gibberbot-debug.apk	com/.../jbosh/ServiceLib.java 148	2/20/13	55
	62	-	Gibberbot-debug.apk	com/.../jbosh/ServiceLib.java 172	2/20/13	26
	62	-	Gibberbot-debug.apk	com/.../jbosh/ServiceLib.java 190	2/20/13	123
	63	-	Gibberbot-debug.apk	net/.../session/SessionImpl.java 335	2/20/13	84
	63	-	Gibberbot-debug.apk	net/.../session/SessionImpl.java 351	2/20/13	92
	63	-	Gibberbot-debug.apk	net/.../session/SessionImpl.java 378	2/20/13	39
	63	-	Gibberbot-debug.apk	net/.../session/SessionImpl.java 430	2/20/13	104
	63	-	Gibberbot-debug.apk	net/.../session/SessionImpl.java 505	2/20/13	116
	63	-	Gibberbot-debug.apk	net/.../session/SessionImpl.java 581	2/20/13	56
	64	-	Gibberbot-debug.apk	info/.../app/SignoutActivity.java 61	2/20/13	60
	68	-	Gibberbot-debug.apk	.../trust/StrongTrustManager.java 599	2/20/13	15
	72	-	Gibberbot-debug.apk	info/.../ui/TorServiceUtils.java 135	2/20/13	117
	74	-	Gibberbot-debug.apk	info/.../xmpp/XmppConnection.java 1660	2/20/13	61

Module #	Class #	Module	Location	Fix By	Flaw Id
76	-	Gibberbot-debug.apk	.../xmpp/XmppStreamHandler.java 277	2/20/13	132
76	-	Gibberbot-debug.apk	.../xmpp/XmppStreamHandler.java 283	2/20/13	100

Code Quality(8 flaws)

Fix Required by Policy

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:

Leftover Debug Code (CWE ID 489)(8 flaws)

Description

A method may be leftover debug code that creates an unintended entry point in a web application. Although this is an acceptable practice during product development, classes that are part of a production J2EE application should not define a main() method. Whether this method can be remotely invoked depends on the configuration of the J2EE container and the application itself.

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

Recommendations

Remove debug code prior to deploying the application. Eliminate unnecessary entry points in deployed web applications to reduce the attack surface.

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
25	-	Gibberbot-debug.apk	.../DiscoverServices.java 62	2/20/13	57
26	-	Gibberbot-debug.apk	.../DiscoverServiceTypes.java 58	2/20/13	81
29	-	Gibberbot-debug.apk	info/.../asn1/util/Dump.java 14	2/20/13	120
39	-	Gibberbot-debug.apk	info/.../bouncycastle/LICENSE.java 61	2/20/13	29
40	-	Gibberbot-debug.apk	samples/ListServices.java 53	2/20/13	93

Module #	Class #	Module	Location	Fix By	Flaw Id
50	-	Gibberbot-debug.apk	samples/OpenJmDNS.java 40	2/20/13	101
58	-	Gibberbot-debug.apk	.../RegisterService.java 66	2/20/13	16
70	-	Gibberbot-debug.apk	.../TestShutdownHook.java 17	2/20/13	19

Credentials Management(6 flaws)

Fix Required by Policy

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 Password (CWE ID 259)(6 flaws)

Description

A method uses a hard-coded password that may compromise system security in a way that cannot be easily remedied. The use of a hard-coded password significantly increases the possibility that the account being protected will be compromised. Moreover, 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.

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

Recommendations

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

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
45	-	Gibberbot-debug.apk	.../MemorizingTrustManager.java 237	2/20/13	111
61	-	Gibberbot-debug.apk	.../smack/SASLAuthentication.java 386	2/20/13	110

Module #	Class #	Module	Location	Fix By	Flaw Id
68	-	Gibberbot-debug.apk	.../trust/StrongTrustManager.java 134	2/20/13	130
68	-	Gibberbot-debug.apk	.../trust/StrongTrustManager.java 139	2/20/13	95
74	-	Gibberbot-debug.apk	info/.../xmpp/XmppConnection.java 912	2/20/13	46
74	-	Gibberbot-debug.apk	info/.../xmpp/XmppConnection.java 919	2/20/13	119

Cryptographic Issues(20 flaws)

Fix Required by Policy

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:

Improper Validation of Host-specific Certificate Data (CWE ID 297)(1 flaw)

Description

The failure to validate host-specific certificate data may mean that, while the certificate read was valid, it was not for the site originally requested.

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

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
75	-	Gibberbot-debug.apk	org/.../smack/XMPPConnection.java 890	2/20/13	74

Insufficient Entropy (CWE ID 331)(19 flaws)

Description

Standard random number generators do not provide a sufficient amount of entropy when used for security purposes. Attackers can brute force the output of pseudorandom number generators such as rand().

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 key or session identifier, use a trusted cryptographic random number generator instead. These can be found on the Windows platform in the CryptoAPI or in an open source library such as OpenSSL.

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
5	-	Gibberbot-debug.apk	net/.../AuthContextImpl.java 200	2/20/13	114
22	-	Gibberbot-debug.apk	.../DESedeKeyGenerator.java 48	2/20/13	83
23	-	Gibberbot-debug.apk	info/.../DESKeyGenerator.java 40	2/20/13	13
24	-	Gibberbot-debug.apk	.../DigestMD5SaslClient.java 789	2/20/13	17
27	-	Gibberbot-debug.apk	org/.../smack/util/DNSUtil.java 59	2/20/13	89
28	-	Gibberbot-debug.apk	info/.../util/DNSUtil.java 61	2/20/13	67
31	-	Gibberbot-debug.apk	org/xbill/DNS/Header.java 147	2/20/13	30
46	-	Gibberbot-debug.apk	org/.../mime/MultipartEntity.java 119	2/20/13	82
46	-	Gibberbot-debug.apk	org/.../mime/MultipartEntity.java 121	2/20/13	31
49	-	Gibberbot-debug.apk	org/.../auth/NTLMEngineImpl.java 222	2/20/13	63
49	-	Gibberbot-debug.apk	org/.../auth/NTLMEngineImpl.java 234	2/20/13	52
55	-	Gibberbot-debug.apk	org/.../impl/tasks/Prober.java 85	2/20/13	11
57	-	Gibberbot-debug.apk	.../ReconnectionManager.java 24	2/20/13	69
59	-	Gibberbot-debug.apk	org/.../impl/tasks/Responder.java 80	2/20/13	109
67	-	Gibberbot-debug.apk	org/.../util/StringUtils.java 548	2/20/13	70
71	-	Gibberbot-debug.apk	.../tls/TlsRSAKeyExchange.java 123	2/20/13	121
73	-	Gibberbot-debug.apk	org/.../xbill/DNS/UDPClient.java 38	2/20/13	77
74	-	Gibberbot-debug.apk	info/.../xmpp/XmppConnection.java 504	2/20/13	27
74	-	Gibberbot-debug.apk	info/.../xmpp/XmppConnection.java 505	2/20/13	141

Insufficient Input Validation(3 flaws)

Fix Required by Policy

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:

Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') (CWE ID 470)(3 flaws)

Description

A call uses reflection in an unsafe manner. An attacker can specify the class name to be instantiated, which may create unexpected control flow paths through the application. Depending on how reflection is being used, the attack vector may allow the attacker to bypass security checks or otherwise cause the application to behave in an unexpected manner. Even if the object does not implement the specified interface and a ClassCastException is thrown, the constructor of the user-supplied class name will have already executed.

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

Recommendations

Validate the class name against a combination of white and black lists to ensure that only expected behavior is produced.

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
53	-	Gibberbot-debug.apk	org/.../PacketParserUtils.java 881	2/20/13	85
62	-	Gibberbot-debug.apk	com/.../jbosh/ServiceLib.java 153	2/20/13	75
66	-	Gibberbot-debug.apk	.../smack/SmackConfiguration.java 306	2/20/13	18

Time and State(1 flaw)

Fix Required by Policy

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

Module #	Class #	Module	Location	Fix By	Flaw Id
30	-	Gibberbot-debug.apk	.../FileBackedOutputStream.java 193	2/20/13	86

Low (7 flaws)

Code Quality(6 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)(1 flaw)

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

Module #	Class #	Module	Location	Fix By	Flaw Id
75	-	Gibberbot-debug.apk	org/.../smack/XMPPConnection.java 734		118

Use of Wrong Operator in String Comparison (CWE ID 597)(5 flaws)

Description

Using '==' to compare two strings for equality or '!=' for inequality actually compares the object references rather than their values. It is unlikely that this reflects the intended application logic.

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

Recommendations

Use the equals() method to compare strings, not the '==' or '!=' operator

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
3	-	Gibberbot-debug.apk	.../AccountSettingsActivity.java 115		41
41	-	Gibberbot-debug.apk	org/.../smack/LLPresence.java 273		131
54	-	Gibberbot-debug.apk	info/.../io/pem/PemHeader.java 53		20
56	-	Gibberbot-debug.apk	com/kenai/jbosh/QName.java 197		136
69	-	Gibberbot-debug.apk	.../SubscriptionProvider.java 46		72

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:

Information Exposure Through Sent Data (CWE ID 201)(1 flaw)

Description

Sensitive information may be exposed as a result of outbound network connections made by the application. This can manifest in a couple of different ways.

In C/C++ applications, sometimes the developer fails to zero out a buffer before populating it with data. This can cause information leakage if, for example, the buffer contains a data structure for which only certain fields were populated. The uninitialized fields would contain whatever data is present at that memory location. Sensitive information from previously allocated variables could then be leaked when the buffer is sent over the network.

Mobile applications may also transmit sensitive information such as email or SMS messages, address book entries, GPS location data, and anything else that can be accessed by the mobile API. This behavior is common in mobile spyware applications designed to exfiltrate data to a listening post or other data collection point. This flaw is categorized as low severity because it only impacts confidentiality, not integrity or availability. However, in the context of a mobile application, the significance of an information leak may be much greater, especially if misaligned with user expectations or data privacy policies.

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

Recommendations

In C/C++ applications, ensure that all struct elements are initialized or zeroed before being sent. In mobile appliactions, 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

Module #	Class #	Module	Location	Fix By	Flaw Id
65	-	Gibberbot-debug.apk	org/.../DNS/SimpleResolver.java 56		58

Very Low (0 flaws)

No flaws of this type were found

Info (0 flaws)

No flaws of this type were found

About Veracode's Methodology

The Veracode platform uses static and dynamic analysis (for web applications) to inspect executables and identify security flaws in your applications. Using both static and dynamic analysis helps reduce false negatives and detect a broader range of security flaws. The static binary analysis engine models the binary executable into an intermediate representation, which is then verified for security flaws using a set of automated security scans. 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 a 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 on a five point scale:

Severity	Name	Description
5	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.
4	High	The offending line or lines of code have significant weakness, and the code should be modified immediately to avoid potential attacks.
3	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.
2	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.
1	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.
0	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 0) 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 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
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

Current Version		Prior Version	
Application Version:	27 Feb 2013 Static - Not Specified	Application Version:	
Scan Date:	2/27/13	Scan Date:	2/20/13
Application Rating:	C	Application Rating:	

Flaws not detected in current scan

The following is a list of all flaws found in the prior scan of this application that were not detected in the current scan.

High (2 flaws)

Fix Required by Policy

SQL Injection(2 flaws)

Fix Required by Policy

Associated Flaws by CWE ID:

Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (CWE ID 89)(2 flaws)

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
11	-	Gibberbot-debug.apk	.../ChatSessionAdapter.java 537		25
15	-	Gibberbot-debug.apk	.../ContactsPickerActivity.java 91		22

Medium (5 flaws)

Fix Required by Policy

CRLF Injection(5 flaws)

Fix Required by Policy

Associated Flaws by CWE ID:

Improper Output Neutralization for Logs (CWE ID 117)(5 flaws)

Instances found via Static Scan

Module #	Class #	Module	Location	Fix By	Flaw Id
9	-	Gibberbot-debug.apk	com/.../jbosh/BOSHClient.java 961		107
32	-	Gibberbot-debug.apk	info/.../proxy/HttpManager.java 93		108
32	-	Gibberbot-debug.apk	info/.../proxy/HttpManager.java 161		87
43	-	Gibberbot-debug.apk	.../LoggingSessionOutputBuffer.java 103		135
47	-	Gibberbot-debug.apk	org/.../auth/NegotiateScheme.java 316		80

Appendix B: Referenced Source Files

Id	Filename	Path
1	AbstractMessageParser.java	org/apache/http/impl/io/
2	AccountActivity.java	info/guardianproject/otr/app/im/app/
3	AccountSettingsActivity.java	info/guardianproject/otr/app/im/app/
4	AndroidDebugger.java	de/measite/smack/
5	AuthContextImpl.java	net/java/otr4j/session/
6	BlockedContactsActivity.java	info/guardianproject/otr/app/im/app/
7	BodyParserSAX.java	com/kenai/jbosh/
8	BodyParserXmlPull.java	com/kenai/jbosh/
9	BOSHClient.java	com/kenai/jbosh/
10	ChatListenerAdapter.java	info/guardianproject/otr/app/im/app/adapter/
11	ChatSessionAdapter.java	info/guardianproject/otr/app/im/service/
12	ChatSessionListenerAdapter.java	info/guardianproject/otr/app/im/app/adapter/
13	ChunkedInputStream.java	org/apache/http/impl/io/
14	ContactPresenceActivity.java	info/guardianproject/otr/app/im/app/
15	ContactsPickerActivity.java	info/guardianproject/otr/app/im/app/
16	ContentLengthInputStream.java	org/apache/http/impl/io/
17	DefaultClientConnection.java	org/apache/http/impl/conn/
18	DefaultRedirectHandler.java	org/apache/http/impl/client/
19	DefaultRedirectStrategy.java	org/apache/http/impl/client/
20	DefaultRequestDirector.java	org/apache/http/impl/client/
21	DefaultResponseParser.java	org/apache/http/impl/conn/
22	DESedeKeyGenerator.java	info/guardianproject/bouncycastle/crypto/generators/
23	DESKeyGenerator.java	info/guardianproject/bouncycastle/crypto/generators/
24	DigestMD5SaslClient.java	com/novell/sasl/client/
25	DiscoverServices.java	samples/
26	DiscoverServiceTypes.java	samples/
27	DNSUtil.java	org/jivesoftware/smack/util/
28	DNSUtil.java	info/guardianproject/util/
29	Dump.java	info/guardianproject/bouncycastle/asn1/util/
30	FileBackedOutputStream.java	com/google/common/io/
31	Header.java	org/xbill/DNS/
32	HttpManager.java	info/guardianproject/onionkit/proxy/
33	HttpRequestParser.java	org/apache/http/impl/io/
34	HttpResponseParser.java	org/apache/http/impl/io/
35	ImApp.java	info/guardianproject/otr/app/im/app/
36	Imps.java	info/guardianproject/otr/app/im/provider/
37	ImpsProvider.java	info/guardianproject/otr/app/im/provider/

Id	Filename	Path
38	ImUrlActivity.java	info/guardianproject/otr/app/im/app/
39	LICENSE.java	info/guardianproject/bouncycastle/
40	ListServices.java	samples/
41	LLPresence.java	org/jivesoftware/smack/
42	LoggingSessionInputBuffer.java	org/apache/http/impl/conn/
43	LoggingSessionOutputBuffer.java	org/apache/http/impl/conn/
44	MemorizingActivity.java	de/duenndns/ssl/
45	MemorizingTrustManager.java	de/duenndns/ssl/
46	MultipartEntity.java	org/apache/http/entity/mime/
47	NegotiateScheme.java	org/apache/http/impl/auth/
48	NetworkConnectivityListener.java	info/guardianproject/otr/app/
49	NTLMEngineImpl.java	org/apache/http/impl/auth/
50	OpenJmDNS.java	samples/
51	OtrDebugLogger.java	info/guardianproject/otr/
52	OtrEngineImplTest.java	net/java/otr4j/test/
53	PacketParserUtils.java	org/jivesoftware/smack/util/
54	PemHeader.java	info/guardianproject/bouncycastle/util/io/pem/
55	Prober.java	org/jmdns/impl/tasks/
56	QName.java	com/kenai/jbosh/
57	ReconnectionManager.java	org/jivesoftware/smack/
58	RegisterService.java	samples/
59	Responder.java	org/jmdns/impl/tasks/
60	ResponseProcessCookies.java	org/apache/http/client/protocol/
61	SASLAuthentication.java	org/jivesoftware/smack/
62	ServiceLib.java	com/kenai/jbosh/
63	SessionImpl.java	net/java/otr4j/session/
64	SignoutActivity.java	info/guardianproject/otr/app/im/app/
65	SimpleResolver.java	org/xbill/DNS/
66	SmackConfiguration.java	org/jivesoftware/smack/
67	StringUtils.java	org/jivesoftware/smack/util/
68	StrongTrustManager.java	info/guardianproject/onionkit/trust/
69	SubscriptionProvider.java	org/jivesoftware/smackx/pubsub/provider/
70	TestShutdownHook.java	samples/
71	TlsRSAKeyExchange.java	info/guardianproject/bouncycastle/crypto/tls/
72	TorServiceUtils.java	info/guardianproject/onionkit/ui/
73	UDPClient.java	org/xbill/DNS/
74	XmppConnection.java	info/guardianproject/otr/app/im/plugin/xmpp/
75	XMPPConnection.java	org/jivesoftware/smack/

Id	Filename	Path
76	XmppStreamHandler.java	info/guardianproject/otr/app/im/plugin/xmpp/