Oracle® Healthcare Foundation

Secure Development Guide

Release 7.2.1

F11014-01

October 2018

The Secure Development Guide provides an overview for developers using the *Oracle® Healthcare Foundation Admin Console user interface* and the *File upload and Omics data ingestion API services* on how to assist in mitigating common security risks.

The Open Web Application Security Project (OWASP) publishes a list of top 10 critical web application security vulnerabilities identified each year. The OWASP Top 10 vulnerability list is technology agnostic and does not contain language or framework specific examples, explanations, hints, or tips. This document provides software developer insights into how the API was created and can be used while addressing the top 10 security vulnerabilities documented in the OWASP Top 10 for 2013. Since defense in depth is a principle strategy for a secure product, do not exclusively rely on the techniques documented herein. Implement and extend these techniques in your own code as you develop your interface to the API specification.

The set of recommendations in this document is not exhaustive and no guarantee is given that implementing all the suggestions in this document provides sufficient protection for all security threats. The reason for this disclaimer is that you cannot delegate responsibility for secure application development to a third party or to a single document. This document is to help developers that know the security tools and features that they can use to implement application security. This document does not replace a formal review process.

Introduction

To guide developers on how to protect against threats, the Open Web Application Security Project publishes an annual document that lists the 10 most critical security vulnerabilities identified in a year. Addressing these ten security vulnerabilities doesn't provide total security, but is a good start in raising awareness about the current major security threats. This document explains how the API and API developers should address security vulnerabilities and risks documented by OWASP for 2013.

OWASP Top 10 Security Vulnerabilities in 2013

The OWASP Top 10 list for the year 2013 doesn't differ much from lists published for previous years, except for changes in ranking. The listed security threats are probably the most severe threats and application developers have to be aware of and protect against these threats.

General descriptions for the OWAP Top 10 list of security risks for 2013 are documented at:

https://www.owasp.org/index.php/Top_10_2013-Top_10

along with an overview describing application security risks at:

https://www.owasp.org/index.php/Top_10_2013-Risk.

Security Awareness and Education

Education is the best investment in application security. Developers and project leads must be mindful of security issues and have an understanding of secure coding practices. Training must include an in depth explanation of the potential risks as well as features of the development and deployment platforms that help mitigate exploits.

The most important design principle for application security is to implement security by design and by default. Secure coding guidelines should be made available, adhered to and enforced in all development organizations, irrespective of the tools and platforms used.

A good example for security by default is the expectation we all have for how elevators should behave in case of a power outage. In such a scenario, we expect elevators to apply the breaks for the safety of the passengers in the cabin. Elevator brakes are closed by default and use an electrical mechanism to hold them open. The approach is not to apply the brakes in case of a power failure, but to require electricity to unlock the brakes that are otherwise closed.

This illustrates the concept of security by default. Before thinking about how to prevent external attacks, it makes sense to identify secure defaults for an application internally. This, however, requires training and awareness.

The Risk Associated with "Building Your Own"

Developers don't always find the security they need for an application within the security toolset provided by a platform or built into a framework. As a result, "building your own security" is not uncommon among development projects. This is especially true if the application is a replacement of an existing system that uses a non-standard security infrastructure. An example for this would be a database table based authentication and authorization combined with user provisioning and resource granting at runtime.

The risk associated with building your own security is that you are also on your own when it comes to: quality assurance of the security layer, application security propagation, and single sign-on; as well as being responsible for bug fixing and maintaining the security layer.

Not all developers are security experts, but it takes experts to build a custom security layer.

Time spent investigating existing and well vetted security solutions is probably time well spent. It is easier and more cost-effective to apply existing solutions to custom applications than to create an error-prone, self written mechanism.

Top 10 Security Risks for 2013

The sections below identify the controls in the Oracle OHF API that are used or may be used to address the OWASP Top 10 security vulnerabilities. In some cases, the controls are integrated into the product and proper use of the controls by the client is required to validate the integrity of the controls.

#1 - Injection

Injection vulnerabilities occur when data is sent into an interpreter via an interface specification and the party submitting the data does not check the data to ensure only the expected actions are performed by the interpreter on the data. SQL, Code, Command, Log, Path Transversal (XML) are all possible types of injection depending on the interpreter used in the container.

Valid Content Types

The inbound provisioning REST service only supports the JSON data format. The service client must accept and send *'application/json'* as the media type while invoking the REST service.

SQL Injection

To prevent SQL injections, the inbound provisioning REST API uses bind variables in SQL queries, which block SQL injection possibilities.

XML Injection

The inbound provisioning REST API accepts and generates only the JSON data format. Therefore, XML injections are not possible.

LDAP Injection

The inbound user provisioning REST API checks all incoming user input through a preset whitelist of allowed inputs and checks the pattern of the ultimate payload for any attack. However, custom code for protecting against LDAP injection at the client side it always welcome.

#2 - Broken Authentication and Session Management

Risks associated with broken authentication and session management are often due to these functions not being implemented properly. As previously stated, custom authentication mechanisms should not be implemented and have not been implemented. The inbound user provisioning REST API uses BASIC authentication as the authentication mechanism.

#3 - Cross Site Scripting (XXS)

The OHF REST API mitigates cross site scripting by following the recommended methodologies listed on the OWSAP website. The best method to address XXS is to validate all data using whitelists and encode data as necessary given the presentation or headline of the data.

#4 - Insecure Direction Object References

When a developer exposes a reference to an object without proper access or other protection, this reference becomes an attack vector. When developing code and sending data to and from the API, make sure that the authorization model of the API interface is consistent to guard against insecure direction object references.

The authorization model in the SCIM interface protects down to the object level and the SCIM interface has been tested to validate proper authorization constructs within the functions of the defined service. Any client code built to integrate with the SCIM

interface should complement this security model so that proper authorization is controlled at the object level.

#5 - Security Misconfiguration

Consult the product secure configuration document to ensure the product API is locked down properly.

#6 - Sensitive Data Exposure

Not all data is public and caution should be used to hide sensitive information from unauthorized users. Failure in security configuration and the selection of insecure defaults may pose a risk of data leakage.

Web client developers should enforce encrypted data transport when the application transports sensitive data and should validate that all certificates are legitimate and signed by public authorities. Ciphers should be restricted to modern implementations.

#7 - Missing Function Level Access Control

As a best practice, never assume that a specific method will only be called within the context that it was initially designed for. All access to functionality that manipulates data must be protected either by access control on the entity or by guarding the invocation of methods with the appropriate permission checks. The credentials of the identity associated with the access control at the client application must be encrypted and stored in a secure fashion.

#8 - Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery requires a browser container. In general, APIs are not meant to be supported directly in a browser container, so the session is not kept as a browser cookie and CSRF is not a viable threat.

#9 - Using Components with Known Vulnerabilities

The API stack is constantly updated with the latest security fixes and patches. Oracle recommends that developers using the API do the same.

#10 - Non-validated Redirects and Forwards

User Provisioning API responses do not provide URLs to other resources or sub-resources in the returned representations. Therefore, this API is not subject to this vulnerability.

Other Aspects of Security

Application security is useless if the application itself runs in an insecure environment. Perimeter security describes the levels of protection that are added on servers, the network, and other data access channels outside of the API domain. As can be seen in this document, not all of the OWASP Top 10 security vulnerabilities documented for 2013 are relevant for application developers for specific implementations.

Appendix: Recommended Readings

OWASP Home Page:

https://www.owasp.org/index.php/Main_Page

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Oracle Healthcare Foundation Secure Development Guide, Release 7.2.1
F11014-01