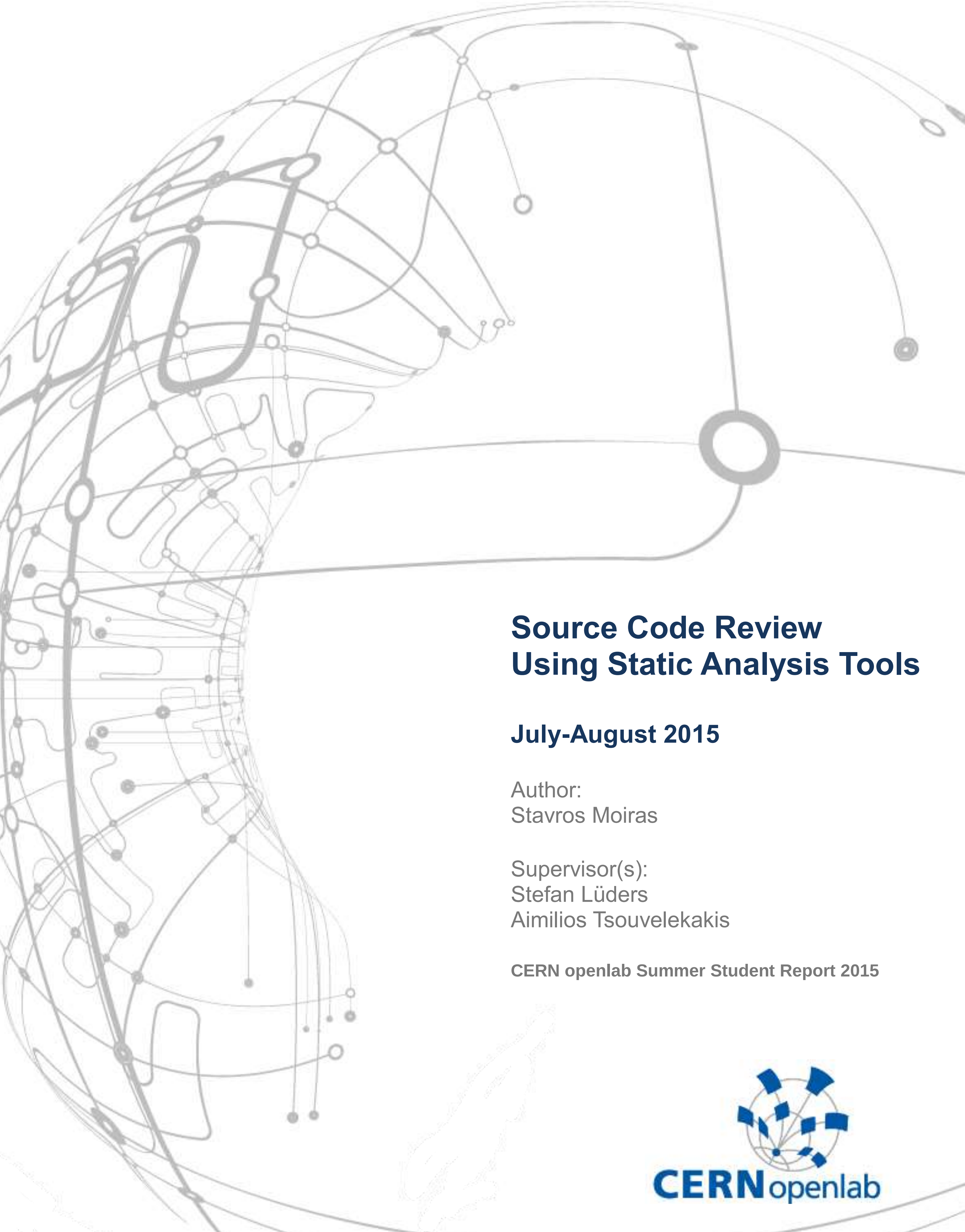


# Source Code Review Using Static Analysis Tools

## July-August 2015

Author:
Stavros Moiras

Supervisor(s):
Stefan Lüders
Aimilios Tsouvelekakis

CERN openlab Summer Student Report 2015

# Abstract

Many teams at CERN, develop their own software to solve their tasks. This software may be public or it may be used for internal purposes. It is of major importance for developers to know that their software is secure. Humans are able to detect bugs and vulnerabilities but it is impossible to discover everything when they need to read hundreds' lines of code. As a result, computer scientists have developed tools which complete efficiently and within minutes the task of analysing source code and finding critical bugs and vulnerabilities. These tools are called static analysis and they are able to find, analyse and suggest solutions to the programmer in the early stages of development.

The goal of this project is to evaluate and compare as many static analysis tools as possible (both freeware and commercial) according to metrics decided by CERN Security Team. The final result should not only be a selection of tools per language that software developers should utilise but also an automated way to use them and get useful reports that will help developers write better software.

# Table of Contents

# 1 Introduction

A bug is a programming error that sometimes can be exploited by an attacker to subvert the functionality of the vulnerable software by feeding it malformed inputs such as network packets or web form data that evade the program's error checks allowing the attacker to execute arbitrary code on the host. In order to exploit a vulnerability, an attacker must have an opportunity to execute the vulnerable code, for instance by sending a message to a service listening on a network port. Such an opportunity is known as an attack vector.

Vulnerabilities could range from buffer overflows, calls to vulnerable library functions to unguarded access to the root privilege (“root privilege escalation”). These may lead to a lot of consequences which could be exploited by an attacker to gain access to the vulnerable system. Fortunately, there are a number of tools to help the programmer check for these errors. While it is impossible to be completely secure, it's possible to minimize these errors.

# 2 Static Analysis Tools

Static analysis tools are designed to analyse a given source code in order to find programming defects. In an ideal world, such tools would automatically find programming defects with high confidence. But this is not the case for many types of programming defects due to the high false positive rate that is reported. As a result, such tools serve as a help for an analyst to detect flaws more efficiently instead of a tool that just automatically finds defects.

The tools that have been tested and evaluated at CERN are listed below:

- Codepro Analytix
- Cppcheck
- Findbugs
- Flawfinder
- Perl-Critic
- PHPca
- PMD
- Pyflakes
- Pylint
- RATS (Rough Auditing Tool for Security)
- RIPS
- SonarQube
- VCG (Visual Code Grepper)
- Commercial Vendor 1
- Commercial Vendor 2

# 3 Advantages and Disadvantages

## 3.1 Advantages

- They are very scalable and can be run repeatedly
- The output is very informative with line highlights
- Automatic scanning of bugs

## 3.2 Disadvantages

- They have a high false positive rate
- They cannot detect configuration issues
- In some cases code compilation is required

# 4 Metrics

Some metrics and results are presented below. Results derived from the tools that we evaluated and they are categorized per programming language (see detailed installation instructions in the appendix at the end of this report). A notable difference made a tool named VCG (Visual Code Grepper) which was fully customizable, the user had the ability to add new patterns for vulnerabilities to be detected. Also, it provides quick access to the file that is affected highlighting the exact line with a single click, this drastically increases the process of a manual review.

On the other hand the tool from the Commercial Vendor 2 was also very customizable, had a reasonable balance between false and true positives and most of its findings were indeed something that required attention and manual review.

Below we have a table with some details of the samples that was tested.

| *Language* | *Files* | *Blank* | *Comment* | *Lines of Code* |
|---|---|---|---|---|
| *C++* | 18661 | 1220503 | 1585615 | 6935350 |
| *C/C++ Header* | 26775 | 710417 | 1086601 | 3061157 |
| *Python* | 9296 | 338607 | 451010 | 1476867 |
| *C* | 1305 | 121910 | 124202 | 606878 |
| *Java* | 970 | 24867 | 36896 | 89181 |
| *PHP* | 854 | 16389 | 48403 | 144309 |
| *Perl* | 275 | 239302 | 176860 | 190896 |

Below there is a table explaining the values that are used in the following metrics tables. The false positives were calculated per file in most cases.

| VALUES | EXPLANATION |
|---|---|
| LOW | Less than 20 false positives |
| MEDIUM | Approximately 20-40 false positives |
| HIGH | More than 40 false positives |
| YES | The application supports this kind of vulnerability / report |
| NO | The application does not support this kind of vulnerability / report |

## 4.1 C / C++

| Application | False Positives | True Positives | Buffer Overflows | Memory Leak | Uninitialized Pointer / Variable |
|---|---|---|---|---|---|
| Cppcheck | Medium | High | No | No | Yes |
| Flawfinder | High | Medium | Yes | No | No |
| RATS | High | Medium | Yes | No | No |
| VCG | Medium | Medium | Yes | Yes | No |
| Commercial Vendor 1 | Medium | High | Yes | Yes | Yes |
| Commercial Vendor 2 | Medium | High | No | Yes | Yes |
| Commercial Vendor 3 | Low | High | No | Yes | Yes |

## 4.2 Java

| Application | False Positives | True Positives | Document empty method | Internal array exposure | XSS | SQL injections |
|---|---|---|---|---|---|---|
| **Codepro Analytix** | Medium | Low | Yes | Yes | No | No |
| **Findbugs** | Low | Medium | Yes | Yes | No | Yes |
| **PMD** | Low | Medium | Yes | Yes | No | No |
| **SonarQube** | Low | High | Yes | Yes | No | No |
| **VCG** | Medium | Medium | No | Yes | No | Yes |
| **Commercial Vendor 1** | Medium | Medium | No | No | Yes | Yes |

## 4.3 Python

| Application | False Positives | True Positives | Code Injection | Untrusted Regex | TOCTOU Vulnerability | Bad indentation | Unused Variable |
|---|---|---|---|---|---|---|---|
| **Pyflakes** | Medium | Medium | No | No | No | No | Yes |
| **Pylint** | Medium | Low | No | No | No | Yes | Yes |
| **RATS** | High | Low | Yes | Yes | Yes | No | No |
| **SonarQube** | Low | High | No | No | No | No | No |
| **Commercial Vendor 1** | Low | High | Yes | No | No | No | No |

## 4.4 Perl

| Application | False Positives | True Positives | Insecure Random Number Generator | Untrusted User Input | Loop iterator is not lexical |
|---|---|---|---|---|---|
| **Perl-Critic** | High | Medium | No | No | Yes |
| **RATS** | Medium | Medium | Yes | Yes | No |
| **Commercial Vendor 1** | Low | High | No | Yes | No |

## 4.5 PHP

| Application | False Positives | True Positives | Cross Site Scripting | SQL Injection | File Inclusion |
|---|---|---|---|---|---|
| **PHPca** | High | Medium | No | No | No |
| **RIPS** | Medium | Medium | Yes | Yes | Yes |
| **RATS** | Medium | Low | No | No | Yes |
| **SonarQube** | Low | High | No | No | No |
| **VCG** | High | Low | Yes | Yes | Yes |
| **Commercial Vendor 1** | Medium | Medium | Yes | Yes | Yes |

# 5 Types of Reports Generated

| Application | PDF | XML | HTML | Program / Web UI | CSV | Command Line | Email |
|---|---|---|---|---|---|---|---|
| **Cppcheck** | | Yes | | | Yes | | |
| **Flawfinder** | | | Yes | | | Yes | |
| **RATS** | | Yes | Yes | | | Yes | |
| **SonarQube** | Yes | | | Yes | | | Yes |
| **VCG** | | Yes | | Yes | Yes | Yes | |
| **Codepro Analytix** | | | Yes | | | | Yes |
| **Findbugs** | | Yes | Yes | | | | |
| **PMD** | | Yes | Yes | | Yes | Yes | |
| **Pyflakes** | | | | | | Yes | |
| **Pylint** | | | | | | Yes | |
| **Perl-Critic** | | | | | | Yes | |
| **PHPca** | | | | Yes | | | |
| **RIPS** | | | | Yes | | | |
| **Commercial Vendor 1** | Yes | Yes | | Yes | Yes | | Yes |
| **Commercial Vendor 2** | | Yes | Yes | Yes | | | |

# 6 Integration with Jenkins

Jenkins is an open source continuous integration tool and is used by software developers to speed up the development process. Using the tool, a build can be initiated with various ways, for example it can be triggered by commit in a version control system like GIT. That is why it is ideal for integration with static analysis or security tools, because the tools can be set up to run every time a build is taking place and inform the developers if bugs are presented in the code.

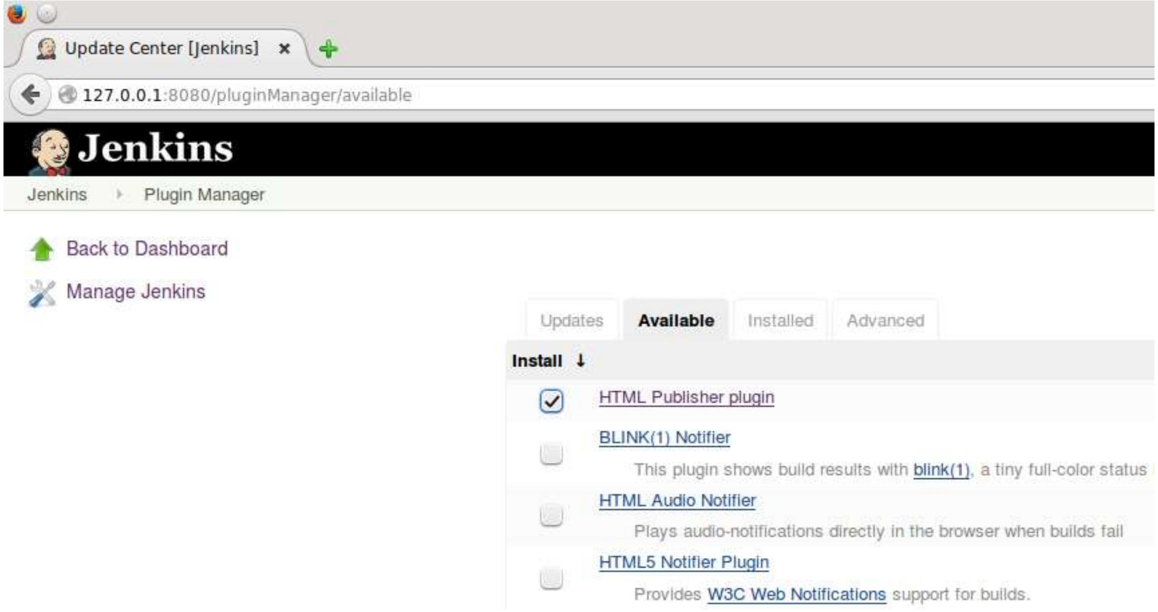
## 6.1 Instructions

In order to integrate our static analysis tools to Jenkins we have to follow the steps below:

After the installation of Jenkins we can start our browser and navigate to http://127.0.0.1:8080 where we will we find ourselves into the Jenkins platform main interface.

First of all we have to install some vital plugins.

1. Manage Jenkins → Manage Plugins → Available Tab
2. Install "Email Extention Template Plugin"
3. Install "Publish HTML Reports"
4. Optional: Install "Findbugs", "PMD", "Cppcheck" plugins
5. Restart Jenkins



☑ HTML Publisher plugin

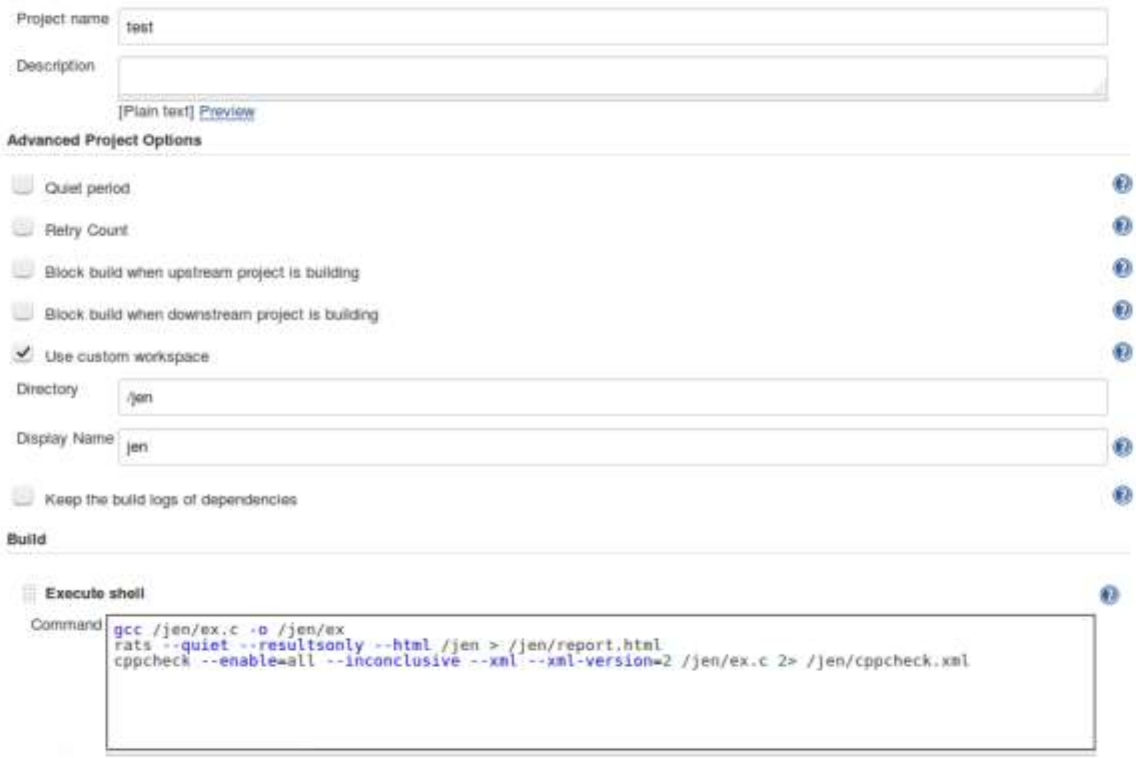
□ BLINK(1) Notifier
This plugin shows build results with blink(1), a tiny full-color status

□ HTML Audio Notifier
Plays audio-notifications directly in the browser when builds fail

□ HTML5 Notifier Plugin
Provides W3C Web Notifications support for builds.

*Figure 1 – Jenkins Plugin Installation*

6. Jenkins main interface → New Item → Freestyle Project
7. Advanced → Check “use custom workspace”
8. Enter the directory where the sources and the reports are going to be stored.
9. Add build step → Execute shell
10. Enter our project's build command followed by the analysis command.
    Example: rats --quiet --resultsonly --html /your_directory > /your_directory/report.html



Project name: test

Description:

[Plain text] Preview

**Advanced Project Options**

☐ Quiet period

☐ Retry Count

☐ Block build when upstream project is building

☐ Block build when downstream project is building

☑ Use custom workspace

Directory: /jen

Display Name: jen

☐ Keep the build logs of dependencies

**Build**

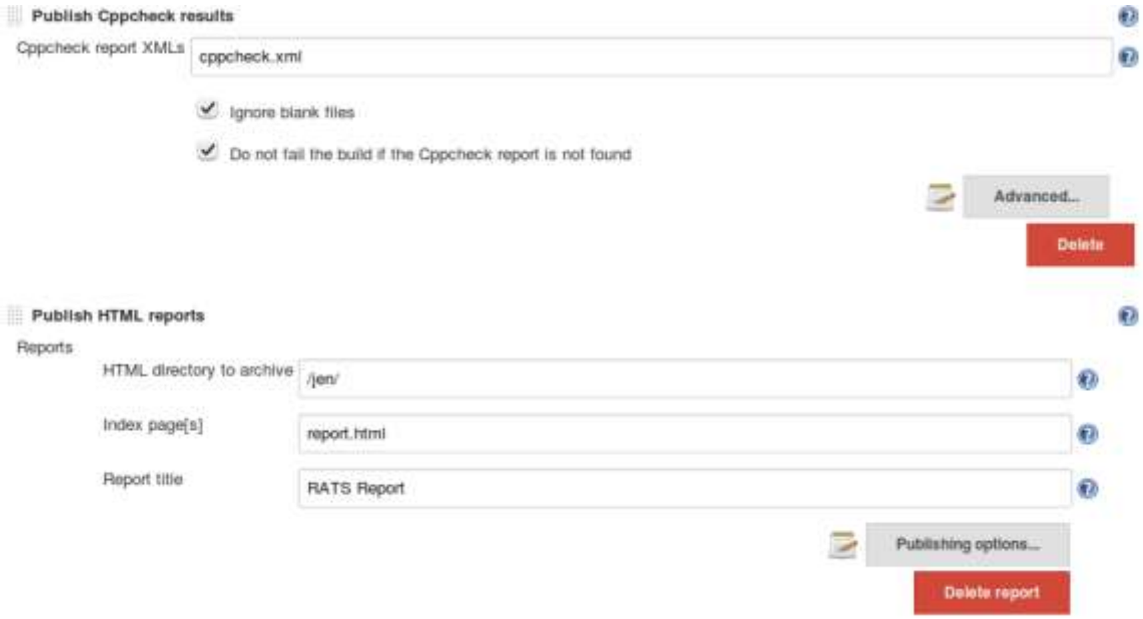
**Execute shell**

Command:

```
gcc /jen/ex.c -o /jen/ex
rats --quiet --resultsonly --html /jen > /jen/report.html
cppcheck --enable=all --inconclusive --xml --xml-version=2 /jen/ex.c 2> /jen/cppcheck.xml
```

*Figure 2 – Jenkins Project Configuration Interface*

It is important to not forget to add the report like the figure below:



**Publish Cppcheck results**

Cppcheck report XMLs: cppcheck.xml

☑ Ignore blank files

☑ Do not fail the build if the Cppcheck report is not found

**Publish HTML reports**

Reports

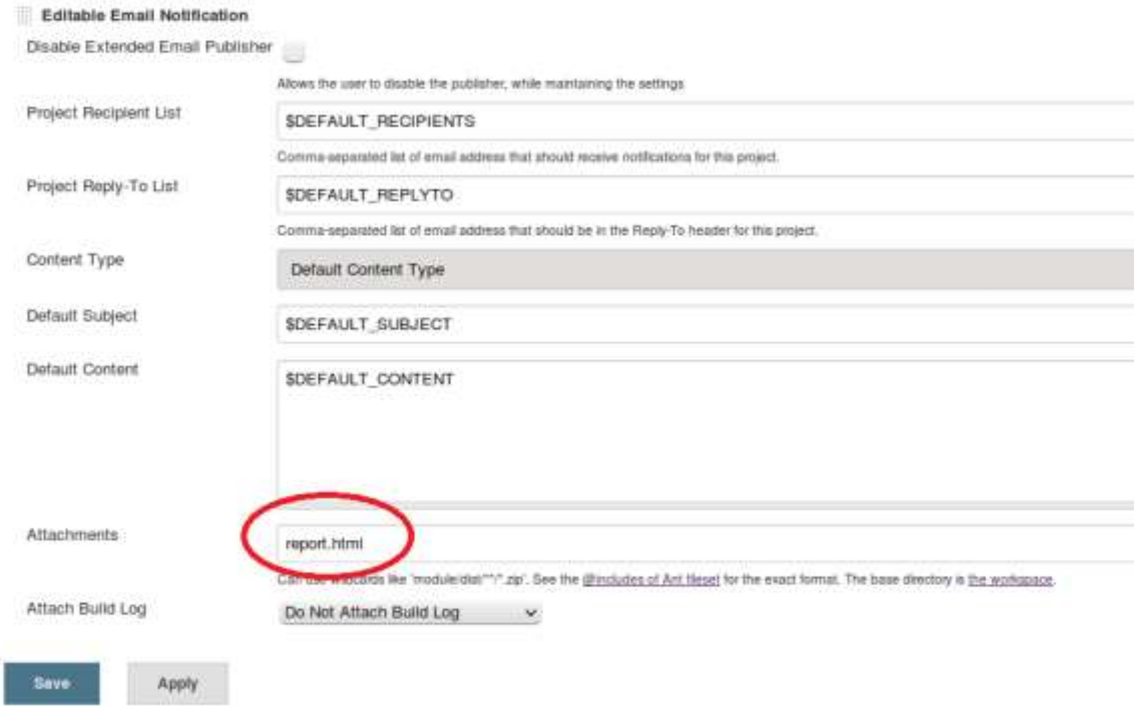
HTML directory to archive: /jen/

Index page[s]: report.html

Report title: RATS Report

*Figure 3 – Jenkins Project Configuration Interface*

Also, we have to add the report as an attachment to make the manual review process easier for the developer.



**Editable Email Notification**

Disable Extended Email Publisher □

Allows the user to disable the publisher, while maintaining the settings

Project Recipient List: $DEFAULT_RECIPIENTS

Comma-separated list of email address that should receive notifications for this project.

Project Reply-To List: $DEFAULT_REPLYTO

Comma-separated list of email address that should be in the Reply-To header for this project.

Content Type: Default Content Type

Default Subject: $DEFAULT_SUBJECT

Default Content: $DEFAULT_CONTENT

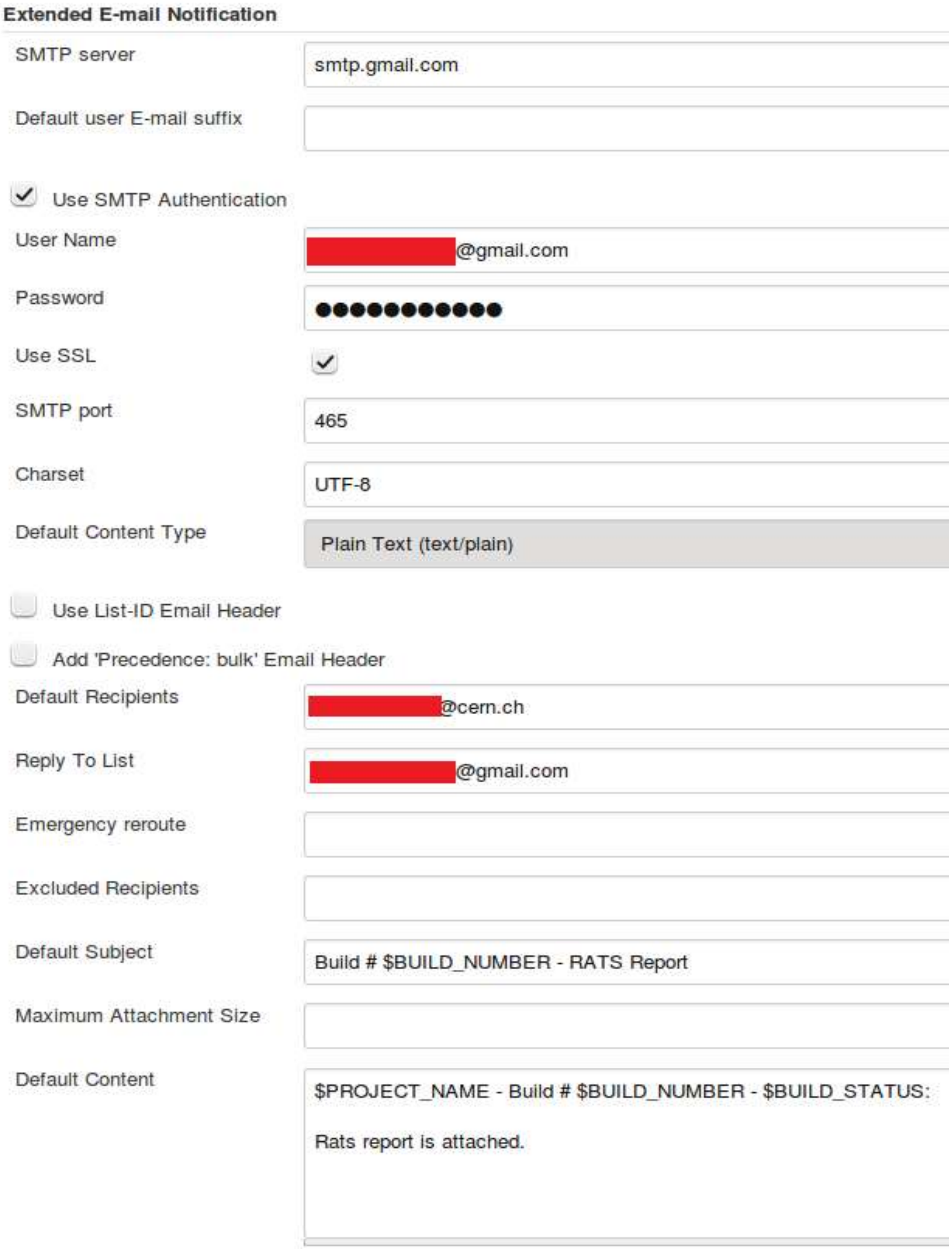
Attachments: report.html

Can use wildcards like 'module/dist/**/*.zip'. See the @includes of Ant fileset for the exact format. The base directory is the workspace.

Attach Build Log: Do Not Attach Build Log

*Figure 4 – Jenkins Project Configuration Interface*

Finally, to complete our project we have to configure the email settings (SMTP server, credentials) in order to send each email with the report of the static analysis tool without any problems.



**Extended E-mail Notification**

| | |
|---|---|
| SMTP server | smtp.gmail.com |
| Default user E-mail suffix | |
| ☑ Use SMTP Authentication | |
| User Name | [redacted]@gmail.com |
| Password | ●●●●●●●●●●●● |
| Use SSL | ☑ |
| SMTP port | 465 |
| Charset | UTF-8 |
| Default Content Type | Plain Text (text/plain) |
| □ Use List-ID Email Header | |
| □ Add 'Precedence: bulk' Email Header | |
| Default Recipients | [redacted]@cern.ch |
| Reply To List | [redacted]@gmail.com |
| Emergency reroute | |
| Excluded Recipients | |
| Default Subject | Build # $BUILD_NUMBER - RATS Report |
| Maximum Attachment Size | |
| Default Content | $PROJECT_NAME - Build # $BUILD_NUMBER - $BUILD_STATUS:<br>Rats report is attached. |

*Figure 5 – Jenkins Email Plugin Configuration Interface*

# 7 Future Work

There are many things to be done to have a complete automated system scanning millions lines of code. At first, we should integrate as many static analysis tools as we can in Jenkins, because as we obverse from the results all the tools have their strength and weaknesses. Furthermore, since not all of them are working both in Windows and Linux we have to research how we can integrate windows tools on a Jenkins instance.

Moreover, there are valuable security tools that have not been tested for this project and could be integrated in Jenkins platform with the same process described above.

# 8 Conclusion

In conclusion, source code static analysis tools help us to spot and eliminate bugs in the early stages of development when they are easy to fix. Many serious bugs can be only detected by analysing the source code which is also called “whitebox testing”. The integration with Jenkins automates this process so the code can be scanned on regular basis and repeatedly like nightly builds while it keeps the output suitable for developers. In the near future, this will lead to better software quality, faster development and easier testing.

CERN’s Computer Security Team provides a web page with the most recent recommendations for static analysis tools along with installation instructions: https://security.web.cern.ch/security/recommendations/en/code_tools.shtml

# 9 Appendix (Installation Instructions)

## Cppcheck

```
Download the installer from http://cppcheck.sourceforge.net/
Run the installer
```

## Flawfinder

```
wget http://www.dwheeler.com/flawfinder/flawfinder-1.31.tar.gz
tar -xzvf flawfinder-1.31.tar.gz
cd flawfinder-1.31
./flawfinder
```

## RATS (Rough Auditing Tool for Security)

```
Installing Dependencies – Expat Library
wget http://downloads.sourceforge.net/project/expat/expat/2.0.1/expat-2.0.1.tar.gz
tar -xvf expat-2.0.1.tar.gz
cd expat-2.0.1
./configure && make && sudo make install
Installing RATS
wget https://rough-auditing-tool-for-security.googlecode.com/files/rats-2.4.tgz
tar -xzvf rats-2.4.tgz
cd rats-2.4
./configure && make && sudo make install
./rats
```

## VCG (Visual Code Grepper)

```
1  Download the installer from http://sourceforge.net/projects/visualcodegrepp/

2  Run the installer
```

## SonarQube

```
   Installing SonarQube

1  Download http://www.sonarqube.org/downloads/

2  Unzip the distribution ie: "C:\sonarqube" or "/etc/sonarqube"

3  Windows / Other OS Execution

3a Execute StartSonar.bat in sonarqube\bin folder

3b Navigate and execute /etc/sonarqube/bin/[OS]/sonar.sh console

   Installing SonarQube Runner

4  Download http://www.sonarqube.org/downloads/

5  Unzip the SonarQube Runner

6  Create Configuration File sonar-project.properties

7  Java Configuration File Sample

   # Required metadata

   sonar.projectKey=UNIQUE:CHOOSE_ANY_UNIQUE_KEYWORD_FOR _PROJECT

   sonar.projectName=LANGUAGE::PROJECT_NAME_HERE

   sonar.projectVersion=1.0

   # Comma-separated paths to directories with sources (required), enter
   '.' for current directory

   sonar.sources=.

   # Language

   sonar.language=java

   # Encoding of the source files

   sonar.sourceEncoding=UTF-8
```

```
    Analyse a Project

8   Windows / Other OS Execution

8a  Navigate to the Sonar-Runner dir and execute \bin\sonar-runner.bat

8b  Navigate and execute  /etc/sonar-runner/bin/sonar-runner

9   Scan Results are in http://localhost:9000

10  Credentials for logging into the system are admin/admin
```

## Findbugs

```
1  wget http://prdownloads.sourceforge.net/findbugs/findbugs-3.0.1.tar.gz

2  tar -xfz findbugs-3.0.1.tar.gz

3  cd findbugs-3.0.1/bin

4  ./findbugs
```

## PMD

```
1   Download pmd-bin-5.3.3.zip from here http://sourceforge.net/projects/pmd/

2   unzip pmd-bin-5.3.3.zip

3   cd pmd-bin-5.3.3/bin

4   Windows / Linux Execution

4a  In Windows execute pmd.bat

4b  In Linux execute run.sh

5   Windows / Linux Example

5a  C:\>pmd-bin-5.3.2\bin\pmd.bat -dir c:\my\source\code -format text -R
    java-unusedcode,java-imports -version 1.5 -language java –debug

    C:\>pmd-bin-5.3.2\bin\pmd.bat -dir c:\my\source\code -f xml -rulesets
    java-basic,java-design -encoding UTF-8

    C:\>pmd-bin-5.3.2\bin\pmd.bat  -d  c:\my\source\code  -rulesets  java-
    typeresolution -auxclasspath commons-collections.jar;derby.jar

    C:\>pmd-bin-5.3.2\bin\pmd.bat -d c:\my\source\code -f html -R java-
    typeresolution -auxclasspath c:\my\classpathfile

5b  pmd-bin-5.3.2/bin/run.sh pmd -dir /home/workspace/src/main/java/code
    -f html -rulesets java-basic,java-design,java-sunsecure

    pmd-bin-5.3.2/bin/run.sh pmd -d ./src/main/java/code -f xslt -R java-
    basic,java-design -property xsltFilename=my-own.xsl

    pmd-bin-5.3.2/bin/run.sh pmd -d ./src/main/java/code -f html -R java-
    typeresolution -auxclasspath commons-collections.jar:derby.jar
```

**List of Rulesets with Description**

http://pmd.sourceforge.net/pmd-5.3.2/pmd-java/rules/java/

## Codepro Analytix

```
1  Download and Install Eclipse 3.7 Indigo

2  Open Eclipse and go to: Help → Install New Software → Add

3  In Name field enter: http://dl.google.com/eclipse/inst/codepro/latest/3.7

4  Click Next and finish the installation.
```

## Pyflakes

```
1  yum install python-pip

2  pip install pyflakes
```

## Pylint

```
1  sudo yum install pylint
```

## Perl-Critic

```
1  sudo yum install perl-Perl-Critic
```

## PHPca

```
1  Download PHPca https://github.com/spriebsch/phpca
2  Extract all the files in your home directory
   Step if you do not have PHP installed
3  sudo yum install php
4  Navigate to the directory where you extracted the files
5  Use PHPca like this:  php src/phpca.php -p "path" "file or directory"
   Where "path" is the path of the php binary such as /usr/bin/php
```

## RIPS

```
1  Download package http://sourceforge.net/projects/rips-scanner/files/
2  Unzip the rips-0.XX.zip in your public html directory of Apache
3  Browse to 127.0.0.1 (localhost) using your browser
```