Advanced Software Protection:
Integration, Research and Exploitation

D4.04
Security Evaluation, Advanced Human Experiments

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 / D4.04
WP and tasks contributing:	WP 4 / Task 4.4
Due date:	Apr 2016 – M30
Actual submission date:	3 June 2016
Responsible Organization:	POLITO
Editor:	Cataldo Basile
Dissemination level:	Public
Revision:	v1.0

Abstract:
This deliverable presents the versions at M30 of the ASPIRE security model, the input/output model of the metrics framework and of the Security Protection Assessment tool, the results of empirical academic studies conducted in the M24-M30 period, and the planning of the industrial studies.

Keywords:
ASPIRE knowledge Base, security evaluation, empirical studies

Editor
Cataldo Basile (POLITO)

Contributors (ordered according to beneficiary numbers)
Bart Coppens, Jeroen Van Cleemput (UGent)
Cataldo Basile, Daniele Canavese, Leonardo Regano, Marco Torchiano (POLITO)
Mariano Ceccato, Paolo Tonella (FBK)
Paolo Falcarin, Elena Gómez-Martínez, Gaofeng Zhang (UEL)

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

This deliverable presents the status of WP4 activities and the outcomes at M30. Achievements are divided in three parts: the first part presents the updates to the ASPIRE Security Model, the second part presents the metrics framework and the security evaluation models, finally, the third one presents the results and planning of academic user studies and the planning of industrial user studies.
The ASPIRE Security Model v1.2 (ASMv1.2) has been released at M30. ASMv1.2 describes all the concepts used in the AKB at M30 for storing information and reasoning about it. These concepts are used by both the ADSS Light and ADSS Full. ASMv1.2, as its previous versions, is composed of a main model and several submodel (assets, application, protections, attacks, metrics, and protection requirements sub-models). Compared to ASMv1.1, the ASMv1.1 presented here only presents minor differences in the assets and protection sub-models. In the assets submodel, the ASMv1.2 only adds the `SingleInstance` class, used to mark applications that have to use anti-cloning techniques. In the protection sub-model, the ASMv1.2 includes the `ProtectionInstantiation` class and `AppliedSWProtectionInstantion` association class to represent potential ways to use protections on an asset, and the `Solution` class to represent combinations of protections.
In task T4.2, the design of the extensions to the ASPIRE Compiler Tool Chain (ACTC) has been performed to support metrics extraction. The metrics subsystem itself consists of three ACTC compilation steps (*BLP00*, *BLP04* and *BLP04_DYN*), each one generating different types of metrics, and the *M01* step, which collects metrics results and formats the output into a set of *metrics file*. Modifying the application the generate the metrics is done by rewriting object files and binary files (or libraries) using the Diablo obfuscation tool. All metrics files output by M01 have a similar format: the filename reports the category of metrics that that be found inside; then, every line in the file reports the metrics computed on a code region. We also report that since the previous deliverable D4.03, no new metrics been have developed.
Moreover, in Task T4.2 the Security Protection Assessment (SPA) tool has been improved. In the last six months, the effort concentrated on the evaluation techniques based on the fitness function already introduced in D4.03. The purpose has been to make available the comparison features of the SPA tool to the ASPIRE Decision Support System (ADSS) Light. Therefore, while the theoretical framework is mainly unchanged (and it is thus not reported here), effort has been put to integrate the SPA tool with the ACTC to extract metrics and with the ADSS to drive the evaluation process performed by the SPA tool, and to test the evaluation techniques on sample applications and on the use cases.
In task T4.3 the ASPIRE user studies have been performed on academic subjects. Two experiments have been conducted. First, a replication with data obfuscation in attack tasks on compiled binary code, which has involved 34 Master students from Universiteit Gent, has showed that these techniques have a dramatic impact on success rate and on the time necessary to complete a successful attack (success rate reduced by a factor 4.5 and attack time increased by a factor 2). This quantitative evidence proves that data obfuscation techniques are powerful deterrents of malicious tampering with binary code, as they render the attacks less economically convenient. Moreover, the experiments also prove that combining multiple protections has an even better impact. This is also confirmed by insights on attack strategies. Successful attacks were never targeted directly against an ASPIRE protection, rather they took advantage of side channels attacks or brute force. Therefore, if the application assets are designed to be safe against side channels attack or brute force, the job of attackers becomes extremely difficult and economically inconvenient if ASPIRE techniques are applied.
Second, the first replication (out of three) of the experiments to assess Client Server Code Splitting effectiveness has been conducted with 10 participants at the University of East London, which gave us only partial data. Indeed, due to the small number of participants and to the fact that only two participants could successfully complete the attack task, no significant result is highlighted by statistical tests. Since the same experimental design will be reused in the forthcoming replications

on the other academic project partners, we will have more data points to draw statistically sound conclusions. The complete set of experiments on code splitting and the new (statistically sound) conclusions will be reported in deliverable D4.06, to be delivered at M36.
In task T4.4 the ASPIRE user studies with industrial participants have been designed. The experiment is a long running case study, with loose control on the involved subjects and mostly qualitative data collected during the execution of the experiment. It will consists of a single replication with two (or more) consecutive phases, to give hackers a larger continuous amount of time to work on their task (with freedom to switch among phases) and a more flexible allocation of time to replications (in case the ASPIRE Consortium would decide to change phases and their difficulty). Currently, experiments at NAGRA have been started, experiments at SFNT have started and are temporary stopped to allow hackers to improve their background on the ASPIRE context, while experiments at GTO are planned for M32-M33.

Contents

List of Figures

List of Tables

1 Introduction

Section authors:
Cataldo Basile (POLITO)

The goal of this deliverable (see GA Annex II DoW part A) is to document the updates and the tool support for the security models and the evaluation techniques in the ASPIRE's Work Package 4 at M30. WP4 is creating the necessary infrastructure to allow the ASPIRE Decision Support System (ADSS) to select the best combination of protections for preserving the assets in the application to protect. The WP4 outcomes include the ASPIRE security model, the ASPIRE knowledge base, the metrics framework, and the evaluation techniques based on analytic methods (fitness function) and simulation models (Petri nets). Moreover, WP4 includes the evaluation of protection techniques with real human beings, to check if (ex-post/ex-ante) estimates on the protection level are correct, and the public challenge.
Therefore, this deliverable presents the updates on:

- the ASPIRE Security Model (developed in T4.1 "Security Model and Evaluation Methodology"), which allows the formal representation of the ASPIRE concepts , stored by the ASPIRE Knowledge Base in a way to perform the sophisticated reasoning used by the ADSS;
- the metrics framework (developed in T4.2 "Complexity metrics"), designed to evaluate software protection strength through the use of software complexity and protection resilience metrics ;
- the Security Protection Assessment tool (developed in T4.2 "Complexity metrics"), designed to evaluate the impact of protections and impact of attacks against the application assets;
- the user studies, which have conducted with academic and industrial parties to evaluate the effectiveness of the ASPIRE protections (developed in T4.3 "Experiments with academic subjects" and T4.4 "Experiments with industrial tiger teams").

We report here that the public challenge to be performed in task T4.5 is delayed with 6 weeks because the additional effort required at UGent to support all the partners for the protection of the use case applications for the tiger team experiments. However, this delay does not affect the project in a major way. In fact, the public challenge design is documented in the internal WD4.05, it will run for six months as expected even if it will be completed after the end of the project. The costs that for the period will not be claimed and the analysis of the obtained information will be available in time for the Y3 review. Therefore, the public challenge code and setup will be delivered in M31 with D3.05 and documented, together with the results of the analysis, with D4.06. An update of D4.06 is expected before the Y3 review in case further interesting data would be available after the end of the project.
This deliverable is organized as follows. Part I presents the updates to the ASPIRE Security model. Section 2 reports the new version of the ASPIRE Security Model (ASMv1.2) and the changes compared to the last version (ASMv1.1). ASMv1.1 has been documented in the deliverable D4.03, Section 3.
Part II presents the updates to the ASPIRE Security Evaluation. Section 3 presents the input/output model of the metrics framework and sketches the modifications to the ACTC to support it. Section 4 reports the input/output model of Software Protection Assessment tool used by the ADSS Light to assess the protection strength and compare combinations of protections, based on the metrics computed by the metrics framework.
Finally, Part III reports the design and the results of the empirical studies conducted to evaluate the effectiveness of the ASPIRE protections. Section 5 presents the academic studies conducted at UGent to assess strenght of data obfuscation techniques against attacks conducted at binary level. Section 6 presents the academic studies conducted to assess the strength of Client Server Code

splitting. In this deliverable only the outcomes of the experiments at UEL are available. Section 7 presents the planning of the industrial studies, which will be conducted by the industrial partners during the next months.

Part I
The ASPIRE Knowledge Base

2 The final ASPIRE Security Model and ASPIRE Knowledge base

Section authors:
Cataldo Basile, Daniele Canavese, Leonardo Regano (POLITO)

This deliverable presents the ASPIRE Security Model v1.2 (ASMv1.2), which updates the ASPIRE Security Model v1.1 (ASMv1.1) described in the deliverable D4.03. The main model and all the sub-model do not present differences, but the Asset sub-model, and the Software Protection sub-model, which show minor moor updates. For the Asset and the Software Protection sub-models, we report the entire sub-model description together with a changelog that presents the differences compared to the ASMv1.1.
We also report here that there are no changes in the ASPIRE Knowledge base (AKB). The AKB purpose and organization of the information is unchanged. The AKB is still represented as an OWL-DL ontology, which have been updated to reflect changes into the ASPIRE Security Model. No additional inferences have been added at ontology level. Moreover, the AKB still features the support for the Enrichment Framework and Enrichment Modules already presented in the deliverable D5.07, which have been slightly updated in the last six months (and will be reported in D5.10 and D5.11).

2.1 The Asset submodel

Changelog: Compared to ASMv1.1, the ASMv1.2 only adds the `SingleInstance` class, used to mark applications that have to use anti-cloning techniques.

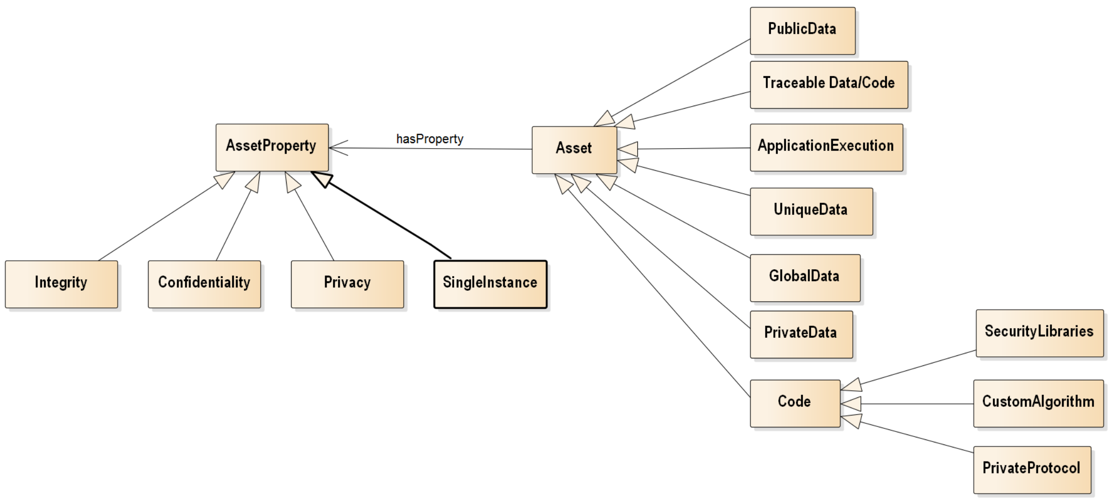


Figure 1: The asset sub-model.

Figure 1 reports the categorization of assets presented in D1.02. The different asset categories have been modelled with the sub-classing paradigm. The classes `PublicData`, `TraceableData`, `TraceableCode`, `ApplicationExecution`, `UniqueData`, `GlobalData`, `PrivateData`, and `Code` are subclasses of the main `Asset` class. Moreover, the `Code` class has been further sub-classed in `SecurityLibrary`, `CustomAlgorithm`, and `PrivateProtocol`. There are different asset properties, obtained as subclasses of the `AssetProperty` class: the `Integrity` class, used for the assets that must be protected from modifications, the `Confidentiality` class, for the asset that must remain secret, the `Privacy` class, for the assets that data whose disclosure affects the privacy of the person owning the application, and the `SingleIstance` class, used for applications that must be only used on a registered platform (e.g., no clones of the application must be executed).

2.2 The Software Protection submodel

Changelog: The SW protection sub-model was expanded to include the `ProtectionInstantiation` class and `AppliedSWProtectionInstantion` association class (and related associations) to represent a possible way to use a protections to protect an asset. It also includes the `Solution` class (and related associations) to represent combinations of protections.

Figure 2 shows the categorization of the protection techniques that are considered of interest for the ASPIRE project. First of all, it is possible to see the five lines of defence, i.e. the main, more abstract categories of protections as detailed in the ASPIRE DoW. These techniques are represented by means of the `DataHiding`, `AlgorithmHiding`, `AntiTampering`, `RemoteAttestation`, and `Renewability` classes, all subclasses of the `SWProtectionType` class. Together with the five lines of defence, we also added some more concrete protection types that play a major role in the ASPIRE project. They are represented by `ClientSideCodeSplitting`, `ClientServerCodeSplitting`, `ClientServerDataSplitting`, and `ReactiveTechnologies`.
All the previously mentioned classes are the first level of categorization. We also specialized some of these techniques, for instance:

- `DataHiding` has been sub-classed in `Source2SourceDataObfuscation` and `WBC` (white-box crypto);
- `AlgorithmHiding` has been sub-classed in:
  - `SourceLevelAlgorithmHiding`, further sub-classed in `PatternRemoval`;
  - `ClientSideVM`;
  - `BinaryCodeObfuscation`, further sub-classed in `CodeFlattening`, `BranchFunctions` and `OpaquePredicates`;
- `AntiTampering`, has been sub-classed in `AntiCodeInjection`, `AntiLibraryCallback`, `AntiDebug` and `CodeGuards`;
- `Renewability` has been sub-classed in `RenewabilityInSpace` and `RenewabilityInTime`
- `ReactiveTechnique` has been sub-classed in `TimeBombs`.

Integrity protection techniques, like remote attestation and anti-tampering techniques, are also categorized as either static and dynamic. However, instead of using the sub-classing paradigm we plan to add a Boolean attribute (`dynamic`) in those classes.
A protection technique exposes one or more protection profiles (see D5.01), implemented as instances of the `SWProtectionProfile` class. The `hasProfile` annotation is used to link the profiles to their related `SWProtection` objects.
Some protection techniques cannot be applied on the same asset in any order, so that a (partial) ordering is needed. The ASM models this by making use of the `cannotBePrecededBy` association, used to express the fact that a protection technique cannot used after another one.
Protection techniques can be used in different ways by changing their own configuration parameters. However, there are more meaningful ways to use them (which are selected by human beings

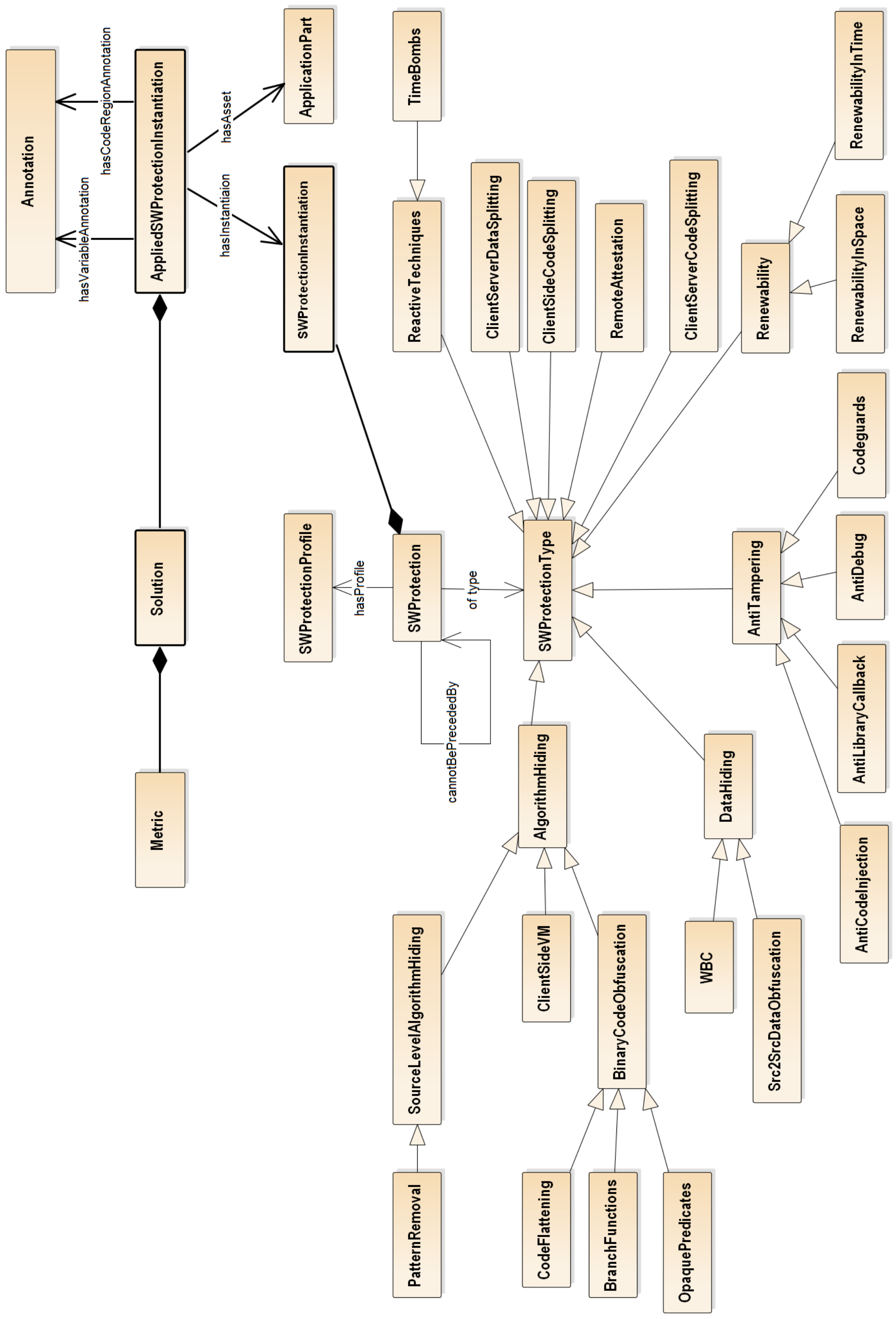


Figure 2: The SW protections sub-model.

and made available to the decision process)[1]. We introduced the `ProtectionInstantiation` class to represent a set of ways to use protections. The `AppliedSWProtectionInstantion` association class is then used to associate a `ProtectionInstantion` class instance (via the `hasInstantiation` association) to the `ApplicationPart` class instance to protect (via the `hasAsset` association) and the annotation to be inserted in the application source code (via the `hasVariableAnnotation` and `hasCodeRegionAnnotation` associations depending on the asset type.) Finally, the `Solution` class represent a combination of protections to be use to protect an entire application as an aggregation of `AppliedSWProtectionInstantion` association class instances. Moreover, `Solution` class instances also aggregate all the values of the metrics computed on the application protected with the protection instantiations it aggregates.

[1]For instance, in some case it is worth considering if 5%, 10%, 15% of the code needs to be obfuscated, it is not really interesting considering if 5%, 5.1%, or 5.5% of the code should be obfuscated.

Part II
Security Evaluation

3 Final Complexity metrics

Section authors:
Bart Coppens, Jeroen Van Cleemput (UGent)

This section gives a global overview of the metrics collection in the ACTC framework and discusses the file format used to represent the different metrics. This perspective is interesting as it presents the metrics subsystem as a black box component that is used by other tools developed in WP4 and by the ADSS developed in WP5. A complete overview of the ACTC metrics subsystem can be found the *D5.08* deliverable.
Since the previous deliverable, no new metrics been have developed. Therefore no updates are reported in this deliverable. Detailed analyses of the existing metrics can be found in the D4.03 deliverable. In the next months, more (dynamic) metrics will be implemented, which will be reported in the D4.06 deliverable at M36.

3.1 Metrics generation in the ACTC

Modifying the application the generate the metrics is done by rewriting object files and binary files (or libraries) using the Diablo obfuscation tool.
The global input-output behavior of the metrics subsystem is shown in Figure 3.
The various inputs for the metrics subsystem are shown on the left. The inputs of the metrics generation subsystem are the outputs of source to binary tools and binary to binary tools already present in the ACTC:

- BC08, which are the input (binary) object files
- BC02, which is the binary that has source level protections applied, but not binary level protections
- BLC02, which is are the extracted code fragments for the SoftVM component
- BC03, which are additional inputs object files for the binary protections.
- BC04, which is binary that has been protected with both source and binary protections
- ACTC JSON configuration

All of these inputs are defined in more detail in WD5.02.
The ACTC workflow has been extended with additional steps to generate and collect metrics. The metrics subsystem itself consists of three compilation steps *BLP00*, *BLP04* and *BLP04_DYN* shown in green, each generating different types of metrics, and *M01*, which collects metrics results and formats the output, the *metrics file*. The ACTC metrics steps reported in Figure 3 are:

BLP00 In this step the library or binary is rewritten to support the generation of *runtime profiles*. At the same time, *static metrics* are generated for the application protected with source code transformations, or the "vanilla" application in case not source code protections are applied. The *self-profiling binary* is then run on an actual target board to generate a *runtime profile*. This step is automated using use-case specific scripts that copy the required files to the board and retrieve the runtime profiles when the application has finished. Finally, the library or binary is rewritten based on the information contained in the runtime profiles and *dynamic metrics* are calculated.

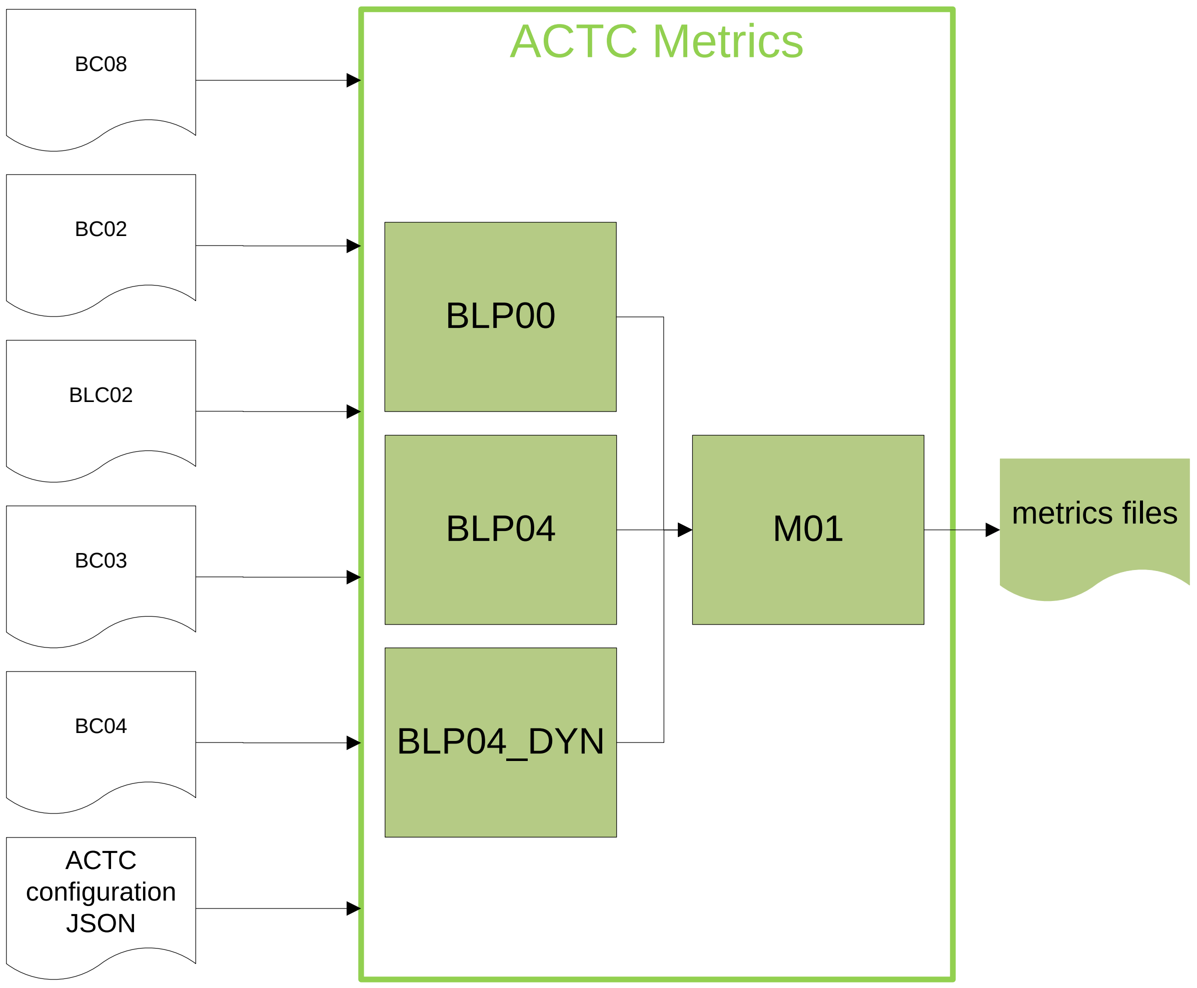


Figure 3: ACTC metrics generation and collection

BLP04 The existing diablo obfuscator step, BLP04, has been extended to generate *static metrics* for both the complete binary or library and for the individual protected regions. This step generates metrics for the application protected with both source code protection techniques and binary protection techniques.

BLP04_DYN When the diablo obfuscator uses the runtime profiles generated in *BLP00* as an additional input, *dynamic metrics* are generated as well. Finally,

M01 the metrics collection step *M01* collects the generated metrics from the individual output folders of the compilation steps in the metrics subsystem and centralizes them in the *M01* directory. The specific files that need to be collected during this phase can be configured in the ACTC configuration JSON file. M01 outputs all the metrics files in the same folder. The format of the metrics files is described in Section 3.2.

The output files produced by the metrics framework are:

BC02_SP (*.stat_complexity_info) These files contain the static complexity metrics computed over the entire program (without binary protections applied, i.e., the BC02 binary). This is produced by analysing the BC08 and BC02 inputs.

BC02_SP/profiles (*.plaintext) These files contain the dynamic profile information of the program without binary protections applied, i.e., BC02.

BC02_DYN (*.dynamic_complexity_info) These files contain the dynamic complexity metrics of the program without binary protections applied, i.e., BC02.

BC05 (*.stat_complexity_info, *.stat_regions_complexity_info) This is the static metrics information computed on the program with binary protections applied. This is produced by analysing the BC08, BC02, BC03 and BLC02 input files, and optionally by including the profile information from BC02_SP/profiles that can guide the binary protections.

BC05_DYN (*.dynamic_complexity_info) This is the dynamic metrics information computed on the program with binary protections applied, i.e., this is the dynamic complexity info that corresponds to the BC05 complexity info.

The current version of the metrics subsystem outputs a subset of the metrics proposed in the D4.02 deliverable. Moreover, it includes are additional metrics not described in D4.02, which are labelled with §. All the currently include the metrics are shown in Table 1.

Metric	Acronym	Description
nr_ins_static	INS	Number of instructions§.
nr_src_oper_static	SRC	Number of source operands§.
nr_dst_oper_static	DST	Number of destination operands§.
halsted_program_size_static	SPS	Static Program Size.
halsted_program_size_dynamic	DPL	Dynamic Program Size.
halsted_program_size_dynamic_coverage	DPS	Coverage Dynamic Program Size.
nr_edges_static	EDG	Number of static edges§.
nr_indirect_edge_CFIM_static	CFIM	Control Flow Indirection Metric
cyclomatic_complexity_static	CC	Static Cyclomatic Complexity

Table 1: Metrics resulting from the current ACTC execution.

These metrics are used by the Software Protection Assessment (SPA) tool and by the ADSS. These tools fetch them directly from the ACTC. Moreover, the ADSS stores the metrics data extracted into the AKB, following the data class structure in Section 2.

3.2 The Metrics file format

All metrics files output by M01 have a similar format. First of all, the name of the metric categories in Table 1 is used as filename. The first line of each file works as header and indicates by column the metrics values that will be found in the next rows. Then all the other lines report the metrics for each code region, following the metrics order described in the first line. All the lines with information on code regions start with a numerical identifier of the code region that is described in that line. Moreover, the `stat_complexity_info` files contain information about the *entire* program. Metrics about the entire program are reported as a single 'region' labelled with the index of '-1'. The `stat_regions_complexity_info` files contain information about the regions that have been annotated with ASPIRE protection annotations. An example of `stat_regions_complexity_info` file (which has a reduced number of columns to fit on this page) is as follows:

```
#region, nr_ins_static, halstead_size_static, cycl_complexity_static
0,       22,            83,                   2,
1,       142,           455,                  16,
```

This indicates that this file has two annotation regions; that the first region contains 22 instructions and that the second region contains 142 instructions. In addition, the first region has a cyclomatic complexity of 2, whereas the second region has a cyclomatic complexity of 16, etc. All of these metrics are described in much more detail in Deliverable D4.02.
The metrics files for the dynamic complexity information (`dynamic_complexity_info` files) follow the same structure. However, these have additional fields containing the dynamic metrics. These dynamic metrics are computed based on the dynamic profile files (`plaintext`), which

consist simply of one line for each basic block containing the start address of this basic block, and its dynamic execution count. All this information can be computed both on the application that has no binary protections applied (BC02) as the application with the binary protections applied (BC05).

4 Software Protection Assessment: New features

Section authors:
Paolo Falcarin, Elena Gómez-Martínez, Gaofeng Zhang (UEL)

The Software Protection Assessment (SPA) tool is the component of the ASPIRE framework to assess and compare different protection solutions. It is used by the ADSS Light, the component of the ADSS that permits the evaluation of the strength of combinations of protections against a set of attacks against the application assets. As introduced in deliverables D4.03, the SPA tool has three main components: PN editor, fitness function and PN simulator, illustrated in Figure 4.

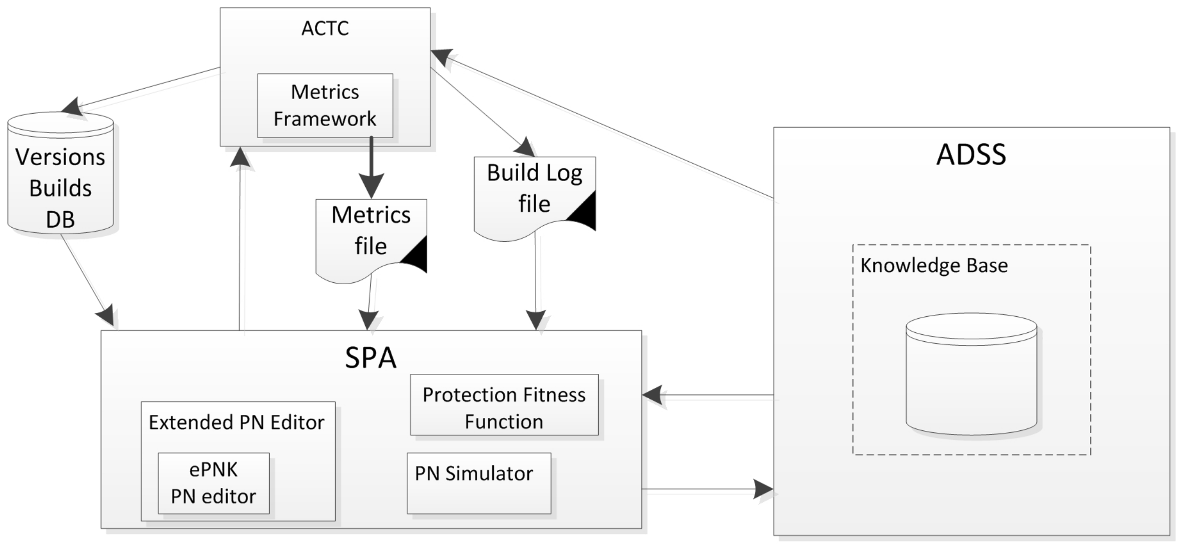


Figure 4: SPA architecture (taken from D4.03).

Since the delivery in D4.03, the SPA tool has been updated to be integrated with the ADSS Light and the ACTC. The SPA tool obtains the attack models from the ADSS and metrics data from the ACTC and feeds back assessment results. In other words, the current version of the SPA tool can do protection assessment on Petri Net based attack models, and run the same assessment functions on the attack paths available in the AKB, as computed by the Enrichment Framework (presented in Section 15 of D5.01 and updated in D5.07). Hence, the SPA can be executed for protection assessment with all data inputs from ADSS and ACTC, whose interaction is made by means of the fitness function. The extended PN editor could be an alternative option for the attack model input.
Compared to D4.03, PN editor and PN simulator have no significant updates in this deliverable. Minor updated will be needed for the full integration with ACTC and ADSS and will be delivered later in D4.06.
To carry out the assessment, the SPA tool collects some input data from ACTC and ADSS (both Light and Full).

- Metrics files: These files are generated by ACTC, which include the software metrics on vanilla and protected software. These metric data are essential for the assessment processes in the SPA. The metrics file format has been introduce in Section 3.2, which recently includes 9 metrics values for assessment.

- Attacks and Protections data from AKB: these data come from AKB and ADSS. The SPA can load directly the attack paths and protection solutions from AKB to carry out the assessment.
- Version Builds and Build Log files are generated by ACTC to identify previous metric files. Currently, due to the lack of sample codes, this input has not been used for assessment in SPA. In future work, SPA will use this input to improve related assessments.

In the SPA outputs/results, there are two parts: the fitness level of one protection solution and detailed logs.

- The fitness level of a protection solution on attack paths or one specific attack path is a number (type is double) that determines the effectiveness of protection solutions, the higher the level the better the solution. Based on this number, the ADSS Light can compare different protection solutions to identify the best one.
- The detailed logs on the assessment process include all formulas and data for assessment on each attack step. It is saved as txt files where the SPA tool is executed. Data in the detailed logs are only used for manual inspection and are not used by the ADSS Light.

Currently, in order to integrate SPA, ADSS and ACTC together, the ADSS light version is presented, which is a WP5 outcome. Compared to the ADSS Full, it focuses on the assessment and comparison of the existing protection solutions. The SPA tool is the main component of this ADSS Light. In this ADSS Light, users can configure all data for assessment firstly, including selecting attack paths, protection solutions, metrics files and so on. Then, ADSS Light can send all these data to SPA to carry out the assessment. Lastly, the assessment results (the fitness level of one protection solution on attack paths or one attack path) can be send back to the ADSS Light for solution comparison to find out the gold solution.

Part III
Experiments

The objective of the experiments is to investigate the level of protection offered by ASPIRE protections from the empirical point of view. Experiments are divided into experiments with academic participants and experiments with industrial participants.
In the experiments with academic participants, the experimental setting is represented by an artificial environment (i.e., in vitro experiment) where the experimenter controls and objectively measures all the relevant variables. This allows us to apply statistical analysis to elaborate objective observations. Academic participants are involved in multiple rounds of experiments to test ASPIRE protections, each run in isolation.
To make the presentation of the experiments self-contained, some fragments of text that describe the design have been taken from Deliverable D4.03. When text was reused from the previous deliverable this is clearly marked in the beginning of each section or subsection and the copied text is marked in blue.

5 Data obfuscation experiment

Section authors:
Mariano Ceccato, Paolo Tonella (FBK), Marco Torchiano (POLITO), Bjorn De Sutter, Bart Coppens (UGent)

In this section, we present a new replication of the experiment with data obfuscation. The replication has involved 34 students (32 Master, 2 Bachelor) students from Universiteit Gent in attack tasks on compiled binary code. We collected the following pieces of evidence:

1. The *success rate* of attack tasks on code obfuscated with the combination of *RNC* and *Array Reordering* is reduced to zero;
2. The *success rate* of attack tasks on code obfuscated with *Dynamic RNC* is substantially lower than the success rate of attackers working on clear code;
3. The *amount of time* required to attack code obfuscated with *RNC* and *Dynamic RNC* is higher than the time required to attack clear code;
4. Attacks to code obfuscated with *RNC* consist both of static analysis (using IDA-Pro) and dynamic analysis (using OllyDbg); and
5. Attacks to code obfuscated with *Dynamc RNC* consist mainly of dynamic analysis (using OllyDbg).

5.1 Experimental Definition

The planning and formal definition of the experimental settings have been conducted in a structured way, following the guidelines by [8].
The *goal* of this study (see Table 2) is to analyse the effect of data obfuscation – specifically *RNC*, *ArrayReordering*, *ArrayReordering* + *RNC* and *Dynamic RNC* obfuscation – with the *purpose* of evaluating its capability of making sensitive data resilient to malicious attacks. The *quality focus* regards how these obfuscation techniques reduce the attacker’s ability to correctly and efficiently extract data from the code. Investigating the effect of obfuscation on the attack efficiency is a crucial point in our experimentation: although we are aware that an attacker might be ultimately able to complete an attack on obfuscated code, he can be discouraged if such an attack requires a substantial effort/time. Results of this study can be interpreted from multiple *perspectives*: (i) a researcher

Table 2: Data obfuscation experiment definition

Goal :	Analyse the ability of data obfuscation to protect sensitive data inside the code
Treatments :	T0 = Clear code; T1 = Array Reordering; T2 = RNC data obfuscation; T3 = RNC + Array Reordering data obfuscation; T4 = Dynamic RNC data obfuscation;
RQ1 :	How effective is data obfuscation in protecting data inside the code as compared to the clear code?
RQ2 :	What is the effort to attack data obfuscated with data obfuscation as compared to clear code?
RQ3 :	What strategies and tools have been used to complete successful attacks?
Participants :	Students from Universiteit Gent
Systems :	P1=Lottery, P2=Lotto (compiled code): programs for the extraction of lottery/lotto numbers
Tasks :	Force the program to extract only numbers between 1 and 20; leak the winning sequence from the program
Metrics :	Success rate; time to mount a successful attack
Design :	Balanced design: two tasks, two objects, (five treatments)

	Group1	Group2	Group3	Group4
Lab1	P1-T0	P1-T4	P1-T2	P1-T0
Lab2	P2-T2	P2-T0	P2-T1	P2-T3

interested in the empirical assessment of data obfuscation; and (ii) a practitioner, who wants to ensure high resilience to attacks to sensitive data of applications delivered to clients, where they run in an untrusted environment.
The data obfuscation techniques that we have investigated in this experiment are:

Residue Number Coding (RNC): state-of-the-art obfuscation before ASPIRE. Numeric values are protected by encoding them as a tuple of integers computed as the respective residues over a tuple of moduli. Moduli are static constants in the code. This obfuscation is described in detail in Deliverable D2.01.

ArrayReordering (AR): In a variant of array reordering as described in D2.01, the initialization values of a read-only array are scrambled in this experiment. No indexing operations in the code needed to be adapted in this case, as the only purpose of the array is to feed the algorithm with a list of unique, strictly positive numbers. In the original, clear program, this is a simple list of consecutive natural numbers starting from 1, as a result of which each value equals its index + 1. The list of numbers therefore featured almost no entropy. By reordering the numbers, a significant amount of entropy was introduced, and the all too obvious link between index and value was removed from the program.

RNC+ArrayReordering (RNC+Reordering): combination of the aforementioned techniques, to protect either the sensitive assets and the side channel (array indexes) that can be used to circumvent obfuscation.

Dynamic Residue Number Coding (DynRNC): novel code obfuscation technique developed in the project. Numeric values are protected by encoding them as a tuple of integers computed as the respective residues over a tuple of moduli. Moduli are computed at run-time with a function that is hard to analyse statically, because it is based on an NP-complete problem. This obfuscation has been elaborated by ASPIRE and is described in detail in Deliverable D2.08.

This experiment allows for a direct measurement of the reduced success rate/increased attack effort associated with the novel protection elaborated by the ASPIRE project. The reference is not just the clear code, but also state-of-the-art data obfuscation techniques, potentially combined. This experiment has been conducted at Universiteit Gent. As part of the course, the students had access to IDA Pro 6.1. They were also given access to gdb and OllyDbg, LordPE, and any tool they wanted to bring or install themselves.
The experiment aims at answering the following three research questions:

- **RQ1**: How effective are data obfuscations RNC/AR/RNC+AR/DynRNC in protecting data inside the code as compared to the clear code?
- **RQ2**: What is the effort required to complete a successful attack on data obfuscated with RNC/AR/RNC+AR/DynRNC as compared to clear code?
- **RQ3**: What strategies and tools have been used to complete successful attacks?

The first research question deals with the effectiveness of data obfuscation protection, while the second deals with the effort required to conduct a successful attack. The third research question investigates the techniques and tools used to complete a successful attack. Data obfuscation might result in a higher time to successfully complete an attack task or might result in the failure to complete the attack task when this is conducted on protected code. There is an economic advantage in adopting a data obfuscation protection if the probability that an attack is successfully completed within a limited amount of time is drastically reduced by the protection.

5.2 Context: Systems

The content of this section is copied from Section 7.2 of Deliverable D4.03
The *objects* of this experiment are two C programs, Lotto and Lottery, obtained from the web and developed by third parties who were not involved in the preparation of the experiment.
The *tasks* of this experiment are as follows: *Lotto:* Determine the JACKPOT sequence, consisting of 7 winning numbers, which is embedded in the binary code (specifically, inside some variables) of the Lotto program. While executing the task, subjects are allowed to read, debug and execute the code. Participants are asked to write down the winning sequence on paper. *Lottery:* Modify the program Lottery so that it extracts only numbers between 1 and 20 in a legal extraction (i.e., one matching the logged challenge). When any of the 7 extracted numbers is greater than 20, the extraction should be redone, by requesting a new challenge to the remote server. In fact, the extracted numbers are checked for validity against the challenge stored in the remote server. After successfully completing the task, the reported frequencies shall be equal to zero for all numbers greater than 20.

5.3 Hypothesis Formulation and Variable Selection

The content of this section is mostly copied from Section 7.3 of Deliverable D4.03.
The two metrics collected to answer research questions RQ1 and RQ2 are:

SR (Success Rate): Proportion of attack tasks completed successfully by participants.

AT (Attack Time): Time needed to complete the attack task, measured only for successfully completed tasks.

Participants are asked to mark down the start and end time when starting and after finishing the attack task, so one key metric collected during the experiment is the attack time (AT). Participants are also asked to show their attack, in case they deem it successful, to one of the research assistants available during the experiment, who verified if the attack was indeed successful or not. Students could spend three hours on the task (corresponding to two consecutive lab sessions in the course schedule).

The success of each attack task on binary code was assessed by one of the course's teaching assistants directly during the laboratory session, by looking at the modifications operated on the execution of the programs by means of the debugger.

Based on the metrics chosen to quantify the effectiveness of the protections, we can formulate the following null hypotheses, respectively for research questions RQ1 and RQ2:

- **H0**$_S$: There is no difference in average SR between participants working on obfuscated and participants working on clear code;
- **H0**$_T$: There is no difference in average AT between participants working on obfuscated code and participants working on clear code.

No null hypothesis is formulated for RQ3, because it will be addressed by asking direct questions to participants.

The main factor of the experiment—that acts as independent variable—is the treatment (clear vs. obfuscated code). As shown in Table 2, in the experiment the five alternative treatments are: (i) clear binary code, (ii) arrays obfuscated with array reordering, (iii) binary code obfuscated with RNC, (iv) binary code obfuscated with RNC and array reordering; and (v) binary code obfuscated with dynamic RNC.

In addition to the metrics SR and AT, we asked participants to answer a pre-questionnaire and a post-questionnaire. The pre-questionnaire collects information about the abilities and experience of the involved participants. This is important to analyse the effect of ability and experience in the successful completion of attack tasks, either on clear or on obfuscated code. The post-questionnaire collects information about clarity and difficulty of the task, availability of sufficient time to complete it, and about the tools used and the activities carried out to complete the task.

Among the co-factors that can potentially affect the results, we identified and measured the following ones:

- *Experience in C*: self reported in the profiling questionnaire (*pre1*);
- *Experience in Assembly*: self reported in the profiling questionnaire (*pre2*);
- *Experience in Reverse Engineering*: self reported in the profiling questionnaire (*pre3* and *pre4*);
- *The Lab*: whether there is a learning effect across subsequent experiment laboratories.

5.4 Design

The design of this experiment is balanced so as to ensure that each participant works both on clear code/code with reordered arrays and on obfuscated code, and to ensure at the same time that each participant works on different systems during different labs.

The bottom of Table 2 reports the balanced design of the experimental sessions. Participants are divided into four groups. However, participants work on their own, without any collaboration within the group. Systems (P1 = Lottery; P2 = Lotto) are provided as clear code (T0), code with arrays reordered (T1) or code obfuscated by RNC/RNC+AR/DynRNC (T2/T3/T4). In the two consecutive laboratories sessions (Lab1, Lab2), each group of participants receives both a different system and a different treatment.

Table 3: Summary of the variables.

Variable	Description	Scale	Range
Treatment	Type of obfuscation	Nominal	$\in$ { Clear, AR, RNC, RNC+AR, DynRNC}
AS	Attack success	Nominal	$\in \{ 0, 1 \}$
AT	Time to complete attack	Ratio	≥ 0
Lab	Lab session of the task	Ordinal	$\in \{ 1, 2 \}$
Pre1	Experience in C	Ordinal	$\in$ { <3 Months, 6 Months, 1 Year, 2 Years, >3 Years }
Pre2	Experience in Assembly	Ordinal	$\in$ { <3 Months, 6 Months, 1 Year, 2 Years, >3 Years }
Pre3	Previous experience in Reverse Engineering	Nominal	$\in$ { Yes, No }
Pre4	Experience in Reverse Engineering	Ordinal	$\in$ { <3 Months, 6 Months, 1 Year, 2 Years, >3 Years }

5.5 Experimental Procedure

As the students had already received classes and labs on reverse engineering, no additional warm-up exercises were necessary.
All students were informed that they were not evaluated on their performance in doing the experiment, they were only rewarded for their participation (as they were for other activities in the course).
Right before the experiment, we provided participants with a detailed explanation of the tasks to be performed during the lab. A reference was made to the study hypotheses: the students knew they were participating in an experiment on analysing the strength of data obfuscations, and they had studied the overview of such obfuscations in ASPIRE deliverable D2.01.
We distributed the following material to our participants:

- A short textual documentation of the system they had to attack, including how to run it;
- The compiled code to attack, either clear or obfuscated depending on the group the participant belonged to (see Table 2);
- A textual description of the goal to achieve with the attack task.

The binary code was compiled for Windows with the Visual Studio compiler with optimisations for speed and size maximally enabled, favouring size over speed (to avoid that divisions are converted to hard-to-interpret multiplications).
The experiment was carried out according to the following procedure. Participants had to:

1. Fill a profiling questionnaire.
2. Read the application description.
3. Run the application (Lotto or Lottery) to familiarise with it.
4. Read the task; mark the start time; perform the task; mark the stop time.
5. After completing all tasks, show the attack to the experimenter by manipulating an actual execution with the help of the debugger.

6. Complete a post-experiment survey questionnaire.

During the experiment, teaching assistants and professors were in the laboratory to prevent collaboration among participants, and to check that participants properly followed the experimental procedure.
Before the experiment, participants were required to fill a profiling questionnaire. Profiling questions deal with the participants' experience with C, e.g. as professional programmers, with debuggers, with Assembly and its reverse engineering.
After the experiment, participants were required to fill a post-experiment survey questionnaire, aimed at both gaining insights about the participants' behaviour during the experiment and finding justifications for the quantitative results. The survey includes questions on the clarity of the task and on the time granted for the task, the task easiness, the tools used, the actions undertaken to attack the code and the time devoted to the attack task.

5.6 Analysis Procedure

Most of the content of this section is copied from Section 7.6 of Deliverable D4.03
The difference between the output variable (SR and AT) obtained under different treatments (clear code vs. obfuscated code) is tested using non-parametric statistical tests, assuming significance at a 95% confidence level (α=0.05). So, we reject the null-hypotheses when p-value<0.05. All the data processing is performed using the R statistical package [4].
To analyse whether data obfuscation reduces the success rate of attack tasks, we used a test on categorical data, because the tasks can be either correct (completed successfully) or incorrect (completed unsuccessfully). In particular, we used Fisher's exact test [2], which is more accurate than the χ^2 test for small sample sizes, another possible alternative to test the presence of differences in categorical data. The same analysis was used in previous empirical works [5].
To be conservative (because of the small sample size and non-normality of the data), a non-parametric test has been used to test the hypotheses related to differences in the attack time. In particular, we perform the one-tailed Mann-Whitney U test on all samples [6]. Such a test allows to check whether differences exhibited by participants under different treatments (clear and obfuscated code) over the two labs are significant.
Since multiple obfuscations have been used, we are interested in comparing pairwise all the cases (i.e., against clear code, but also direct comparison between obfuscations). However, when multiple pairwise comparisons are performed with overlapping data, the number of hypotheses in a test increases and so does the likelihood of witnessing a rare event. Hence, the chance to reject true null hypotheses may also increase (type I error). To control this problem, we adopt the *Holm* correction which is more complex but also more powerful than the *Bonferroni* correction. The *Holm* correction consists of using different significance levels on different tests. P-values from the n dependent hypotheses are sorted in ascending order. Then, on each ordered $p\text{-}value_i$, a decreasing correction factor $n - i + 1$ is used, i.e., an increasing significance level $\alpha/(n - i + 1)$ is applied. We reject the null hypotheses until the minimum index k for which the null hypothesis cannot be rejected is encountered ($p\text{-}value_k > \alpha/(n - k + 1)$). All subsequent hypotheses cannot be rejected ($p\text{-}value_i : i > k$).
While these tests allow for checking the presence of significant differences, they do not provide any information about the magnitude of such a difference. This is particularly relevant in our study, since we are interested in investigating to what extent the use of obfuscation reduces the likelihood of completing an attack and increases the time needed for an attack. As such, two kinds of effect size measures have been used, the *odds ratio* for the categorical variable SR and the Cliff's delta effect size [3] for the AT variable. The effect size is computed using the `effsize` package [7].
The odds ratio is a measure of effect size that can be used for dichotomous categorical data. An odds indicates how likely it is that an event will occur as opposed to it not occurring [6]. The odds ratio is defined as the ratio of the odds of an event occurring in one group (e.g., experimental group) to the odds of it occurring in another group (e.g., control group), or to a sample-based

estimate of that ratio. If the probabilities of the event in each of the groups are indicated as p (experimental group) and q (control group), then the odds ratio is defined as:

$$OR = \frac{p/(1-p)}{q/(1-q)} \tag{1}$$

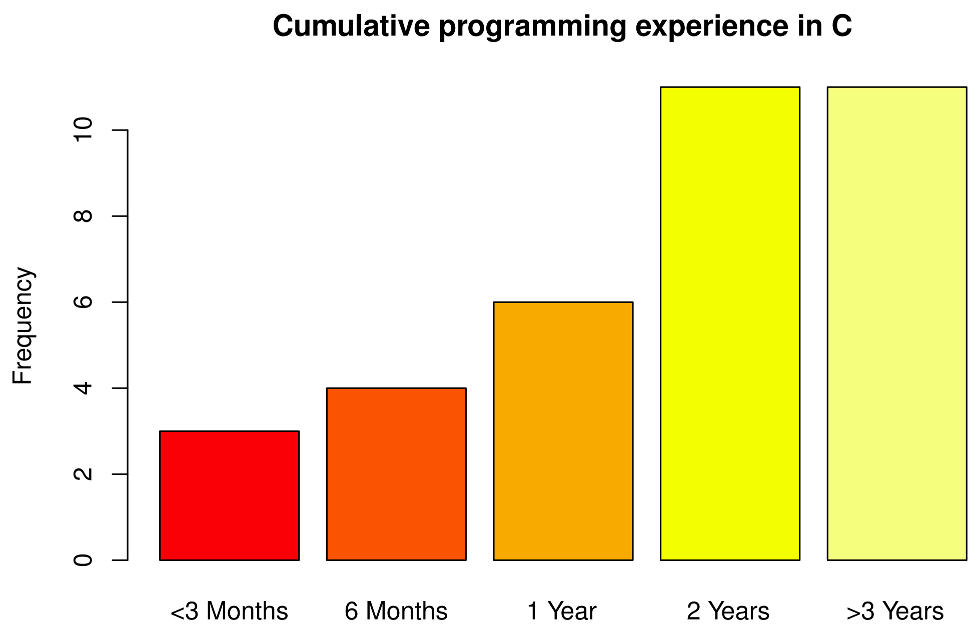
An odds ratio of 1 indicates that the condition or event under study is equally likely in both groups. An odds ratio greater than 1 indicates that the condition or event is more likely in the first group. An odds ratio less than 1 indicates that the condition or event is less likely in the first group.
For independent samples, Cliff's delta provides an indication of the extent to which two (ordered) data sets overlap, i.e., it is based on the same principles of the Mann-Whitney test. Cliff's Delta ranges in the interval $[-1, 1]$. It is equal to $+1$ when all values of one group are higher than the values of the other group and -1 when the opposite is true. Two overlapping distributions would have a Cliff's Delta equal to zero. An effect size d is considered small when $0.148 \leq d < 0.33$, medium when $0.33 \leq d < 0.474$ and large when $d \geq 0.474$ [1].
The analysis of co-factors is performed using a General Linear Model (GLM). It consists of fitting a linear model of the *dependent* output variables (success rate or attack time) as a function of the *independent* input variables (all factors, including the treatment, i.e., the presence of obfuscation). A general linear model allows to test the statistical significance of the influence of all factors on the attack success rate and attack time. In case of relevant factors, interpretations are formulated by visualising the associated interaction plots.
The analysis is conducted on the main co-factors (reported in Table 3) and reports the individual coefficients and their significance level. The coefficients are used to understand magnitude and direction of the effects.
The rest of this section reports the results of the experiment, with the aim of answering the research questions elaborated previously. Since different treatments have been applied to program Lotto and Lottery, statistical analysis is conducted separately on these two programs.

5.7 Participants Characterization

The *Participants* to the experiment are 32 Master and 2 Bachelor students from Universiteit Gent. They take part in the experiment as part of the "Software hacking and protection course", an elective course at Universiteit Gent. Figure 5 and Figure 6 show some descriptive statistics about the participants. These data were collected by means of the pre-experiment questionnaire.
The majority of the participants declared two or more years of experience with the C programming language. Their experience with the x86 assembly language was a bit shorter; still, most of them have experience reading assembly for at least one year. Before participating to the course, most students had no experience in reverse engineering. For those who had some prior experience, it was mostly less than 3 months. During the course, however, all students had already studied static and dynamic reverse engineering techniques, as well as applied them during hands-on labs. Moreover, the students had read the data obfuscation techniques overview from deliverable D2.01.

5.8 Analysis of Success Rate for program Lotto

Figure 7 shows the bar plots of success rate for the attack tasks, divided by treatment (clear code, code obfuscated with RNC/DynRNC). Successful and failed tasks are represented, respectively, in red and in yellow. Bars for different treatments have different height because different treatments have been executed a different number of times (the graph reports absolute values).
From the plot, we can observe that the success rate on clear code is always higher than the success rate on obfuscated code. Moreover, the success rate is higher on static RNC than on Dynamic RNC.



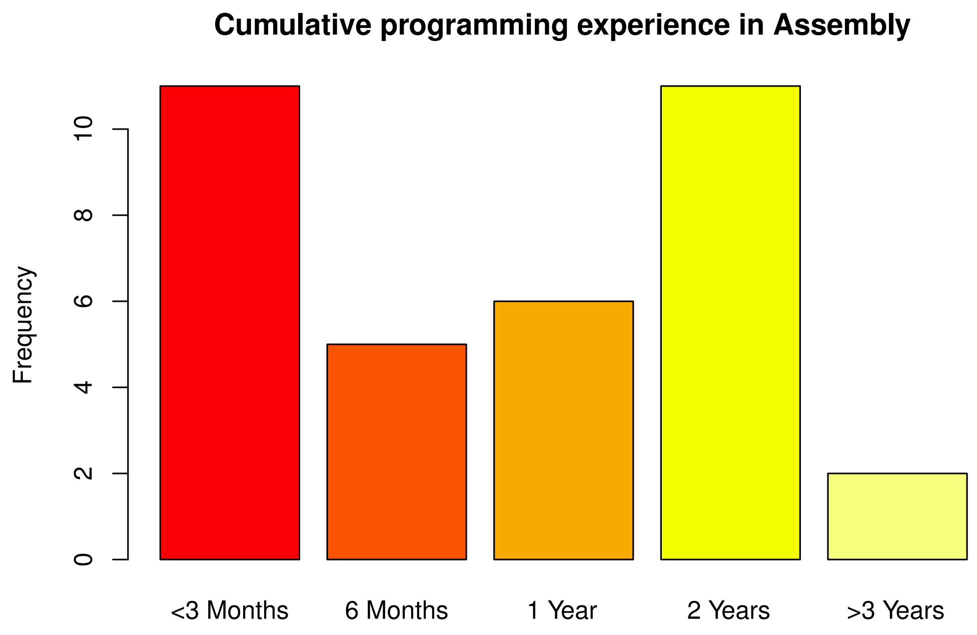
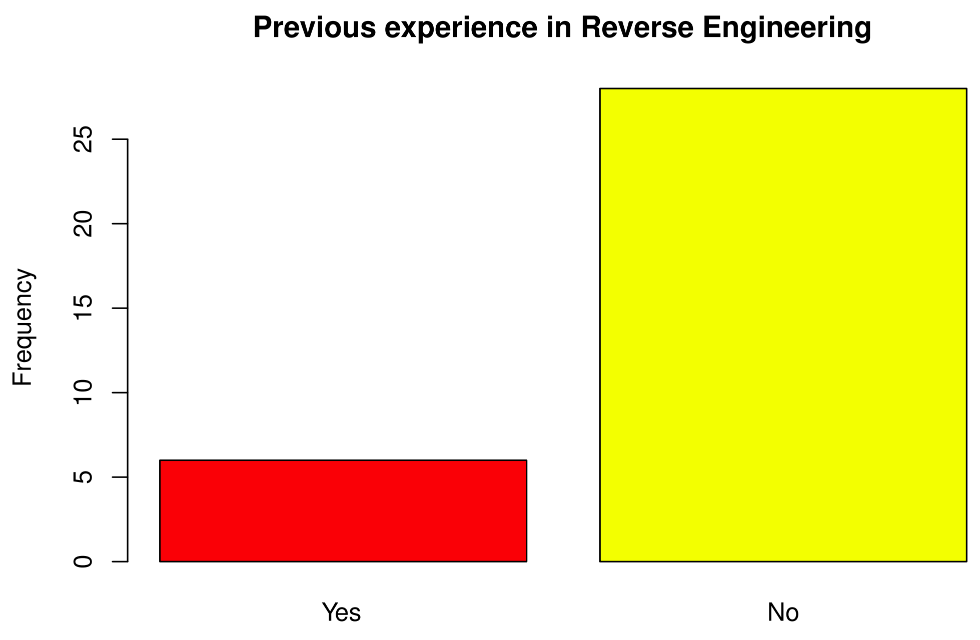


Figure 5: Demographics of participants (a)



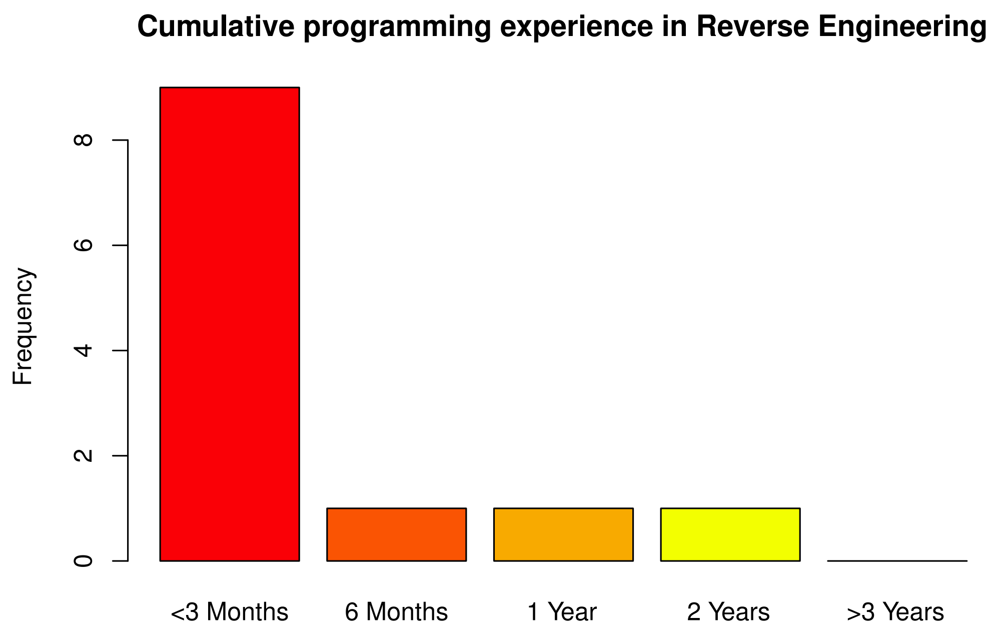


Figure 6: Demographics of participants (b) – reverse engineering experience does not include knowledge acquired during the course.

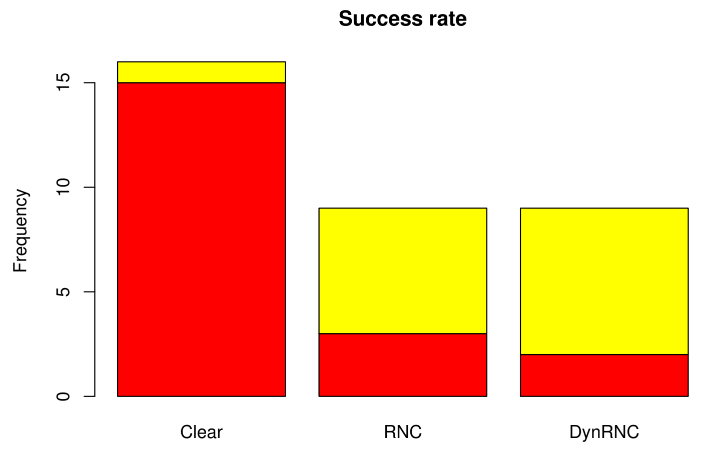


Figure 7: Bar plot of attack success rate (red=successful attack, yellow=failed attack)

	Clear	RNC	DynRNC
Success	15	3	2
Failure	1	6	7

Table 4: Success rate

Table 4 reports the precise number of successful and failed attack tasks for each treatment. Table 5 and Table 6 show the analysis of success rate for the attack tasks performed by participants, reporting the p-value resulting from Fisher's test and the effect size, computed as the odds ratio. An odds ratio > 1 indicates that the chances of success are higher with the first treatment than with the second one. Considering the Holm correction, significant cases (shown in bold face) are when p-values $< 0.05/k$. This holds for the smallest p-value (p-value$_1 < 0.05/3$) and the second smallest p-value (p-value$_2 < 0.05/2$).
The lack of significance in the remaining case (RNC vs. DynRNC) might be due to the presence of multiple treatments, which caused a reduction of participants in each group, thus reducing the number of observations per treatment.
These results suggest that the success rate of attacks on code obfuscated with RNC/DynRNC is significantly lower than the success rate of attacks on clear code. On the other hand, no statistical difference is observed in the direct comparison between obfuscations, because of insufficient statistical power of the sampled data. The output of the tool GPower[2] run on the sampled data confirms that their statistical power is low (less than the commonly accepted threshold $1-\beta = 0.8$). The output of GPower corroborates the hypothesis that significance could not be obtained due to low statistical power of the sampled data.
The odds ratio for RNC/DynRNC indicates that the chances of correctly porting an attack task when code is obfuscated with RNC/Dynamic RNC is more than 24 (resp. 39) times lower than on clear code. Even if the difference is not statistically significant, we can observe that, according to odds ratio, the dynamic variant of RNC reduces that chances of porting a successful attack by a factor 1.7 with respect to the state-of-the-art variant, (static) RNC.
Overall, the null hypothesis $H0_{SR}$ on success rate can be rejected using Fisher's test for the comparison with clear code in two cases: for the RNC and for Dynamic RNC.
Therefore, we can formulate the following alternative hypothesis:

- $H_{SR,RNC}$: The success rate of attackers working on code obfuscated with RNC is lower than

[2]http://www.gpower.hhu.de/en.html

Treatments	Clear	RNC	DynRNC
Clear		**0.0029**	**0.0005**
RNC			1.0000

Table 5: Analysis of success rate (Fisher test)

Treatments	Clear	RNC	DynRNC
Clear		24.55	39.59
RNC			1.70

Table 6: Analysis of success rate (Odds ratio)

the success rate of attackers working on clear code.

- $H_{SR,DynRNC}$: The success rate of attackers working on code obfuscated with Dynamic RNC is lower than the success rate of attackers working on clear code.

5.9 Analysis of Success Rate for program Lottery

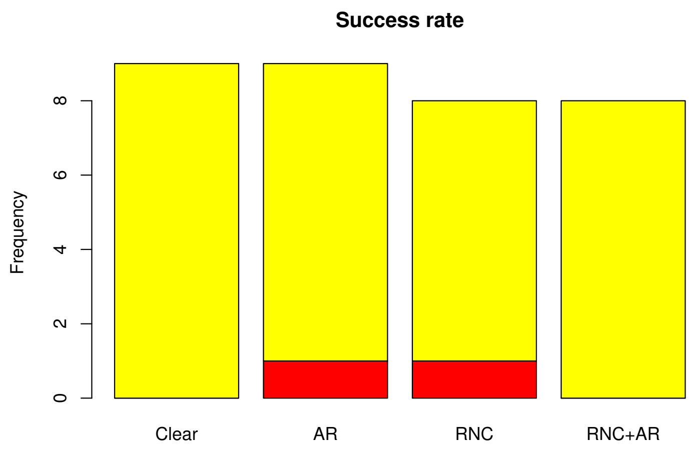


Figure 8: Bar plot of attack success rate (red=successful attack, yellow=failed attack)

Figure 8 shows the bar plots of success rate for the attack tasks, divided by treatment (clear code, code with arrays reordered, code obfuscated with RNC/RNC+AR). Successful and failed tasks are represented, respectively, in red and in yellow. Bars for different treatments have different height because different treatments have been executed a different number of times (the graph reports absolute values).
From the plot, we can observe that the success rate is almost zero across treatments. This clearly indicates that program Lottery is remarkably more complex than program Lotto, even when provided as clear code. No participant could complete a successful attack when the code was clear or was obfuscated with a combination of RNC and AR. Only 1 subject could complete a successful attack in the other two cases.
Table 7 reports the precise number of successful and failed attack tasks for each treatment. Table 8 shows the analysis of success rate for the attack tasks performed by participants, reporting the p-value resulting from Fisher's test. Even without considering the Holm correction, there are no significant cases (p-value < 0.05).

	Clear	AR	RNC	RNC+AR
Success	0	1	1	0
Failure	9	8	7	8

Table 7: Success rate

Treatments	Clear	AR	RNC	RNC+AR
Clear		1.0000	0.4706	1.0000
AR			1.0000	1.0000
RNC				1.0000

Table 8: Analysis of success rate (Fisher test)

The lack of statistical significance of results is due to the presence of multiple treatments, which caused a reduction of participants in each group, thus reducing the number of observations per treatment. Since there are no statistically significant results, the odds ratio is not analysed for this program. No statistical difference is observed in the direct comparisons because of insufficient statistical power of the sampled data. The output of the tool GPower[3] run on the sampled data confirms that their statistical power is low (less than the commonly accepted threshold $1-\beta = 0.8$). The output of GPower corroborates the hypothesis that significance could not be obtained due to low statistical power of the sampled data.
The null hypothesis $H0_{SR}$ on success rate cannot be rejected because of low statistical power of the collected data. Although there is no statistical power to reject $H0_{SR}$, the results of this experiment are still useful for ASPIRE. In fact, this experiment shows that RNC combined with Array Reordering could successfully defeat all attack attempts. This is not true of RNC alone, because array indexes leak information about the array content, even if the array content is protected by means of RNC. Actually, one participant was able to successfully complete an attack task against RNC. On the other hand, AR alone is not a very powerful protection (it was also defeated by one participant), since it just scrambles the array content without actually obfuscating it. Once combined, the two approaches eliminate the possibility to obtain information about the array content without actually attempting to de-obfuscate them, which turned out to be a very difficult task that no participant was able to complete successfully.

5.10 Co-factors for Success Rate

Here we report the analysis of co-factors that could have influenced the success rate. Table 9 and Table 10 summarise the questions asked in the pre-questionnaire and post-questionnaire respectively.
Table 11 reports the analysis of co-factors obtained by applying the general linear model method. Statistically significant coefficients are in boldface. The *Lab* co-factor significantly influences the success rate. The significantly non zero value for the intercept of the linear model (first row of Table 11) indicates that the two classes out output (Success vs. Failure) are not equally likely, hence one of the two (i.e., Failure) can be predicted even without any knowledge of co-factors.
From the interaction plot in Figure 9 we can notice that in the second lab the success rate is consistently lower than in the first lab (the trends for AR, RNC+AR, DynRNC are missing, because these obfuscations were not used in both labs). This trend goes in the opposite direction of the learning effect, which should make participants achieve higher success rate in the second lab. This can be explained considering that different systems were used in different labs (Lotto in the first Lab and Lottery in the second lab) and a program can be intrinsically more difficult to attack than the other. Lottery was actually reported by the involved subjects as substantially more difficult to

[3]http://www.gpower.hhu.de/en.html

Question	Question	Answers
Pre1	What is your cumulative programming experience in C?	$<$ 3 Months; 6 Months; 1 Year; 2 Years; $>$ 3 Years
Pre2	What is your cumulative programming experience in Assembly?	$<$ 3 Months; 6 Months; 1 Year; 2 Years; $>$ 3 Years
Pre3	Did you have any experience with Reverse Engineering before this course?	Yes; No
Pre4	If so, what is your cumulative programming experience in Reverse Engineering?	$<$ 3 Months; 6 Months; 1 Year; 2 Years; $>$ 3 Years
Pre4	If so, which tools do you have prior experience with outside of this course?	

Table 9: Overview of the pre-questionnaire questions

Question	Question	Answers
Post1	The task was clear to me.	Strongly agree; Agree; Not certain; Disagree; Strongly disagree
Post2	There was enough time to perform the task.	Strongly agree; Agree; Not certain; Disagree; Strongly disagree
Post3	The task was easy to perform.	Strongly agree; Agree; Not certain; Disagree; Strongly disagree
Post4	Which tools did you use in trying to solve this challenge?	
Post5	What is the specific sequence of activities that you performed?	
Post6	Can you order these activities by difficulty (from the more difficult to the more easy)?	
Post7	Can you order these activities by time consumed (from the one that required more time to the one that required less time)?	
Post8	What protections, if any, do you think it was applied to the program? Did identifying them help you?	

Table 10: Overview of the post-questionnaire questions

attack than Lotto.

Another important factor that could have influenced the success rate is the background knowledge of each participant. We collected participant background information through the profile questionnaire. However, the profile of the participants (experience in C, Assembly and Reverse Engineering) has no significant influence on the success rate. Participants involved in this ex-

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2.3912	0.9064	2.64	**0.0205**
Clear	-0.2666	0.3362	-0.79	0.4420
DynRNC	-1.2615	0.4848	-2.60	**0.0219**
RNC	-0.4091	0.2621	-1.56	0.1425
RNC+AR	-0.3441	0.2889	-1.19	0.2549
Lab	-0.8930	0.2103	-4.25	**0.0010**
Pre1 (C experience)	-0.0153	0.1019	-0.15	0.8833
Pre2 (Assembly experience)	0.0471	0.0891	0.53	0.6061
Pre3 (Previous RE experience)	-0.1090	0.2065	-0.53	0.6065
Pre4 (RE experience)	-0.1308	0.1178	-1.11	0.2872

Table 11: Success rate co-factors (General linear model).

periment have all a similar background, as apparent from the pre-questionnaires, and they all attended the course "Software hacking and protection" at Universiteit Gent. As a consequence, the involved students were quite homogeneous in terms of binary code analysis competences and their background did not influence their performance in the experiment.

5.11 Analysis of Attack Time for Lotto

We now analyse the effort required to port a successful attack. Since we consider how long it took to complete a *successful* attack task, we drop those tasks that were not successfully completed. Moreover, since only two attacks (associated with 2 out of 4 treatments applied) were successful on program Lottery, there is not enough information to carry out statistical tests on this program. Hence, analysis of attack time is conducted only on Lotto.
Figure 10 shows the distribution of the attack time required to deliver correct results when the attacked code is clear, obfuscated with RNC or obfuscated with Dynamic RNC. From the plot, we can observe that obfuscated code consistently requires more time to be attacked than clear code and that to attack the dynamic variant of RNC more time is required than for the static variant.

Treatment	N	Mean	Median	SD (σ)
Clear	15	61.00	52.00	24.80
RNC	3	122.00	125.00	34.60
DynRNC	2	148.50	148.50	14.85

Table 12: Descriptive statistics of Attack Time.

Table 12 reports the descriptive statistics of the time required to port successful attacks, respectively on clear and obfuscated code. The table reports the number of successful attacks (second column), mean, median and standard deviations of attack time (respectively on third, fourth and fifth columns).

Treatments	RNC	DynRNC
Clear	0.0176	0.0438
RNC		0.4000

Table 13: Analysis of attack time (Mann-Whitney U test).

Table 13 and Table 14 report analysis of the time required to port successful attacks, respectively as the p-value (significant cases in boldface) of the Mann-Whitney U test with Holm correction and Cliff's delta effect size.

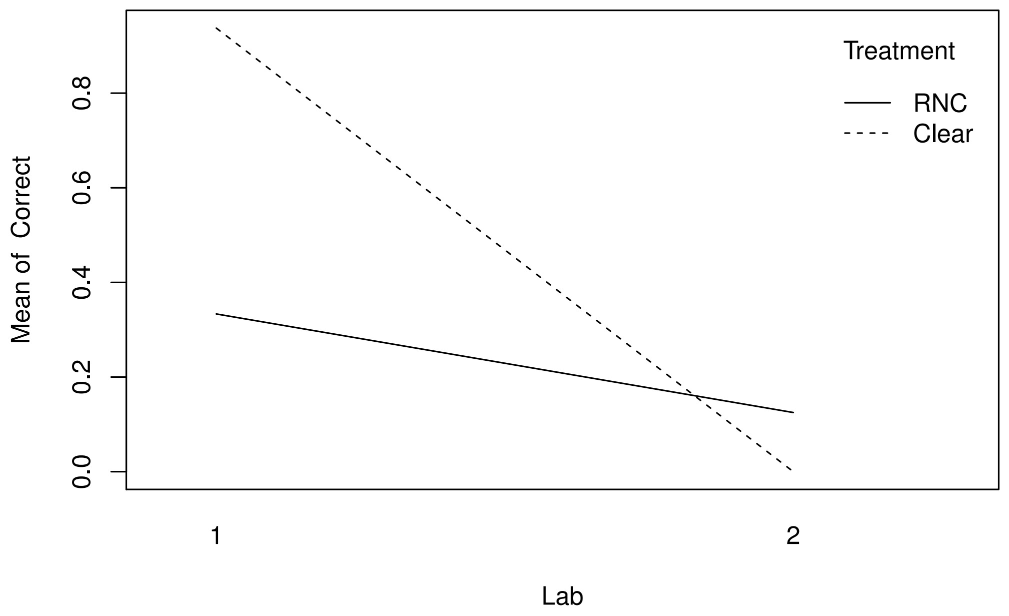


Figure 9: Interaction plot of Success Rate and Lab.

Treatments	RNC	DynRNC
Clear	-2.03	-4.28
RNC	-	-1.00

Table 14: Analysis of attack time (Effect size).

Although statistical significance is not achieved in the comparison between clear code and code obfuscated with RNC and DynRNC when the Holm correction is applied, p-values are quite low (p-value < 0.05 for both RNC and DynRNC) and are associated with a large effect size. We conjecture that a larger sample size would have provide statistical significance even after the application of the Holm correction. Power analysis conducted by means of the GPower tool confirmed that the statistical power of the data collected for RNC vs. Dynamic RNC is low ($1 - \beta < 0.8$). For similar reasons, the attack time difference visible in Figure 10 does not reach statistical significance.
Overall, hypothesis $H0_{AT}$ cannot be rejected using the Mann-Whitney U test with Holm correction when RNC/DynRNC obfuscation is compared with clear code. It can be rejected if Holm correction is not applied. Therefore, the following alternative hypothesis requires further validation, by collection of a larger sample of data:

- $H_{AT,RNC/DynRNC}$: The amount of time required to attack code obfuscated with RNC/DynRNC is higher than the time required to attack clear code.

Lack of the large sample size necessary to achieve statistical significance when comparing attack times is mostly due to the small number of successful attacks (only 3 on RNC and 2 on DynRNC, as compared to 15 on clear code). In fact, in this experiment the primary metrics is success rate and on this metrics there is indeed statistical significance of the differences with large effect size (see Section 5.8). Attack time is a secondary metrics that can be computed only on the subset of tasks completed successfully. Since the experimented protections were quite difficult to defeat for attackers, only a small sample of attack times could be collected. Hence, the lack of significance (in the presence of Holm correction) for $H_{AT,RNC/DynRNC}$ simply means that protections obstructed

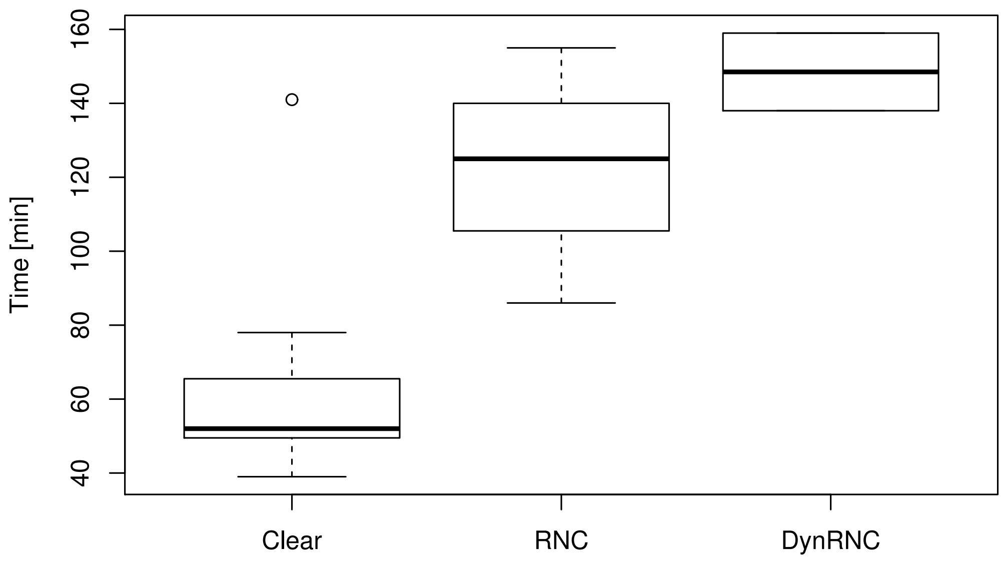


Figure 10: Boxplot of attack time (only for successful attacks)

most of the attack attempts, leaving us with a few attack time values for the statistical analysis.

5.12 Co-factors for Attack Time

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	169.9677	28.2646	6.01	**0.0266**
Clear	-60.8871	10.5604	-5.77	**0.0288**
RNC	-65.9677	10.7396	-6.14	**0.0255**
Pre1 (C experience)	-12.7097	5.9682	-2.13	0.1670
Pre2 (Assembly experience)	12.7419	4.5115	2.82	0.1058
Pre3 (Previous RE experience)	-31.6129	9.9185	-3.19	0.0859
Pre4 (RE experience)	13.1129	5.7141	2.29	0.1487

Table 15: Attack Time co-factors (general linear model).

Similarly to what we did for success rate, we also analyse the impact of co-factors on attack time. Table 15 shows the analysis of co-factors with general linear model. Treatments (RNC, Clear) have an impact, which is significant only if no Holm correction is applied (consistently with the Mann-Withney U test reported in Table 13). The statistically significant intercept indicates that one of the two outputs (namely, attack task Failure) can be predicted even without knowledge of any co-factor just because it is associated with a class that contains more samples.
Similarly to the case of success rate, also for attack time, the profile of the participants (experience in C, Assembly and Reverse Engineering) collected with the profiling pre-questionnaire has no significant influence on the result.

Treatment	Post1: Clear	Post2: Enough time	Post3: Easy
Clear	1 (Strongly agree)	2 (Agree)	2 (Agree)
AR	2 (Agree)	3.5 (Not Certain/Disagree)	4 (Disagree)
RNC	1 (Strongly agree)	3.5 (Not Certain/Disagree)	4 (Disagree)
RNC+AR	2 (Agree)	4 (Disagree)	5 (Strongly disagree)
DynRNC	1 (Strongly agree)	4.5 (Disagree/Strongly disagree)	5 (Strongly disagree)

Table 16: Analysis of feedback post-questionnaire.

5.13 Analysis of post-questionnaire

In this section, we analyse the answers provided by participants to the feedback post-questionnaire. Table 16 reports the median of the answers to questions with closed answers on a Likert scale. Overall, participants *agree* or *strongly agree* that the task was clear. Participants *agree* that the time was enough only when attacking clear code (*post1*). They are either *uncertain* or they *disagree* that the time was enough to attack obfuscated code (*post2*). Participants *agree* that the task was easy (*post3*) only when performed on clear code, they *disagree* that the task was easy on obfuscated code.

Now we are going to analyse the answers to open questions, provided by those participants who were able to attack obfuscated code. The questions are intended to investigate the used tools (*Post4*), the activities performed to complete the attack (*Post5*), the difficulty (*Post6*) and the time spent (*Post7*) on these activities, as well as the protections participants think that had been applied to the code (*Post8*).

RNC: The four participants who managed to attack code protected with RNC reported that they mostly started analysing the code with IDA Pro to perform static analysis and OllyDbg to perform dynamic analysis. In one case, a participant additionally used Python to automate a part of the attack (with a script). Another participant also tried to use a "Chinese remainder calculator" to break the data obfuscation scheme.

Two different strategies have been reported by participants: (1) white-box attack to obfuscation and (2) brute force attack. The first strategy consists of using static and dynamic analysis (with IDA Pro and OllyDbg) to identify the relevant point in the code where data was accessed and decoded. Then, decoding operations were reverse engineered to recover the decoding algorithm. Once known, the decoding algorithm has been used to recover the clear values of the protected assets. While this strategy was successful for a participant, it was not so for the other participant who attempted it. The latter participant, in fact, reported that at a certain point he gave up and switched to the second strategy, adopted in the rest of the successful attacks, i.e., brute force. The time to complete the attack was approximately two times higher for the participant who switched from attack strategy (1) to (2) as compared to the participant who insisted with strategy (1) until he was successful.

According to the answers to the feedback post-questionnaire, brute force attack still starts with static/dynamic analysis to locate the point where data are decoded. Then, several attempts are made with a trial-and-error approach, in which data combinations are enumerated and tried, successful sub-sequences of data are recognised, and the complete sequence is obtained incrementally and greedily, by concatenating the correct sub-sequences.

The hard part was understanding the Chinese Remainder Theorem and the way values are decoded. Brute forcing of values was also reported as a hard activity. In all the cases, the attack tasks started with the easiest tasks and, while proceeding in the attack, more and more difficult tasks have been faced. Not surprisingly, most of the time was spent in difficult tasks where human effort was required.

Three out of four participants realised that Residue Number Coding was used to protect asset values. Even if able eventually to attack the code, the fourth participant did not realise the presence of this specific data obfuscation algorithm. Instead, he wrongly reported the presence of XOR masking and opaque predicates. The presence of additional protections was wrongly reported by two of the other participants who correctly identified RNC.

Dynamic RNC: Just two participants were able to attack the code protected with Dynamic RNC. The used tools are consistent with the previous case: IDA Pro for static analysis and OllyDbg for dynamic analysis. However, the attack strategy was quite different, because participants realised that static analysis was useless, since it was not useful to detect the values of the modules used to decode the protected values. In fact, none could reverse engineer the decoding algorithm nor apply the Chinese Remainder Theorem to complete the attack. In one case, the attack was successful only when a participant resorted to OllyDbg to intercept the print statement and read the clear values of the assets from the memory. In the other case, the attack was a waste of time until the participant decided to switch to a brute force strategy.

Similarly to the previous cases, the steps supported by tools were easy and fast, while the mentally and manual intensive tasks were reported as the most hard and time consuming.

While a participant was able to guess that some form of Residue Number Coding was present in the code, the other participant did not realise this. Instead, he reported that some form of control obfuscation was present, with many complex branches ("tons of unneeded branching"). These branches probably result from the many uses of encoded variables in if-checks. RNC replaces individual comparisons with short-circuiting chains. Anyway, the attack was successful even if the deployed protection was not identified correctly.

5.14 Threats to validity

The content of this section is mostly copied from Section 7.13 of Deliverable D4.03

The main threats to the validity of this experiment belong to the conclusion, internal, construct and external validity threat categories.

Conclusion validity threats concern the relationship between treatment and outcome. We use statistical tests to draw our conclusions. Inability to reject the null hypothesis exposes us to type II errors (incorrectly accepting a false null hypothesis). We mitigate this threat by increasing the number of participants to this study. In fact, the probability of a type II error can be reduced by increasing the sample size. In a few instances, the application of the Holm correction prevented us from achieving statistical significance. While a larger sample size might solve the problem, we think such results are still interesting because they are very close to statistical significance (they are statistically significant without Holm correction).

Internal validity threats concern external factors that may affect the independent variable. Subjects were aware of the experimental hypotheses. Participants were informed that they were not evaluated on their performance in doing the experiment.

Construct validity threats concern the relationship between theory and observation. They are mainly due to how we measure the effectiveness of data obfuscation. We manually assessed the successful completion of each task in order to measure SR. During the labs, experimenters made sure that times were accurately marked in the time sheets upon start and completion of the attack tasks.

External validity concerns the generalisation of the findings. In our experiment we considered two small programs, Lotto and Lottery, to allow for potentially successful attacks within experimental sessions with a limited time bound (3h). Results may not generalise to larger programs. On the other hand, we expect that larger programs will be more difficult, not easier, to attack, both with and without protections. Moreover, the chosen programs have been written by third parties, completely unrelated with ASPIRE, so as to ensure that they are not crafted to provide optimal application conditions for the ASPIRE protections. The possibility to complete a successful attack on these simple programs might not necessarily translate to the fact that a more complex program can be also attacked.

The study was performed in an academic environment which may differ substantially from the industrial one. However, we mitigate this threat by asking participants to work on binary code, a setting similar to a real attack scenario. Results cannot be extrapolated to (i) more advanced attackers, (ii) different types of assets, (iii) different attack goals. Furthermore, the specific form of Array Reordering (see Section 5.1) used in this study differs significantly from the more traditional form as was presented in deliverable D2.01. Results obtained for this specific form cannot be extrapolated to other forms of this protection. Further replications of the study on additional objects, with different assets and attack goals, as well as execution of case studies involving professional hackers, will corroborate the external validity of our findings.

5.15 Lessons Learned

In this section we discuss the implications of the experimental results for the ASPIRE project.
Effects of protections. The protections tested in this experiment had a dramatic impact on success rate and on the time necessary to complete a successful attack. On Lotto, the success rate was reduced by a factor 3.3 and the attack time was more than doubled. This provides quantitative evidence for the shift in the economic convenience of code attacks when attackers deal with protected binaries. The reduced probability of success and the increased effort necessary to complete an attack are powerful deterrents that prevent to a major extent malicious tampering with binary code.
Combination of protections. Attacks can be conducted on multiple, different assets and code regions. By combining multiple protections it is possible to block multiple fronts of attacks at the same time. On Lottery, we evaluated the combination of RNC with Array Reordering and we observed that the combined protection could never be attacked successfully, while the two individual protections, applied in isolation, could be successfully attacked by one participant each. Practical usage of the ASPIRE tool chain should activate as many protections as possible (considering of course their impact on performance).
Attack strategies. Successful attacks were never targeted directly against an ASPIRE protection. In fact, all successful attacks took advantage of side attack channels or brute force. This indicates that careful analysis of the security assets and careful selection of the appropriate protections may eventually inhibit all possibilities of side attacks. If the involved computation is complex enough to prevent brute force attacks, the job of attackers becomes extremely difficult and economically inconvenient.

6 Client/server code splitting experiment

Section authors:
Mariano Ceccato, Paolo Tonella (FBK)

According to the ASPIRE *Description of Work* (DoW), a second round of experiments is planned, where a second protection is assessed. In this section, we present the results of the first replication of the second round of experiments, intended to investigate client/server code splitting. The replication has been conducted with 10 academic participants from University of East London. Due to the small number of participants and to the fact that only two participants could successfully complete the attack task, no significant result is highlighted by statistical tests.
This experimental design will be reused in the forthcoming replications on the other academic project partners, and more participants will be involved. More data points will allow us to draw statistically sound conclusions. The complete set of experiments on code splitting and the new (statistically sound) conclusions will be reported in deliverable D4.06 scheduled for month M36.

6.1 Experimental Definition

The content of this section is mostly copied from the first part of Section 8 of Deliverable D4.03
Table 17 provides a schematic overview of the code splitting experiment. The *goal* of this experiment is to analyse the degree of protection offered by the ASPIRE code splitting technique, when

Goal	Analyze the ability of code splitting to prevent malicious attacks and measure the performance penalty to be paid
Treatments	T0 = original code; T1 = small code portion split; T2 = medium code portion split
RQ1	How effective is code splitting for preventing code tampering as compared to the clear code?
RQ2	How effective is code splitting for increasing the attack time as compared to the attack time for the clear code?
Subjects	Students from FBK, UEL, POLITO
Objects	P1: space game
Tasks	Make spacecraft move faster by doubling the effect of a move
Metrics	Success rate; time to mount a successful attack
Design	Lab1: P1-T0; P1-T1/2

Table 17: Code splitting experiment

it is applied to increasingly larger or more complex code portions. Splitting larger code portions is expected to bring increased protection but also to cause increasing performance degradation. Large portions of code executed on the server may involve a significant server execution overhead, especially when serving many clients at the same time. Moreover, splitting a more complex portion of code may require more frequent synchronisations between the code remaining on the client and the code moved to the trusted server. Because of the implementation of this protection, the more dependencies between client and server, the more synchronizations are required (see Deliverable D3.1 for technical details about this protection). In order to evaluate the impact of the split code size on both the level of protection and the performance penalty to be paid, we consider two treatments, in addition to the baseline T0, which is the original code: T1/T2, associated with a small/medium code portion being moved from the client to the trusted server.
In each replication of the experiment, one of the two split code sizes (T1/T2) is evaluated in comparison with the clear code (T0). This allows for a direct measurement of the increased attack effort associated with each level of protection. Such measurement are paired with the associated performance degradation. Moreover, by comparing the data collected in the three replications where the split code portion has variable size, it is possible to quantitatively assess the trade off between degree of protection and associated performance penalty.
The three replications of this experiment with variable split code size will be conducted at POLITO, FBK and UEL, and will evaluate the treatment (T1/T2) in comparison with the baseline treatment T0 (clear code).

6.2 Research questions

The experiment aims at answering the following two research questions:

- **RQ1**: How effective is code splitting for preventing code tampering as compared to the clear code?
- **RQ2**: How effective is code splitting for increasing the attack time as compared to the attack time for the clear code?
- **RQ3**: What strategies and tools are used to complete successful attacks?

The first research question is about the effectiveness of code splitting as a code protection technique that limits the likelihood for attackers to be able to complete a successful attack. The second question is about the cost for porting a successful attack in terms of human time. The evidence collected during the experiment will be used to assess the average increased attack time provided by code splitting, regardless of the split code size. We will measure also the performance degradation associated with code splitting, depending on the size of the code portion being split, so as to be able to pair the increased attack effort with the increased cost of protection (performance penalty). The last research question is intended to investigate, more qualitatively, how attackers work to complete their task.

6.3 Object

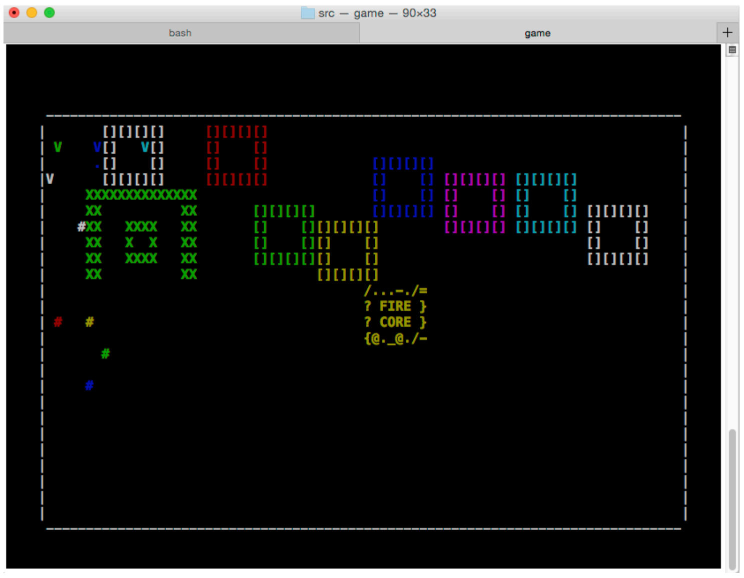


Figure 11: Screenshot of SpaceGame

The object of this experiment is an open source C program, SpaceGame, obtained from SourceForge. SpaceGame is a demonstrator for the framework GAME (Geometrical Ascii Multigame Environment), a C language framework for creating geometrical games using `ncurses` text screens. While GAME was originally created for Unix platforms, it can be ported to more systems, because it uses standard ANSI C. The framework and the demonstrator amount to 1,873 SLoC (measured by `sloccount`), including header files. Players can move their spacecraft on the screen by means of the numeric keyboard or by pressing the keys for characters 'h', 'j', 'k', 'l' ('s' stops the game, while 'q' quits it). Figure 11 shows a screenshot of SpaceGame.
The attack task to be executed by the experiment subjects on SpaceGame aims at gaining an unfair advantage over other competing players:

Attack task: Modify the source code of SpaceGame so as to move twice as fast as allowed by the game rules. Specifically, each key press must translate into a 2-character length move, instead of a 1-character move.

While in the clear code the required modification consists just of changing the unitary increment or decrement of the position into a double increment/decrement, when the code handling player movements is moved from client to server, the modification to be done becomes increasingly more difficult, depending on the split code size. It might for instance involve a double function call that replaces a single call, or a modification of some client-server messages. Maybe, an attacker could also observe the input/output data exchanged with the server, *learn* the behaviour of the split part and reimplement locally the missing features in a fake-server, that could be later tampered with to complete the attack.

6.4 Analysis of Runtime Overhead

The execution time (ET) measures the time for a complete execution of SpaceGame under a predefined interaction scenario. In order to obtain meaningful and comparable execution times, a program driver is used to stimulate the program without requiring any user intervention. The driver executes the program in batch mode and it sends a predefined key sequence to SpaceGame, so as to simulate the interaction of the user with the game. The length of the chosen key sequence was 120 (which is the maximum value for the execution in batch mode), which means that an execution of the driver simulates the user pressing the game keys 120 times. The key-press rate is not relevant for a batch execution, because key-press events are stored in a file and the game consumes exactly one key-press event for each step of the game. To accommodate for random fluctuations in the execution time measurements, ET is measured multiple (100) times.
The performance overhead (PO) is the relative increase of the average execution time between split code and original code:

$$PO = \frac{\overline{ET(P')} - \overline{ET(P)}}{\overline{ET(P)}} \tag{2}$$

where P, P' indicate the original and protected program, respectively.

Version	Average time [sec]	SD (σ)	PO
Original	0.020	0	-
Small	2.159	0.028	107
Medium	2.106	0.048	104

Table 18: Execution times for split code.

Table 18 reports the average execution times (expressed in seconds), the standard deviation and the performance overhead for the three different versions of SpaceGame (original code, small split, medium split). We can observe that the performance overhead for the two split versions is significant. Protected code takes, respectively, 107 times and 104 times longer than the original code. This is meanly due to the interaction between client and trusted server, because applying this protection to SpaceGame means to turn a client-only program into a client-server architecture. Anyway, this overhead was measured in a batch execution, and the performance overhead does not impact negatively the user experience.
The difference between the two split versions is small, but opposed to the slice size. In fact, the small split version is affected by a larger overhead than the medium split version. This is due to the fact that, even if the portion of the code being moved from the client to server is larger for medium split, the total number of synchronization actions required is slightly smaller. Thus, when considering the runtime overhead, network messages are more important than the size of the code to move to the server.

6.5 Metrics

The metrics collected to answer research questions RQ1 and RQ2 are:

AT: Attack time

SR: Success rate

Subjects are asked to mark down the start and end time when starting and after finishing the attack task, so one key metrics collected during the experiment is the attack time (AT). Subjects are also asked to send the attacked code to the experimenters, who manually verify if the attack was successful or not. Metrics AT is meaningful only for successful attacks. The proportion of

successful attacks provides a second metrics, which complements AT, called success rate (SR). SR measures the proportion of subjects that successfully completed the attack task either on the original code or on code protected by code splitting.
Based on the metrics chosen to quantify the effectiveness of the code splitting protection, we can formulate null and alternative hypotheses associated with research questions RQ1, RQ2:

- **H0**$_{SR}$: There is no difference in the average SR between participants working on code protected with code splitting and participants working on clear code;
- **H0**$_{AT}$: There is no difference in the average AT between participants working on code protected with code splitting and participants working on clear code.

In addition to the metrics AT, SR, we ask subjects to answer a pre-questionnaire and a post-questionnaire. The pre-questionnaire collects information about the abilities and experience of the involved subjects. This is very important to analyse the effect of ability and experience in the successful completion of the attack tasks either on original or on split code. The post-questionnaire collects information about clarity and difficulty of the task, availability of sufficient time to complete it, and on the tools used and the activities carried out to complete the task.

6.6 Design

The content of this section is copied from Section 8.4 of Deliverable D4.03
The design consists of a family of three experiments associated with replications aimed at exploring the trade off between increased protection and performance penalty associated with different levels of code splitting. Specifically, two levels of code splitting are applied in the three replications, T1/T2 (with respectively a small/medium code portion being split), while T0 indicates no code splitting at all.

Lab	**P1-T0**	**P1-T′**
Lab1	G1	G2

Table 19: Design for the code splitting experiment: group G1 is assigned object P1 in its original form, while group G2 is assigned P1 protected with code splitting T′ (either of T1/T2), in a single lab (Lab1)

Table 19 shows the design of each experimental session (Lab1). Subjects are divided into two groups, G1 and G2. Object P1 (SpaceGame) is provided as clear code (T0) or split code code (T′). In the latter case, the amount of code splitting varies across replications (T′ = T1 or T2 depending on the replication).

6.7 Statistical analysis

The content of this section is copied from Section 8.5 of Deliverable D4.03
The difference between the output variable (AT, SR) obtained under different treatments (original code vs. split code) is tested using the Wilcoxon non-parametric statistical test, assuming significance at a 95% confidence level (α=0.05); so we reject the null-hypotheses having p-value<0.05.

6.8 Participants Characterization

Figure 12 shows some statistics about the participants involved in the first replication of the experiment, conducted at University of East London. These data were collected by means of the pre-experiment questionnaire. Participants have a quite different background. They are: 1 bachelor student, 4 master students, 2 PhD students and 3 post-doc researches. Half of them have no

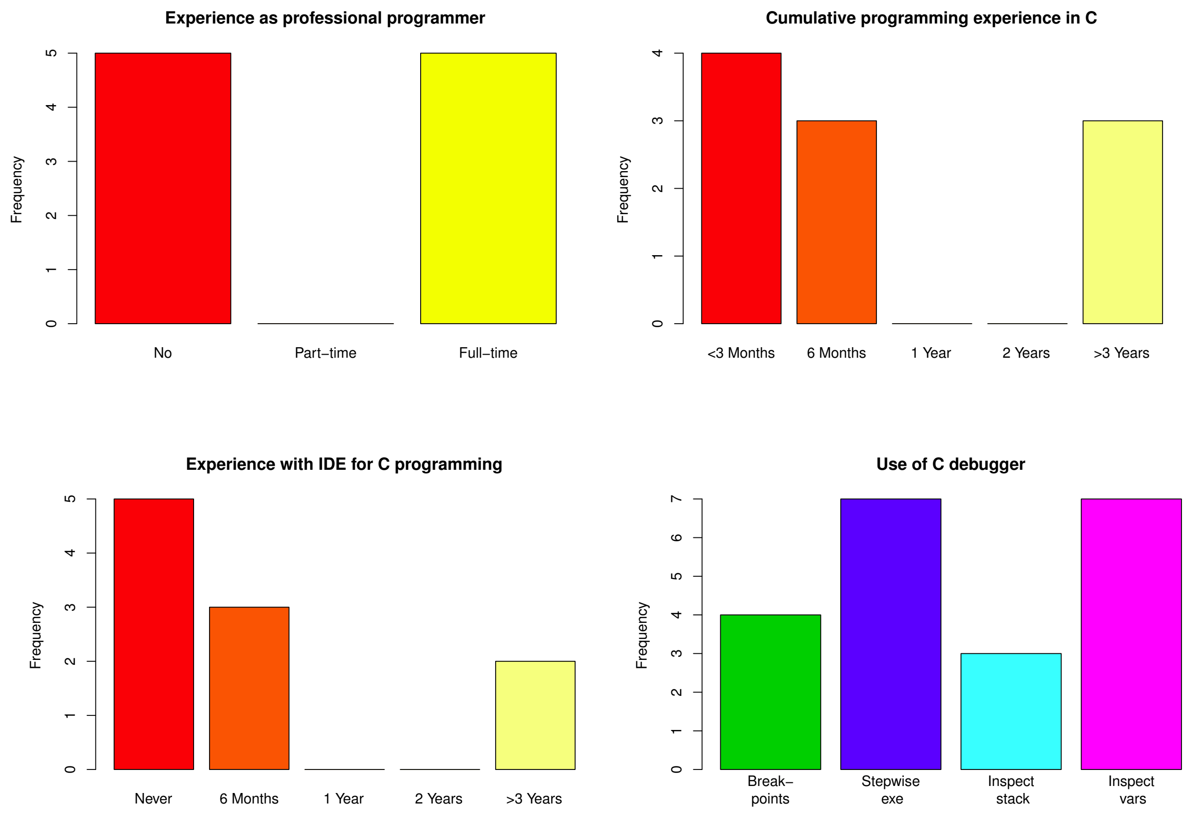


Figure 12: Demographics of participants at UEL

experience as professional programmers, while the other half have experience as full-time professional programmer. Some of them have limited experience in C and in using IDE to program in C, while others have more that three years of experience in C. For this reason, participants from University of East London have been involved in a crash course of C before the experimental lab.

6.9 Analysis of Success Rate

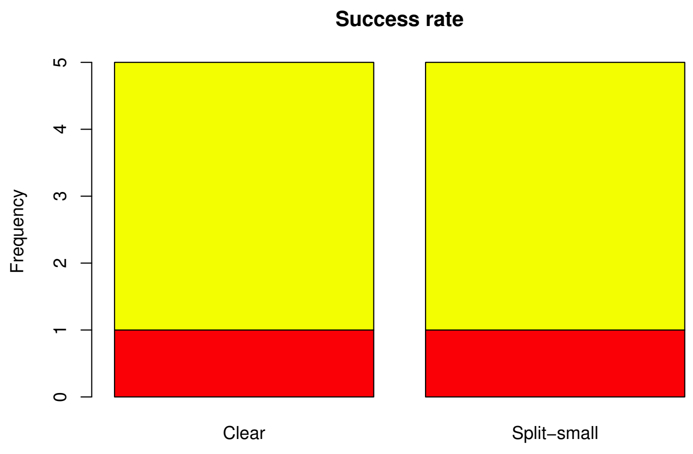


Figure 13: Bar plot of attack success rate

Figure 13 shows the bar plots of success rate of the attack tasks, comparing clear code and code protected with code splitting. From the plot, we can observe that the success rate is quite low. In both cases, only one participant could elaborate a correct attack.
Given the symmetry of success rate between protected and unprotected code, and the low success rate, it does not make sense to apply statistical analysis to these data. All in all, with these data, we can not reject the null hypotheses on success rate.

6.10 Analysis of Attack Time

Treatment	N	Mean	Median	sigma
Clear	1	34	34	-
Split-small	1	47	47	-

Table 20: Descriptive statistics of Attack Time

Now we analyse the time required to elaborate and port a successful attack. Similarly to what done for data obfuscation, here we only consider *successful* attack tasks and we drop tasks that were not successfully completed. Thus, we have just two data points. Table 20 and Figure 14 show the distribution of the attack time required to deliver correct results when the attacked code is clear or protected. From the graph, we can observe that protected code required more time (47 minutes) to be attacked than clear code (34 minutes), however the fact that we observed just two cases does not allow us to use any statistical test and to generalise our findings. More data points on next replications of this study are required to formulate observations.

6.11 Analysis of post-questionnaire

Figure 15 shows the distribution of questions to the feedback post-questionnaire. For many participants the task was clear and the time to complete the task was enough and the task was easy

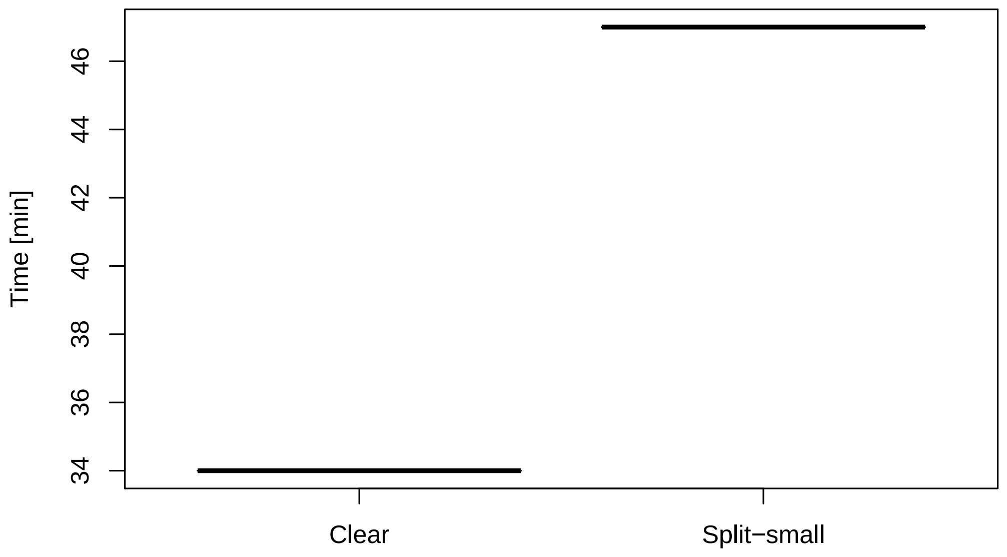


Figure 14: Boxplot of attack time (only for successful attacks)

to perform. However, this is apparently in contrast with the fact that only few participants completed the task successfully. For this reason, a more detailed question is required in the future replications, to ask in detail what would have been the precise problems and difficulties experienced by participants.
To work on the attack task, participants mostly resorted to the *editor*, to the *compiler* and to *internet search*. Eventually, most of the time was devoted to *reading and understanding* the code, and less time was devoted to *changing and executing* the code.
Differently from previous questions, *Post6*, *Post7* and *Post8* are open questions, meant to let the participants freely formulate their answers and describe their attack.
The sequence of activities (*Post6*) performed by the participant who solved the task on clear code was quite simple. After identifying some static locations in the code, the participant added some *printf* statements to do dynamic analysis and to verify the assumptions. The attack strategy for the participant who was successful on the protected code was more complex. Similarly to the other participant, the attack started by detecting the relevant function. The participant realised that some computation was missing, because it was delegated to the split server. The participant, then, attempted many different changes to guess and replicate the feature moved to the server. This represents and attempt to learn the functionality moved to the server, possibly with the intention of replicating it locally. The participant reported that, after wasting a lot of time in useless attempts, eventually the attack was successful.
For the participant working on clear code, the most difficult task (*Post7*) was identifying the point to attack. After that, it was easy to edit the code. On the other hand, for the participant working on the protected code, understanding the missing feature and editing the code were reported as the hardest parts. This result supports our hypothesis that the protection was effective in making the attack's job harder.
Considering where most of time was spent (*Post8*), the two participants agree that the most time consuming task was editing the code and running it to check the results of edits.

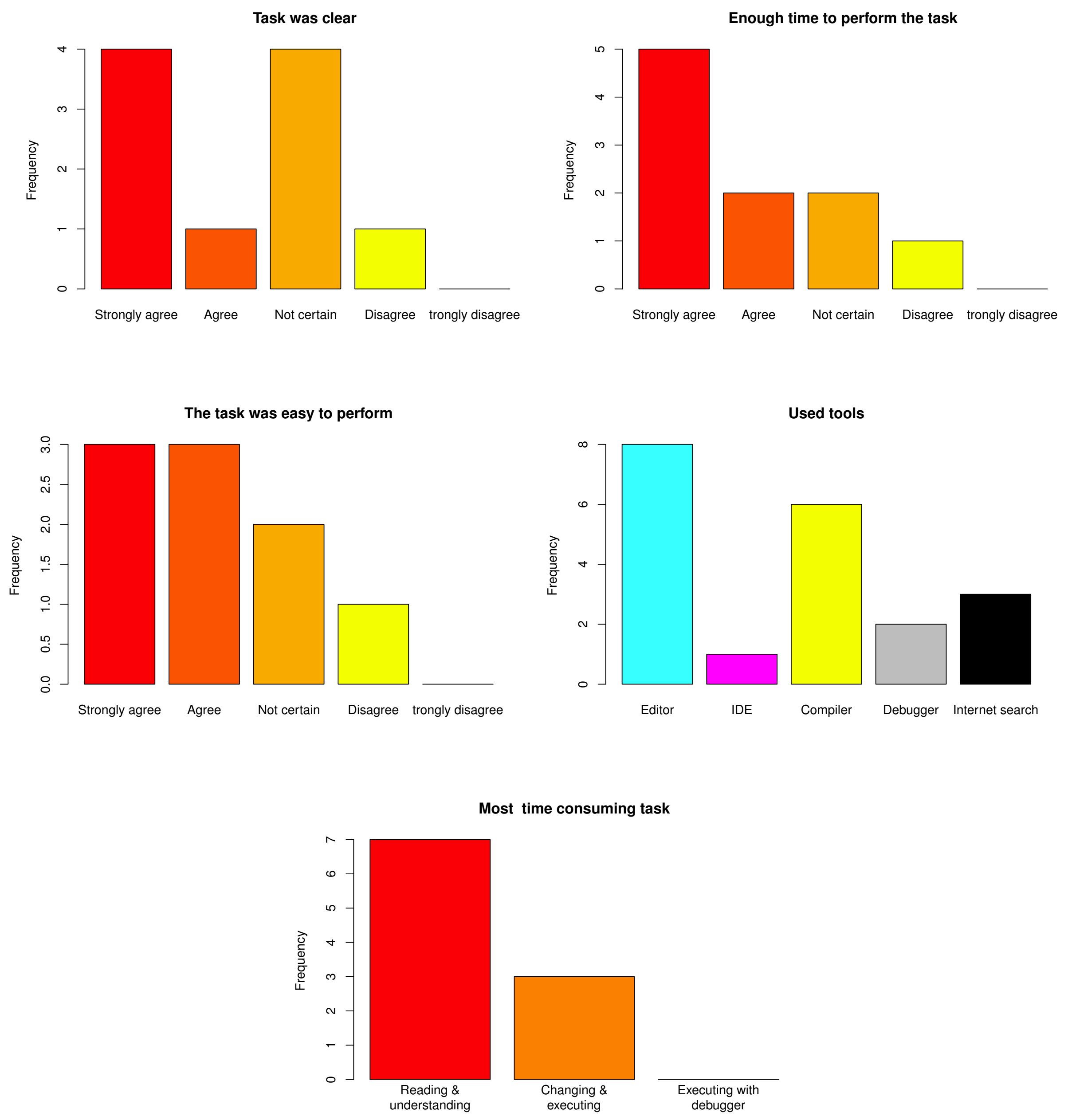


Figure 15: Post-questions answered by UEL's subjects.

6.12 Threats to validity

The content of this section is mostly copied from Section 8.6 of Deliverable D4.03
The main threats to the validity of this experiment belong to the conclusion, internal, construct and external validity threat categories.
Conclusion validity threats concern the relationship between treatment and outcome. We could not use statistical tests to draw our conclusions. Inability to reject the null hypothesis exposes us to type II errors (incorrectly accepting a false null hypothesis). This was expected and already acknowledged in the beginning of this section: a quite limited number of participants (they were 10) have been involved in this first replication. Moreover, among them, only a small fraction (only 2 participants) had the right knowledge and skill to complete the attack task. We will mitigate this threat by replicating the experiment multiple times, so as to increase the number of participants. In fact, the probability of a type II error can be reduced by increasing the sample size. Subjects were provided with the source code, not the binary code, because students are not proficient enough in binary and assembly code analysis at FBK, UEL and POLITO.
Internal validity threats concern external factors that may affect the independent variable. Subjects were not aware of the experimental hypotheses. Subjects were not rewarded for the participation in the experiment and they were not evaluated on their performance in doing the experiment.
Construct validity threats concern the relationship between theory and observation. They are mainly due to how we measure the effectiveness of code splitting and the related performance overhead. We manually assess the successful completion of each task in order to measure SR. During the labs, the experimenters make sure that times are accurately marked in the time sheets upon start and completion of the attack task. Performance measurements were repeated 100 times so as to remove the effect of the possible random fluctuations of the measured execution time under slightly different conditions.
External validity concerns the generalisation of the findings. In our experiment we considered one program, SpaceGame. Results may not generalise to different programs. However, the chosen program has been written by third parties, completely unrelated with ASPIRE, so as to ensure that it is not crafted to provide optimal application conditions for the ASPIRE protection. Moreover, the chosen program is publicly available as an open source project in SourceForge. Hence, it can be regarded as a representative for this category of software. Further replications of the study on additional objects will corroborate the external validity of our findings.

6.13 Lessons Learned

In this section we discuss the implications of the experimental results for the ASPIRE project.
Effects of protections. The code splitting protection increased the attack time by a factor 1.4, while at the same time degrading the execution time of the application by 107 times. This provides quantitative evidence for the shift in the economic convenience of code attacks when attackers deal with split code.
Attack strategies. Successful attacks against protected code required the reconstruction of server side computation on the client side. This might be extremely challenging and time consuming of realistic applications, larger than Space Game, for which a substantial portion of complex computations is moved to the server. The key implication is that the code splitting protection is extremely powerful and offers little room for attacks when complex computations that are difficult to reconstruct from black box observations are moved to the server. Such advantage is paid in terms of performance degradation, which in our experiment with Space Game was quantified as 107 times longer execution time. However, the performance overhead largely depends both on the application subject to splitting and on the chosen split configuration, so this quantification of slowdown can not assigned universal validity.

7 Experiments with Industrial Participants

Section authors:
Mariano Ceccato, Paolo Tonella (FBK)

This section reports the detailed definition of the experiments that will be conducted at the industrial project partners. At the moment of writing, some of the experiments are started and they are monitored with weekly conference calls between industrial participants and academic experimenters. However, considering the long period of execution (30+ days), no final experimental data is available yet. Results of the experiments with industrial participants will be reported on Deliverable D4.06 scheduled for month M36.
The plan originally was for two replications of the each experiment at each industrial partner. However, the consortium decided to merge the two replications into a single longer replication at each industrial partner. In this way, industrial hackers will be able to work continuously for a longer time and deliver more complete and valuable attack reports. The benefit of merging two separate sessions into a longer single session are discussed in more detail in Section 7.4.

7.1 Experimental Definition

The content of this section is copied from the first part of Section 9 of Deliverable D4.03

Goal	Analyze the ability of ASPIRE to prevent DRM attacks
Treatments	T1 = protection configuration 1; T2 = protection configuration 2; T3 = protection configuration 3
RQ1	To what extent do ASPIRE protections prevent attacks against DRM?
RQ2	What ASPIRE protections are most effective in preventing attacks against DRM?
Subjects	Hackers from the NAGRA tiger team
Objects	DemoPlayer (binary code)
Tasks	Violate specific DRM protection
Data	Report about the different reverse engineering and attack activities (e.g., data de-obfuscation; identifier renaming; control flow reconstruction; code understanding; decompilation) carried out by attackers
Design	Long running (30+ days) case study

Table 21: Nagravision case study

Tables 21, 22, 23 provide a schematic overview of the industrial case studies. The *goal* of these case studies is to evaluate the degree of protection offered by the ASPIRE techniques as a whole, considering those operating on the source code as well as those operating on the binary code. The entire ASPIRE tool chain is applied to the industrial case studies, so as to ensure maximum protection. The subjects involved in these case studies are professional pentesters employed by the industrial partners of ASPIRE.
The case studies aim at answering the following research questions:

- **RQ1** To what extent do ASPIRE protections prevent attacks against DRM/license management/secure authentication?
- **RQ2** What ASPIRE protections are most effective in preventing attacks against DRM/license management/secure authentication?

Goal	Analyze the ability of ASPIRE to prevent attacks against license protections
Treatments	T1 = no information about assets; T2 = detailed information about assets
RQ1	To what extent do ASPIRE protections prevent attacks against license management?
RQ2	What ASPIRE protections are most effective in preventing attacks against license management?
Subjects	Hackers from the SFNT tiger team
Objects	Diamante (binary code)
Tasks	Forge valid license
Data	Report about the different reverse engineering and attack activities (e.g., data de-obfuscation; identifier renaming; control flow reconstruction; code understanding; decompilation) carried out by attackers
Design	Long running (30+ days) case study

Table 22: SafeNet case study

These research questions deal with the effectiveness of the ASPIRE protections, when these are applied to the industrial case studies. We want to assess the capability of the ASPIRE protections to resist to a massive attack mounted by professional hackers during a long time period. Moreover, we want to assess the relative importance of different defence lines implemented by the various components in the ASPIRE tool chain, by analysing the activities carried out by the professional hackers to defeat each specific ASPIRE protection.

7.2 Objects

The content of this section is copied from Section 9.2 of Deliverable D4.03
The objects of this experiment are programs provided by the industrial ASPIRE partners:

DemoPlayer: Media player provided by Nagravision and requiring DRM protection

Diamante: License manager provided by SafeNet

OTP: One time password authentication server and client

Table 24 shows some size data about the involved objects (reported SLoC include any library that must be compiled with the application code). The tasks that hackers are asked to perform on these programs are respectively:

- **Nagravision:** Violate a specific DRM protection of DemoPlayer
- **SafeNet:** Forge a valid license key that is accepted by Diamante
- **Gemalto:** Authenticate on OTP without having any valid credential

7.3 Data

The content of this section is mostly copied from Section 9.3 of Deliverable D4.03
Professional hackers are asked to fill a *profiling questionnaire*, to collect data about their expertise and experience. Participants will be required to answer the following questions:

1. What is your programming experience in C? What C programming environment do you use?

Goal	Analyze the ability of ASPIRE to prevent attacks against secure authentication
Treatments	T1 = no information about assets; T2 = detailed information about assets
RQ1	To what extent do ASPIRE protections prevent attacks against secure authentication?
RQ2	What ASPIRE protections are most effective in preventing attacks against secure authentication?
Subjects	Hackers from the GTO tiger team
Objects	OTP (binary code)
Tasks	Authenticate with no valid credentials available
Data	Report about the different reverse engineering and attack activities (e.g., data de-obfuscation; identifier renaming; control flow reconstruction; code understanding; decompilation) carried out by attackers
Design	Long running (30+ days) case study

Table 23: Gemalto case study

Object	C SLoC	H SLoC	Java SLoC	Cpp SLoC	**Total**
DemoPlayer	2,595	644	1,859	1,389	6,487
Diamante	53,065	6,748	819	-	58,283
OTP	284,319	44,152	7,892	2,694	338,103

Table 24: Size of case study objects (measured by `sloccount`), divided by file type.

2. What is your programming experience in assembly? What assembly programming environment do you use?
3. What is your experience in disassembly? What tools do you use for disassembling binary code?
4. What is your experience in analysing compiled binary code?
5. What is your experience in tampering/altering binary code?
6. What are the static analysis tools that you use when tampering/altering binary code (e.g., Ida-pro)?
7. What are the dynamic analysis tools that you use when tampering/altering binary code (e.g., Olly-dbg)?

After completing the attack, professional hackers are asked to complete a *Final Attack Report*. The attack report will cover the following points in detail:

1. **Type of activities carried out during the attack**: detailed indications about the type of activities carried out to perform the attack and the proportion of time devoted to each activity. For instance, hackers may want to indicate the following activities: (1) data de-obfuscation; (2) identifier renaming; (3) control flow reconstruction; (4) code understanding; (5) decompilation; (6) execution inspection (e.g., via debugger); (7) execution modification (e.g., via debugger scripts). Hackers should provide such classification for each working day, not just for the whole attack session.

2. **Encountered obstacles**: detailed description of the obstacles encountered during the attack attempts. In particular, hackers should report any software protection that they think was put into place to prevent the attack and that actually represented a major obstacle for their work.

3. **Attack strategy**: description of the attack strategy and how it was adjusted whenever it proved ineffective. Hackers should describe the initial attempts and the decisions (if any) to change the strategy and to try alternative approaches.

4. **Return of the attack effort**: quantification of the attack effort, if possible economically, so as to provide an estimate of the kind of remuneration that would justify the amount of work done to carry out the attack.

5. **Level of expertise required**: which of the successful actions required a lot of expertise, which could be done by script kiddies?

6. **Identification vs. Exploitation**: for attacks succeeded once in the lab, attackers should describe what work would be required to exploit them in the real world (i.e., on a large scale, on software running on standard devices instead of on lab infrastructure, with other keys, etc.).

The adoption of a common template to collect feedback from professional hackers will allow us to perform comparisons among different experiments. For example, we might formulate considerations on whether similar or different strategies are used by different attackers to attack the same protections, or to compare the effort required to attack different assets in different applications when they are protected with the same protection.

7.4 Design

The design of these three experiments is a long running case study, with loose control on the involved subjects and mostly qualitative data collected during the execution of the experiment. The plan originally included two replications of the experiment on each industrial partner. Since an exact replication would bring limited value to the project, we decided to replicate each experiment under different conditions, in order to collect more information about protections and attacks. The different conditions that we intend to investigate are:

- **Protection configuration**: The same case study protected with different combinations of ASPIRE protection.
- **Information**: The same case study with the same combination of ASPIRE protections, but with more or less information provided to the participants. The information varies in terms of what are the sensitive assets protected with ASPIRE protections, what protections are deployed and, possibly, how these protections work.

Moreover, instead of conducting the two replications in separate moments in time, we decided to merge the two replications and conduct them one after the other, as two (or more) consecutive phases of the same experiment, to give hackers a larger continuous amount of time to work on their task. This allows a more flexible allocation of time to replications. In fact, if the first (harder) phase requires more time than expected, and if we consider it useful to the project, with this setup we can decide to let the first phase last longer and to consume a fraction of the time originally allocated to the subsequent phase, that is hence shortened in time. On the contrary, if the industrial hackers are able to complete the attack before the end of the first phase, they can anticipate the start of the subsequent phase, before the official beginning, thus limiting the waste of time and delivering interesting results sooner.

7.4.1 Protection Configuration

The first dimension considered in the experiment aims at assessing configurations such that protections can reinforce each other. In particular, the Nagravison use case will be subject to 3 different configurations at decreasing level of protection. The first configuration consists of applying a selection of ASPIRE protections, so as to evaluate professional hackers attacking the hardest case. In the second configuration some protections are removed, to experiment with an application that should be easier to attack. The third configuration of protections includes a minimal set of protections.
Other than testing different configurations, this strategy allows us to minimise the risk of collection of no useful data, in case protections are too difficult to attacks, by providing easier and easier code to attack. In case professional hackers are not able to complete a successful attack on the most protected version of the case study, they can try again on an easier version and still provide useful feedback to the project.
In Phase 1, the Nagravision use case is protected with the following protection techniques:

- All the protections available and applicable to the case study. They are:
  - Data obfuscation (convert static data to procedure);
  - White box cryptography.
  - SoftVM (client side splitting);
  - Anti debugging;
  - Call stack check;
  - Code guards;
  - Binary code obfuscation (with flatten function and opaque predicate);
  - Code mobility;
  - Remote attestation;

In Phase 2, the configuration used in phase 1 will be changed by removing these protection techniques:

- Anti-Debugging
- Remote attestation

In Phase 3, the configuration will be further reduced by removing the following techniques:

- Code mobility

In Phase 3, the Nagravision use-case is solely protected with offline protection techniques. The program under attack can thus be analysed without the complexity associated with the server execution.
Conversely, the SafeNet use-case and the Gemalto use-case will be protected with a single set of protection techniques (although different configurations of protections for each use-case). Instead of investigating with different configurations of protections across phases, for these two use-cases we will investigate the role of the amount of information provided to the attackers, as described in the following.

7.4.2 Asset Information

The second dimension is intended to investigate the role of additional information provided to attackers (about protected assets and about protections) to make attackers complete the attack. This dimension will be investigated when conducting the experiments in SafeNet and in Gemalto.

In Phase 1, the attack to perform is formulated as a high level attack goal, i.e. the description of the final goal that participants should achieve in order to violate a high level protection requirement (e.g., forge a valid license).
Of course, there are multiple strategies that an attacker might adopt to achieve this objective (e.g., derive a key, skip a validity check, tamper with the storage). An attacker will adopt the strategy that is closer to her/his experience, or that exploits the weakest protection. Maybe a participant could be able to identify an attack that works around all the deployed protections, and we have to accept this as a realistic attack case. The attack goal needs to be defined at a high level in order not to introduce any bias and not to suggest any (correct or wrong) attack strategy, so as to keep the experiments realistic. This first phase is supposed to measure globally the level of protection offered by the ACTC as a whole, when all protections are deployed at once.
In Phase 2, after the high level attack goal is achieved or a time out has expired, hackers are required to switch to detailed attacks to specific assets. Each of these latter tasks is an attack specifically defined on a program asset protected by an ASPIRE protection, e.g. a key or a portion of a program.
The purpose of this second phase is to explicitly ask hackers to break single ASPIRE protections (or single combination of protections) and to report on their level of security and, possibly, on their weak points.
The description of detailed attacks to assets leaks how the application is structured and what/where are its sensitive assets. These attack tasks guide the attackers on specific strategies, to defeat specific ASPIRE protection, for which we want empirical data in this second phase. Hence, they are less realistic, but still very useful to understand the relative strength of the ASPIRE protections.

7.5 Qualitative analysis

The content of this section is copied from Section 9.5 of Deliverable D4.03
Qualitative analysis of the reports collected from hackers will be the basis to answer RQ2. Evidence about the activities performed and the obstacles encountered will be mapped to the ASPIRE protections that were most effective in blocking the attacks mounted by professional hackers.
For what concerns RQ1, the answer may be boolean, i.e., the attack was or was not successful. However, in case of a non-successful attack, there might still be some degree of exploitation that was achieved, such as leakage of sensitive information, denial of service, or any other attack that was not the direct goal of the case study task. For this reason RQ1 is formulated in terms of the *extent* to which the attack is prevented. Again, qualitative data analysis will be employed to answer this question.

7.6 Threats to validity

The content of this section is mostly copied from Section 9.6 of Deliverable D4.03
The main threats to the validity of the case studies belong to the conclusion, internal, construct and external validity threat categories.
Conclusion validity threats concern the relationship between treatment and outcome. We assume that unsuccessful attacks can be attributed to the ASPIRE protections and that the obstacles encountered during successful attacks can be also attributed to the ASPIRE protections, while we do not know what would have happened without the ASPIRE protections. To mitigate this threat, we will collect extensive feedback from the professional hackers, in order to be able to support our conjectures with objective evidence collected in the field.
Internal validity threats concern external factors that may affect the independent variable. While subjects are professional hackers who are used to the kind of attack tasks requested to them during the study, there might be factors out of our control that affect their performance. Being a long running case study, the degree of control that we can have on the activities performed daily by the professional hackers is limited. We reduce this threat to validity by introducing a structured and

systematic data collection procedure and by scheduling regular conference calls with industrial participants.
Construct validity threats concern the relationship between theory and observation. They are mainly due to how we measure the effectiveness of the ASPIRE protections. We will manually assess the successful completion of the attack tasks to decide on the answer to RQ1. For RQ2, we will perform a qualitative analysis of the reports to obtain evidence in support to our conjectures. To minimise the risk of committing errors in the design of these experiments, the design has been discussed and revised by those project P.I. who are more expert in empirical studies with human participants. Moreover, to minimise the risk of committing errors in the setup of the material required to run the experiments, the industrial partners and the protection owners have been involved in the preparation.
External validity concerns the generalisation of the findings. Being based on a set of three case studies, results may not generalise to different cases. On the other hand, the considered objects are industrial applications, which make them quite meaningful cases, and the protected assets span across a wide and meaningful range (i.e., DRM, licenses, authentication), so we expect some degree of generalisability to similar applications and to similar industrial contexts.

7.7 Dates

The experimental material presented in this section has been revised by the experimenters and it is ready to be used in the actual experiments with industrial participants. The actual experiments will be conducted in these time frames:

- Nagravision: April-May, 2016;
- SafeNet: May-August, 2016;
- Gemalto: June-July, 2016.

At the moment of writing, the experiment in Nagravision and in SafeNet are started and they are monitored by the experimenters with weekly conference calls. Weekly calls are intended to monitor the smooth execution of the attack tasks and to provide clarifications and informations whenever needed to industrial participants, to avoid that an experiment gets stuck and waste valuable participants' time. Moreover, a weekly monitoring will allow to identify potential issues soon and timely react to solve them, possibly by changing some details of the experimental settings, to guarantee that useful informations and lessons are collected for the project.

List of abbreviations

ACM CCS Association for Computing Machinery Conference on Computer and Communications Security

ACTC ASPIRE Compiler Tool Chain

ADSS ASPIRE Decision Support System

AKB ASPIRE Knowledge Base

API Application Programming Interface

AR Array Reordering

ASPIRE Advanced Software Protection: Integration, Research, and Exploitation

ASM ASPIRE Security Model

AS Attack Success

AT Attack Time

CC Cyclomatic Complexity

CFIM Control Flow Indirection Metric

CRUD Create, Read, Update, Delete

DOP Number of destination register operands

DoW Description of Work

DPL Dynamic Program Length

DL Description Logic

DRM Digital Rights Management

DST Number of destination operations

EDG Number of edges

ET Execution Time

E/R DB Entity-Relationship DataBase

EXE Number of executed instructions

GUI Graphical User Interface

ICSE International Conference on Software Engineering

IDA Pro For clarification: IDA is not an abbreviation

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

INS Number of instructions

IND Number of index edges

JDK Java Development Kit

JMP Number of computed jumped instructions

MCAS Monte Carlo-based Attack Simulation

NB Number of bytes

OTP One Time Password

OWL Web Ontology Language

PF Protection Fitness Function

PO Performance Overhead

PN Petri Net

PNML Petri Net Markup Language

PNDV Petri Nets with Discrete Values

PSAM PN-based Software Attack Model

RNC Residue Number Coding

SAPS Single Attack Process Simulation

SAMMPN Software Attack Model based on Marked Petri Net

SLoC Source Lines Of Code

SOP Number of source register operands

SPA Software Protection Assessment

SPRO Software Protection (Workshop)

SR Success Rate

SRC Number of source operations

UML Unified Modeling Language

WP Work Package

XOR Exclusive OR

XML eXtensible Markup Language

References

[1] J. Cohen. *Statistical power analysis for the behavioral sciences (2nd ed.)*. Lawrence Earlbaum Associates, Hillsdale, NJ, 1988.

[2] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury Press; 7 edition, 2007.

[3] Robert J. Grissom and John J. Kim. *Effect sizes for research: A broad practical approach*. Lawrence Earlbaum Associates, 2nd edition edition, 2005.

[4] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[5] F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, and P. Tonella. Using acceptance tests as a support for clarifying requirements: a series of experiments. *Information & Software Technology*, 51:270–283, 2009.

[6] D.J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures (4th Ed.)*. Chapman & All, 2007.

[7] Marco Torchiano. *effsize: Efficient Effect Size Computation*, 2015. R package version 0.5.5.

[8] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.