Advanced Software Protection:
Integration, Research and Exploitation

D5.13
ASPIRE Open Source Manual

Project no.:	609734
Funding scheme:	Collaborative project
Start date of the project:	November 1, 2013
Duration:	36 months
Work programme topic:	FP7-ICT-2013-10
Deliverable type:	Report
Deliverable reference number:	ICT-609734 / D5.13
WP and tasks contributing:	WP 5 / Task 5.3
Due date:	Oct 2016 – M36
Actual submission date:	6 December 2016
Responsible Organization:	UGent
Editor:	Bart Coppens
Dissemination level:	Public
Revision:	1

Abstract:
In this deliverable, we present which ASPIRE components have been open sourced, where they can be found online, and under which licenses they have been released. We have set up scripts to create Docker containers that allows users to easily set up all open sourced protection tools and the ACTC. Furthermore, we present manuals for setting up and getting started with this open sourced Docker protection tool flow, and to set up and get started with the ADSS Full suite.
Keywords:
Open Source, Docker, Manual, ACTC, ADSS, protection techniques

Editor
Bart Coppens (UGent)

Contributors (ordered according to beneficiary numbers)
Bart Coppens (UGent)
Daniele Canavese, Leonardo Regano, Cataldo Basile (POLITO)

The ASPIRE Consortium consists of:

Ghent University (UGent)	Coordinator & Beneficiary	Belgium
Politecnico Di Torino (POLITO)	Beneficiary	Italy
Nagravision SA (NAGRA)	Beneficiary	Switzerland
Fondazione Bruno Kessler (FBK)	Beneficiary	Italy
University of East London (UEL)	Beneficiary	UK
SFNT Germany GmbH (SFNT)	Beneficiary	Germany
Gemalto SA (GTO)	Beneficiary	France

Coordinating person: Prof. Bjorn De Sutter
E-mail: coordinator@aspire-fp7.eu
Tel: +32 9 264 3367
Fax: +32 9 264 3594
Project website: www.aspire-fp7.eu

Disclaimer The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 609734. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Executive Summary

Almost all protection tools written by the academic ASPIRE partners have been open sourced, including the ACTC which was a collaboration between UGent and NAGRA and GTO. The only exception are the source-to-source protections written by FBK, which they want to commercialize. Furthermore, we have created a Docker environment that contains the open sourced ACTC and all open sourced protection tools. This allows users to get started with applying the ASPIRE tools to their code very quickly.
In this deliverable, we present an overview of all these open sourced components: where the code can be found on the internet, and under which license they have been released.
This deliverable also contains manuals for both the Dockerized protection flow and the ADSS Full decision support suite.

Contents

1 Introduction

In this deliverable we describe and document different aspects of the open sourced ASPIRE protection tools. First, we give an overview of all open sourced components in Section 2. The main part of this document are the manuals of the open sourced tool chain in Sections 3 and 4. In Sections 3, we explain step by step how to set up the entire ASPIRE tool chain in a Docker container. Furthermore, we will explain how to run the ASPIRE Compiler Tool Chain (ACTC) in this Docker container to protect a demo application with both offline and online protection techniques. This section will also be published online as documentation for people who want to experiment with the ASPIRE tools. Section 4 describes how to set up and get started with the open sourced ADSS Full system.
Note that this document is not intended to be a developer guide for the ASPIRE tool chain. Deliverable D5.11 serves as a developer guide, containing detailed descriptions of how the ACTC and the different components interact.
Finally, note that this document servers as documentation of D5.12, the actual open source code that is available in the GitHub repositories listed below.

2 Overview of Open Sourced projects

All of the open sourced tools are published on GitHub. These are all published as individual repositories. These repositories belong either to the generic ASPIRE GitHub organization (which can be found at `https://github.com/aspire-fp7/`), or belong to a GitHub organization of to the tool owner. The open sourced projects have been released under different but compatible licenses. This list documents the locations where the source code of the tools has been released, and under which license.

- *Diablo* is released at `https://github.com/diablo-rewriter/diablo/` under the GPLv2 license. The anti-debugging component is released at `https://github.com/diablo-rewriter/diablo-selfdebugger` under the Creative Commons Attribution Noncommercial ShareAlike 4.0 international license.
- *Remote Attestation* is released at `https://github.com/aspire-fp7/remote-attestation` under a 3-clause BSD license.
- *ACCL* is released at `https://github.com/uel-aspire-fp7/accl/` under a 2-clause BSD license.
- *ASCL* is released at `https://github.com/uel-aspire-fp7/ascl/` under a 2-clause BSD license.
- *Code Mobility* is released at `https://github.com/uel-aspire-fp7/code-mobility/` under a 2-clause BSD license.
- *Code Guards* is released at `https://github.com/aspire-fp7/code-guards/` under a 3-clause BSD license.
- *Renewability* components are released at `https://github.com/uel-aspire-fp7/renewability/` under a 2-clause BSD licese.
- *ACTC* is released at `https://github.com/aspire-fp7/actc/` under a 3-clause BSD license.
- *Annotation Extractor* is released at `https://github.com/aspire-fp7/annotation_extractor` under the MIT license.
- Patches to enable dynamic metrics computation, which build on top of the *PANDA* dynamic analysis framework. These are released at `https://github.com/aspire-fp7/panda-metrics` and is released under the GPLv2 license, which is the license under which the original PANDA tool is distributed as well.
- ADSS Full is released at `https://github.com/SPDSS/adss` under the Eclipse Public License - v1.0.
- ADSS Light is released at `https://github.com/uel-aspire-fp7/adss-lightundertheEclipsePublicL` `0`. For documentation on the ADSS Light, we refer to the public deliverable D4.06 Section 6.

All these different repositories (except the PANDA patches, as they part of a standalone tool) are linked together in a framework repository located at `https://github.com/aspire-fp7/framework`. The commits in this repository contain references to specific revisions of all the other repositories that are known to work well together. Thus, to get the source code of all tools that work together, a user simply has to clone this repository and initialise all the subrepositories by executing `git submodule update --init`. Note that this is automatically done when a user sets up the Docker container as described in Section 3.
It is important to note that the source-to-source protection techniques developed by FBK, i.e. the data protections and the client-to-server code splitting technique, are not open sourced as they plan to commercialize these techniques. Furthermore, while the binary of the annotation_extractor developed by FBK has been released and is available, the code itself has not been released.

3 Manual of the ACTC Docker container

In this manual, we will explain how to set up the ASPIRE tools and how to use them to protect a simple application.
The set up of the open sourced version ASPIRE tools is simple thanks to the use of the Docker virtualization framework[1]. The rest of this manual describes how to set up a Docker container with the ASPIRE tools in it. It will also explain how to apply different ASPIRE protections with the ASPIRE Compiler Tool Chain (ACTC).

3.1 Setting up the Docker container

We assume that you have already Docker installed on your machine. If not, you can follow the instructions from the Docker website at `https://www.docker.com/`.
To set up an ASPIRE Docker container, you have to clone the repository that contains the base Docker files, which is located at `https://github.com/aspire-fp7/docker`:

```
# git clone https://github.com/aspire-fp7/docker/
```

This repository contains the Docker file, and contains some useful scripts. Running the `build.sh` script fetches the correct ASPIRE repositories from GitHub to the directory given as argument (which is automatically cleaned), and proceeds to build the ASPIRE docker image called *aspire*:

```
# cd docker
# mkdir actc
# cd actc
# ../build.sh .
```

When building the Docker container, most ASPIRE projects are built from scratch inside the container. This allows you to immediately start developing and extending any existing tools, without having to worry about how to build the projects and how to overwrite the pre-built files. The only down side is that setting up the initial container will take some time. The build process takes about 4 minutes on a decently modern machine.
The only files that are not built inside the container are:

- The patched binary tool chains. These can be rebuilt from source by cloning Diablo's toolchains repository located at `https://github.com/diablo-rewriter/toolchains` and following the instructions in the `README.MD` file.
- Versions of the OpenSSL, libwebsockets and libcurl libraries that have been built with Diablo-compatible tool chains. These can be rebuilt from source by cloning the `https://github.com/aspire-fp7/3rd_party/` repository and following the instructions in the `README.MD` file.

This script also prepares a 'projects' directory which already has the actc-demos repository checked out into it. It is one of the samples in that demos repository that we will be using in the remainder of this manual.

3.2 Starting the Docker

We have provided a script to start the Docker container with the correct parameters:

[1] `https://www.docker.com/`

```
# ./run.sh
root@33f36aef63c2:/projects#
```

This opens a bash shell in the *aspire* container in which we will be executing all further commands. Furthermore, this script sets up the correct ports on your Docker container for the online protection techniques. Finally, it maps the `projects/` directory into the container as `/projects/`.

3.3 Running the ACTC without any protections

The `build.sh` sets up a repository with a demo project in `/projects/actc-demos/`. We will be protecting the bzip2 compression utility in this demos repository. As an example, we have already added some annotations to the `bzip2.c` source file.
First, we will use the ACTC to build an unprotected bzip2 binary for the ARM Linux platform. We have provided an ACTC configuration file for this purpose. You can view this file at `actc/bzip2_linux.json`. This configuration file instructs the ACTC in such a way that the ACTC will build the binary, but it will not apply any protections. To run the ACTC with this configuration file, do the following:

```
# cd actc-demos/bzip2-1.0.6/
# /opt/ACTC/actc.py -d -f actc/bzip2_linux.json
```

This produces a final binary in `actc/build/bzip2_linux/BC05/bzip2`. If you have an ARM development board, you can copy this file and run it as you would do any other binary.
Warning: It is possible that when you stop your container and restart it, the MySQL server that is required by some of the steps in the ACTC has stopped. In that case, you will see MySQL-related errors either when running the ACTC or when running a protected application (in which case the errors will appear in the log files). You can restart the server by running

```
# /etc/init.d/mysql restart
```

3.4 Running the ACTC with only offline protections

The ACTC now has applied no protections at all to our binary. As a first step, we will apply some offline protection techniques. The annotations to instruct the protection tools which code fragments need to be protected have already been added in the source code (you can find these by looking for `_Pragma("ASPIRE begin` and `_Pragma ("ASPIRE end")` in the source files). As you can see, we have added annotations for *call stack checks*, *binary obfuscations*, and *code guards*. We only need to enable their application in the configuration file of the ACTC.
The configuration file is a JSON file that can be edited easily. The Docker container has vim installed, but of course you can use any editor of your liking.

```
# vim actc/bzip2_linux.json
```

Code guards is a technique that consists of a source-to-source transformation and a binary transformation. First, we enable the source-to-source part. We search for `"SLP08"`, which is the name of the source step responsible for this. We see that the `"traverse"` is currently set to `true`, which means that the action for this step simply copies the files from the input directory to the output directory, rather than applying the action in this step. Thus, set this action to **`false`**.
Next, we have to edit the configuration where the binary protection techniques are described. Search for `"BLP04"`, which is the name of the final binary protection step in the ACTC. In the configuration for this step, we will now enable all the aforementioned protections: set `"obfuscations"` and `"call_stack_check"` to **`true`** (the binary part of the code guards protection is automatically enabled or disabled depending on how we configured the `"SLP08"` step). We can now run the ACTC as before:

```
# /opt/ACTC/actc.py -d -f actc/bzip2_linux.json
```

This again produces the protected binary in `actc/build/bzip2_linux/BC05/bzip2`. This time, you can check the log files in the `actc/build/bzip2_linux/BC05/` directory to verify that the protections have indeed been applied. For example, the file `bzip2.diablo.obfuscation.log` contains information of which binary obfuscations have been applied to which code fragments.

3.5 Running the ACTC with code mobility

Now that we have created a version of our binary with offline protections applied, it is time to apply our first online protection called code mobility. This will split off binary code fragments from the application, and replace them with stubs that at run time ask for these code fragments to be downloaded from a protection server. Only once they are downloaded at run time, are these code fragments executed.
The first step is to enable this protection in the JSON configuration. There are two steps needed to enable this: enabling the technique itself, and configuring the server. To enable the technique, simply change the value of **"code_mobility"** to **true** in the "BLP04" section of the configuration file.
To configure the server in the JSON configuration file, go to the "COMPILE_ACCL" section of the configuration, and modify the value of "endpoint" to be the IP of the machine on which your Docker container is running.
Again, run the ACTC as before:

```
# /opt/ACTC/actc.py -d -f actc/bzip2_linux.json
```

You can first verify that the code mobility protection was applied by inspecting the log file called `bzip2.diablo.code_mobility.log`. Next, verify that mobile code blocks were indeed produced. If you look up at the output of the ACTC, near the end it will write information to the terminal similar to:

```
.  SERVER_P20
   code mobility       /projects/actc-demos/bzip2-1.0.6/actc/build/bzip2_linux
      /BC05/mobile_blocks

/opt/code_mobility/deploy_application.sh -a D7846E47BB09D62A2824CA9CF5000AE8 -
   p 20 -i YOUR_IP_HERE /projects/actc-demos/bzip2-1.0.6/actc/build/
   bzip2_linux/BC05/mobile_blocks && touch /projects/actc-demos/bzip2-1.0.6/
   actc/build/bzip2_linux/BC05/mobile_blocks/.p20done

APPLICATION ID  = D7846E47BB09D62A2824CA9CF5000AE8
.  SERVER_RENEWABILITY_CREATE
```

This indicates that the ACTC is deploying the mobile blocks to the location in which the code mobility server expects these blocks to be. This location depends on the `APPLICATION ID` (AID) mentioned in the output of the ACTC: they are located in `/opt/online_backends/<AID>/code_mobility/00000000/`:

```
# ls  /opt/online_backends/D7846E47BB09D62A2824CA9CF5000AE8/code_mobility
   /00000000/
mobile_dump_00000000  mobile_dump_00000001  mobile_dump_00000002
   mobile_dump_00000003  mobile_dump_00000004  mobile_dump_00000005
   mobile_dump_00000006  mobile_dump_00000007  mobile_dump_00000008
   mobile_dump_00000009  mobile_dump_0000000a  source.txt
```

As you can see in the source code, the code mobility annotation was applied to the `uncompress` function. So if you now copy the protected file to an ARM board and try to decompress a file, the code mobility protection will be triggered. You can afterwards verify the logs in the server to see which blocks were requested:

```
# scp actc/build/bzip2_linux/BC05/bzip2 user@armboard:.
# ssh user@armboard
user@armboard:~$ dd if=/dev/zero of=./zeroes bs=1M count=10 ; bzip2 zeroes
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.080404 s, 130 MB/s
user@armboard:~$ ./bzip2 -d zeroes.bz2
user@armboard:~$ logout
# tail /opt/online_backends/code_mobility/mobility_server.log
Fri Nov 25 15:02:24 2016 [Code Mobility Server] Actual revision for app_id
   D7846E47BB09D62A2824CA9CF5000AE8 is 00000000

Fri Nov 25 15:02:24 2016 [Code Mobility Server] BLOCK_ID 8 requested
```

```
Fri Nov 25 15:02:24 2016 [Code Mobility Server] BLOCK_ID 8 (filename: /opt/
    online_backends/D7846E47BB09D62A2824CA9CF5000AE8/code_mobility/00000000/
    mobile_dump_00000008) is going to be served.

Fri Nov 25 15:02:24 2016 [Code Mobility Server] BLOCK_ID 8 is 52 bytes long.

Fri Nov 25 15:02:24 2016 [Code Mobility Server] BLOCK_ID 8 correctly sent to
    ASPIRE Portal.
```

Warning: the server ports of the ASPIRE servers should not be firewalled on your Docker machine. Similarly, if your Docker is running inside a virtual machine, your virtual machine monitor should forward these ports to the VM itself. The ports are all ports between 8080 to 8099 (inclusive) and port 18001.

3.6 Running the ACTC with remote attestation

Now that we have protected the application with code mobility, we will enable another online protection technique called remote attestation. This technique uses a server to verify the integrity of code fragments during the application execution. The application connects to the server, which then instructs the application to send attestations of certain code regions back to the server. The results of these attestations can be linked to the code mobility protection technique to stop serving code to applications that have been compromised.
First, we enable the remote attestation step in the ACTC configuration file. The name of the step source-to-source part of the remote attestation protection is named SLP07. So, to enable remote attestation, simply change the `"excluded"` value in the `"SLP07"` section of the configuration file to **`false`**. The binary part of this protection technique is again automatically enabled and disabled based on the value in `"SLP07"`.
Again, to build the protected application with the ACTC, just run as before:

```
# /opt/ACTC/actc.py -d -f actc/bzip2_linux.json
```

The output of the ACTC immediately shows that the remote attestation was deployed:

```
Add attestator in the DB (nr: 00000000000000000000, name: remote_ra_region, f:
    10)
Generating inital 100 prepared data for current attestator (launching
    extractor)
Attestator inserted with ID: 5
Inserting startup areas in the DB, found 1 areas
Startup area: 0

**** RA application components deployed on server successfully ****
```

To verify that the binary itself indeed connects to the protection server of the remote attestation technique, and that this server indeed receives valid attestations, we can again copy the protected binary to a development board, run it to compress and decompress the file we created earlier to demonstrate code mobility, and check the attestation logs on the server:

```
# scp actc/build/bzip2_linux/BC05/bzip2 user@armboard:.
# ssh user@armboard
user@armboard:~$ ./bzip2 zeroes ; ./bzip2 -d zeroes.bz2
[1927760:9055] NOTICE: Initial logging level 7
<... some additional logging information about the connection is displayed
    ...>
user@armboard:~$ logout
# tail /opt/online_backends/remote_attestation/verifier.log
03E2A03010E28DBB00E3922A020A02D3
75E190009DE39330F8E3FE305CE19F00
00E5933078EB014D9AE1FE2C45E39CCA
38EB00FFD40A5330AFEB02490AE19220
9DE30300BCE39CCA06E2A03027E1900A
03E24F3848E2911A00E58D1010E30D20
03E2900A0EE3A0DB
(Verifier) Response verification result: SUCCESS
(Verifier) Response verified in = 0.001435 s
```

```
(Verifier) Execution finished at: Fri Nov 25 15:55:07 2016
```

This shows that the protected application indeed connected to the server, and that the server sent an annotation request back to this client, and that this client successfully responded to that request.

4 Manual of the ADSS Full

Section authors: Daniele Canavese, Leonardo Regano, Cataldo Basile
In this manual we present how to install the ADSS as a normal user or as a developer. Moreover, we provide basic information on how to use the ADSS to protect software applications.

4.1 Installation

In this section we describe the procedure for installing the ADSS. We distinguish between the user installation, for using the ADSS "as is", and the developer installation, for adding new features to the ADSS.

4.1.1 User installation

The ADSS comes in form of an Eclipse exported product, which is essentially a stand-alone Eclipse installation containing all the required plug-ins to execute the ADSS. The supported OS are Windows and Linux, for 64 bit architectures. A Java Runtime Environment, version 7.x or above, must be installed on the machine.
The ADSS requires, for the attack inference phase, that SWI-Prolog[2] (version 7.2.3 or above) is installed on the machine. For Windows is provided a self-installing executable, while for Linux the recommended option is installing from source code (is also available as a PPA). To be invoked properly, SWI-Prolog may also need, depending on the OS, these environment variables to be set:

- in Windows, add an environment variable called *SWI_HOME_DIR*, and set its value to the installation directory of SWI-Prolog;
- in Linux, add to the *LD_LIBRARY_PATH* environment variable the path

  ```
  <SWI_Prolog_Install_Dir>/lib/amd64
  ```

  substituting <*SWI_Prolog_Install_Dir*> with the installation directory of SWI-Prolog (the default path is */usr/lib/swi-prolog*).

To display the attack graphs, GraphViz[3] must be installed on the machine. A self-installing executable is available both for Windows and Linux. Only on Windows machines, the path to the *bin* folder in the GraphViz installation directory must be added to the *PATH* environment variable.
For the L2P evaluation phase, a working installation of IBM ILOG CPLEX Optimization Studio [4] is needed. For both Windows and Linux is provided a self-installing executable. Only in Linux, the user must add to the *LD_LIBRARY_PATH* environment variable the path <*CPLEX_Install_Dir*>*/bin/x86-64_linux*, substituting <*CPLEX_Install_Dir*> with the installation directory of CPLEX, by default

```
/opt/ibm/ILOG/CPLEX_Enterprise_Server<VERSION>
```

where <VERSION> is the version number of the installed CPLEX).
For the source code analysis phase and the deployment phase, a working ASPIRE virtual machine is needed.
Given this, the user simply needs to download the archive from
`https://github.com/SPDSS/adss`
by selecting the version of the ADSS for the OS (Windows or Linux) running on the machine, extract the downloaded archive and run the contained executable.

[2] http://www.swi-prolog.org
[3] http://www.graphviz.org
[4] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

4.1.2 Developer installation

In this section we provide the instruction for setting a working Eclipse installation to extend the ADSS. Before starting, all the requirement stated in the previous section must be met.
First of all a working Eclipse for RCP and RAP Developers installation is needed. The version of Eclipse for which the instruction are given is Mars.2 (4.5.2). Newer versions of Eclipse should be compatible.
Then the following plug-ins must be installed inside Eclipse (Install New Software in the Help menu):

- from `http://download.eclipse.org/releases/mars` install EMF - Eclipse Modeling Framework Xcore SDK;
- from `http://download.eclipse.org/tools/cdt/releases/8.8.1` install C/C++ Development Tools;
- from `http://www2.compute.dtu.dk/~ekki/projects/ePNK/indigo/update/` install:
  1. ePNK Core;
  2. ePNK Basic Extensions;
  3. ePNK HLPNGs;
- from `http://download.eclipse.org/nattable/releases/1.4.0/repository/` install:
  1. NatTable Core;
  2. NatTable GlazedLists Extension Feature.
- from `http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/` install:
  1. Xtext Complete SDK;
  2. Xtext Redistributable.

After this, from Eclipse all the ADSS plug-ins must be imported from the GitHub repository.
Then an Eclipse launch configuration must be created and then edited, from the *Run Configurations* windows, in this way:

- in the *Main* tab, select an execution environment based on Java 7 or above;
- in the *Arguments* tab:
  - if Linux is running on the machine, add to the program arguments the string

    ```
    -Djava.library.path="/usr/lib/:/usr/lib/swi-prolog/lib:
    /usr/local/lib/:<CPLEX_Install_Dir>/CPLEX_Studio/cplex/bin/x86
        -64\_linux"
    ```

    substituting <*CPLEX_Install_Dir*> with the installation directory of CPLEX (the default path is */opt/ibm/ILOG/CPLEX_Enterprise_Server*<*VERSION*>, where <VERSION>is the version number of the installed CPLEX);
  - if Windows is running on the machine, add to the VM arguments the string

    ```
    -Djava.library.path="<CPLEX_Install_Dir>\CPLEX_Studio<VERSION
        >\cplex\bin\x64\win64;
    <SWI_Prolog_Install_Dir>\bin"
    ```

    substituting <*CPLEX_Install_Dir*> with the installation directory of CPLEX (the default path is *C:\Program Files\IBM\ILOG*), <*VERSION*> with the version number of the installed CPLEX, and <SWI_Prolog_Install_Dir>with the installation directory of SWI Prolog (the default path is *C:\Program Files\swipl*);
- in the *Plug-ins* tab, select from the *Launch with* drop-down menu the option *all workspace and enabled target plug-ins*.

4.2 Getting started

In this section we provide a walk-trough of the main features of the ADSS. For further information on the ADSS work-flow phases, please refer to the deliverable D5.11, Part III, Section 14.

4.2.1 Creating an ADSS project

A new ADSS project can be created using the Eclipse *New Project* wizard (from the *File* menu, *New*, *Project...*): in the wizard, select *ADSS Full Software Protection Project* in the folder *ADSS*.
In the first page of the wizard are available the main settings for the project:

- *Project Name*: the name of the ADSS project;
- *Location*: the path where the ADSS project will be created;
- in the *Source Code* section, the paths for the Source Code Analysis and the Protection Deployment phases:
  - *Working directory*: the main folder of the sources of the target application, which must contain the files listed below;
  - *ACTC configuration file*: the name of the ACTC configuration file;
  - *ADSS patch file* and *ADSS JSON file*: the names of the ADSS patch and JSON files, will be generated in the *Deploy a solution phase*;
- *Target Architecture*: the CPU architecture on which the target application will run; at the time of writing, only the *ARMv7-a* architecture is supported;
- *Solution Estimation Profile*: contains a slider for choosing the level of accuracy of the ADSS algorithms used in the Protection Discovery phase; setting an high level of accuracy causes a longer execution time of the algorithms used to find the golden combinations.

In the second page of the wizard the user can set the parameters for the connection with the Aspire VM. If the check-box *Remote connection* is unselected, the ADSS will assume that a working ACTC is present on the running machine; otherwise the ADSS will use the following parameters for establishing an SSH connection with the Aspire VM:

- *Remote host*: the host name or IP address of the Aspire VM;
- *Remote port*: the TCP port on which the SSH daemon running on the Aspire VM is listening;
- *Remote user name* and *Remote password*: the user name and password for logging in on the Aspire VM;
- *Remote server file separator*: the path separator symbol used by the OS running on the Aspire VM.

In the last page of the wizard the user can set the paths for running the following external applications (if the remote connection is enabled, the application will be executed in the Aspire VM):

- *ACTC*;
- *Perl* interpreter;
- *dot*, a graphing tool part of the *GraphViz* distribution, used by the ADSS for drawing the attack graphs.

Clicking on *Finish* the ADSS project is created, and the *Overview* tab is displayed.

4.2.2 Overview

The *Overview* tab contains the main information of the ADSS project:

- the number of assets of the target application, available after executing the source code analysis phase;
- the number of protections supported by the ADSS;
- the number of attack paths found against the target application, available after executing the attack paths detection phase;
- the working directory, ACTC configuration file, ADSS patch file and JSON file, as explained in the previous section.

In this tab the user can launch the ADSS on the target application in two ways: clicking on *Build All*, all the phases of the ADSS are launched automatically; otherwise, the user can launch the single phases separately. Using the step-by-step approach, the user can interact with the ADSS between the execution of the single phases, in order to drive the protection process: for example, after executing the first and/or second level protection discovery, the user can select which solution to deploy; with the fully automatic approach, the ADSS deploys the best available solution.

4.2.3 Logs tab

In this tab, the logs produced by the ADSS are displayed. The user can select 7 different level of verbosity, or alternatively can completely shut down the logging.

4.2.4 Preferences tab

In this tab, the user can set all the preferences for the execution of the ADSS. The preferences are organized in sets.
In the remote connection set, the user can set the parameters for the SSH connection with the Aspire VM; the parameters are the same already presented in Section 4.2.1.
In the commands set, the user can provide the paths for the external applications used by the ADSS, as explained in Section 4.2.1.
In the parsing set, the user can set the following preferences for the source code analysis phase:

1. *Parse headers*: if also the headers of the target application source code must be taken into account when analyzing the code;
2. *Ignore system files*: if selected, the ADSS ignores the system headers (e.g. *stdio.h*) when analyzing the code.

In the attack finder set, the user can set the various constraints to speed up the attack paths detection phase, as for example a time limit and a depth limit for searching in the call graph.
In the protection finder set, the user can set the minimum level of mitigation that a protection must have, against at least one attack type, in order to be taken into account in the protection detection phase. Using high level of mitigation for this setting speeds up the computation and decreases the accuracy of the protection detection phase.
In the overheads set, the user can specify the maximum overheads introduced in the application by the founded combination of protection; the overheads taken into account are:

1. *client time*, a multiplicative factor that, applied to the CPU time required on the client machine for executing the vanilla application, gives the same measure but for the protected application;
2. *server time*, a multiplicative factor that, applied to the CPU time required on the server machine for executing the vanilla application, gives the same measure but for the protected application;
3. *client memory*, the quantity of additional memory needed on the client machine for executing the protected application in respect to the vanilla;
4. *server memory*, the quantity of additional memory needed on the server machine for executing the protected application in respect to the vanilla;
5. *network*, the quantity of additional network usage introduced for sending the packets of on-line protections.

In the first level protections set the user can customize the execution of the correspondent phase by setting the following parameters:

1. the number of solution to find;
2. if an additional Petri net assessment of the solutions must be executed on the founded solutions;
3. the termination criteria for the phase, e.g. a time limit;
4. for each overhead type, the percentage of the total amount of allowed overhead that must be used by the first level protections, in order to leave sufficient available overhead for the second level protections;
5. some advanced options for configuring the minimax algorithm.

In the second level protection set the user can customize the execution of the correspondent phase by setting the following parameters:

1. the number of solutions to find starting from single first level solution;
2. the termination criteria for solving the MILP problem, e.g. a time limit;
3. some advanced options for configuring the generation of the MILP problem.

4.2.5 Rules tab

This tab contains a text editor that allows the user to specify its own custom inference rules. These rules are automatically executed by the ADSS during the source code analysis phase and are used to model additional a-priori knowledge that cannot be inferred by the standard ADSS framework, such as the list of custom encryption functions.
The rules follow a simple if-then paradigm, where the user can specify a set of conditions and an action to execute. When all the conditions match, the action is executed. For instance, the rule:

```
if enc(ciphertext, plaintext, cryptographicKey, _) then
encrypt:AES128-ECB
```

Specifies that the `enc()` function should be tagged as an encryption function with AES-128 in ECB mode and that its first parameter is the ciphertext, the second one the plaintext, the third one the key and the fourth can be ignored.

4.2.6 Assets tab

After executing the source code analysis phase, the user can see the founded application parts of the target application source code, arranged in a table with the following columns:

1. *Name*, the name of the application part;
2. *Type*, the type of the application part, can be a function, a generic variable, an integer, an integer array, a cryptographic key, a plaintext, a ciphertext, an initialization vector or a parameter of a function;
3. *Source File : Line*, the source file path in which the application part can be found; if the application part is a variable, the line of declaration is also provided, otherwise if is a function the start and end line are provided;
4. *Properties*, if the application part is an asset, the relative security requirements; can be edited by the user;
5. *Weight*, if the application part is an asset, the weight, i.e. the importance of the asset; can be edited by the user.

If an application part contains one or more application parts, these are arranged in a tree. Double clicking on an application part, a window showing detailed information on the application part is displayed.

4.2.7 Attacker tab

Before executing the attack discovery phase, the user can select the attacker profile in the *Attacker* tab. In particular the user can select four increasing levels of expertise for the attacker: amateur, geek, expert, guru. In this tab are also listed the attacker tools, in form of a table with the following columns:

1. *Name*, the name of the tool;
2. *Types*, the types of attacker tools to which the tool belongs;
3. *Expertise*, the minimum level of expertise needed by the attacker to successfully use the tool; if the selected expertise level is lower than the one stated in this column, the tool will be disabled and will not be taken into account in the attack discovery phase.

4.2.8 Attacks tab

This tab shows the attacks found against the security requirements of the target application assets. Therefore, this tab is filled in only after executing the attack discovery phase. The attack paths are displayed in a table with the following columns:

1. *ID*, an unique identifier for the attack;
2. *Attack steps*, the ordered sequence of attack steps constituting the path;
3. *Targets*, the pairs asset/security requirements that are breached by the attack path;

4. *Protections*, available after executing the protection detection phase, the list of the possible protections to mitigate the attack;
5. *Maximum expected mitigation*, the maximum mitigation of the attack available between the protection listed in the previous column.

Also all the inferred attack steps are displayed individually in another table, with their attack step type.

4.2.9 Protections tab

This tab contains a table listing the available protections; for every protection is displayed its name and the number of available protection instantiations relative to the protection, and if the protection is enabled or not. In the last column is also present a check-box, that permits the user to enable or disable the protection.

4.2.10 Solutions tab

The solutions tab display the combination of protections found by the first and second level protection discovery phases. The solutions are displayed in a table with three columns: an unique identifier, the sequences of applied protections constituting the solution, and a protection index.
The execution of the first level solution phase can be started by the user by clicking on the relative link in the *Overview* tab. After the execution of this phase, the user can either start the second level solution phase (again, clicking on the relative link in the *Overview* tab), or alternatively can deploy one of the first level solutions in the way described in the next section. In the first case, a window listing the available first level solution will be displayed: the user can select one or more of them. The second level protection detection phase will be executed separately on every first level solution selected by the user, and the resulting second level solution will be listed in the *Solutions* tab. For second level protections, in the ID column is also showed in parenthesis the ID of the first level solution on which the second level one is based.

4.2.11 Deploying a solution

The user can deploy a solution by simply clicking on the *Deploy a solution* link in the *Overview* tab: a window listing the available solutions, both first and second level ones, will be shown; from this window the user can select the solution to deploy on the target application.

List of abbreviations

Abbreviation	Meaning
ACTC	ASPIRE Compiler Tool Chain
ADSS	ASPIRE Decision Support System
ASPIRE	Advanced Software Protection: Integration, Research and Exploitation
BCxx	Binary code directory nr. x
BLPxx	Binary-level software processing step nr. xx
JSON	JavaScript Object Notation
RA	Remote Attestation
SLPxx	Source-level software processing step nr. xx