Secure Programming (06-20010)
Chapter 8: Code Review

Christophe Petit

University of Birmingham

Lectures Content (tentative)

1. Introduction
2. General principles
3. Code injection (SQL, XSS, Command)
4. HTTP sessions
5. Unix Access Control Mechanisms
6. Race conditions
7. Integer and buffer overflows
8. Code review

Why code reviews ?

Over the last 10 years, the team involved with the OWASP Code Review Project have performed thousands of application reviews, and found that every single application has had serious vulnerabilities. If you haven't reviewed your code for security holes the likelihood that your application has problems is virtually 100%.

OWASP Code Review Guide V1.1

Why code reviews ?

- May look expensive and a waste of time
- But can actually save you both money and time
  - Better protection on your assets
  - Less reputation damage
  - Less software updates needed
- Note : code may be your only advantage over attackers (so better take full advantage of it !)
- Do it yourself or not, but get it done !

Security Code Review

Process of auditing the source code for an application to verify that the proper security controls are present, that they work as intended, and that they have been invoked in all the right places.

OWASP Code Review Guide V1.1

Outline

Code Review Process

Manual Code Review

Some Automated Tools

Summary

Outline

Code Review Process

- Understand the context
- Analyze the code
  - Manually and with automated tools
  - Static and dynamic analysis
- Report your findings

Application Purpose

- Understand the application business purpose
- Identify the key business assets

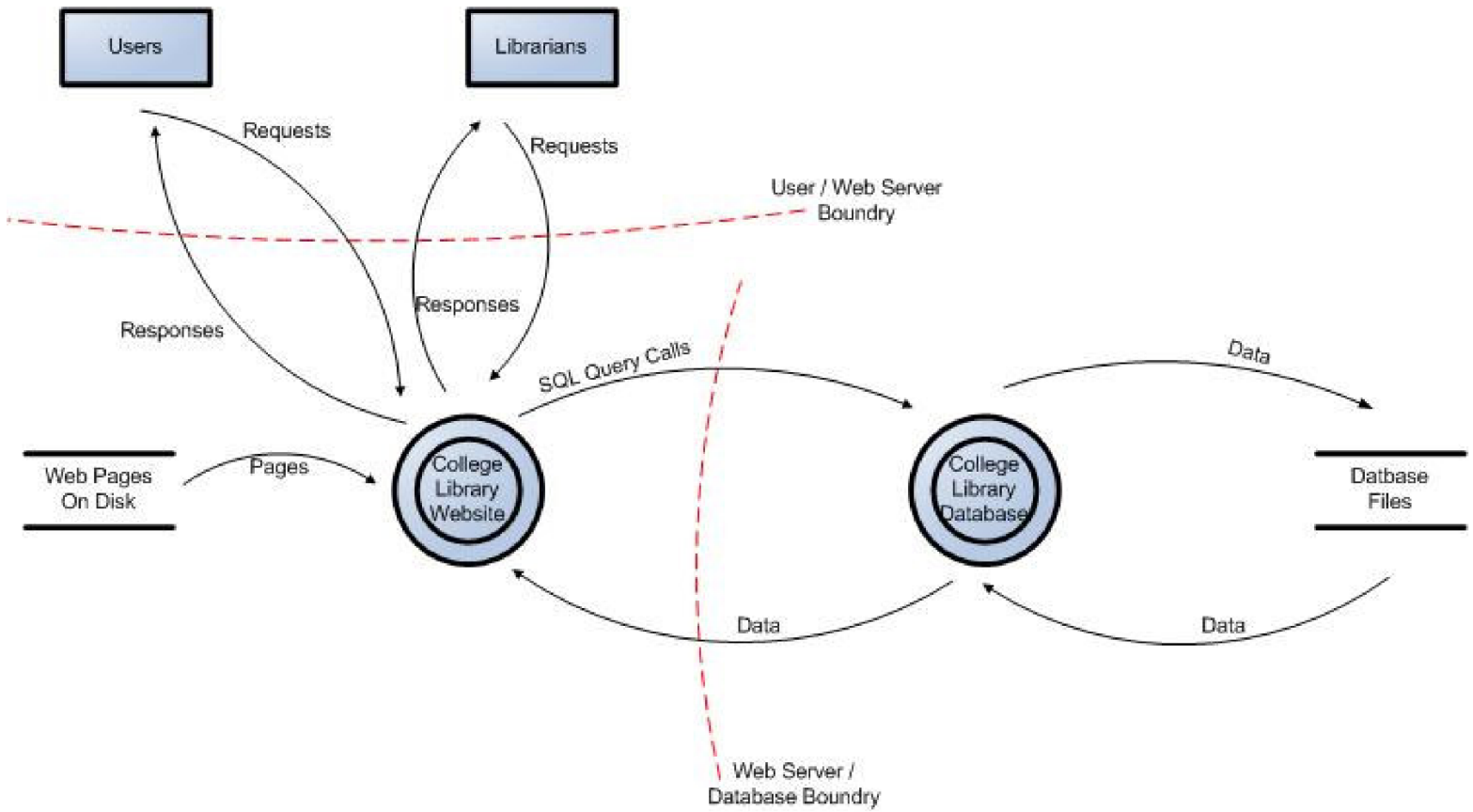
$\rightarrow$ Read business documentation

$\rightarrow$ Talk to business people

Attack surface

- Identify key entities interacting with application (internal/external users, database user/administrator, web server user/administrator, ...)
- Identify trust relationships between entities
- Identify access to assets that application should provide to each agent
- Create use cases to understand how application is used

Example : library application



Picture credit : OWASP Code Review Guide V1.1

Application Design

- Understand the application design and architecture
- Understand where the application will be located
- Identify entry points to the application
- Decompose the application in submodules
- Understand interactions and trust between submodules

$\rightarrow$ Read design documents, functional specifications, etc

$\rightarrow$ Talk to application architects and developers

Threats

- Determine and rank all threats
- Threat category lists help you identify them all
  - From an attacker's point of view
  - From a defensive point of view

Threat Categories : STRIDE

STRIDE Threat List	
Type	**Examples**
Spoofing	Threat action aimed to illegally access and use another user's credentials, such as username and password
Tampering	Threat action aimed to maliciously change/modify persistent data, such as persistent data in a database, and the alteration of data in transit between two computers over an open network, such as the Internet
Repudiation	Threat action aimed to perform illegal operation in a system that lacks the ability to trace the prohibited operations.
Information disclosure.	Threat action to read a file that they were not granted access to, or to read data in transit.
Denial of service.	Threat aimed to deny access to valid users such as by making a web server temporarily unavailable or unusable.
Elevation of privilege.	Threat aimed to gain privileged access to resources for gaining unauthorized access to information or to compromise a system.

Picture credit : OWASP Code Review Guide V1.1

Threat Categories : Application Security Frame

Category	Description
Input and Data Validation	How do you know that the input your application receives is valid and safe? Input validation refers to how your application filters, scrubs, or rejects input before additional processing. Consider constraining input through entry points and encoding output through exit points. Do you trust data from sources such as databases and file shares?
Authentication	Who are you? Authentication is the process where an entity proves the identity of another entity, typically through credentials, such as a user name and password.
Authorization	What can you do? Authorization is how your application provides access controls for resources and operations.
Configuration Management	Who does your application run as? Which databases does it connect to? How is your application administered? How are these settings secured? Configuration management refers to how your application handles these operational issues.
Sensitive Data	How does your application handle sensitive data? Sensitive data refers to how your application handles any data that must be protected either in memory, over the network, or in persistent stores.
Session Management	How does your application handle and protect user sessions? A session refers to a series of related interactions between a user and your Web application.
Cryptography	How are you keeping secrets (confidentiality)? How are you tamper-proofing your data or libraries (integrity)? How are you providing seeds for random values that must be cryptographically strong? Cryptography refers to how your application enforces confidentiality and integrity.
Exception Management	When a method call in your application fails, what does your application do? How much do you reveal? Do you return friendly error information to end users? Do you pass valuable exception information back to the caller? Does your application fail gracefully?
Auditing and Logging	Who did what and when? Auditing and logging refer to how your application records security-related events.

Picture credit : `msdn.microsoft.com/en-us/library/ff649461.aspx`

Countermeasures and Mitigations

- Identify mechanisms and countermeasures in place to address specific threats
- Identify mitigated and non mitigated threats

Review the Code

- Check that security mechanisms are in place and properly implemented
- Look for known vulnerability patterns
- Run static and dynamic analysis tools

Security Risk : DREAD model

- Damage : how big can the damage be ?
- Reproducibility : how easy to reproduce ?
- Exploitability : remotely ? without authentication ? can it be automated ?
- Affected users : what percentage ?
- Discoverability : how easy is it ?

- All ranked from 1-10 ; average value is DREAD score
- Alternative formula : Risk $=$ Likelihood $\times$ Impact

Reporting

- Prioritize your findings, important things first
  - Show severe security problems first
  - Show general architectural problems first
- A problem is severe when
  - You need to change the entire design to fix it
  - It can be directly exploited by an attacker

A good form of presenting results

- Write a brief summary of your findings
  - Outline the architectural problems
  - Recommend countermeasures
- Outline the worst problems in the code
- Then provide a list of every finding
  - When a certain finding happens at many different lines of code, aggregate them
  - Prioritize your results here

Outline

Code review : organization

- Ad hoc review
- Passaround
- Pair programming
- Walkthrough
- Team review
- Inspection

Review Priorities

- Where is the most valuable data stored ?
- Which attacks would cause the worst impact ?
- Which threat is most likely ?
- Which parts have already been reviewed ?

What to look at ?

- Security mechanisms in place related to particular threat categories
- Known vulnerability patterns related to
  - Specific languages
  - Specific applications
- Checklists are very useful !

What to look at : authentication

- Ensure all internal and external connections (user and entity) go through an appropriate and adequate form of authentication. Be assured that this control cannot be bypassed.
- Ensure all pages enforce the requirement for authentication.
- Ensure that whenever authentication credentials or any other sensitive information is passed, only accept the information via the HTTP “POST” method and will not accept it via the HTTP “GET” method.
- Any page deemed by the business or the development team as being outside the scope of authentication should be reviewed in order to assess any possibility of security breach.
- Ensure that authentication credentials do not traverse the wire in clear text form.
- Ensure development/debug backdoors are not present in production code.

Picture credit : OWASP Code Review Guide V1.1

What to look at : authorization

- Ensure that there are authorization mechanisms in place.
- Ensure that the application has clearly defined the user types and the rights of said users.
- Ensure there is a least privilege stance in operation.
- Ensure that the Authorization mechanisms work properly, fail securely, and cannot be circumvented.
- Ensure that authorization is checked on every request.
- Ensure development/debug backdoors are not present in production code.

Picture credit : OWASP Code Review Guide V1.1

What to look at : Cookie Management

- Ensure that sensitive information is not comprised.
- Ensure that unauthorized activities cannot take place via cookie manipulation.
- Ensure that proper encryption is in use.
- Ensure secure flag is set to prevent accidental transmission over “the wire” in a non-secure manner.
- Determine if all state transitions in the application code properly check for the cookies and enforce their use.
- Ensure the session data is being validated.
- Ensure cookie contains as little private information as possible.
- Ensure entire cookie should be encrypted if sensitive data is persisted in the cookie.
- Define all cookies being used by the application, their name and why they are needed.

Picture credit : OWASP Code Review Guide V1.1

What to look at : Data/Input Validation

- Ensure that a DV mechanism is present.
- Ensure all input that can (and will) be modified by a malicious user such as http headers, input fields, hidden fields, drop down lists & other web components are properly validated.
- Ensure that the proper length checks on all input exist.
- Ensure that all fields, cookies, http headers/bodies & form fields are validated.
- Ensure that the data is well formed and contains only known good chars is possible.
- Ensure that the data validation occurs on the server side.
- Examine where data validation occurs and if a centralized model or decentralized model is used.
- Ensure there are no backdoors in the data validation model.
- ***Golden Rule: All external input, no matter what it is, is examined and validated.***

Picture credit : OWASP Code Review Guide V1.1

What to look at : Error Handling / Information leakage

- Ensure that all method/function calls that return a value have proper error handling and return value checking.
- Ensure that exceptions and error conditions are properly handled.
- Ensure that no system errors can be returned to the user.
- Ensure that the application fails in a secure manner.
- Ensure resources are released if an error occurs.

Picture credit : OWASP Code Review Guide V1.1

What to look at : Logging/Auditing

- Ensure that no sensitive information is logged in the event of an error.
- Ensure the payload being logged is of a defined maximum length and that the logging mechanism enforces that length.
- Ensure no sensitive data can be logged; E.g. cookies, HTTP “GET” method, authentication credentials.
- Examine if the application will audit the actions being taken by the application on behalf of the client (particularly data manipulation/Create, Update, Delete (CUD) operations).
- Ensure successful and unsuccessful authentication is logged.
- Ensure application errors are logged.
- Examine the application for debug logging with the view to logging of sensitive data.

Picture credit : OWASP Code Review Guide V1.1

What to look at : Cryptography

- Ensure no sensitive data is transmitted in the clear, internally or externally.
- Ensure the application is implementing known good cryptographic methods.

Picture credit : OWASP Code Review Guide V1.1

What to look at : Secure Code Environment

- Examine the file structure. Are any components that should not be directly accessible available to the user?
- Examine all memory allocations/de-allocations.
- Examine the application for dynamic SQL and determine if it is vulnerable to injection.
- Examine the application for “main()” executable functions and debug harnesses/backdoors
- Search for commented out code, commented out test code, which may contain sensitive information.
- Ensure all logical decisions have a default clause.
- Ensure no development environment kit is contained on the build directories.
- Search for any calls to the underlying operating system or file open calls and examine the error possibilities.

Picture credit : OWASP Code Review Guide V1.1

What to look at : Session Management

- Examine how and when a session is created for a user, unauthenticated and authenticated.
- Examine the session ID and verify if it is complex enough to fulfill requirements regarding strength.
- Examine how sessions are stored: e.g. in a database, in memory etc.
- Examine how the application tracks sessions.
- Determine the actions the application takes if an invalid session ID occurs.
- Examine session invalidation.
- Determine how multithreaded/multi-user session management is performed.
- Determine the session HTTP inactivity timeout.
- Determine how the log-out functionality functions.

Picture credit : OWASP Code Review Guide V1.1

Other OWASP Checklists

- Buffer Overflows
- Command Injection
- SQL injection
- Cross-site scripting
- Cross-site request forgery
- Race conditions
- MySQL security
- Flash applications
- Web services
- ...

Data flow analysis

- Idea : instead of looking at modules separately, follow the data
- Where does data enter the application ?
- Where is it stored ?
- Is every input properly sanitized ?
- Is there a proper authentication and authorization every time someone retrieves or modifies data ?
- Is data sent to client properly escaped for output format ?

Data flow analysis : Advantages

- Huge applications are split into different modules, more or less exposed to outside world (the attacker)
- You may focus more on modules with huge attack surface
- Data flow analysis helps you identify those modules

General Code Quality

- Are variables properly named ?
- Are APIs used in a safe way ?
- Is there any code that is duplicated ?
- Is everything immutable that can be made immutable ?
- Is everything covered by tests ?
- Are all classes and methods so short that they can be easily analysed ?

Outline

Automated Tools vs Manual Reviews

- Automated tools are faster than humans
- They may know more vulnerability patters than you
- They are developed by experts
- You will still be better at
  - Understanding context
  - Identifying false positives
  - Identifying new vulnerability patterns
  - Summarizing the findings

$\rightarrow$ Use automated tools but do not rely solely on them

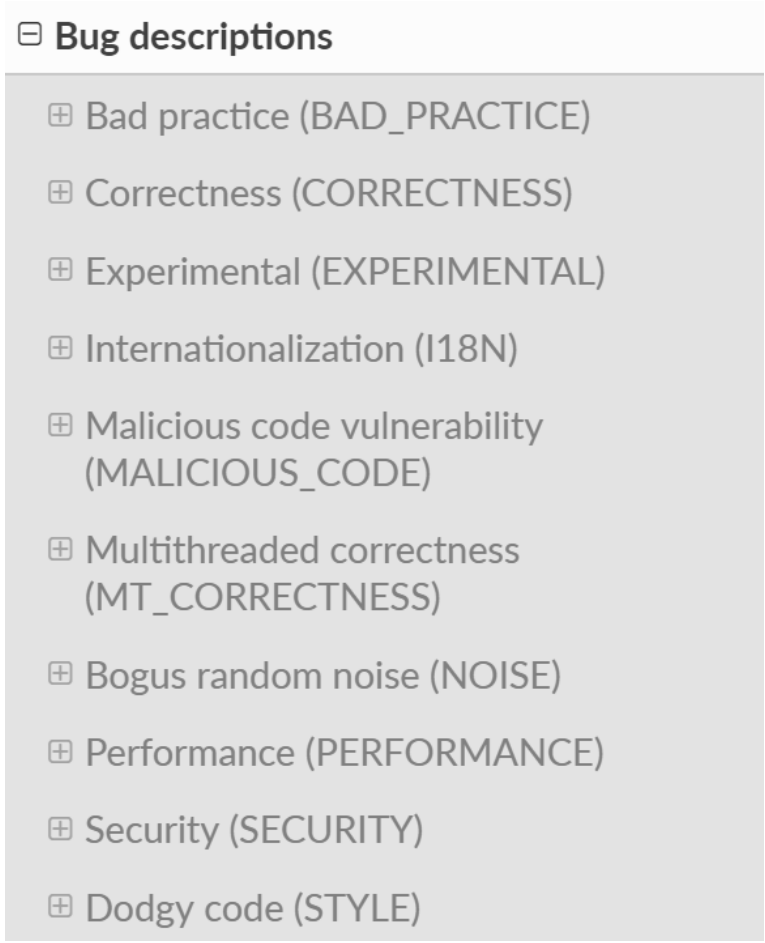
Static and Dynamic Analysis

- Static analysis : look at the code but does not run it
- Dynamic analysis : execute the code, monitor and analyze its behaviour, but does not look at it
- Dynamic analysis can uncover subtle bugs involving several modules, but only for the parameters considered

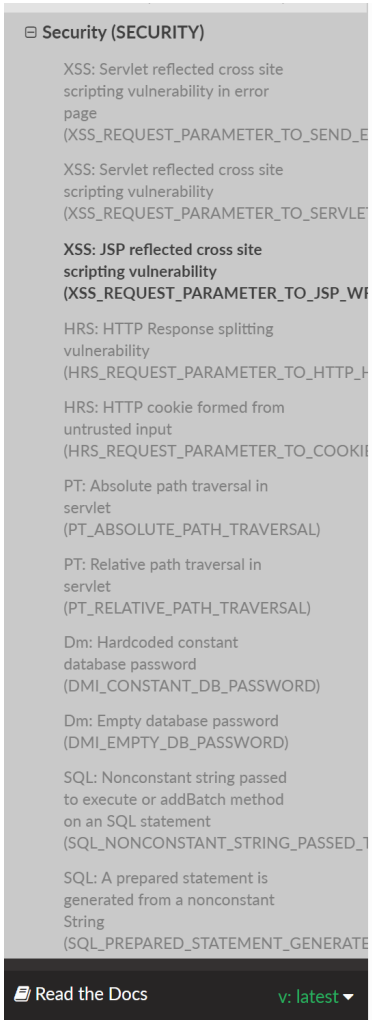
SpotBugs


SpotBugs
Find bugs in Java Programs
Check it out on GitHub
SpotBugs is a program which uses static analysis to look for bugs in Java code. It is free software, distributed under the terms of the Lesser GNU Public License.
SpotBugs is the spiritual successor of FindBugs, carrying on from the point where it left off with support of its community. Please check official manual site for details.
SpotBugs requires JRE (or JDK) 1.8.0 or later to run. However, it can analyze programs compiled for any version of Java, from 1.0 to 1.9.
/// Bug Descriptions
SpotBugs checks for more than 400 bug patterns. Bug descriptions can be found here
Descriptions are also available in Japanese
/// Using SpotBugs
SpotBugs can be used standalone and through several integrations, including:
Ant
Maven
Gradle
Eclipse
Bug Descriptions
Using SpotBugs
Extensions
License of SpotBugs Logo
Support or Contact

SpotBugs


Bug descriptions
Bad practice (BAD_PRACTICE)
Correctness (CORRECTNESS)
Experimental (EXPERIMENTAL)
Internationalization (I18N)
Malicious code vulnerability (MALICIOUS_CODE)
Multithreaded correctness (MT_CORRECTNESS)
Bogus random noise (NOISE)
Performance (PERFORMANCE)
Security (SECURITY)
Dodgy code (STYLE)

SpotBugs



XSS: JSP reflected cross site scripting vulnerability (XSS_REQUEST_PARAMETER_TO_JSP_WRITER)

This code directly writes an HTTP parameter to JSP output, which allows for a cross site scripting vulnerability. See http://en.wikipedia.org/wiki/Cross-site_scripting for more information.

SpotBugs looks only for the most blatant, obvious cases of cross site scripting. If SpotBugs found *any*, you *almost certainly* have more cross site scripting vulnerabilities that SpotBugs doesn't report. If you are concerned about cross site scripting, you should seriously consider using a commercial static analysis or pen-testing tool.

HRS: HTTP Response splitting vulnerability (HRS_REQUEST_PARAMETER_TO_HTTP_HEADER)

This code directly writes an HTTP parameter to an HTTP header, which allows for a HTTP response splitting vulnerability. See http://en.wikipedia.org/wiki/HTTP_response_splitting for more information.

SpotBugs looks only for the most blatant, obvious cases of HTTP response splitting. If SpotBugs found *any*, you *almost certainly* have more vulnerabilities that SpotBugs doesn't report. If you are concerned about HTTP response splitting, you should seriously consider using a commercial static analysis or pen-testing tool.

HRS: HTTP cookie formed from untrusted input (HRS_REQUEST_PARAMETER_TO_COOKIE)

This code constructs an HTTP Cookie using an untrusted HTTP parameter. If this cookie is added to an HTTP response, it will allow a HTTP response splitting vulnerability. See http://en.wikipedia.org/wiki/HTTP_response_splitting for more information.

SpotBugs looks only for the most blatant, obvious cases of HTTP response splitting. If SpotBugs found *any*, you *almost certainly* have more vulnerabilities that SpotBugs doesn't report. If you are concerned about HTTP response splitting, you should seriously consider using a commercial static analysis or pen-testing tool.

PMD

PMD Source Code Analyzer | Documentation | Bugs | About | News | Downloads | Support | Plugins

PMD

An extensible cross-language static code analyzer.

GITHUB

Latest Version: 5.8.1 (1st July 2017)
Release Notes | Download | Documentation

QuickStart

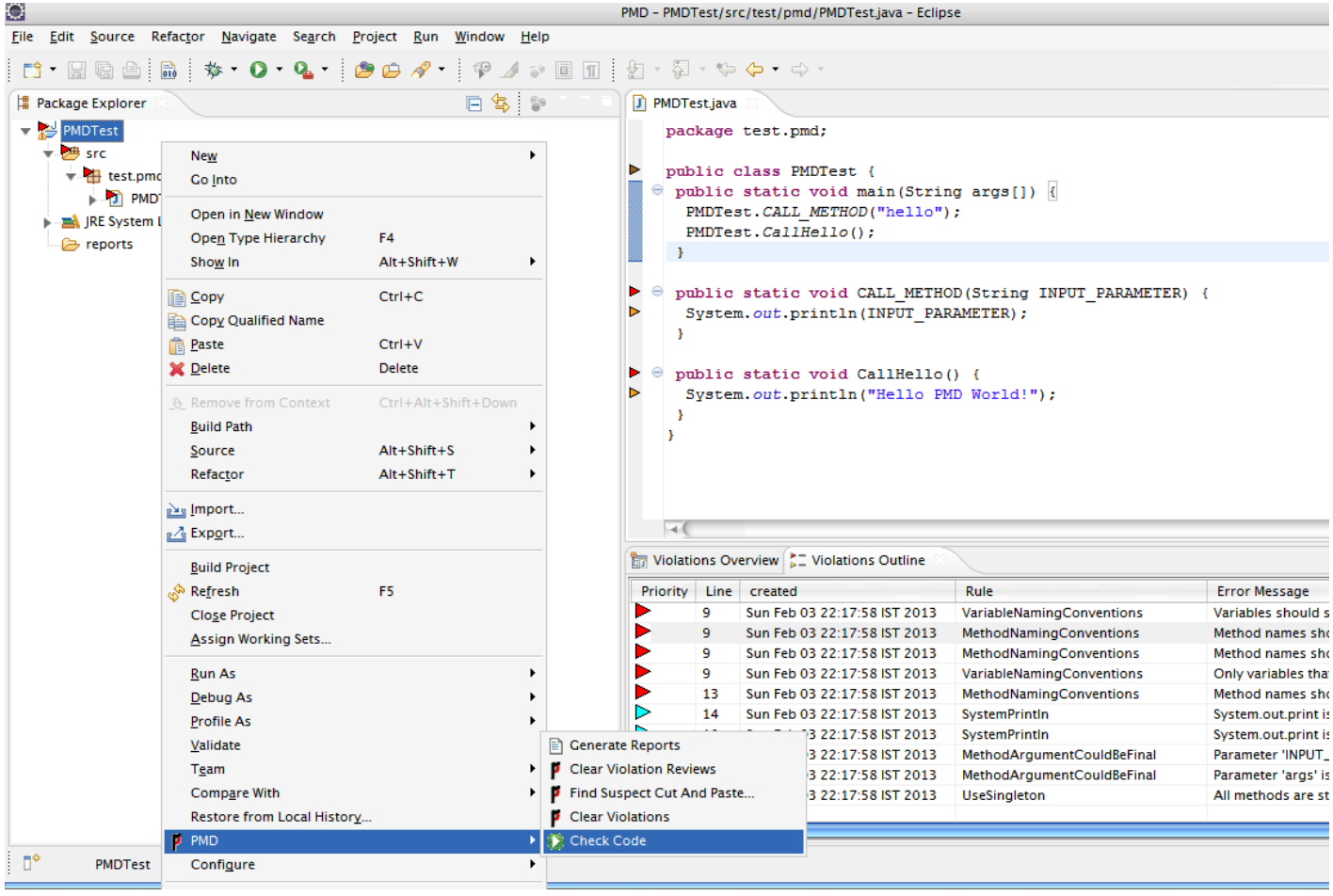
See Installation and Command Line Usage

Linux | MacOS | Windows

```
$ cd $HOME
$ wget https://github.com/pmd/pmd/releases/download/pmd_releases%2F5.8.1/pmd-bin-5.8.1.zip
$ unzip pmd-bin-5.8.1.zip
$ alias pmd="$HOME/pmd-bin-5.8.1/bin/run.sh pmd"
$ pmd -d /usr/src -R java-basic -f text
```

Checkout the existing rules.

PMD



Picture credit : `www.javatips.net/blog/pmd-in-eclipse-tutorial`

PMD : related tools

Similar to PMD

Open Source

- Checkstyle - Very detailed, supports both Maven and Ant. Uses ANTLR.
- DoctorJ - Uses JavaCC. Checks Javadoc, syntax and calculates metrics.
- ESC/Java - Finds null dereference errors, array bounds errors, type cast errors, and race conditions. Uses Java Modeling Language annotations.
- FindBugs - works on bytecode, uses BCEL. Source code uses templates, nifty stuff!
- Hammurapi - Uses ANTLR, excellent documentation, lots of rules
- Jamit - bytecode analyzer, nice graphs
- JCSC - Does a variety of coding standard checks, uses JavaCC and the GNU Regexp package.
- Jikes - More than a compiler; now it reports code warnings too
- JLint - Written in C++. Uses data flow analysis and a lock graph to do lots of synchronization checks. Operates on class files, not source code.
- JPathFinder - A verification VM written by NASA; supports a subset of the Java packages
- JWiz - Research project, checks some neat stuff, like if you create a Button without adding an ActionListener to it. Neat.

Commercial

- AppPerfect - 750 rules, produces PDF/Excel reports, supports auto-fixing problems
- Assent - The usual stuff, seems pretty complete.
- Aubjex - Rules aren't listed online. Appears to have some code modification stuff, which would be cool to have in PMD. $299.
- AzoJavaChecker - Rules aren't listed online so it's hard to tell what they have. Not sure how much it costs since I don't know German.
- CodePro AnalytiX - Eclipse plug-in, extensive audit rules, JUnit test generation/editing, code coverage and analysis
- Enerjy Java Code Analyser - 200 rules, lots of IDE plugins
- Flaw Detector - In beta, does control/data flow analysis to detect NullPointerExceptions
- JStyle - $995, nice folks, lots of metrics and rules
- JTest - Very nice with tons of features, but also very expensive and requires a running X server (or Xvfb) to run on Linux. They charge $500 to move a license from one machine to another.
- Lint4J - Lock graph, DFA, and type analysis, many EJB checks
- SolidSDD - Code duplication detection, nice graphical reporting. Free licensing available for Educational or OSS use.

Similar to CPD

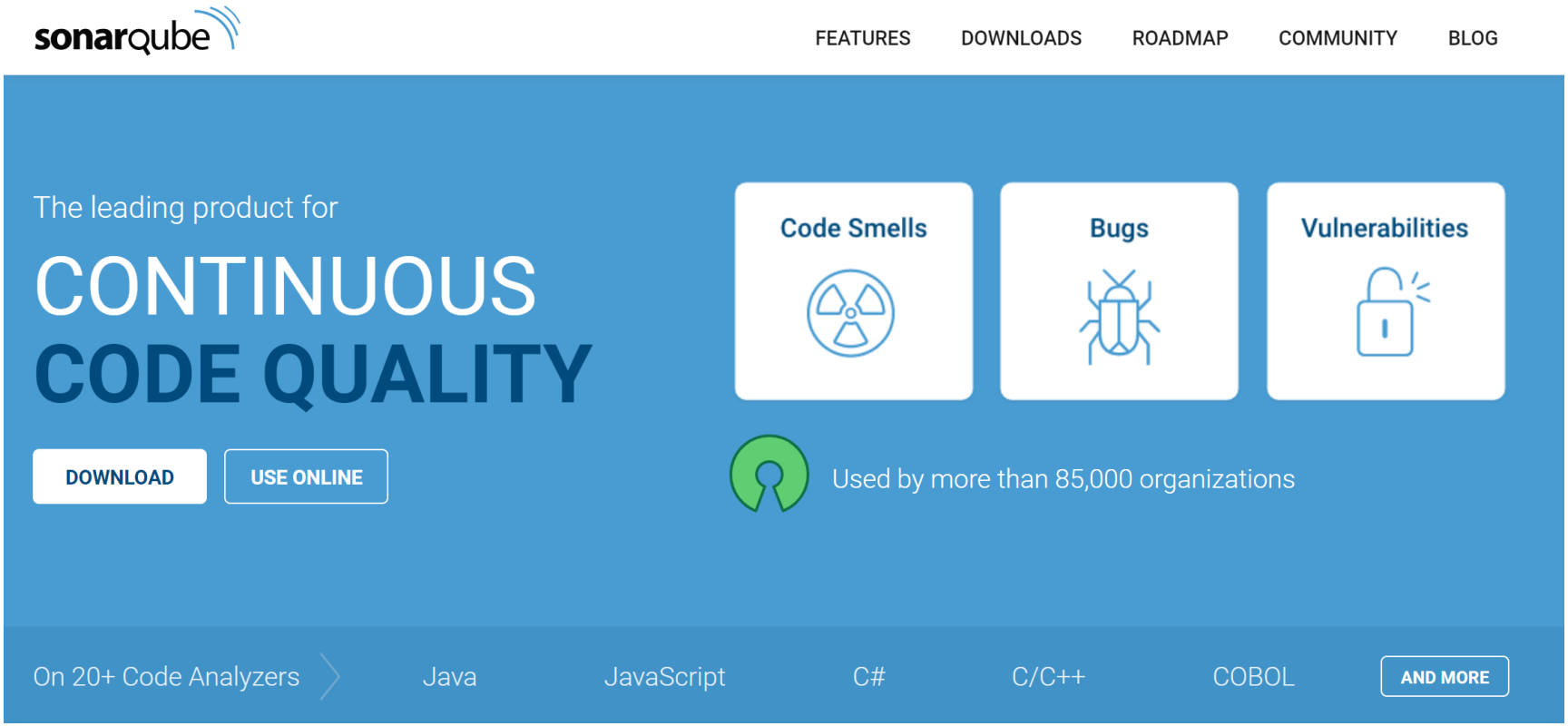
Commercial

- Simian - fast, works with Java, C#, C, CPP, COBOL, JSP, HTML
- Simscan - free for open source projects

High level reporting

- XRadar - Agregates data from a lot of code quality tool to generate a full quality dashboard.
- Sonar - Pretty much like XRadar, but younger project, fully integrated to maven 2 (but requires a database)
- Maven Dashboard - Same kind of agregator but only for maven project.
- QALab - Yet another maven plugin…

SonarQube


sonarqube
FEATURES
DOWNLOADS
ROADMAP
COMMUNITY
BLOG
The leading product for
CONTINUOUS
CODE QUALITY
DOWNLOAD
USE ONLINE
Code Smells
Bugs
Vulnerabilities
Used by more than 85,000 organizations
On 20+ Code Analyzers
Java
JavaScript
C#
C/C++
COBOL
AND MORE

SonarQube

sonarsource

WHY US | PRODUCTS | PLANS AND PRICING | CUSTOMERS | RESOURCES | COMPANY | BLOG

Products > Code Analyzers > SonarJS

Type

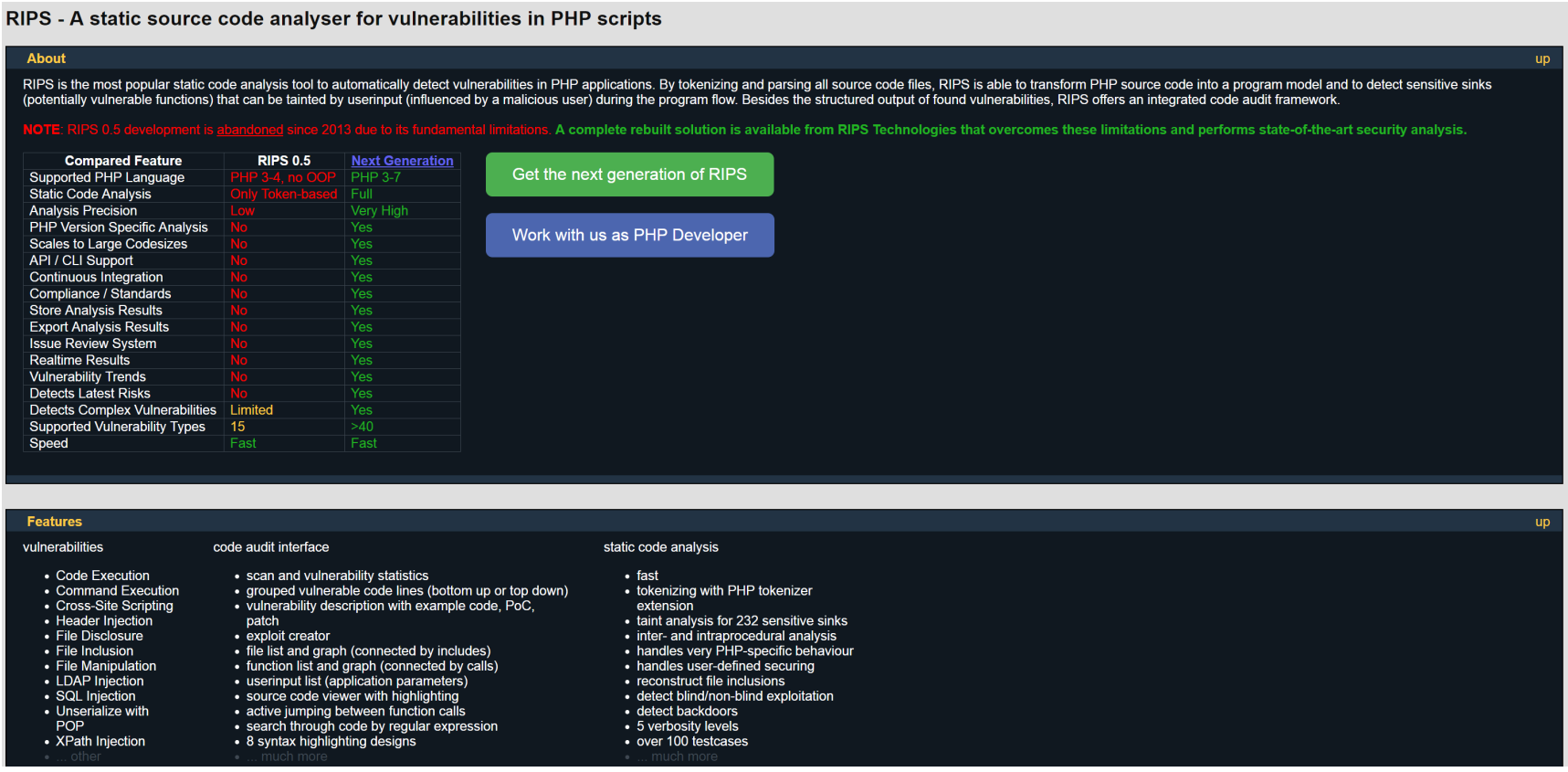
Bugs | Code Smells | Vulnerabilities

Standard

CWE | SANS_TOP_25 | OWASP | MISRA | CERT

- Code should not be dynamically injected and executed
- Cross-document messaging domains should be carefully restricted
- Function constructors should not be used
- "alert(...)" should not be used
- Debugger statements should not be used
- Web SQL databases should not be used
- Local storage should not be used
- Untrusted content should not be included

PHP Applications : RIPS



RIPS - A static source code analyser for vulnerabilities in PHP scripts

About up

RIPS is the most popular static code analysis tool to automatically detect vulnerabilities in PHP applications. By tokenizing and parsing all source code files, RIPS is able to transform PHP source code into a program model and to detect sensitive sinks (potentially vulnerable functions) that can be tainted by userinput (influenced by a malicious user) during the program flow. Besides the structured output of found vulnerabilities, RIPS offers an integrated code audit framework.

NOTE: RIPS 0.5 development is abandoned since 2013 due to its fundamental limitations. **A complete rebuilt solution is available from RIPS Technologies that overcomes these limitations and performs state-of-the-art security analysis.**

Compared Feature	RIPS 0.5	Next Generation
Supported PHP Language	PHP 3-4, no OOP	PHP 3-7
Static Code Analysis	Only Token-based	Full
Analysis Precision	Low	Very High
PHP Version Specific Analysis	No	Yes
Scales to Large Codesizes	No	Yes
API / CLI Support	No	Yes
Continuous Integration	No	Yes
Compliance / Standards	No	Yes
Store Analysis Results	No	Yes
Export Analysis Results	No	Yes
Issue Review System	No	Yes
Realtime Results	No	Yes
Vulnerability Trends	No	Yes
Detects Latest Risks	No	Yes
Detects Complex Vulnerabilities	Limited	Yes
Supported Vulnerability Types	15	>40
Speed	Fast	Fast

Get the next generation of RIPS

Work with us as PHP Developer

Features up

vulnerabilities

- Code Execution
- Command Execution
- Cross-Site Scripting
- Header Injection
- File Disclosure
- File Inclusion
- File Manipulation
- LDAP Injection
- SQL Injection
- Unserialize with POP
- XPath Injection
- ... other

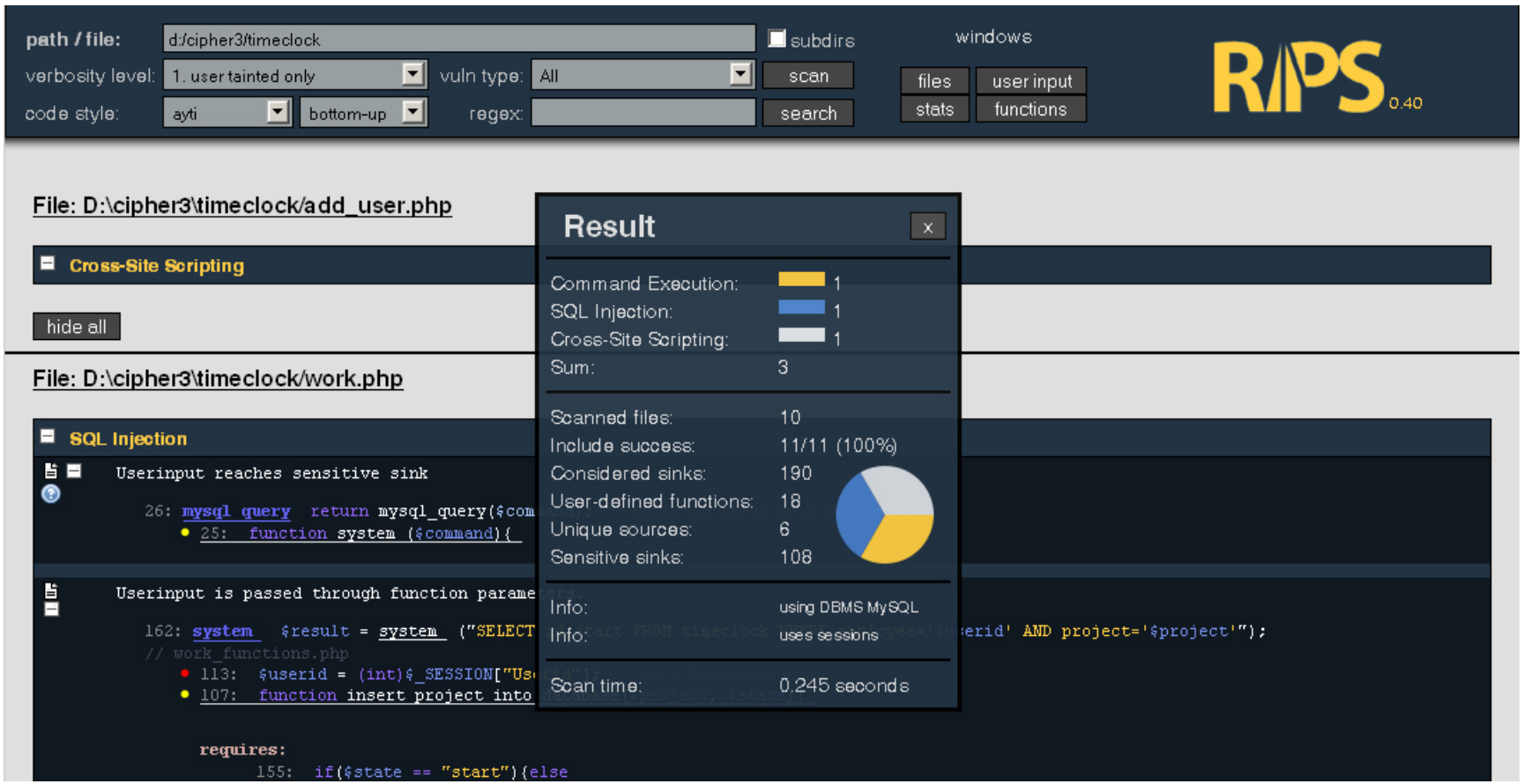
code audit interface

- scan and vulnerability statistics
- grouped vulnerable code lines (bottom up or top down)
- vulnerability description with example code, PoC, patch
- exploit creator
- file list and graph (connected by includes)
- function list and graph (connected by calls)
- userinput list (application parameters)
- source code viewer with highlighting
- active jumping between function calls
- search through code by regular expression
- 8 syntax highlighting designs
- ... much more

static code analysis

- fast
- tokenizing with PHP tokenizer extension
- taint analysis for 232 sensitive sinks
- inter- and intraprocedural analysis
- handles very PHP-specific behaviour
- handles user-defined securing
- reconstruct file inclusions
- detect blind/non-blind exploitation
- detect backdoors
- 5 verbosity levels
- over 100 testcases
- ... much more

PHP Applications : RIPS



And also

About CBMC

CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. It also supports SystemC using Scoot. We have recently added experimental support for Java Bytecode.

CBMC verifies array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

While CBMC is aimed for embedded software, it also supports dynamic memory allocation using `malloc` and `new`. For questions about CBMC, contact Daniel Kroening.

CBMC is available for most flavours of Linux (pre-packaged on Debian, Ubuntu and Fedora), Solaris 11, Windows and MacOS X. You should also read the CBMC license.

CBMC comes with a built-in solver for bit-vector formulas that is based on MiniSat. As an alternative, CBMC has featured support for external SMT solvers since version 3.3. The solvers we recommend are (in no particular order) Boolector, MathSAT, Yices 2 and Z3. Note that these solvers need to be installed separately and have different licensing conditions.

Information Source Code **Documentation** Contact How to Help Gallery

| Table of Contents | Quick Start | FAQ | User Manual | Download Manual | Research Papers | Books |

<< Home Page

Valgrind User Manual

7. Helgrind: a thread error detector

Table of Contents

To use this tool, you must specify `--tool=helgrind` on the Valgrind command line.

7.1. Overview

Helgrind is a Valgrind tool for detecting synchronisation errors in C, C++ and Fortran programs that use the POSIX pthreads threading primitives.

The main abstractions in POSIX pthreads are: a set of threads sharing a common address space, thread creation, thread joining, thread exit, mutexes (locks), condition variables (inter-thread event notifications), reader-writer locks, spinlocks, semaphores and barriers.

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

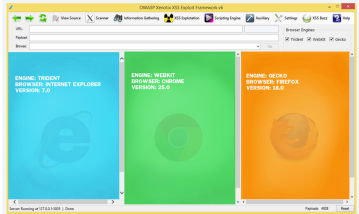
```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.0.5.63#dev}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent i
s illegal. It is the end user's responsibility to obey all applicable local, state and fed
eral laws. Developers assume no liability and are not responsible for any misuse or damage
 caused by this program

[*] starting at 17:43:06

[17:43:06] [INFO] testing connection to the target URL
[17:43:06] [INFO] heuristics detected web page charset 'ascii'
[17:43:06] [INFO] testing if the target URL is stable
[17:43:07] [INFO] target URL is stable
[17:43:07] [INFO] testing if GET parameter 'id' is dynamic
[17:43:07] [INFO] confirming that GET parameter 'id' is dynamic
[17:43:07] [INFO] GET parameter 'id' is dynamic
[17:43:07] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
 (possible DBMS: 'MySQL')
```

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

The Valgrind distribution currently includes six production-quality tools: a memory error detector, two thread error detectors, a cache and branch-prediction profiler, a call-graph generating cache and branch-prediction profiler, and a heap profiler. It also includes three experimental tools: a stack/global array overrun detector, a second heap profiler that examines how heap blocks are used, and a SimPoint basic block vector generator. It runs on the following platforms: X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, MIPS32/Linux, MIPS64/Linux, X86/Solaris, AMD64/Solaris, ARM/Android (2.3.x and later), ARM64/Android, X86/Android (4.0 and later), MIPS32/Android, X86/Darwin and AMD64/Darwin (Mac OS X 10.12).

Valgrind is Open Source / Free Software, and is freely available under the GNU General Public License, version 2.

OWASP Xenotix XSS Exploit Framework



OWASP Xenotix XSS Exploit Framework is an advanced Cross Site Scripting (XSS) vulnerability detection and exploitation framework. Xenotix provides Low False Positive XSS Detection by performing the Scan within the browser engines where in real world, payloads get reflected. Xenotix Scanner Module is incorporated with 3 intelligent fuzzers to reduce the scan time and produce better results. If you really don't like the tool logic, then leverage the power of Xenotix API to make the tool work like you wanted it to be. It is claimed to have the world's 2nd largest XSS Payloads of about 4800+ distinctive XSS Payloads. It is incorporated with a feature rich Information Gathering module for target Reconnaissance. The Exploit Framework includes real world offensive XSS exploitation modules for Penetration Testing and Proof of Concept creation. Say no to alert pop-ups in PoC. Pen testers can now create appealing Proof of Concepts within few clicks.

And also (OWASP)

- Bandit - bandit is a comprehensive source vulnerability scanner for Python
- Brakeman - Brakeman is an open source vulnerability scanner specifically designed for Ruby on Rails applications
- Codesake Dawn - Codesake Dawn is an open source security source code analyzer designed for Sinatra, Padrino for Ruby on Rails applications. It also works on non-web applications written in Ruby
- FindBugs - Find Bugs (including a few security flaws) in Java programs
- FindSecBugs - A security specific plugin for FingBugs that significantly improves FindBug's ability to find security vulnerabilities in Java programs
- Flawfinder Flawfinder - Scans C and C++
- Google CodeSearchDiggity - Uses Google Code Search to identifies vulnerabilities in open source code projects hosted by Google Code, MS CodePlex, SourceForge, Github, and more. The tool comes with over 130 default searches that identify SQL injection, cross-site scripting (XSS), insecure remote and local file includes, hard-coded passwords, and much more. *Essentially, Google CodeSearchDiggity provides a source code security analysis of nearly every single open source code project in existence – simultaneously.*
- PMD - PMD scans Java source code and looks for potential code problems (this is a code quality tool that does not focus on security issues)
- PreFast (Microsoft) - PREfast is a static analysis tool that identifies defects in C/C++ programs. Last update 2006.
- Puma Scan - Puma Scan is a .NET C# open source static source code analyzer that runs as an IDE plugin for Visual Studio and via MSBuild in CI pipelines.
- .NET Security Guard - Roslyn analyzers that aim to help security audits on .NET applications. It will find SQL injections, LDAP injections, XXE, cryptography weakness, XSS and more.
- RIPS - RIPS is a static source code analyzer for vulnerabilities in PHP web applications. Please see notes on the sourceforge.net site.
- phpcs-security-audit - phpcs-security-audit is a set of PHP_CodeSniffer rules that finds flaws or weaknesses related to security in PHP and its popular CMS or frameworks. It currently has core PHP rules as well as Drupal 7 specific rules.
- SonarQube - Scans source code for more than 20 languages for Bugs, Vulnerabilities, and Code Smells. SonarQube IDE plugins for Eclipse, Visual Studio, and IntelliJ provided by SonarLint.
- VisualCodeGrepper (VCG) - Scans C/C++, C#, VB, PHP, Java, and PL/SQL for security issues and for comments which may indicate defective code. The config files can be used to carry out additional checks for banned functions or functions which commonly cause security issues.
- Xanitizer - Scans Java for security vulnerabilities, mainly via taint analysis. The tool comes with a number of predefined vulnerability detectors which can additionally be extended by the user.

and many more !

Outline

Summary

- Code review will save you time and money
- First step is to understand the context
- Checklists are very useful
- Automated tools very useful (but humans still needed)
- Make a useful report

References and Acknowledgements

- OWASP Code Review Guide
- While preparing these slides I also used teaching material developed by Erik Tew at the University of Birmingham (kindly provided to me). Some of my slides are heavily inspired from his (but blame me for any errors !)