



Advanced Software Protection:
Integration, Research and Exploitation

D4.05

Public Challenge

Project no.:	609734
Funding scheme:	Collaborative project
Start date of the project:	1 st November 2013
Duration:	36 months
Work programme topic:	FP7-ICT-2013-10
Deliverable type:	Report
Deliverable reference number:	ICT-609734 / D4.05 / 1.0
WP and tasks contributing:	WP4 / Tasks 4.5
Due date:	M30 / April 2016
Actual submission date:	15 July 2016
Responsible Organization:	UGent
Editor:	Bart Coppens
Dissemination Level:	Public
Revision:	1.0

Abstract:

We have designed a set of challenges for the public challenge, implemented these, and integrated them into a website that attackers can sign up to. The website has been released on <https://bounty.aspire-fp7.eu/>. Attackers can win a prize of up to €200 for breaking one of the 8 challenges.

Keywords:

Public challenge, website



Editor

Bart Coppens (UGent)



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 n°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

In this Deliverable, we document how we designed and implemented the public challenge for the ASPIRE project. Initially, we had foreseen to select a pre-existing piece of software to protect with the ASPIRE protections and to distribute this. As this turned out to be not possible, we came up with a new design for the public challenge, where a set of 8 smaller programs is protected. The challenge will run from August 2nd until September 30th.

We automated the generation of randomized instances of these challenges, and integrated this in a custom website in which attackers can request new instances to break.

Table of Contents

Section 1	Introduction	1
Section 2	Public Challenge Design	2
2.1	Introduction	2
2.2	Public Challenge Protections	2
2.3	Challenge individualization	4
Section 3	Website	5
Section 4	Rules	8
4.1	Introduction and rationale	8
4.2	General terms and conditions for the ASPIRE Public Challenge	9
Section 5	List of Abbreviations	11
Appendix A	Challenge source code.....	12
A.1	Challenge 1	12
A.2	Challenge 2	14
A.3	Challenge 3	16
A.4	Challenge 4	17
A.5	Challenge 5	19
A.6	Challenge 6	22
A.7	Challenge 7	26
A.8	Challenge 8	29

List of Figures

Figure 1 Landing page of the public challenge website	5
Figure 2 Dashboard for logged-in users	6
Figure 3 Public ranking	6
Figure 4 Page containing the Terms & Conditions	7

List of Tables

Table 1 Protections applied to the challenges	2
---	---

Introduction

Section Author:

Bart Coppens (UGent)

This report documents Deliverable D4.05 of type prototype (consisting of protected binaries to attack, a website and scripts to deliver them, scripts to handle responses, and a database to track downloads and responses) We describe how the Public Challenge was designed and implemented. This Document is based on the internal Working Document WD40.5.

Due to unforeseen use of extra resources from UGent for other tasks (such as the preparation of the tiger experiments of Task T4.4), this Task T4.5 was started significantly later than originally planned. While the prototype deliverable D4.05 contains a public component, that is, the public challenge itself, the design decisions should be kept from the attackers until the end of the public challenge. This is why this corresponding document is classified as confidential.

This document is structured as follows. First, in 0 we describe how we designed the different challenges. Then, in 0, we describe how the attackers will interact with us using a website we designed for the public challenge. Finally, in 0, we describe how we arrived at the terms and conditions that the attackers will have to follow.

Public Challenge Design

Section Author:

Bart Coppens (UGent)

1.1 Introduction

In the original ASPIRE Description of Work, we had foreseen to extend the Anti-Cheat Engine (ACE) for the Unreal game engine by protecting it with the ASPIRE protections. However, the author of ACE, Stijn Volckaert, was no longer able to contribute to the public challenge because he had already started a post-doctoral fellowship at the University of California, Irvine when Task T4.5 started. Thus, we needed to start from scratch with the public challenge design.

Because external attackers need to attack the public challenge, the goal of the challenge needs to be perfectly clear. That is to say, the application to be protected and attacked should contain very concrete assets that we can concisely describe and that should be easy for an attacker to verify as correctly attacked. While the ASPIRE Use Cases seem like the perfect match, in the end we do not use any of these due to IP concerns.

We instead decided on using a set of self-written target applications. The asset to be protected in all these cases is a random string of 64 characters in length, which we will refer to as the key. As valid keys can contain both upper and lower case characters and numbers, the search space is sufficiently large enough that attackers will not be able to brute force the correct number. The command-line program itself checks its first argument against this key, and prints out whether or not the argument matches the key. Thus on entering the correct key as input, it is immediately clear that the attack was successful.

The reason for deciding on a small set of 8 challenges is that then we can easily deploy different protection combinations to evaluate and compare these combinations individually.

Even though the main focus of the ASPIRE project is on protecting Android binaries, we decided to also offer GNU/Linux binaries of all challenges in addition to Android binaries. This way, experienced hackers and reverse engineers who are skilled in GNU/Linux will not be put off by the prospect of having to port their hacking tools and environment to an Android environment.

1.2 Public Challenge Protections

Table 1 provides an overview of the challenges and their combinations of protections. Besides the indicated protections, all challenges are protected with control-flow obfuscations (opaque predicates, branch functions and function flattening), call stack checks, and offline code guards.

Table 1 Protections applied to the challenges

Challenge Number	Data Obfuscation	Anti-Debugging	Remote Attestation	Code Mobility	Client-Server Split	SoftVM	WBC
1		X	X	X			
2	X						



3	X	X					
4		X				X	
5		X					X
6			X	X			
7		X					
8		X			X		

Even though the asset is the same in all challenges (i.e., a key with the same function; obviously the key value differs from binary to binary), and the program I/O behaviour is the same in all challenges, we have decided to tailor the source code of each challenge to suit the protections that are applied in each of those challenges. We do so because we think that different protection combinations can yield different attack paths.

For each of the challenges summarized in Table 1, we arrived at the following source code descriptions and protections that have to be applied:

1. In this challenge, an array is constructed by a block of code that is protected with the anti-debugger component. Next, mobile code is used to XOR this array with mobile data, and this mobile code then calls the `strncmp` C function. Attackers might try to attack this by hooking the `strncmp` function.
2. The key is split into 16 integers, each of which is encoded using the Residue Number Coding. The input to the program is also split into 16 integers, and these integers are then compared with the RNC-encoded key. Attackers might try to find the key by
3. This challenge also uses the RNC data obfuscation, and has anti-debugging applied at the entry of the `main` function and to the RNC decoding function.
4. The `main` function is protected by moving it into the SoftVM.
5. The key is used as ciphertext for WBC with a fixed key. This is then ‘decrypted’ when a challenge instance is generated, and the decrypted key is stored in the challenge instance. When run, the challenge similarly decrypts its input and compares it to the stored decrypted key. Furthermore, the WBC code has been protected with anti-debugging.
6. The key is checked in two parts: the first half of the key is checked byte per byte. The code that checks the key performs a very long delay loop (a double nested loop, both of which only finish after looping through 2^{64} values) and a sleep of about 11 days after each character is checked. These delays are to encourage the attackers to try to modify the binary to remove or shorten the delay to verify the key, which is protected with Remote Attestation. The next part of the binary is protected using mobile code, so that if the RA component detects tampering, the protection server can trigger a tamper response in the code mobility component.
7. This challenge starts from code that has ‘ugly’ control flow at the source level. Then this code has binary control flow obfuscations applied, and is furthermore protected with anti-debugging.
8. The code is protected with client-server code splitting, where each character is sent individually to the server, and the client only asks for the correctness of the next key character once the previous one was correct. One way that we think this can be attacked is by a side channel: attackers can deduce whether a single character is correct by observing the time it takes the challenge to complete, or by observing the amount of communication happening between the client and the server.

The full sources of these challenges (including their annotations) have been included in Appendix A.

1.3 Challenge individualization

Some of the challenges have been protected with networked anti-tampering protections. In particular, in the case of a failed Remote Attestation request, the server will instruct the code mobility server to trigger its anti-tamper response (which results in the code mobility no longer sending back any code). Thus, once an attacker tries circumventing the RA and fails, the circumvented challenge will no longer function. If we would send all attackers the same challenge binary, no one would be able to correctly run the application anymore. Thus, we have to provide attackers with individualised copies of the challenges. Furthermore, we want to provide attackers the opportunity to learn from their mistakes, and send them fresh, working copies of a challenge if the anti-tampering has been triggered. Moreover, we want to enable collusion attacks, in which attackers attack multiple instances of an application protected with the same protections (possibly applied differently) but different key values. We decided to allow attackers to request a limited number of fresh instances of each challenge.

To allow for this, we implemented the following flow:

1. Whenever we create a new instance of a challenge, we in a way that the ACTC will assign a unique and reproducible identifier to that challenge instance.
2. This identifier is used as the initialisation to generate the randomness for the key asset in the challenge.
3. This identifier is also used in the control flow obfuscation and layout randomization of the binary rewriting step in the tool chain. This means that every instance will be completely randomized at the binary level. We verified that the binaries indeed differ at the binary level by comparing some different challenge instances with the binary diffing tool BinDiff.
4. When an instance is produced, it is automatically tested on either an Android board or on a GNU/Linux board as appropriate. We do so as a quality assurance, to ensure that we do not provide non-working binaries to attackers.

We pre-generate a set of challenge instances, which are put automatically into a database that is accessed by the public challenge's website. Whenever an attacker requests a new challenge, the database is queried and the next available binary is mailed to the requesting attacker. In case the number of challenge instances is running low, the public challenge website admins are sent an e-mail, that allow us to generate additional instances.

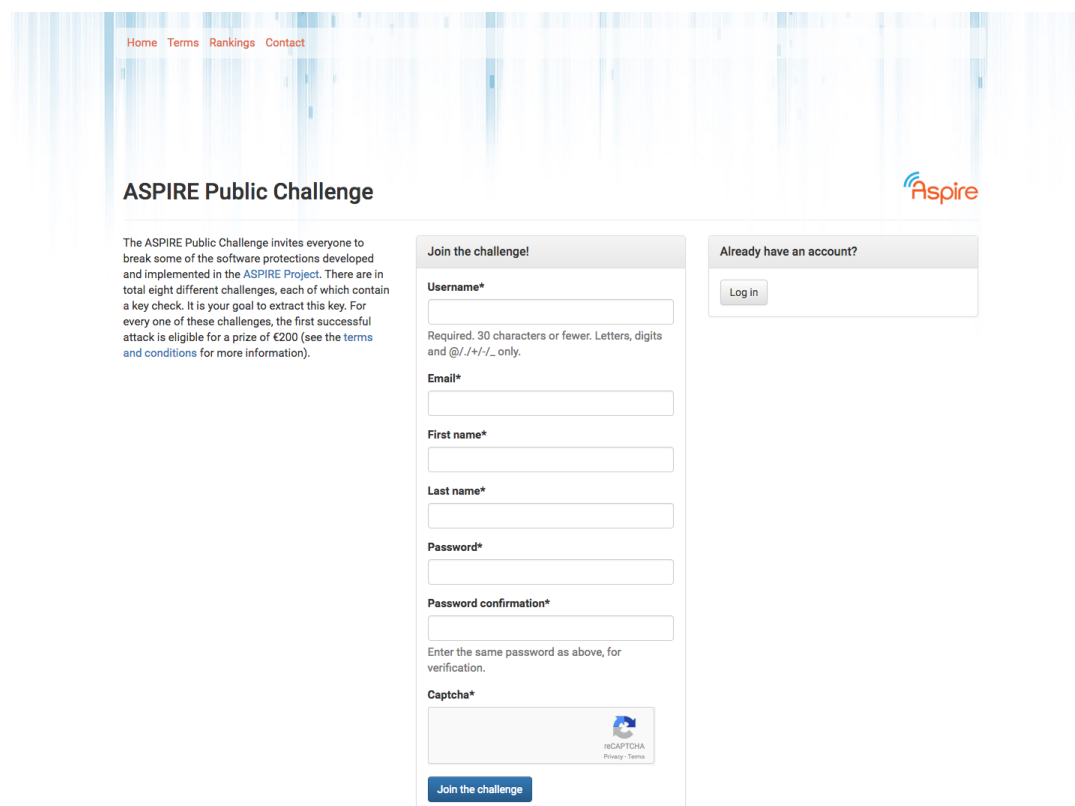
In a database, we furthermore store hashes of the correct keys for each challenge instance. To verify whether or not a user has cracked an instance, he will submit the combination of the name of the broken challenge instance, and the matching key. We can then query the database for this information and easily check whether the attacker succeeded in recovering the key. Once we know that an attack succeeded, we also know who the successful attacker is, as challenge instances are linked to users (and their e-mail addresses).

Website

Section Author:

Bart Coppens (UGent)

Attackers will interact with us through a public website we released on <https://bounty.aspire-fp7.eu/>. Figure 1 shows the site's landing page for new users. It contains a short description of what the goal of the challenge is, and also contains the registration form where attackers can sign up, and a 'Log in' button that can be used for registered attackers. The banner of the page also contains links to the Terms & Conditions, to a Ranking page, and to a Contact page.



The screenshot shows the landing page of the ASPIRE Public Challenge. At the top, there is a navigation bar with links: Home, Terms, Rankings, and Contact. The main heading is "ASPIRE Public Challenge". Below this, there is a descriptive paragraph about the challenge. To the right, there is a registration form titled "Join the challenge!" with fields for Username*, Email*, First name*, Last name*, Password*, and Password confirmation*. A reCAPTCHA widget is also present. A "Join the challenge" button is at the bottom of the form. To the right of the registration form, there is a "Log in" button under the heading "Already have an account?". The Aspire logo is in the top right corner.

Home Terms Rankings Contact

ASPIRE Public Challenge

The ASPIRE Public Challenge invites everyone to break some of the software protections developed and implemented in the [ASPIRE Project](#). There are in total eight different challenges, each of which contain a key check. It is your goal to extract this key. For every one of these challenges, the first successful attack is eligible for a prize of €200 (see the [terms and conditions](#) for more information).

Join the challenge!

Username*

Required: 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

First name*

Last name*

Password*

Password confirmation*

Enter the same password as above, for verification.

Captcha*

[Join the challenge](#)

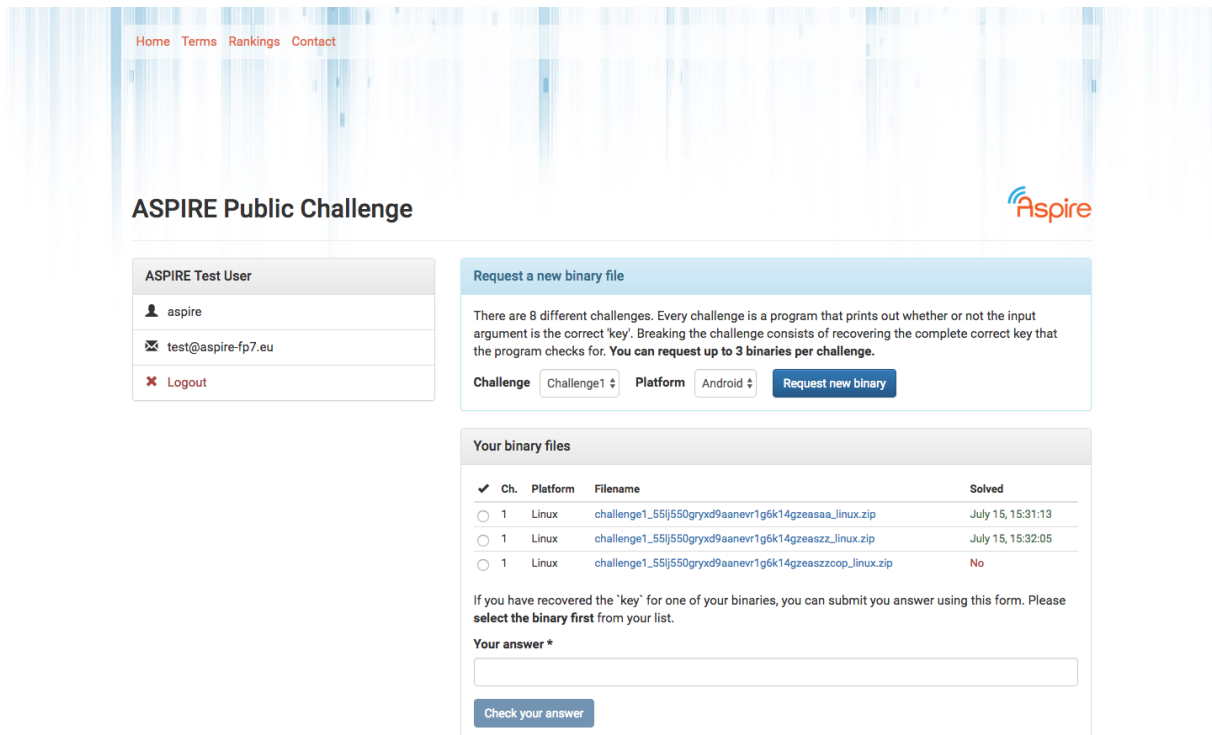
Already have an account?

[Log in](#)

Aspire

Figure 1 Landing page of the public challenge website

When a user has logged in, surfing to the same landing page presents the user with a dash board, which is show in Figure 2. From this screen, attackers can request new challenge instances, and submit their solutions to one of their challenge instances.



The screenshot shows the ASPIRE Public Challenge dashboard for a user named 'aspire'. The user is logged in as 'ASPIRE Test User' with email 'test@aspire-fp7.eu'. The dashboard includes a section for requesting a new binary file, a table of binary files, and a form to submit an answer.

ASPIRE Public Challenge

Home Terms Rankings Contact

ASPIRE Test User

aspire

test@aspire-fp7.eu

Logout

Request a new binary file

There are 8 different challenges. Every challenge is a program that prints out whether or not the input argument is the correct 'key'. Breaking the challenge consists of recovering the complete correct key that the program checks for. **You can request up to 3 binaries per challenge.**

Challenge Challenge1 Platform Android Request new binary

Your binary files

✓	Ch.	Platform	Filename	Solved
<input type="radio"/>	1	Linux	challenge1_55ij550gryxd9aanevr1g6k14gzeasaa_linux.zip	July 15, 15:31:13
<input type="radio"/>	1	Linux	challenge1_55ij550gryxd9aanevr1g6k14gzeasz_linux.zip	July 15, 15:32:05
<input type="radio"/>	1	Linux	challenge1_55ij550gryxd9aanevr1g6k14gzeaszcoop_linux.zip	No

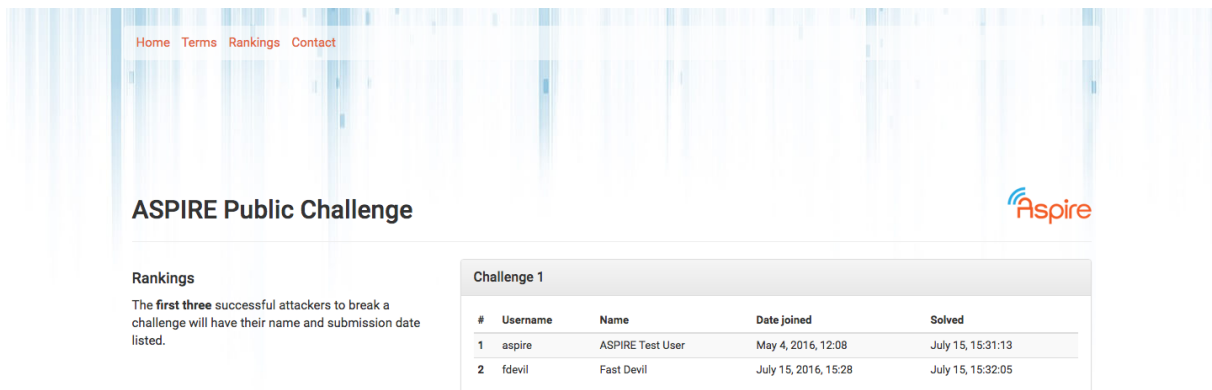
If you have recovered the 'key' for one of your binaries, you can submit you answer using this form. Please **select the binary first** from your list.

Your answer *

Check your answer

Figure 2 Dashboard for logged-in users

Once a user has successfully broken a challenge, this will show up on the public ranking, as is shown in Figure 3. Finally, Figure 4 shows the page with the Terms & Conditions.



The screenshot shows the ASPIRE Public Challenge public ranking page. It displays the rankings for Challenge 1, listing the top three successful attackers: 'aspire', 'fdevil', and 'ASPIRE Test User'.

ASPIRE Public Challenge

Home Terms Rankings Contact

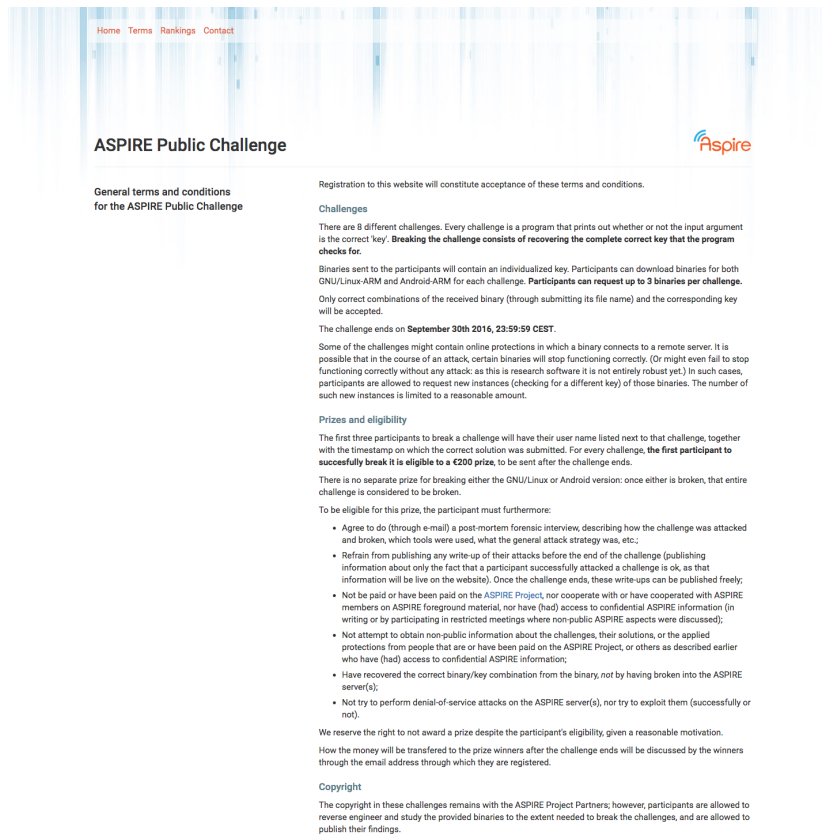
Rankings

The **first three** successful attackers to break a challenge will have their name and submission date listed.

Challenge 1

#	Username	Name	Date joined	Solved
1	aspire	ASPIRE Test User	May 4, 2016, 12:08	July 15, 15:31:13
2	fdevil	Fast Devil	July 15, 2016, 15:28	July 15, 15:32:05

Figure 3 Public ranking



Home Terms Rankings Contact

ASPIRE Public Challenge

General terms and conditions for the ASPIRE Public Challenge

Registration to this website will constitute acceptance of these terms and conditions.

Challenges

There are 8 different challenges. Every challenge is a program that prints out whether or not the input argument is the correct 'key'. **Breaking the challenge consists of recovering the complete correct key that the program checks for.**

Binaries sent to the participants will contain an individualized key. Participants can download binaries for both GNU/Linux ARM and Android-ARM for each challenge. **Participants can request up to 3 binaries per challenge.**

Only correct combinations of the received binary (through submitting its file name) and the corresponding key will be accepted.

The challenge ends on **September 30th 2016, 23:59:59 CEST**.

Some of the challenges might contain online protections in which a binary connects to a remote server. It is possible that in the course of an attack, certain binaries will stop functioning correctly. (Or might even fail to stop functioning correctly without any attack; as this is research software it is not entirely robust yet.) In such cases, participants are allowed to request new instances (checking for a different key) of those binaries. The number of such new instances is limited to a reasonable amount.

Prizes and eligibility

The first three participants to break a challenge will have their user name listed next to that challenge, together with the timestamp on which the correct solution was submitted. For every challenge, **the first participant to successfully break it is eligible to a €200 prize**, to be sent after the challenge ends.

There is no separate prize for breaking either the GNU/Linux or Android version; once either is broken, that entire challenge is considered to be broken.

To be eligible for this prize, the participant must furthermore:

- Agree to do (through e-mail) a post-mortem forensic interview, describing how the challenge was attacked and broken, which tools were used, what the general attack strategy was, etc.;
- Refrain from publishing any write-up of their attacks before the end of the challenge (publishing information about only the fact that a participant successfully attacked a challenge is ok, as that information will be live on the website). Once the challenge ends, these write-ups can be published freely;
- Not be paid or have been paid on the ASPIRE Project, nor cooperate with or have cooperated with ASPIRE members on ASPIRE foreground material, nor have (had) access to confidential ASPIRE information (in writing or by participating in restricted meetings where non-public ASPIRE aspects were discussed);
- Not attempt to obtain non-public information about the challenges, their solutions, or the applied protections from people that are or have been paid on the ASPIRE Project, or others as described earlier who have (had) access to confidential ASPIRE information;
- Have recovered the correct binary/key combination from the binary, not by having broken into the ASPIRE server(s);
- Not try to perform denial-of-service attacks on the ASPIRE server(s), nor try to exploit them (successfully or not).

We reserve the right to not award a prize despite the participant's eligibility, given a reasonable motivation.

How the money will be transferred to the prize winners after the challenge ends will be discussed by the winners through the email address through which they are registered.

Copyright

The copyright in these challenges remains with the ASPIRE Project Partners; however, participants are allowed to reverse engineer and study the provided binaries to the extent needed to break the challenges, and are allowed to publish their findings.

Figure 4 Page containing the Terms & Conditions

Rules

1.4 Introduction and rationale

Even though we want to motivate the attackers to break the challenges, their doing so must be somewhat orderly if we are to learn anything from them. Thus, all attackers must follow a set of ground rules when participating in the public challenge. These ground rules were codified into the ‘General terms and conditions for the ASPIRE Public Challenge’, which is shown verbatim in the next section.

To arrive at these precise rules, we first discussed some general requirements in a Steering Board meeting:

1. Motivate attackers by appealing to their pride, by listing their name immediately on a public leader board when they break a challenge.
2. Attackers should be motivated to participate not only through their pride, but also by offering prize money for each challenge that is successfully broken.
3. We want to learn as much as possible from the public challenge. Thus, in order to claim their prizes, attackers must provide us with detailed information on how they broke the challenge. If they do not do this, they will not get the prize.
4. Attackers might still want to break a challenge even after it was already broken by someone else, i.e., when it might not lead to any monetary reward. We still want to learn as much as possible from these attackers, though, as they might try different approaches compared to the first successful attempt. In order not to ‘taint’ the people coming after them, successful attackers should not publish the information until after the challenge ends.
5. Attackers are free to publish information about how they attacked the challenges in a public write-up, but again, only after the challenge has ended.
6. We would like to be able to have results before the end of the project.

We fleshed out these requirements, and added some initial description of how exactly the public challenge works. Furthermore, as some of our protections involve server-side components that were not necessarily written with security in mind, we explicitly ask that the attackers not try to attack the servers, but that they focus on the protected application. This is similar to the tiger team experiments, where the teams were explicitly told that the servers were out of scope. This is consistent with the attack model we have used throughout the project. As for the prize money, we settled on a €200 per-challenge prize, for the first successful attack.

As for the submission deadline, we originally planned the public challenge to end after the end of the ASPIRE project, as we already discussed in Deliverable D4.04. However, as it would be beneficial to be able to analyze and discuss the results during the project, we will publicly state that the competition finishes at the end of September. However, internally we will allow for a possibility of extending this deadline until the end of the calendar year, in accordance to what was stated in Deliverable D4.04 Section 1. We will do so only if we can still learn more from the challenges, for example if some of them have not yet been broken and when we notice (in the security server logs) that attackers are still active.

Furthermore, we decided to launch the challenge itself on August 2nd, even though all work to launch it has already been completed by July 15th. The reason is that the public challenge itself is completely organized by UGent, which has a fixed holiday period in the second half of July due to a city festival, which is thus also the period where most people decide to go travel abroad. As we cannot guarantee that everything will immediately work smoothly once a significant number of attackers try to run the server-connected applications, and we thus

might to do manual interventions, launching the challenge without the possibility of low-latency support by the people who designed the challenge would be irresponsible.

UGent sent out a draft of these rules to some of the project partners for feedback. As there was no additional feedback apart from some typos, these typo-fixed draft rules became the final rules.

1.5 General terms and conditions for the ASPIRE Public Challenge

- There are 8 different challenges. Every challenge is a program that prints out whether or not the input argument is the correct 'key'.
- Breaking the challenge consists of recovering the complete correct key that the program checks for.
- Binaries sent to the participants will contain an individualized key.
- Only correct combinations of the received binary (through submitting its file name) and the corresponding key will be accepted.
- Participants can request binaries for both GNU/Linux-ARM and Android-ARM for each challenge.
- There is no separate prize for breaking either the GNU/Linux or Android version: once either is broken, that entire challenge is considered to be broken.
- The first three participants to break a challenge will have their user name listed next to that challenge, together with the timestamp on which the correct solution was submitted.
- For every challenge, the first participant to successfully break it is eligible to a €200 prize, to be sent after the challenge ends.
- To be eligible for this prize, the participant must furthermore:
 - Agree to do (through e-mail) a post-mortem forensic interview, describing how the challenge was attacked and broken, which tools were used, what the general attack strategy was, etc.;
 - Refrain from publishing any write-up of their attacks before the end of the challenge (publishing information about only the fact that a participant successfully attacked a challenge is ok, as that information will be live on the website). Once the challenge ends, these write-ups can be published freely;
 - Not be paid or have been paid on the ASPIRE Project, nor cooperate with or have cooperated with ASPIRE members on ASPIRE foreground material, nor have (had) access to confidential ASPIRE information (in writing or by participating in restricted meetings where non-public ASPIRE aspects were discussed);
 - Not attempt to obtain non-public information about the challenges, their solutions, or the applied protections from people that are or have been paid on the ASPIRE Project, or others as described earlier who have (had) access to confidential ASPIRE information;
 - Have recovered the correct binary/key combination from the binary, /not/ by having broken into the ASPIRE server(s)
 - Not try to perform denial-of-service attacks on the ASPIRE server(s), nor try to exploit them (successfully or not).
- We reserve the right to not award a prize despite the participant's eligibility, given a reasonable motivation.
- How the money will be transferred to the prize winners after the challenge ends will be discussed by the winners through the email address through which they are registered.
- The challenge ends on September 30th 2016, 23:59:59 CEST.
- Some of the challenges might contain online protections in which a binary connects to a remote server. It is possible that in the course of an attack, certain binaries will stop

functioning correctly. (Or might even fail to stop functioning correctly without any attack: as this is research software it is not entirely robust yet.) In such cases, participants are allowed to request new instances (checking for a different key) of those binaries. The number of such new instances is limited to a reasonable amount.

- The copyright in these challenges remains with the ASPIRE Project Partners; however, participants are allowed to reverse engineer and study the provided binaries to the extent needed to break the challenges, and are allowed to publish their findings.
- Registration to this website will constitute acceptance of these terms and conditions.

List of Abbreviations

ACE	Anti-Cheat Engine
ACCL	ASPIRE Client-side Communication Logic
ASPIRE	Advanced Software Protection: Integration, Research and Exploitation
RNC	Residue Number Coding
WBC	White-Box Cryptography

Appendix A Challenge source code

A.1 Challenge 1

```

/* C-standard headers */
#include <stdio.h>
#include <string.h>

#include "common.h"
#include "xor.h"

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    _Pragma("ASPIRE begin protection(remote_attestation, static_ra(RW_NORMAL, HF_SHA256, NI_1, NG_1,
MA_1, DS_1), label(first_attestator), frequency(10))")
    _Pragma("ASPIRE end")

    /* Validate the arguments */
    validateArguments(argc, argv);

    _Pragma("ASPIRE end") /* obfuscations, call check */

    _Pragma("ASPIRE begin protection(remote_attestation, static_ra_region, attestator(first_attestator),
attest_at_startup(true))")
    /* Build up the key */
    char key[KEY_LENGTH];

    COMPILER_BARRIER();
    _Pragma("ASPIRE begin protection(anti_debugging)")
    key[59] = part1[59];
    key[8] = part1[8];
    key[14] = part1[14];
    key[0] = part1[0];
    key[37] = part1[37];
    key[48] = part1[48];
    key[41] = part1[41];
    key[6] = part1[6];
    key[51] = part1[51];
    key[3] = part1[3];
    key[23] = part1[23];
    key[10] = part1[10];

```

```
key[49] = part1[49];
key[26] = part1[26];
key[17] = part1[17];
key[40] = part1[40];
key[22] = part1[22];
key[18] = part1[18];
key[15] = part1[15];
key[62] = part1[62];
key[61] = part1[61];
key[53] = part1[53];
key[55] = part1[55];
key[63] = part1[63];
key[19] = part1[19];
key[11] = part1[11];
key[50] = part1[50];
key[12] = part1[12];
key[52] = part1[52];
key[58] = part1[58];
key[44] = part1[44];
key[35] = part1[35];
key[45] = part1[45];
key[27] = part1[27];
key[16] = part1[16];
key[34] = part1[34];
key[33] = part1[33];
key[60] = part1[60];
key[42] = part1[42];
key[7] = part1[7];
key[36] = part1[36];
key[13] = part1[13];
key[38] = part1[38];
key[32] = part1[32];
key[46] = part1[46];
key[20] = part1[20];
key[4] = part1[4];
key[24] = part1[24];
key[30] = part1[30];
key[56] = part1[56];
key[25] = part1[25];
key[47] = part1[47];
key[39] = part1[39];
key[31] = part1[31];
key[43] = part1[43];
key[28] = part1[28];
key[2] = part1[2];
```

```

key[5] = part1[5];
key[54] = part1[54];
key[29] = part1[29];
key[21] = part1[21];
key[1] = part1[1];
key[9] = part1[9];
key[57] = part1[57];
_Pragma("ASPIRE end") /* anti_debugging */
_Pragma("ASPIRE end") /* RA */
_Pragma("ASPIRE end") /* CG */

COMPILER_BARRIER();

_Pragma("ASPIRE begin protection(code_mobility, status(mobile), data(mobile))")
int iii;
for (iii = 0; iii < KEY_LENGTH; iii++)
    key[iii] ^= part2[iii];

/* Do the actual compare */
int res = strncmp(key, argv[1], KEY_LENGTH);
_Pragma("ASPIRE end") /* code_mobility */

_Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
_Pragma("ASPIRE                                     begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function)))")
/* Print out the result */
if (res == 0)
    printf("Success!\n");
else
    printf("Wrong key.\n");
_Pragma("ASPIRE end") /* obfuscations */
_Pragma("ASPIRE end") /* CG */
}

```

A.2 Challenge 2

```

/* C-standard headers */
#include <stdint.h>
#include <stdio.h>

#include "common.h"
#include "rnc_key.h"

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")

```

```

_Pragma("ASPIRE
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check))
begin
/* Validate the arguments */
validateArguments(argc, argv);

/* Get the key and check it part by part */
int res = 0;
char* key = argv[1];
uint32_t* int_key = (uint32_t*) key;

if (int_key[0] != rnc_part0)
    res = 1;
if (int_key[1] != rnc_part1)
    res = 1;
if (int_key[2] != rnc_part2)
    res = 1;
if (int_key[3] != rnc_part3)
    res = 1;
if (int_key[4] != rnc_part4)
    res = 1;
if (int_key[5] != rnc_part5)
    res = 1;
if (int_key[6] != rnc_part6)
    res = 1;
if (int_key[7] != rnc_part7)
    res = 1;
if (int_key[8] != rnc_part8)
    res = 1;
if (int_key[9] != rnc_part9)
    res = 1;
if (int_key[10] != rnc_part10)
    res = 1;
if (int_key[11] != rnc_part11)
    res = 1;
if (int_key[12] != rnc_part12)
    res = 1;
if (int_key[13] != rnc_part13)
    res = 1;
if (int_key[14] != rnc_part14)
    res = 1;
if (int_key[15] != rnc_part15)
    res = 1;

/* Print out the result */
if (res == 0)

```

```

    printf("Success!\n");
else
    printf("Wrong key.\n");
_Pragma("ASPIRE end") /* obfuscations */
_Pragma("ASPIRE end") /* CG */
}

```

A.3 Challenge 3

```

/* C-standard headers */
#include <stdint.h>
#include <stdio.h>

#include "common.h"
#include "rnc_key.h"

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    /* Validate the arguments */
    validateArguments(argc, argv);
    _Pragma("ASPIRE end") /* obfuscations */

    _Pragma("ASPIRE begin protection(anti_debugging)")
    /* Get the key and check it part by part */
    int res = 0;
    char* key = argv[1];
    uint32_t* int_key = (uint32_t*) key;
    _Pragma("ASPIRE end") /* anti_debugging */

    _Pragma("ASPIRE
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    if (int_key[0] != rnc_part0)
        res = 1;
    if (int_key[1] != rnc_part1)
        res = 1;
    if (int_key[2] != rnc_part2)
        res = 1;
    if (int_key[3] != rnc_part3)
        res = 1;
    if (int_key[4] != rnc_part4)
        res = 1;
    if (int_key[5] != rnc_part5)
        res = 1;
}

```

```

    if (int_key[6] != rnc_part6)
        res = 1;
    if (int_key[7] != rnc_part7)
        res = 1;
    if (int_key[8] != rnc_part8)
        res = 1;
    if (int_key[9] != rnc_part9)
        res = 1;
    if (int_key[10] != rnc_part10)
        res = 1;
    if (int_key[11] != rnc_part11)
        res = 1;
    if (int_key[12] != rnc_part12)
        res = 1;
    if (int_key[13] != rnc_part13)
        res = 1;
    if (int_key[14] != rnc_part14)
        res = 1;
    if (int_key[15] != rnc_part15)
        res = 1;

    /* Print out the result */
    if (res == 0)
        printf("Success!\n");
    else
        printf("Wrong key.\n");
    _Pragma("ASPIRE end") /* obfuscations */
    _Pragma("ASPIRE end") /* CG */
}

```

A.4 Challenge 4

```

/* C-standard headers */
#include <stdio.h>

#include "common.h"
#include "key.h"

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    /* Validate the arguments */
    validateArguments(argc, argv);
    char* input = argv[1];

```



```
int res = 0;
_Pragma("ASPIRE begin protection(softvm)")
res = res || !(protected_key[28] == input[28]);
res = res || !(protected_key[20] == input[20]);
res = res || !(protected_key[47] == input[47]);
res = res || !(protected_key[25] == input[25]);
res = res || !(protected_key[21] == input[21]);
res = res || !(protected_key[0] == input[0]);
res = res || !(protected_key[27] == input[27]);
res = res || !(protected_key[45] == input[45]);
res = res || !(protected_key[58] == input[58]);
res = res || !(protected_key[38] == input[38]);
res = res || !(protected_key[39] == input[39]);
res = res || !(protected_key[7] == input[7]);
res = res || !(protected_key[62] == input[62]);
res = res || !(protected_key[61] == input[61]);
res = res || !(protected_key[5] == input[5]);
res = res || !(protected_key[46] == input[46]);
res = res || !(protected_key[18] == input[18]);
res = res || !(protected_key[42] == input[42]);
res = res || !(protected_key[8] == input[8]);
res = res || !(protected_key[37] == input[37]);
res = res || !(protected_key[33] == input[33]);
res = res || !(protected_key[49] == input[49]);
res = res || !(protected_key[2] == input[2]);
res = res || !(protected_key[59] == input[59]);
res = res || !(protected_key[22] == input[22]);
res = res || !(protected_key[44] == input[44]);
res = res || !(protected_key[40] == input[40]);
res = res || !(protected_key[29] == input[29]);
res = res || !(protected_key[23] == input[23]);
res = res || !(protected_key[57] == input[57]);
res = res || !(protected_key[14] == input[14]);
res = res || !(protected_key[53] == input[53]);
res = res || !(protected_key[32] == input[32]);
res = res || !(protected_key[24] == input[24]);
res = res || !(protected_key[17] == input[17]);
res = res || !(protected_key[41] == input[41]);
res = res || !(protected_key[11] == input[11]);
res = res || !(protected_key[63] == input[63]);
res = res || !(protected_key[19] == input[19]);
res = res || !(protected_key[10] == input[10]);
res = res || !(protected_key[51] == input[51]);
res = res || !(protected_key[4] == input[4]);
```

```

    res = res || !(protected_key[31] == input[31]);
    res = res || !(protected_key[15] == input[15]);
    res = res || !(protected_key[43] == input[43]);
    res = res || !(protected_key[13] == input[13]);
    res = res || !(protected_key[9] == input[9]);
    res = res || !(protected_key[48] == input[48]);
    res = res || !(protected_key[52] == input[52]);
    res = res || !(protected_key[6] == input[6]);
    res = res || !(protected_key[35] == input[35]);
    res = res || !(protected_key[34] == input[34]);
    res = res || !(protected_key[54] == input[54]);
    res = res || !(protected_key[12] == input[12]);
    res = res || !(protected_key[16] == input[16]);
    res = res || !(protected_key[50] == input[50]);
    res = res || !(protected_key[60] == input[60]);
    res = res || !(protected_key[1] == input[1]);
    res = res || !(protected_key[26] == input[26]);
    res = res || !(protected_key[30] == input[30]);
    res = res || !(protected_key[55] == input[55]);
    res = res || !(protected_key[3] == input[3]);
    res = res || !(protected_key[36] == input[36]);
    res = res || !(protected_key[56] == input[56]);
    _Pragma("ASPIRE end")

/* Print out the result */
if (res == 0)
    printf("Success!\n");
else
    printf("Wrong key.\n");
_Pragma("ASPIRE end") /* obfuscations, call check */
_Pragma("ASPIRE end") /* CG */
}

```

A.5 Challenge 5

```

/* C-standard headers */
#include <stdio.h>
#include <stdint.h>
#include <assert.h>

#include "common.h"
#include "challenge.h"

/*
 * @brief

```



```

*   function used to encrypt a 64-byte buffer using AES
*
* Copyright (C) 2016-2016 Nagravision S.A.
*/
/*****
/*
/*
/*           INCLUDE FILES
/*
/*
/*
*****/
#include <stdlib.h>

/*****
/*
/*
/*           CONSTANTS
/*
/*
/*
*****/
#ifdef __cplusplus
extern "C" {
#endif /* C++ */

#ifdef NULL
#define NULL    ((void *)0)
#endif /* NULL */

/* confidentiality key */
const char  gWbcAesChallengeKey[16]  __attribute__((ASPIRE("protection(wbc,   role(key),   size(16),
label(WbcAesChallenge)"))))
= {
    0x5e, 0xac, 0xab, 0x01, 0xab, 0xae, 0x11, 0xa1,
    0xaf, 0x0c, 0xac, 0xc1, 0xa1, 0xe1, 0xca, 0xfe
};
/* confidentiality IV (C2016Nagravision) */
const char  gWbcAesChallengeIv[16]  __attribute__((ASPIRE("protection(wbc,   role(iv),   size(16),
label(WbcAesChallenge)"))))
= {
    0xa9, 0x32, 0x30, 0x31, 0x36, 0x6e, 0x61, 0x67,
    0x72, 0x61, 0x76, 0x69, 0x73, 0x69, 0x6f, 0x6e
};

/*****
/*
/*
/*           TYPES & STRUCTURES
/*
/*
/*
*****/
**

```

```

* @brief
*   byte buffer
*/
typedef struct
{
    uint8_t*    pBuffer;
    /**< data */
    size_t      size;
    /**< size, in bytes */
} TBuffer;

#include "key_wbc_decrypted.h"
int doCheck(const char* in_key)
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE                                     begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    uint8_t encrypted_key_from_input[KEY_LENGTH];
    uint8_t* pInput;
    TBuffer  inputBuffer;
    TBuffer* pInputBuffer    __attribute__((ASPIRE("protection(wbc,      role(input),    size(0),
label(WbcAesChallenge))"))));
    uint8_t* pOutput    __attribute__((ASPIRE("protection(wbc,      role(output),    size(0),
label(WbcAesChallenge))"))));
    int i;

    pInputBuffer = &inputBuffer;
    pInputBuffer->pBuffer = in_key;
    pInputBuffer->size = KEY_LENGTH;

    pInput = in_key;
    pOutput = encrypted_key_from_input;

    /* Decrypting instead of encrypting because (potentially) of a bug(?) in the WBC/OpenSSL */
    _Pragma("ASPIRE begin protection (wbc, algorithm(aes), operation(decrypt), mode(CBC),
label(WbcAesChallenge))")
    decryptBuffer(pInput, KEY_LENGTH, pOutput);
    _Pragma("ASPIRE end")

    if (memcmp(encrypted_key_from_input, encrypted_key, KEY_LENGTH) == 0)
        return 1 /* true */;
    else
        return 0; /* false */
    _Pragma("ASPIRE end") /* obfuscations, call check */
    _Pragma("ASPIRE end") /* CG */
}

```

```

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    /* Validate the arguments */
    validateArguments(argc, argv);

    const char* in_key = argv[1];
    if (doCheck(in_key))
        printf("Success!\n");
    else
        printf("Wrong key.\n");

    _Pragma("ASPIRE end") /* obfuscations, call check */
    _Pragma("ASPIRE end") /* CG */
} /* main */

#ifdef __cplusplus
} /* C++ */
#endif /* C++ */

```

```
/* wbc_aes_challenge.c */
```

A.6 Challenge 6

```

/* C-standard headers */
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>

/* Linux headers */
#include <unistd.h>

#include "common.h"
#include "key.h"

#define WAITLOOP iii = 0;\
    while(++iii != 0)\
    {\
        jjj = 0;\
        while(++jjj != 0);\
    }

#define SLEEPYTIME 999999

```

```

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE begin protection(remote_attestation, static_ra(RW_NORMAL, HF_SHA256, NI_1, NG_1,
MA_1, DS_1), label(first_attestator), frequency(10))")
    _Pragma("ASPIRE end")

    _Pragma("ASPIRE begin protection(remote_attestation, static_ra_region, attestator(first_attestator),
attest_at_startup(true))")
    _Pragma("ASPIRE begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    /* Validate the arguments */
    validateArguments(argc, argv);

    const char* input = argv[1];
    bool res = true;
    volatile uint64_t iii, jjj;

    /* Compare the characters in an extremely slow manner, so that attackers are forced to tamper */
    res = res && (protected_key[16] == input[16]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[15] == input[15]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[55] == input[55]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[30] == input[30]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[35] == input[35]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[40] == input[40]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[6] == input[6]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[39] == input[39]);
    sleep(SLEEPYTIME);
    WAITLOOP
    res = res && (protected_key[28] == input[28]);
    sleep(SLEEPYTIME);

```



```
WAITLOOP
res = res && (protected_key[38] == input[38]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[10] == input[10]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[26] == input[26]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[51] == input[51]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[1] == input[1]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[42] == input[42]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[36] == input[36]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[13] == input[13]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[34] == input[34]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[37] == input[37]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[53] == input[53]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[44] == input[44]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[48] == input[48]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[63] == input[63]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[46] == input[46]);
sleep(SLEEPYTIME);
```

```

WAITLOOP
res = res && (protected_key[57] == input[57]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[24] == input[24]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[2] == input[2]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[14] == input[14]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[12] == input[12]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[4] == input[4]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[41] == input[41]);
sleep(SLEEPYTIME);
WAITLOOP
res = res && (protected_key[11] == input[11]);
sleep(SLEEPYTIME);
_Pragma("ASPIRE end") /* obfuscations, call check */
_Pragma("ASPIRE end")
_Pragma("ASPIRE end") /* CG */

_Pragma("ASPIRE begin protection(code_mobility, status(mobile), data(mobile))")
res = res && (protected_key[17] == input[17]);
res = res && (protected_key[20] == input[20]);
res = res && (protected_key[19] == input[19]);
res = res && (protected_key[25] == input[25]);
res = res && (protected_key[5] == input[5]);
res = res && (protected_key[33] == input[33]);
res = res && (protected_key[43] == input[43]);
res = res && (protected_key[23] == input[23]);
res = res && (protected_key[56] == input[56]);
res = res && (protected_key[47] == input[47]);
res = res && (protected_key[58] == input[58]);
res = res && (protected_key[7] == input[7]);
res = res && (protected_key[45] == input[45]);
res = res && (protected_key[62] == input[62]);
res = res && (protected_key[61] == input[61]);
res = res && (protected_key[54] == input[54]);

```

```

    res = res && (protected_key[3] == input[3]);
    res = res && (protected_key[60] == input[60]);
    res = res && (protected_key[31] == input[31]);
    res = res && (protected_key[9] == input[9]);
    res = res && (protected_key[29] == input[29]);
    res = res && (protected_key[8] == input[8]);
    res = res && (protected_key[50] == input[50]);
    res = res && (protected_key[27] == input[27]);
    res = res && (protected_key[18] == input[18]);
    res = res && (protected_key[59] == input[59]);
    res = res && (protected_key[21] == input[21]);
    res = res && (protected_key[52] == input[52]);
    res = res && (protected_key[49] == input[49]);
    res = res && (protected_key[0] == input[0]);
    res = res && (protected_key[22] == input[22]);
    res = res && (protected_key[32] == input[32]);
    _Pragma("ASPIRE end")

COMPILER_BARRIER();

_Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
_Pragma("ASPIRE
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
/* Print out the result */
if (res)
    printf("Success!\n");
else
    printf("Wrong key.\n");
_Pragma("ASPIRE end") /* obfuscations, call check */
_Pragma("ASPIRE end") /* CG */
}

```

A.7 Challenge 7

```

/* C-standard headers */
#include <stdio.h>

#include "common.h"
#include "key.h"

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")

```



```

/* Validate the arguments */
validateArguments(argc, argv);

/* We will calculate the two results: one that should always be 0 and one that should always be 1 */
int res0 = 0;
int res1 = 1;
char* input = argv[1];
_Pragma("ASPIRE end") /* obfuscations, call check */

COMPILER_BARRIER();
_Pragma("ASPIRE begin protection(anti_debugging)")
if (protected_key[55] == input[55])
{
    res0 = res0 || !(protected_key[54] == input[54]);
    res0 = res0 || !(protected_key[4] == input[4]);
    res0 = res0 || !(protected_key[56] == input[56]);
    res0 = res0 || !(protected_key[1] == input[1]);
    res0 = res0 || !(protected_key[40] == input[40]);
    res0 = res0 || !(protected_key[63] == input[63]);
    res0 = res0 || !(protected_key[58] == input[58]);
}
else
    res0 = 1;
_Pragma("ASPIRE end")
COMPILER_BARRIER();
_Pragma("ASPIRE
protection(obfuscations(enable_obfuscation(branch_function:percent_apply=75)))")
if (protected_key[15] == input[15])
{
    res1 = res1 && (protected_key[34] == input[34]);
    res1 = res1 && (protected_key[61] == input[61]);
    res1 = res1 && (protected_key[48] == input[48]);
    res1 = res1 && (protected_key[31] == input[31]);
    res1 = res1 && (protected_key[52] == input[52]);
    res1 = res1 && (protected_key[59] == input[59]);
    res1 = res1 && (protected_key[35] == input[35]);
}
else
    res1 = 0;
if (protected_key[12] == input[12])
{
    res0 = res0 || !(protected_key[37] == input[37]);
    res0 = res0 || !(protected_key[20] == input[20]);
    res0 = res0 || !(protected_key[62] == input[62]);
    res0 = res0 || !(protected_key[43] == input[43]);
    res0 = res0 || !(protected_key[32] == input[32]);
}

```

begin




```

    res0 = res0 || !(protected_key[16] == input[16]);
    res0 = res0 || !(protected_key[44] == input[44]);
}
else
    res0 = 1;
if (protected_key[7] == input[7])
{
    res1 = res1 && (protected_key[41] == input[41]);
    res1 = res1 && (protected_key[21] == input[21]);
    res1 = res1 && (protected_key[28] == input[28]);
    res1 = res1 && (protected_key[50] == input[50]);
    res1 = res1 && (protected_key[46] == input[46]);
    res1 = res1 && (protected_key[30] == input[30]);
    res1 = res1 && (protected_key[45] == input[45]);
}
else
    res1 = 0;
if (protected_key[27] == input[27])
{
    res1 = res1 && (protected_key[19] == input[19]);
    res1 = res1 && (protected_key[38] == input[38]);
    res1 = res1 && (protected_key[3] == input[3]);
    res1 = res1 && (protected_key[49] == input[49]);
    res1 = res1 && (protected_key[26] == input[26]);
    res1 = res1 && (protected_key[8] == input[8]);
    res1 = res1 && (protected_key[2] == input[2]);
}
else
    res1 = 0;
if (protected_key[13] == input[13])
{
    res0 = res0 || !(protected_key[18] == input[18]);
    res0 = res0 || !(protected_key[57] == input[57]);
    res0 = res0 || !(protected_key[25] == input[25]);
    res0 = res0 || !(protected_key[11] == input[11]);
    res0 = res0 || !(protected_key[47] == input[47]);
    res0 = res0 || !(protected_key[14] == input[14]);
    res0 = res0 || !(protected_key[23] == input[23]);
}
else
    res0 = 1;
if (protected_key[0] == input[0])
{
    res1 = res1 && (protected_key[9] == input[9]);
    res1 = res1 && (protected_key[36] == input[36]);

```

```

    res1 = res1 && (protected_key[24] == input[24]);
    res1 = res1 && (protected_key[6] == input[6]);
    res1 = res1 && (protected_key[42] == input[42]);
    res1 = res1 && (protected_key[5] == input[5]);
    res1 = res1 && (protected_key[10] == input[10]);
}
else
    res1 = 0;
if (protected_key[17] == input[17])
{
    res0 = res0 || !(protected_key[51] == input[51]);
    res0 = res0 || !(protected_key[60] == input[60]);
    res0 = res0 || !(protected_key[33] == input[33]);
    res0 = res0 || !(protected_key[29] == input[29]);
    res0 = res0 || !(protected_key[53] == input[53]);
    res0 = res0 || !(protected_key[22] == input[22]);
    res0 = res0 || !(protected_key[39] == input[39]);
}
else
    res0 = 1;

/* Print out the result */
int res = res0 || !res1;
if (res == 0)
    printf("Success!\n");
else
    printf("Wrong key.\n");
_Pragma("ASPIRE end") /* obfuscations, call check */
_Pragma("ASPIRE end") /* CG */
}

```

A.8 Challenge 8

```

/* C-standard headers */
#include <stdio.h>
#include <stdlib.h>

#include "common.h"
#include "key.h"

void doCheck(char key, int iiii)
{

    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE begin protection(barrier_slicing, criterion(res), label(keys))")

```

```

    _Pragma("ASPIRE begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")

    int res = (key != protected_key[iii]);

    if (res != 0)
    {
        printf("Wrong key.\n");
        exit(0);
    }

    _Pragma("ASPIRE end") /* obfuscations, call check */
    _Pragma("ASPIRE end")
    _Pragma("ASPIRE end") /* CG */
        return;
}

int main(int argc, char* argv[])
{
    _Pragma("ASPIRE begin protection(guarded_region,label(generic_region))")
    _Pragma("ASPIRE begin
protection(obfuscations(enable_obfuscation(opaque_predicate:percent_apply=75,branch_function:percent_ap
ply=75,flatten_function))), protection(call_stack_check)")
    /* Validate the arguments */
    validateArguments(argc, argv);

    /* Compare the keys character per character */
    const char* in_key = argv[1];
    int iii;
    for (iii = 0; iii < KEY_LENGTH; iii++)
    {
        doCheck(in_key[iii], iii);
    }

    /* If we got to the end, it's a success */
    printf("Success!\n");
    _Pragma("ASPIRE end") /* obfuscations, call check */
    _Pragma("ASPIRE end") /* CG */
}

```