

OWASP Orizon project

safe coding and beyond

20080908

© 2002-2008 OWASP Foundation

This document is licensed under the Creative Commons [Attribution-ShareAlike 2.5](http://creativecommons.org/licenses/by-sa/2.5/) license

# Table of Contents

[Table of Contents 2](#_Toc166867344)

[Introduction 3](#_Toc166867345)

[Summary 4](#_Toc166867346)

[Methodology 5](#_Toc166867347)

[A1 – Cross Site Scripting (XSS) 8](#_Toc166867348)

[A2 – Injection Flaws 11](#_Toc166867349)

[A3 – Malicious File Execution 14](#_Toc166867350)

[A4 – Insecure Direct Object Reference 18](#_Toc166867351)

[A5 – Cross Site Request Forgery (CSRF) 20](#_Toc166867352)

[A6 – Information Leakage and Improper Error Handling 23](#_Toc166867353)

[A7 – Broken Authentication and Session Management 25](#_Toc166867354)

[A8 – Insecure Cryptographic Storage 27](#_Toc166867355)

[A9 – Insecure Communications 29](#_Toc166867356)

[A10 – Failure to Restrict URL Access 31](#_Toc166867357)

[Where To Go From Here 33](#_Toc166867358)

[References 35](#_Toc166867359)

# Introduction

As Internet usage grows, people feel the need to publish services over the Net. No matter if we are talking about small companies or big international corporate, services offered to customers must meet security requirements.

Security is the keyword in the “Web2.0 era”. Everybody needs security and most important everybody knows as to provide people security. In facts, there is a lot of confusion when people talk about security in applications.

It is not so uncommon to meet people pretending a firewall to solve their application related security issues. It is not so uncommon to meet people that do not feel the need about having security embedded in their SDLC[[1]](#footnote-2).

Software engineers does not care about security when showing people theory behind software development. A reason could be the fact that security arises as topic in software development in these years and not when SDLC was first designed.

A lot of security experts have filled such gap with documentation, code snippets, know-how, best practices and everything related to build an application in a secure[[2]](#footnote-3) way.

(more to fill here)

A lot of tools are available in the wild (both *open* than closed as *source* code) to help people in building hardened code.

Orizon is an opensource project, sponsored by Owasp [1], born in the late 2006 to answer the need about having a common code review engine that people can use to build source code assessment tools.

This engine goal is to provide both APIs to developers than a collection of safe coding best practices to provide a common ground to open source tools built upon Orizon.

(more to fill here)

## about the author

Paolo Perego (aka thesp0nge) is a Senior Security consultant for Spike Reply where he works as penetration tester, code reviewer and SSDLC designer.

He started as Linux Kernel hacker in 1996 with a security project with the ambitious goal to turn a linux box into an unoffensive party in a network. AngeL project was a LSM that hijack linux system call table and netfilter hooks placing sanity checks before packet leaving the box or system call execution granted to the user.

He is now devoted to source code assessment and code reviewing mission and he leads the Owasp Orizon project, an opensource code review engine.

He is also a Owasp Code Review Guide author, contributing to defining source code flaws categories and a scoring system for a code review tool.

## Aim

This document’s goal is to describe Owasp Orizon internals.

## Acknowledgements

Owasp Orizon Project Lead: Paolo Perego

# The Owasp Orizon Framework architecture

During the OWASP Summer of code 2008 edition, the framework architecture dramatically evolved. As you may see in Figure 1, besides well known per engine vertical structure is tied with an horizontal row of engine to be used internally.

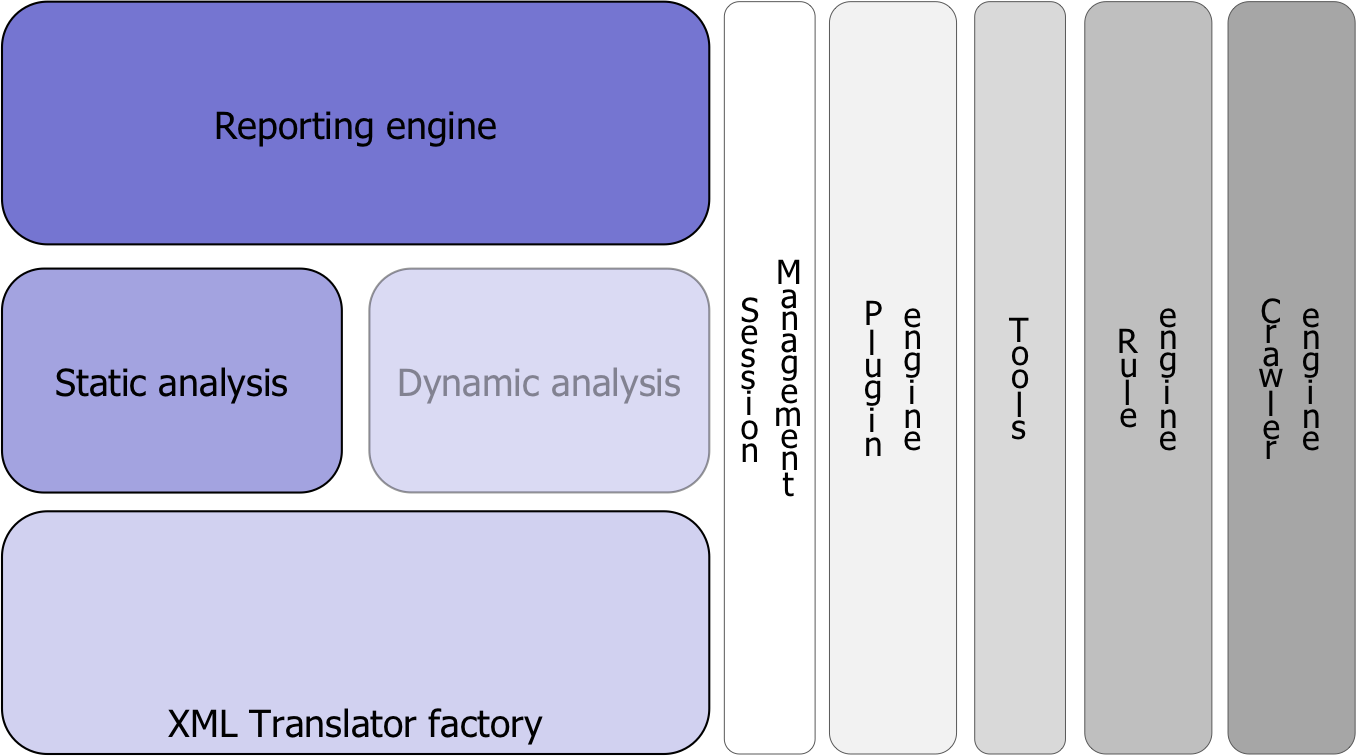


Figure 1 Owasp Orizon 1.0 architecture

The blue services are the ones provided to regular users.

## Translation factory

## Static analysis engine

## Report engine

## Session management

## Plugin engine

## Rule engine

## Crawler engine

# Hacking the Orizon

Hacking Owasp Orizon project is an activity for people interested in evolving the project, adding new features and working over the existing ones.

Owasp Orizon is a Java application deployed as standalone JAR file and based upon ANT as building system. The main workstation used for development is an Apple Macbook laptop, with Mac OS X 10.5.4 operating system. However the source is buildable and is running in every operating system supported by Sun Java SE 6 or later.

The source code is divided in packages grouped by logical block as shown in Figure 1. Main package is org.owasp.orizon containing source code managing sessions and the framework very core services.

## Core Services

Core services are the ones devoted to tie together all Owasp Orizon classes. Contained in the org.owasp.orizon package, the core services are devoted to:

* Session management;
* Logging facilities
* Versioning
* Support facility (SkyLine class)

Now we are going to describe the core services implementation with a brief details over the source code.

### Your personal butler: the SkyLine class

Named '''core''' in the architectural picture (Figure 1), SkyLine object is one of the most valuable news into Orizon version 1.0.

The idea behind SkyLine is simple: the Orizon architecture becomes wider, regular developers may be scared about understanding a lot of APIs in order to build their security tool, so we can help them providing a "per service" support.

Using SkyLine object, developers can ask services to Orizon framework waiting for their accomplishment.

The main SkyLine input is:

**public** **boolean** launch(**String** service)

Passing the requested service as string parameter, the calling program will receive a boolean true return value if the service can be accomplished or a false value otherwise. The service name is compared to the ones understood by the framework:

**private** **int** goodService(**String** service) {

**int** ret = -1;

**if** (service.equalsIgnoreCase("init"))

ret = Cons.OC\_SERVICE\_INIT\_FRAMEWORK;

**if** (service.equalsIgnoreCase("translate"))

ret = Cons.OC\_SERVICE\_INIT\_TRANSLATE;

**if** (service.equalsIgnoreCase("static\_analysis"))

ret = Cons.OC\_SERVICE\_STATIC\_ANALYSIS;

**if** (service.equalsIgnoreCase("score"))

ret = Cons.OC\_SERVICE\_SCORE;

**return** ret;

}

The secondary feature introduced in this first major framework release is the support for command line option given to the user.

If the calling program passes command line option to Orizon framework using SkyLine, the framework will be tuned accordingly to the given values.

This example will explain better:

**public** **static** **void** main(**String**[] args) {

**String** fileName = "";

OldRecipe r;

DefaultLibrary dl;

SkyLine skyLine = **new** SkyLine(args);

That's all folks! Internally, the SkyLine constructor when it creates a code review session, it uses the values it was able to understand from command line. The command line format must follow this convention

-o orizon\_key=value

or the long format

--orizon orizon\_key=value

And these are the keys that the framework cares about:

* “input-name"
* "input-kind"
* "working-dir"
* "lang"
* "recurse"
* "output-format"
* "scan-type";

The org.owasp.orizon.Cons class contains a detailed section about these keys with some comments and with their default value. The only side effect is that calling program can use -o flag for its purpose.

### Give me something to remind: the Session class

Another big news introduced in Owasp Orizon version 1.0 is the code review session concept. One of the missing features in earlier versions was the capability to track the state of the code review process.

A Session class instance contains all the properties specified using SkyLine and it is their owner giving access to properties upon request. It contains all a SessionInfo array containing information about each files being reviewed.

Ideally a user tool will never call Session directly but it must use SkyLine as interface, of course anyone is free to override this suggestion.

Looking at the launch() method code, inside the SkyLine class, you can look how session instance is prompted to execute services.

**public boolean** launch(**String** service) {

**int** code, stats;

**boolean** ret = **false**;

if ( (code = goodService(service)) == -1)

**return** log.error("unknown service: " + service);

**switch** (code) {

// init service

**case** Cons.OC\_SERVICE\_INIT\_FRAMEWORK:

ret = session.init();

**break**;

// translation service

**case** Cons.OC\_SERVICE\_INIT\_TRANSLATE:

stats = session.collectStats();

**if** (stats > 0) {

log.warning(stats + " files failed in collecting statistics.");

ret = false;

} **else**

ret = true;

**break**;

// static analysis service

**case** Cons.OC\_SERVICE\_STATIC\_ANALYSIS:

ret = session.staticReview();

**break**;

// score service

**case** Cons.OC\_SERVICE\_SCORE:

**break**;

**default**:

**return** log.error("unknown service: " + service);

}

**return** ret;

}

Internally, Session instance will ask each SessionInfo objects to execute services. Let us consider the Session class method that execute the static analysis service.

/\*\*

\* Starts a static analysis over the files being reviewed

\*

\* @return <i>true</i> if static analysis can be performed or <i>false</i>

\* if one or more files fail being analyzed.

\*/

**public** **boolean** staticReview() {

**boolean** ret = true;

**if** (!active)

**return** log.error("can't perform a static analysis over an inactive session.");

**for** (**int** i = 0; i < sessions.length; i++) {

**if** (! sessions[i].staticReview())

ret = false;

}

**return** ret;

}

Where sessions variable is declared as:

**private** SessionInfo[] sessions;

As you may see Session object delegates service accomplishment to SessionInfo ones collecting the final results. In facts, SessionInfo objects are the ones talking with Orizon internals performing the real work.

The following method is stolen from org.owasp.orizon.SessionInfo class.

/\*\*

\* Perform a static analysis over the given file

\*

\* A full static analysis is a mix from:

\*

\* \* local analysis (control flow)

\* \* global analysis (call graph)

\* \* taint propagation

\* \* statistics

\*

\*

\* @return <i>true</i> if the file being reviewed doesn't violate any

\* security check, <i>false</i> otherwise.

\*/

**public** **boolean** staticReview() {

**boolean** ret = false;

s = new Source(getStatFileName());

ret = s.analyzeStats();

...

**return** ret;

}

## Translation factory

One of the Owasp Orizon goals is to be independent from the source language being analyzed. This means that Owasp Orizon will support:

* Java
* C, C++
* C#
* perl
* ...

Such support is granted using an intermediate file format to describe the source code and used to apply the security checks. Such format is XML language.

A source code, before static analysis is started, is translated into XML. Starting from version 1.0, each source code is translated in 6 XML files containing:

* statistical information
* variables tracking information
* program control flow (local analysis)
* call graph (global analysis)
* design information
* used keywords (for crawling)

As the time this document is written (Owasp Orizon v1.0pre1, September 2008) only Java programming language is supported, however other programming language will follow soon.

Translation phase is requested from org.owasp.orizon.SessionInfo.inspect() method. Depending from the source file language, the appropriate Translator is called and the scan() method is called.

/\*\*

\* Inspects the source code, building AST trees

\* @return

\*/

**public boolean** inspect() {

**boolean** ret = false;

**switch** (language) {

**case** Cons.O\_JAVA:

t = new JavaTranslator();

**if** (!t.scan(getInFileName()))

**return** log.error("can't scan " + getInFileName() + ".");

ret = true;

**break;**

**default**:

log.error("can't inspect language: " + Cons.name(language));

**break**;

}

**return** ret;

}

Scan method is an abstract method defined in org.owasp.orizon.translator.DefaultTranslator class and declared as the following:

**public** **abstract** **boolean** scan(String in);

Every class implementing DefaultTranslator must implement how to scan the source file and build ASTs in this method.

Aside from scan() method, there are four abstract method needful to create XML input files.

**public abstract boolean** statService(String in, String out);

**public abstract boolean** callGraphService(String in, String out);

**public abstract boolean** dataFlowService(String in, String out);

**public abstract boolean** controlFlowService(String in, String out);

**public abstract boolean** designService(String in, String out);

**public abstract boolean** keywordService(String in, String out);

All these methods are called in the translator() method, the one implemented directly from DefaultTranslator class.

**public final boolean** translate(String in, String out, int service) {

**if** (!isGoodService(service))

**return** false;

**if** (!scanned)

**if** (!scan(in))

**return** log.error(in+ ": scan has been failed");

**switch** (service) {

**case** Cons.OC\_TRANSLATOR\_STAT:

**return** statService(in, out);

**case** Cons.OC\_TRANSLATOR\_CF:

**return** controlFlowService(in, out);

**case** Cons.OC\_TRANSLATOR\_CG:

**return** callGraphService(in, out);

**case** Cons.OC\_TRANSLATOR\_DF:

**return** dataFlowService(in, out);

**case** Cons.OC\_TRANSLATOR\_DESIGN:

**return** designService(in, out);

**case** Cons.OC\_TRANSLATOR\_KEYWORD:

**return** keywordService(in, out);

**default**:

**return** log.error("unknown service code");

}

}

So, when from a language specific translator is prompted for translate() method, this recalls the language specific service methods.

Every translator contains as private field a language specific scanner containing ASTs to be used in input file generation.

Consider org.owasp.orizon.translator.java.JavaTranslator class, it is declared as follows:

''' public class JavaTranslator extends DefaultTranslator {

''' static SourcePositions positions;

''' private JavaScanner j;

''' ...

JavaScanner is a class from org.owasp.orizon.translator.java package and it uses Sun JDK 6 Compiler API to scan a java file creating in memory ASTs.

Trees are created in scan() method, implemented for Java source language as follow:

''' public final boolean scan(String in) {

''' boolean ret = false;

''' String[] parms = { in };

''' Trees trees;

'''

''' JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

''' if (compiler == null)

''' return log.error("I can't find a suitable JAVA compiler. Is a JDK installed?");

'''

''' DiagnosticCollector<JavaFileObject> diagnostics = new DiagnosticCollector<JavaFileObject>();

''' StandardJavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);

''' Iterable<? extends JavaFileObject> fileObjects = fileManager.getJavaFileObjects(parms);

'''

''' JavacTask task = (com.sun.source.util.JavacTask) compiler.getTask(null,fileManager, diagnostics, null, null, fileObjects);

'''

''' try {

''' trees = Trees.instance(task);

''' positions = trees.getSourcePositions();

''' Iterable<? extends CompilationUnitTree> asts = task.parse();

''' for (CompilationUnitTree ast : asts) {

''' j = new JavaScanner(positions, ast);

''' j.scan(ast, null);

''' }

''' scanned = true;

''' return true;

''' } catch (IOException e) {

''' return log.fatal("an exception occured while translate " + in + ": " +e.getLocalizedMessage());

''' }

''' }

==== Statistical gathering ====

To implement statistic information gathering, DefaultTranslator abstract method statService() must be implemented. In the following example, the method is the JavaTranslator's one.

Statistics information are stored in JavaScanner object itself and retrieved by getStats() method.

''' public final boolean statService(String in, String out) {

''' boolean ret = false;

'''

''' if (!scanned)

''' return log.error(in + ": call scan() before asking translation...");

''' log.debug(". Entering statService(): collecting stats for: " + in);

''' try {

''' createXmlFile(out);

''' xmlInit();

''' xml("<source name=\"" + in+"\" >");

''' xml(j.getStats());

''' xml("</source>");

''' xmlEnd();

'''

''' } catch (FileNotFoundException e) {

''' } catch (UnsupportedEncodingException e) {

''' } catch (IOException e) {

''' ret = log.error("an exception occured: " + e.getMessage());

''' }

''' ret = true;

''' log.debug("stats written into: " + out);

''' log.debug(". Leaving statService()");

''' return ret;

''' }

## Static analysis engine

## Report engine

## Session management

## Plugin engine

## Rule engine

## Crawler engine

# The Orizon handbook for regular users

# Using Orizon: the Milk

1. Software Development LifeCycle [↑](#footnote-ref-2)
2. By “secure way” we means that an application is hardened against attack attempts to its data and it is designed to not allow its behaviour to be subverted by an attacker. [↑](#footnote-ref-3)