Sample Secure Code Review Report

1. The Code Review Process

A Secure Code Review is a specialized task with the goal of identifying types of weaknesses that exist within a given code base. The task involves both manual and automated review of the underlying source code and identifies specific issues that may be representative of broader classes of weakness inherent in the code. A Secure Code Review does not attempt to identify every issue in the code, but instead attempts to identify types of risk within the code such that mitigation strategies can be devised.

During the actual review, members of a review team review the application code for security problems and categorize the findings based on the weakness categories (e.g., authentication, authorization, etc.). Each finding is assigned a risk rating of High, Medium, Low, or Informational. These findings and the broader weakness classes that they represent are presented in this final report that the development team can use as the foundation for improving the overall quality of the code base.

It should be noted that while the review process will be as thorough as possible in finding and reporting security weaknesses, it is not guaranteed to always find every possible weakness. If no issues are found, the review does not implicitly certify that the application is 100-percent “hack proof.”

A Secure Code Review is not a silver bullet, but instead is a strong part of an overall risk mitigation program to protect an application.

2. Review Summary

The secure code review of the Example App application was completed on October 17, 2013 by a review team consisting of [redacted name] and [redacted name]. The review was performed on code obtained from [redacted name] via email attachment on October 11, 2013, and bundled under the file named example_app_v2.tar.gz.

A meeting between the review team, [redacted name] and [redacted name] was held on October 7, 2013, at which time information about the code structure was presented along with high level overviews of how things like authentication, data validation, and logging were implemented in the code. This information was used by the review team to formulate a plan for the impending review.

The actual review involved a manual investigation of the Java code. Specific source files were not assigned to individual members; rather, each member of the review team attempted to review the entire application. Each reviewer recorded their specific findings within a spreadsheet and assigned risk levels as they felt appropriate. At the end of the review, the team looked across the individual spreadsheets to compare common findings and to perform group reviews of the uncommon findings. The specific findings are

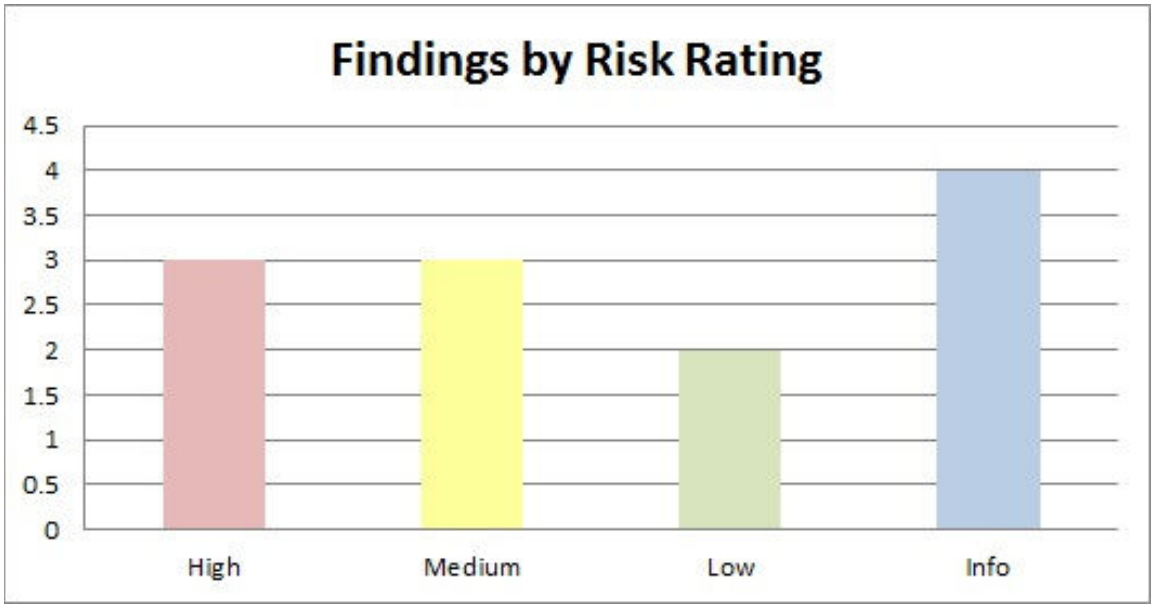
presented in the next section.

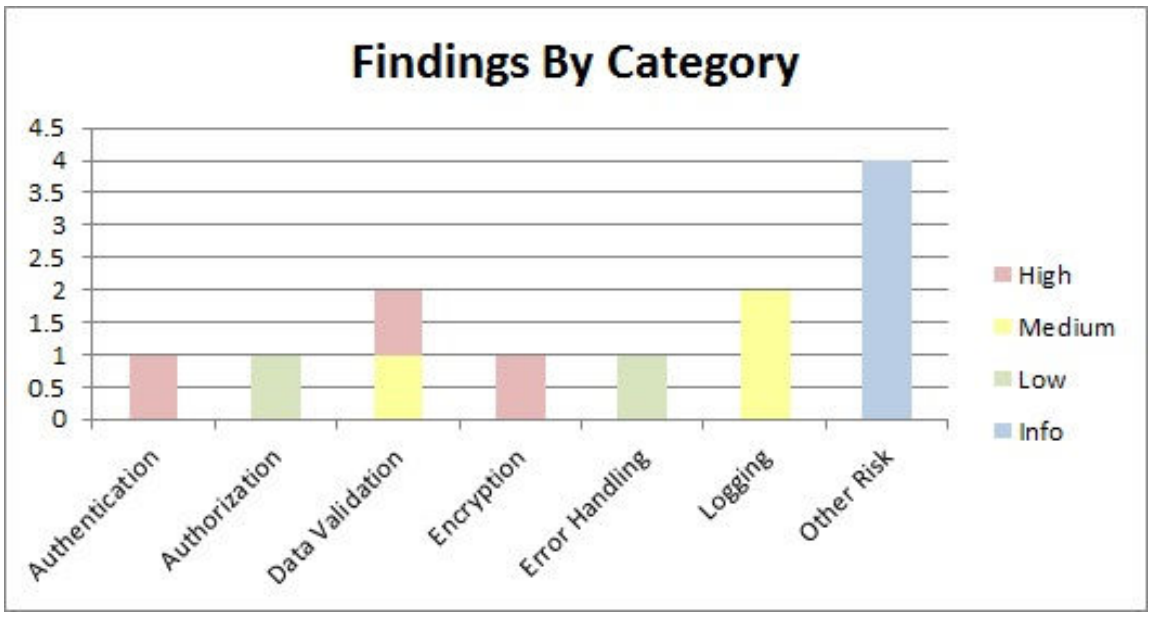
3. Finding Summary

This section provides a summary of the findings resulting from this review.

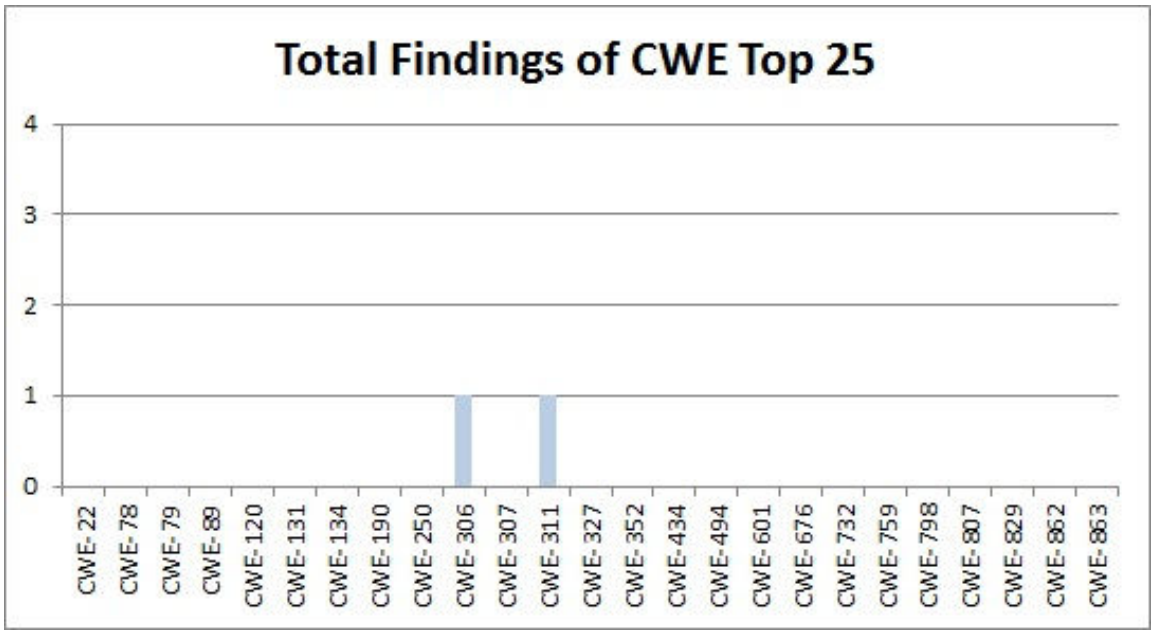
For this application, three high level issues were found related to the areas of authentication and data validation. One of the high level issues resulting from unvalidated attacker input being sent to the JSON parse() function could result in arbitrary commands being executed. Mitigating actions should be considered. Some other medium and low issues have also been found. Details are given below.

The figures below graphically outline the review team's findings by both category and risk level.





The figure below shows the findings related to the CWE Top 25 list. Details related to the specific CWE IDs can be found in the next section. It should be noted that the exact CWE number may not be found in the next section as a child CWE may have been used to report a finding. This child CWE is still counted as an instance of a parent that is in the Top 25.



4. Finding Details

This section provides details about the specific weaknesses that were found during the review. These details are designed to provide the developers with proof that the stated weaknesses exist as well as to provide examples that the developers can use to find and fix similar areas of the code. As mentioned before, the Secure Code Review does not claim to find every issue; as such the development team should use the information in these findings as an opportunity to improve the entire code base. Just fixing the specific examples identified below will most likely not remove the higher level risks from the application.

Each finding is given a qualitative risk rating assigned by the reviewers at the time of the review. The general guidelines used when assigning risk levels are as follows:

- **High** - Serious impact to the application security, generally unmitigated, large-scale issues, such as an attack that is currently exploitable from the Internet.
- **Medium** - Notable impact to the application security, or somewhat mitigated high risks (e.g., being available only to the user's Intranet).
- **Low** - Potential impact to the application security, or heavily mitigated high risk (e.g., being in dead code or after an abort call).
- **Informational** – Does not directly make the code less secure, but bad coding practice.

The risk ratings should be considered risks to the application itself. In other words, the risk that the application behavior could be subverted in an unintended way could lead to a possible compromise. This information should then be used by the appropriate teams (developers/management/Information Security) in conjunction with the additional 'big picture' information that they have, to make the appropriate risk mitigation decisions.

4.1 Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

Category: Data Validation
Weakness: CWE-95 -- The software receives input from an upstream component, but it does not neutralize, or incorrectly neutralizes, code syntax before using the input in a dynamic evaluation call (e.g., "eval").

Source File	Line Number	Description	Risk
src\main\java\org\mitre\example.java	117	On line 117, unvalidated input is passed into the Json parse() function. Due to the use of eval() in the implementation of parse(), this leaves the application subject to command injection attacks. The JsonParser	High

		documentation reads: "Evaluates a trusted JSON string and returns its JSONValue representation. CAUTION! For efficiency, this method is implemented using the JavaScript eval() function, which can execute arbitrary script. DO NOT pass an untrusted string into this method." Some amount of data validation should be performed on the input in an effort to determine if it can be trusted.	

4.2 Authentication Bypass Issues

Category: Authentication
Weakness: CWE-592 -- The software does not properly perform authentication, allowing authentication to be bypassed through various methods.

Source File	Line Number	Description	Risk
src\main\java\org\mitre\client.java	2110-2114	The block of code on line 2110-2114 of HttpClientFetchService.java seems to allow an automatic fall back to a non-OAuth request. The review team was not able to fully explore this; however, it looks suspicious and potentially is a security concern. This code should probably be revisited before the final release.	High

4.3 Cleartext Transmission of Sensitive Information

Category: Encryption
Weakness: CWE-319 -- The software transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.

Source File	Line Number	Description	Risk
src\main\webapp\WEB-INF\spring\appServlet\security-cont ext.xml	65	The credentials are retrieved from the OAuth2 server specified in this file and is not done using https. If the token returned is compromised, an attacker could masquerade as the user and gain access to the data.	High

4.4 Insufficient Logging

Category: Logging
Weakness: CWE-778 -- When a security-critical event occurs, the software either does not record the event or omits important details about the event when logging it.

If security critical information is not recorded, there will be no trail for forensic analysis and discovering the cause of problems or the source of attacks may become more difficult or impossible to identify.

Source File	Line Number	Description	Risk
src\main\java\org\mitrefetch.java	1310-1314	If the OAuth request fails and the application falls back to a non-OAuth request, then a log entry should probably be done noting this.	Medium
src\main\java\org\mitre\fetch.java	941	A log entry was supplied in the previous catch block; one should probably be supplied here as well.	Medium

4.5 Improper Input Validation

Category: Data Validation
Weakness: CWE-20 -- The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.

When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Source File	Line Number	Description	Risk
src\main\java\org\mitre\aggregator.java	294	The userid that is obtained from the request string is never validated. Although this doesn't appear to be harmful in its current use, it is still advisable to perform some amount of rudimentary validation on the input. Data validation is the easiest way to prevent most security issues from occurring.	Medium

4.6 Missing Authentication for Critical Function

Category: Authorization
Weakness: CWE-306 -- The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

Source File	Line Number	Description	Risk
N/A	N/A	The application allows anyone to make a request for consolidated contact information related to a given employee ID. It is understood that this is the current choice of the design team; however this issue should be	Low

		addressed if the application is ever used in a more open environment where not everyone that could make a request should have access to the contact information.	

4.7 Exposure of System Data to an Unauthorized Control Sphere

Category: Error Handling
Weakness: CWE-497 -- Exposing system data or debugging information helps an adversary learn about the system and form an attack plan.

Source File	Line Number	Description	Risk
src\main\java\org\mitre\client.java	170, 173	The error messages generated on lines 170 and 173 contain stacktraces. If these are sent back to the user, they could provide an attacker with valuable information about the underlying system. Stacktraces should be reserved for a log and kept out of error messages that are sent back to the user.	Low

4.8 Suspicious Comment

Category: Other Risk
Weakness: CWE-546 -- The code contains comments that suggest the presence of bugs, incomplete functionality, or weaknesses.

Many suspicious comments, such as BUG, HACK, FIXME, LATER, LATER2, TODO, in the code indicate missing security functionality and checking. Others indicate code problems that programmers should fix, such as hard-coded variables, error handling, not using stored procedures, and performance issues.

Source File	Line Number	Description	Risk
src\main\java\org\mitre\service.java	159	The comment on line 159 contains a "TODO," which implies that more work needs to be done. However, this does not appear to be the case. As written, the comment could lead to a future maintainer of the code attempting to fix something incorrectly. Either the details of the comment should be improved or consideration should be given to removing it.	Info
src\main\java\org\mitre\auth.java	117, 124	Two comments in this file (lines 117 and 124) call out TODO items. These items should be addressed before the code is released -	Info

		especially the one on line 124 that implies improper handling of the secret. Note that we could not determine why the secret comment was there.	
src\main\java\org\mitre\fetch.java	1580	The comment on line 1580 implies an incomplete implementation that could lead to a compromise. "FIXME: HACK AND A HALF" This should be addressed before the code is released.	Info

4.9 Copyright and Confidentiality Statements

Category: Other Risk
Weakness: N/A - copyright --

Clear copyright should be asserted by whoever will be the appropriate party to own copyright on this application.

Source File	Line Number	Description	Risk
N/A	N/A	Clear copyright is missing. It should be part of the header for each code file.	Info

Appendix A References

Christey, S. (2010). 2010 CWE/SANS Top 25 Most Dangerous Programming Errors. Retrieved from http://cwe.mitre.org