Design of the TOIF Adaptors and Framework.

Table of Contents

1 Summary

This document is the design of the TOIF adaptors and framework.

2 Introduction

There are two main roles of the TOIF (Tool Output Integration Format):

- Convert the different reports from 3rd party vulnerability detection and scan tools to a common report format, without modifying the original vulnerability detection tool in any way.
- Extract key data from the outputs and supplement the findings with additional weakness information.

The role of the TOIF (Tool Output and Integration Framework) adaptors are to map the findings of 3rd party vulnerability detection and scan tools, to a normalized TOIF output as defined in the specification of the TOIF XML schema. These adaptors will attempt to gather key facts from the 3rd party tool's output such as the data element relating to the finding, the location in the code where the finding was found, and the weakness description. In addition to gathering facts from the tool output, the adaptors will map the weakness to a Common Weakness Enumeration (cwe.mitre.org) for further, more detailed, information. The adaptors are run by the TOIF adaptor framework which provides utilities for constructing the findings and produces the final output. By using a common framework, the Adaptors can be reduced to only parsing their particular tool's output.

It is essential that the adaptors do not modify the original vulnerability detection tools. The internal working of the detection tools must be preserved, and only the output used. To do this the adaptors will wrap the scan tools, initiating their process, and collecting their results. The adaptors will then construct their own output before writing this to disk.

3 Context

This section describes the context in which the adaptors and their framework operate. This is to provide a brief overview before discussing the inputs and outputs of the TOIF adaptors.

Adaptors described in this doc are being run from their jar files. It is possible however, to wrap the adaptors in an Eclipse RCP application for ease of use.

The aim of the adaptor, when run on a particular source file, is to capture the output of a particular vulnerability tool and translate its output into a common output format.

The adaptors are to be run during the build of the project in question. This is done to allow access to all the files used during the build and also any required arguments, such as include paths, which the adaptors may need. To achieve this in an unobtrusive manner, a wrapper should be constructed around the compiler. In this way, arguments originally intended for the compiler can be redirected via the adaptor process before being forwarded onto the compiler. By wrapping the compiler, instead of being inserted into the makefile, the requirement of not intruding into the original source or build can be maintained.

The TOIF XML outputs produced from each input source files are written to subdirectories relating to which Adaptor produced them. These files can then be used to integrate the TOIF information into KDM.

4 Configuration

In order to run the adaptors, there is some configuration that will be required. It is essential that the required vulnerability detection tools are pre-installed and have been added to path.

House Keeping:
Because the TOIF output is able to provide information relating to the parties and tools involved in the analysis of a project, this information must be entered into a file and handed to the Adaptor as an argument. The text file is a map of keys and values. The keys relate to facts and entities used in the project, while the values are information about that fact or entity.

House Keeping Example:

```
############################
#         Facts
############################

TOIFSegmentIsRelatedToProject=project1
TOIFSegmentIsProducedByOrganization=org1
TOIFSegmentIsOwnedByOrganization=org1
TOIFSegmentIsGeneratedByGenerator=generator1
```

```
TOIFSegmentIsGeneratedByPerson=person1
TOIFSegmentIsSupervisedByPerson=person1

PersonIsInvolvedInProjectAsRole=person1;project1;role1
OrganizationIsInvolvedInProjectAsRole=org1;project1;role2
OrganizationIsPartOfOrganizationAsRole=org1;org2;role2
PersonIsEmployedByOrganizationAsRole=person1;org1;role1

##########################
#        Entities
##########################

SegmentDescription=Segment relating to the findings of the cppcheck tool on the wireshark
project.

#projectId=name;description
project1=Wireshark;Using Wireshark to test the TOIF tool chain.

#generatorId=name;description;version
generator1=cppcheck;Tool for static C/C++ code analysis intended to complement the checking of
the compiler. Checks for: memory leaks, mismatching allocation-deallocation, buffer overrun, and
many more.;1.4

#personId=name;email;phone
person1=Adam Nunn;adam@kdmanalytics.com;555-1234
#person2=Joe Bloggs;blogs@kdmanalytics.com;555-1234

#organizationId=name;description;address;phone;email
org1=KDM;Kdm Analytics;Richmond Road;555-5555;kdm@kdmanalytics.com

#organizationId=name;description;address;phone;email
org2=Acme;Acme Analytics;blah;555-5555;thingamy@thingamy.com

#roleId=name;description
role1=Software Developer;Developer on the TOIF Adaptors project
role2=company;employer
```

Although most entities and facts within the TOIF XML schema are optional, there are few house keeping elements which are mandatory.

- TOIFSegmentIsRelatedToProject
- Project
- TOIFSegmentIsGeneratedByPerson
- Person

Running the Adaptor from command line:

The adaptor is run from the command line. It is actually the framework which is run with the adaptor provided as an argument.

```
java -cp "./*" ToifAdaptor --adaptor <Adaptor Name> --inputFile <full path to input file>
--outputDirectory <path to output directory> --houseKeeping <path to house-keeping file>
```

The class path, “-cp”, must be provided in order for the framework to find the correct adaptor to run. This lets the Java Virtual Machine know where to look for the classes and packages. Multiple paths can be provided by separating the paths with a colon “:”.

“--adaptor” or “-a” is the fully qualified name of the adaptor class. This is the adaptor that is to be used with the input source file. From this class, the framework is able to discover house keeping facts about

the adaptor as well as which generator to call and what options to use.

“--inputFile” or “-i” is the full path to the input source file. This will usually be provided by the wrapper after extracting the correct file from the compiler options.

“--outputDirectory” or “-o” is the path to the output directory. This is the directory where the subdirectories containing the TOIF XML file will be written.

“--houseKeeping” or “-h” is the path to the file containing the facts about the project's house keeping. This file is specific to each adaptor and each project. This is because it is down to the user to provide the project details as well as which generator (scan tool) is running on the system.

“--arguments” or “-x”. These are additional, optional, arguments which may have to be handed to the vulnerability detection tool. An example of this is handing the include paths to the Splint adaptor. The additional arguments must be in the form of a single string.

Integrating with C project's build:
The best way to integrate the adaptors into the build, is by wrapping the compiler and the adaptors into a script. To start, all the arguments to the compiler should be iterated over:

```
#iterate over all the arguments.
for i in $@;do
    …
done
```

Arguments should be gathered:

```
#collect all the include arguments for adaptors like splint who need the include paths.
#if the argument begins with -I, then it is an include argument.
if [[ ${i} == -I* ]]
then
    INCLUDE="${INCLUDE} ${i}"
fi
```

And:

```
#complete the full path.
INPUT=$PWD/${i}

#if the file ends with .c , then the c adaptors can run on it.
if [[ ${INPUT} == *".c" ]]
then
    … #run the c adaptors …
fi
```

Finally, the original arguments should be passed onto the compiler:

```
gcc "$@"
```

Now when the compiler is called, the adaptors will be run for every source file used. To get the build process to use this wrapper instead of the compiler on its own, the compiler flag needs to be set during configure of the make:

```
./configure CC=myGccWrapper
```

The make can be continued as normal:

```
make
make install
```

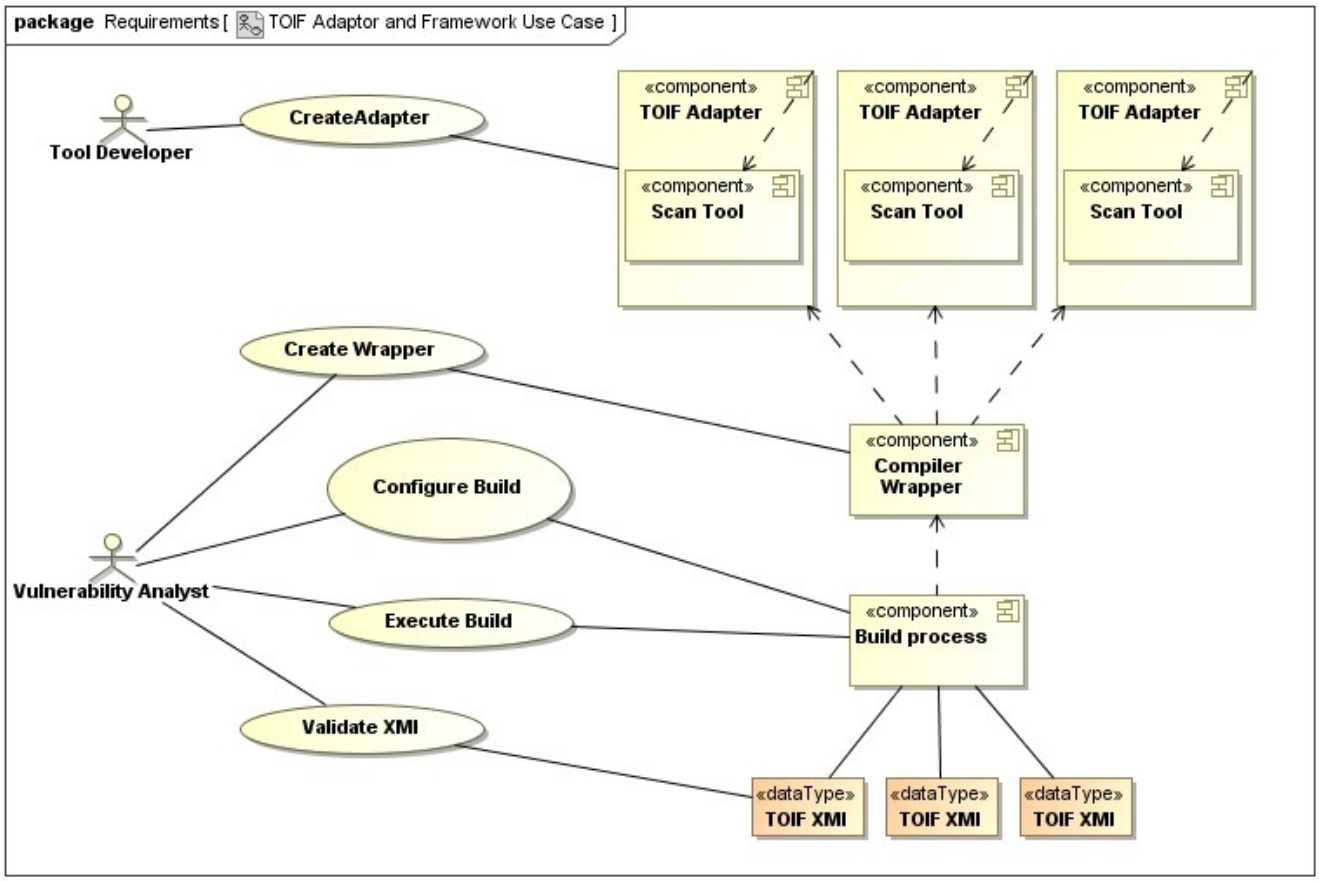
Integrating with Java project's build:
It may be possible to integrate into a Java project's build by modifying Apache Ant's “build.xml” file. Create a new target which will find all the “.class” files in the destination directory of the project. For each file, the Java adaptors will be run:

```
<target name="check" depends="compile_src">
    <foreach param="file" target="run">
        <path>
            <fileset dir="${classes_dir}">
                <filename name="**/*.class" />
            </fileset>
        </path>
    </foreach>
</target>

<target name="run" >
    <exec executable="java">
        <arg line="-cp /home/adam/Desktop/adaptors/* ToifAdaptor
--adaptor FindbugsAdaptor
--outputDirectory /home/adam/Desktop/output/
--houseKeeping cppcheckHouseKeeping
--inputFile"/>
        <arg value="${file}"/>
    </exec>

    <exec executable="java">
        <arg line="-cp /home/adam/Desktop/adaptors/* ToifAdaptor
--adaptor JlintAdaptor
--outputDirectory /home/adam/Desktop/output/
--houseKeeping jlintHouseKeeping --inputFile"/>
        <arg value="${file}"/>
    </exec>
</target>
```

To run the adaptors and compile the project the following command should be given:

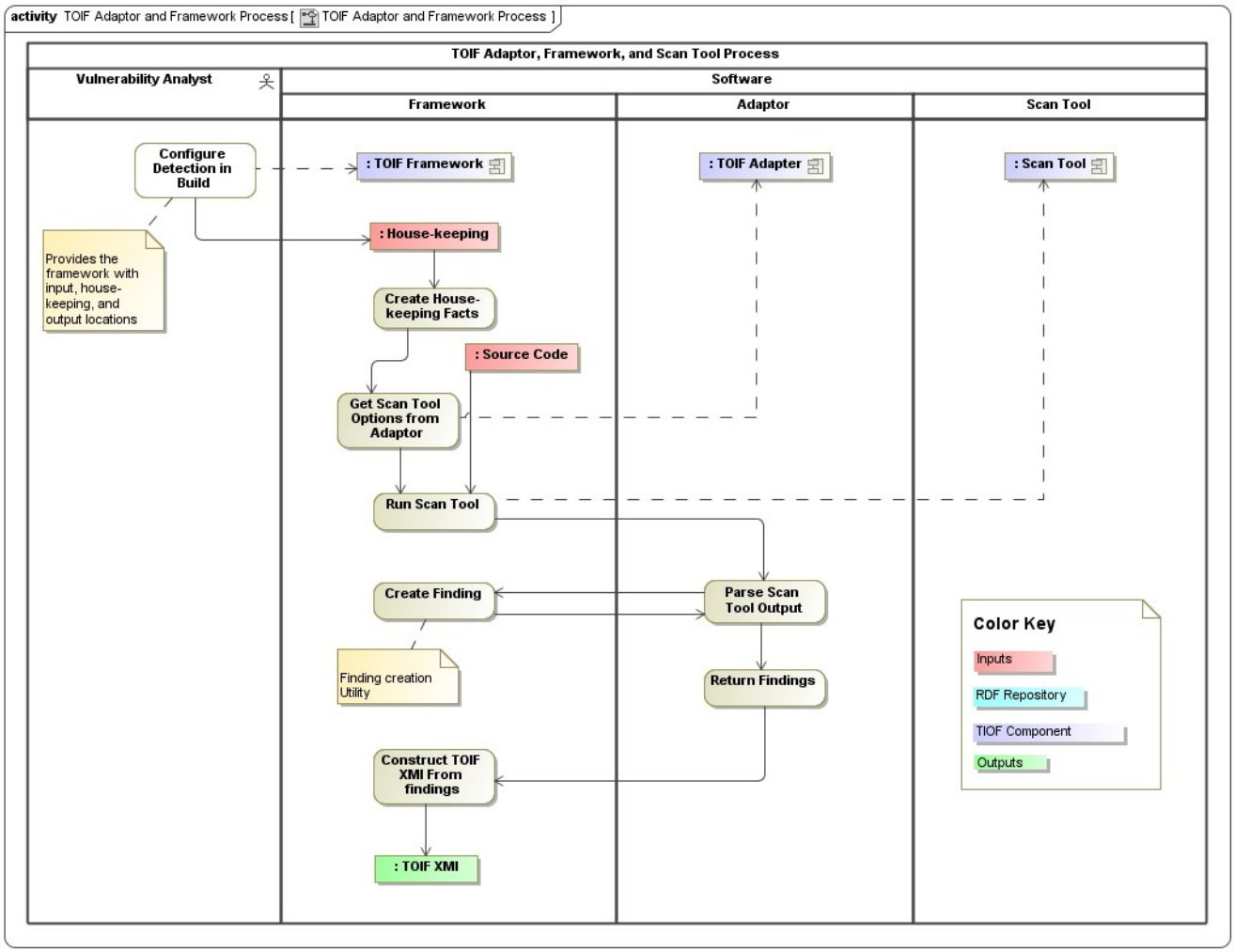
```
ant check
```

5 Use Case



- Tool integrator creates an Adaptor for each Scan Tool.
- The vulnerability analyst creates a wrapper for the compilers which will be used during the build.
- The build is configured to use the wrapper instead of the original compiler.
- The build is run, producing the TOIF output.
- The TOIF output is validated against the TOIF XML schema.

6 Adaptor and Framework Interaction



Framework:
The framework is run with the adaptor as an argument. From this argument, the Framework can find the adaptor class and query that class to find information out about the scan tool, the adaptor's name and description, and the arguments with which to run the vulnerability detection tool.

With the house keeping file as an argument the framework can create the elements which describe the project in detail.

The framework initiates the scan tool with the proper arguments, provided by the adaptor and the user, and hands this running process to the adaptor for parsing.

Once the adaptor has returned the findings it created whilst parsing the scan tool's output, the framework can create the XML and write it to disk.

Adaptor:
The adaptor takes the running process from the framework and parses the output of the scan tool. Because each vulnerability detection tool produces it own unique output, each adaptor must be different

in order to handle each output format.

When the adaptor discovers a finding, it passes the key elements of the finding to the finding creator utility in the framework package. This finding creator utility returns the facts and elements which make up the finding.

Finally the adaptor returns all the finding elements it has found to the framework.

7 Creating an Adaptor

All adaptors must extend the abstract class “AbstractAdaptor”, in the framework package. This class contains a series of methods which must be implemented:

public abstract String getAdaptorName();
This is where the name of the adaptor being written is returned.
Example: “Cppcheck”

public abstract String getAdaptorDescription();
A description of the adaptor should be returned here.
Example: “Adaptor for the cppcheck tool”

public abstract String getAdaptorVersion();
The version of the adaptor is to be returned here.
Example: “1.0”

public abstract ArrayList<Element> parse(Process process, AdaptorOptions options, File file);
This is probably the most important method in the adaptor. The returned ArrayList is a list of all the elements created by passing information extracted from the process to the finding creator utility. How the Adaptor parses the output of the scan tool is unique to each particular scan tool. The parse process will need an instance of the FindingCreator class. This will allow you to call create() on it for each finding that you find. The returned ArrayList is the result of calling getElements() on this instance.

public abstract String[] runToolCommands(AdaptorOptions options);
This method must return a String array of the arguments required to run the generator tool. If the tools output must be in a particular format, now is the time to specify it.
Example: {“cppcheck”, “--enable=all”, “--xml”, “-q”, options.getInputFile().toString()}

In addition to completing the abstract class, a configuration file must be created in a folder called “config” in the root of the jar. The name of the configuration file must be of the form:

```
/config/<Adaptor class simple name>Configuration
```

Within this file there must be a map of weakness Ids, their CWEs, Clusters, and SFPs.

```
#(bufferAccessOutOfBounds, error) Buffer access out-of-bounds: buffer
bufferAccessOutOfBoundsCWE=Path Resolution;SFP-16;CWE-126

#(dangerousStdCin, possible error) Dangerous usage of std::cin, possible buffer overrun
dangerousStdCin=;SFP-9;CWE-242
```

Clusters may be omitted with only a semi-colon marking their possition. If this is the case, then they will be looked up against the SFP.

Adaptors must also implement an extension point “com.kdmanlaytics.toif.adaptor” with the Adaptor class as the extension point class.

8 Tool Output

The output of the adaptors will conform to the TOIF XML schema. This schema describes the structure of the elements within the TOIF XML file.

For each individual input source file, an output TOIF XML file is created. This is written to a directory relating to the adaptor which generated it.

```
.../<output directory>/<adaptor name>/<input filename>.toif.xml
```

This output can then be used by other tools to generate reports or integrated into KDM.

Additional files are created along side the toif.xml files. These files show the standard output and standard err output of the original generator tool. This may be useful for debugging a new adaptor.

9 Creating an RCP Application

It may be desirable to create an RCP application from the TOIF plugins. A TOIF RCP product plugin has been provided on the git hub site and can be used to create the RCP application. You will need to include the TOIF and third party plugins in your workspace in order to use the TOIF RCP product plugin.

Resources for describing how an RCP is created are avaliable:

http://www.vogella.com/articles/EclipseRCP/article.html#tutorial_e4wizard