Secure Coding Guidance

Secure Coding Guidelines

In order to insure limited exposure to security exploits within the opnfv platform, we recommend developers learn and implement secure coding practices. The following contains recommendations from CWE/SANS, The OpenStack Security Group (very good python resource) and CERT, OWASP. If anyone has other resources or contributions they feel are useful additions, please make recommendations to the opnfv security group.

Gerrit Security Impact Code Review

We also strongly recommend to make use of the Gerrit secure code review tag we have. Any commit (or commit amend) with the words 'SecurityImpact' will automatically email the security group for gerrit based review.

Help review!

You can never have enough eyes, doing security code reviews. If you have an interest in helping review code, please sign up to the opnfv-security list

CWE/SANS Top 25

A very good reference for those new (and old) to secure coding conventions, is the CWE/SANS Top 25 Most Dangerous Programming Errors. The CWE give real world examples, of exactly how insecure code can be exploited.

Below is a simplified list, with a link to the CWE page that contains greater detail.

**Improper Input Validation**

Ensure that your input is valid. If you're expecting a number, it shouldn't contain letters! Also why accept 100 chars, limit input to only what you need!

http://cwe.mitre.org/data/definitions/20.html

**Improper Encoding or Escaping of Output**

Improper encoding or escaping can allow attackers to change the commands that are sent to another component, inserting malicious commands instead.

http://cwe.mitre.org/data/definitions/116.html

**Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**

Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data.

http://cwe.mitre.org/data/definitions/89.html

**Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')**

Software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

http://cwe.mitre.org/data/definitions/79.html

**Failure to Preserve OS Command Structure (aka 'OS Command Injection') =**

Software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

http://cwe.mitre.org/data/definitions/78.html

**Cleartext Transmission of Sensitive Information**

Avoid transmit of sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors. It should always be encrypted, using a secure, strong algorithm.

http://cwe.mitre.org/data/definitions/319.html

**Cross-Site Request Forgery (CSRF)**
When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.

http://cwe.mitre.org/data/definitions/352.html

**Race Conditions**

The program contains a code sequence that can run concurrently with other code, and the code sequence requires temporary, exclusive access to a shared resource, but a timing window exists in which the shared resource can be modified by another code sequence that is operating concurrently.

http://cwe.mitre.org/data/definitions/362.html

**Information Exposure Through an Error Message**

Watch those stack traces! An error message should never include sensitive information about its environment, users, or associated data.

http://cwe.mitre.org/data/definitions/209.html

**Buffer Overflows**

Software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer, thereby allowing a hacker to execute malicious instructions.

http://cwe.mitre.org/data/definitions/119.html

**External Control of Critical State Data**

Avoid of be careful of storing critical state data into open readable configuration files, profiles, cookies, hidden form fields, environment variables, registry keys, or other locations, all of which can be modified by an attacker.

http://cwe.mitre.org/data/definitions/642.html

**External Control of File Name or Path**

Never allow user input to control or influence paths or file names that are used in filesystem operations.

http://cwe.mitre.org/data/definitions/73.html

**Untrusted Search Path**

Avoid exposure of critical resources using an externally-supplied search path that can point to resources that are not under the application's direct control.

http://cwe.mitre.org/data/definitions/426.html

**Improper Control of Generation of Code ('Code Injection')**

Software constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment.

http://cwe.mitre.org/data/definitions/94.html

**Download of Code of anything without Integrity Check(checksums)**

Never download source code, patches, updates or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code. This is typically achieved using checksums, or key based signing.

http://cwe.mitre.org/data/definitions/494.html

**Improper Resource Shutdown or Release**

The program does not release or incorrectly releases a resource before it is made available for re-use. 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, such as a set period of time or revocation.

http://cwe.mitre.org/data/definitions/404.html

**Improper Initialization**
Software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used. If you don't properly initialize your data and variables, an attacker might be able to do the initialization for you, or extract sensitive information that remains from previous sessions.

http://cwe.mitre.org/data/definitions/665.html

**Incorrect Calculation**

Software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management. When attackers have control over inputs to numeric calculations, math errors can have security consequences. It might cause you to allocate far more resources than you intended - or far fewer

http://cwe.mitre.org/data/definitions/682.html

**Improper Authorization**

Software does not perform or incorrectly performs an authorization check when an actor attempts to access a resource or perform an action. If you don't ensure that your software's users are only doing what they're allowed to, then attackers will try to exploit your improper authorization and exercise that unauthorized functionality.

http://cwe.mitre.org/data/definitions/285.html

**Use of a Broken or Risky Cryptographic Algorithm**

The use of a non-standard, old or broken algorithm is dangerous because a determined attacker may be able to break the algorithm and compromise whatever data has been protected. Well-known techniques may exist to break the algorithm.

http://cwe.mitre.org/data/definitions/327.html

**Use of Hard-coded Password**

Software contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.

Example:

```java
DriverManager.getConnection(url, "scott", "tiger");
```

```java
javap -c ConnMngr.class
22: ldc #36; //String jdbc:mysql://ixne.com/rxsql
24: ldc #38; //String scott
26: ldc #17; //String tiger
```

http://cwe.mitre.org/data/definitions/259.html

**Incorrect Permission Assignment for Critical Resource**

Software specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

http://cwe.mitre.org/data/definitions/732.html

**Use of Insufficiently Random Values**

Software may use insufficiently random numbers or values in a security context that depends on unpredictable numbers. Be especially careful of this in virtualized environments. Quite often a VNF might be dependent on a host entropy source, and that source should always be trusted.

http://cwe.mitre.org/data/definitions/330.html

**Execution with Unnecessary Privileges**

Your software may need special privileges to perform certain operations; wielding those privileges longer than necessary is risky.

http://cwe.mitre.org/data/definitions/250.html

**Client-Side Enforcement of Server-Side Security**

Don't trust the client to perform security checks on behalf of your server. Attackers can reverse engineer your client and write their own custom clients.

http://cwe.mitre.org/data/definitions/602.html
Other useful reference sources

You may find the following resources also very helpful.

**OSSG Developer Guidelines**

The OSSG (OpenStack Security Group) have authored a very good set of guidelines (mostly specific to python). These can be found on the following Github repository

**OWASP**

OWASP have authored a Secure Coding Practices Quick Reference Guide is a technology agnostic set of general software security coding practices, in a comprehensive checklist format, that can be integrated into the development lifecycle. At only 17 pages long, it is easy to read and digest. Details on the OWASP wiki

Of key interest are the cheatsheets, namely:

- REST Security Cheat Sheet
- SQL Injection

Last, but not least, the OWASP top ten (a need of an update, but still relevant)

**CERT**

The CERT standards are a very good free resource. Languages covered are C, C++, Java, Perl. It is currently hosted on a confluence space at cert over here

**safecode**

The Safe Code Development Practices guide

**IEEE Computer Society's Center for Secure Design (CSD)**

Avoiding the Top 10 Security Flaws