Study Project : Study of compiler options influencing static binary analysis for control flow integrity verification

École Nationale Supérieure des Mines de Saint-Étienne

K. Limouri, T. Gousselot, O. Potin, J. B. Rigaud, et J. M. Dutertre

ENSMSE, Centre Micro-électronique de Provence, F-13541 Gardanne France

May 5, 2024

Abstract

My project is part of the ARSENE project at the SAS department, which focuses on how to design hardening solutions that guarantee the integrity of the control flow and control signals, but also ensure the confidentiality of instructions. The study project focuses on evaluating the impact of compilation options on IoT-Embench programs, in order to develop a strategy that ensures the functioning of the protection developed within the SAS dpt. The main objectives of this project are to list and characterize the options to be tested on a given program, and to compare the results obtained with the results of the MAFIA solution's article.

Keywords : `ARSENE, SAS, control flow integrity (CFI), compilation options, hardening schemes, MAFIA`

Abstract
[French version]

Mon projet s'inscrit dans le cadre du projet ARSENE au sein du département SAS, qui porte sur comment concevoir des solutions de durcissement qui garantissent l'intégrité du flot de contrôle et des signaux de contrôle mais assure aussi la confidentialité des instructions. Le projet d'étude est autour de l'évaluation de l'impact des options de compilation sur des programmes de IoT-Embench, afin d'avoir une stratégie qui assure le fonctionnement de la protection développée au sein du SAS. Les principaux objectifs de ce projet sont de lister et de caractériser les options à tester sur un programme donné, et de comparer les résultats obtenus avec les résultats de l'article de la solution MAFIA.

Mots clés : `ARSENE, SAS, intégrité du flot de contrôle (CFI), options de compilation, schémas de durcissement, MAFIA`

Contents

1 Introduction

In the course of my studies as an ISMIN engineer in my third and final year at École Nationale Supérieure des Mines de Saint-Étienne, I carried out a study project (PE). During the month of March, I worked full-time on this last academic project at the school. This document is a report on my work throughout the study project within the SAS department.

In this report, we will first briefly present the study project. Secondly, we will detail the realization of the project, from both a technical and a managerial point of view. Thirdly and finally, we will conclude the report with a conclusion containing the outcomes and the prospects.

2 Context and subject of the PE

2.1 Context of the study project

The PE is based on another project called ARSENE, for Architectures SEcurisées pour le Numérique Embarqué, meaning Secure Architectures for Embedded Computing. The latter project is a sort of consortium between 12 research labs, including EMSE. The researchers at EMSE work on securing the RISC-V processor, ie. hardening of the RISC-V by developing protection schemes for CFI. The state of the art in this field is given by scientists at CEA and Thales DIS, who developed a protection against fault injection attacks known as MAFIA.

2.2 Subject of the project

The software countermeasures mentioned in the previous part are central aspects of the PE. In fact, its subject leans to the software side of the protection against physical attacks by laser and electromagnetic injection. The analysis of the software discontinuities in a given program's binary code is done statically, meaning before execution. The code obtained is heavily dependent on the compilation options, which conditions the complexity level of CFI solutions. The project's aim is to study the impact of compilation options by static analysis of assembly code targeting RISC-V, in order to enhance the hardening solution of SAS.

3 Project's realization

3.1 Project management of the PE

First of all, the overall progression of the project was clear and smooth. I had weekly meetings or updates with my project's supervisors, in which we discussed what was done, what was to be done, and tackled the challenges encountered. The figure [1], placed in the Annex section, is a Gantt chart which demonstrates the work achieved during this PE from 03/04 to 03/29.
Regarding the harmonization part of the project management, I was alone in the PE. So for the task distribution, I did everything myself with the help and the assistance of my supervisors.

3.2 Technical realization of the project

Environment setup

We launched the project by implementing the work environment and putting in place all the tools that I may need in order to achieve the goals PE. We used the school's servers, notably the domain name tallinn.emse.fr, to have a remote entry point as well as a safe back-up stored in the cloud. We faced some difficulties when establishing the remote access due to changes in my account's access rights. At the same time, I configured my virtual machine, denoted VM in the figure [2] that details how our architecture works. The configuration of the VM was done prior to the project, but I had to do it again due to start-up errors. After that, we tackled the Gitlab setup which was a little bit easier, as I was much more autonomous and only punctually guided by my direct supervisors if needed.

Technical approach

Our final goal in the PE was to perform a series of tests on programs of a Github repository, "IoT-Embench", by running a program called "flow_analysis". My approach was to use the latter program which analyzes the .hex file, in hexadecimal format, and disassembles each instruction, in order to quantify and qualify the discontinuities. I also had to take part in understanding and using the compilation options cited in MAFIA's solution briefly mentioned before. The most relevant compilation option was `-fno-jump-tables` for our programs, but the other options such us `-ffunction-section`, which removes "dead code", were also tested on a local test suite. I've started to initialize this test suite by creating dummy examples which implement simple addition or functions that were never called. In order to see differences in the compiler's behavior, my approach was to add a series of `switch_case` implementations, specifically eight, from one case to eight cases.

Results obtained

Locally, I observed that the threshold value of the difference in compilation for the `switch_case` sequence is five cases. Meaning that the `switch_case` is compiled into branch instructions if there are only four cases or less, and into jump instructions for five cases or more. The two compilation options of the MAFIA's article were tested in the programs I created at the VM. The first option, `-fno-jump-tables`, enabled us to remove all the indirect jump instructions that didn't correspond to a function return. And the second one, `-ffunction-section`, deleted the assembly code of the function in the program `mycode`, which wasn't used but only declared.

On the remote side, I started by pulling the Github repository on my local machine. I've adapted and automated the two scripts, `compile_script` and `hex2dias_hex`, written in bash and python which compiles the source code and formats the hexadecimal respectively. I've noticed that, due to the complexity of the programs, the option `-fno-jump-tables` doesn't make a difference in general as we can see in the tables [1], [2], [3], [4], and [5]. Finally, we can observe that the numbers in the last chart 5 are very similar, which is quite normal because these type of instructions aren't reccurent.

4 Conclusions and perspectives

Based on the results at hand in the context of my study project, I can conclude by saying that the optimization and the security vectors are going in opposite directions. We clearly see that when we optimize more (-O1, -O2...), the number of jumps increases because they replace branches instructions in general. However, in order to apply the solution developed within SAS, the jump usage should be minimal due to precedence and succession issues that require application of patch.

As far as the project's goals are concerned, I couldn't realize everything required, as listed in the subject of the project. And what remains to be done is to establish the control flow graph and to compute the number of basic blocks, which are the sections of code that are delimited by two instructions discontinuity.

5 Annexes

A Illustrations & Images

A.0.1 Figure n°1

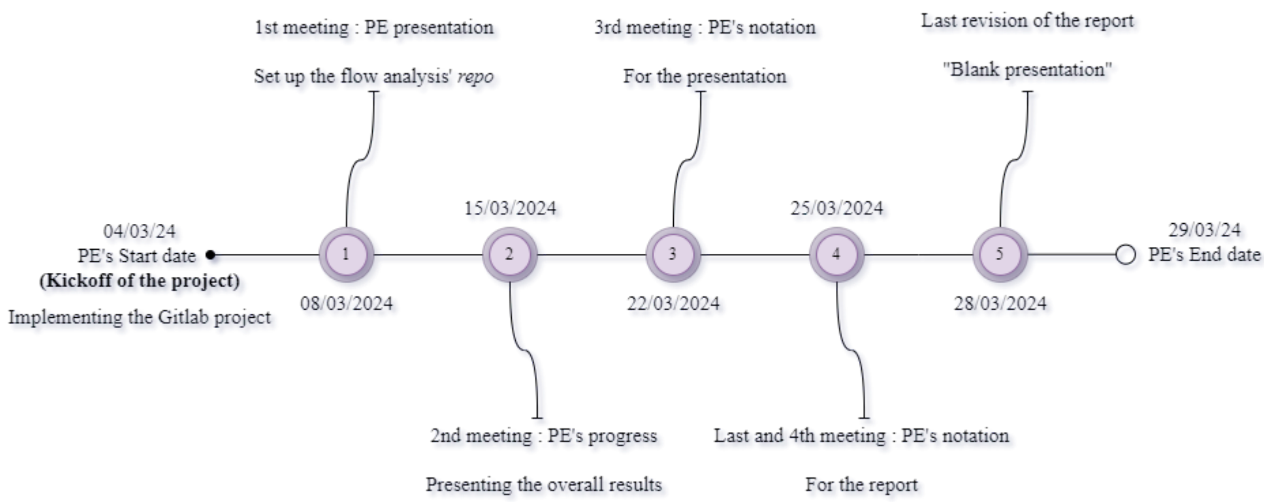

Figure 1: Gantt chart of the study project

A.0.2 Figure n°2

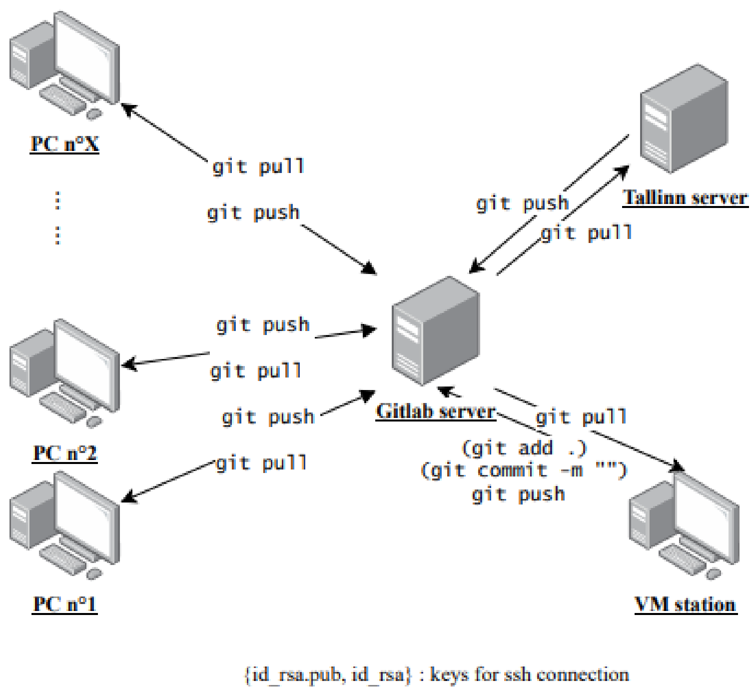

Figure 2: Representation of the functioning of VM, Tallinn, and Gitlab servers

B Charts & Graphics

Option/Program	-O1	-O2	-O3	-Os	-fno-jump-tables
crc32	392	405	409	402	392
cubic	1344	1360	1362	1352	1357
edn	253	254	268	259	253
fibonacci	393	417	420	391	391
huffbench	440	495	506	464	440
matmult-int	242	244	256	249	241
md5sum	405	409	412	409	407
minver	642	650	653	652	643
nbody	552	571	582	579	571
nettle-aes	262	263	270	269	262
nettle-sha256	273	276	283	275	275
nsichneu	865	866	870	865	867
picojpeg	608	609	617	608	608
primecount	399	408	412	410	110
sglibcombined	761	863	1096	938	790
slre	540	575	604	588	543
st	546	583	600	572	546
statemate	423	429	455	437	441
tarfind	403	412	426	419	405
ud	338	307	298	302	326
verifypin_0	371	355	338	367	370
wikisort	836	869	880	872	836

Table 1: Number of branch instructions

Option/Program	-O1	-O2	-O3	-Os	-fno-jump-tables
crc32	132	131	130	131	132
cubic	256	258	261	257	259
edn	92	90	87	88	92
fibonacci	121	121	123	123	123
huffbench	144	145	152	150	144
matmult-int	90	88	84	86	89
md5sum	144	146	147	145	144
minver	120	132	154	149	120
nbody	128	169	205	173	129
nettle-aes	101	101	102	102	102
nettle-sha256	108	105	104	103	108
nsichneu	84	84	82	84	81
picojpeg	311	324	331	327	311
primecount	126	125	130	129	128
sglibcombined	191	189	176	193	190
slre	150	149	150	150	150
st	135	180	201	163	171
statemate	92	90	91	91	91
tarfind	134	134	135	135	134
ud	92	91	85	87	88
verifypin_0	118	104	92	94	118
wikisort	206	213	227	220	206

Table 2: Number of direct jump instructions that correspond to a function call

Option/Program	-O1	-O2	-O3	-Os	-fno-jump-tables
crc32	205	206	207	206	205
cubic	581	583	585	584	581
edn	148	147	146	147	148
fibonacci	207	210	214	210	207
huffbench	227	227	228	228	227
matmult-int	142	142	144	143	142
md5sum	210	210	211	211	210
minver	302	303	305	304	302
nbody	279	275	273	277	273
nettle-aes	153	154	158	157	152
nettle-sha256	157	156	156	156	157
nsichneu	644	648	652	649	643
picojpeg	325	328	335	321	332
primecount	210	209	210	207	210
sglibcombined	319	367	387	375	346
slre	272	271	271	271	272
st	273	275	280	274	271
statemate	186	191	206	198	186
tarfind	211	211	211	211	211
ud	160	158	154	155	157
verifypin_0	194	194	190	192	191
wikisort	400	401	403	402	400

Table 3: Number of direct jump instructions that that doesn't correspond to a function call

Option/Program	-O1	-O2	-O3	-Os	-fno-jump-tables
crc32	125	126	126	126	125
cubic	161	161	161	161	161
edn	102	102	102	102	102
fibonacci	104	105	105	105	104
huffbench	128	128	133	128	128
matmult-int	99	99	98	99	98
md5sum	126	127	127	127	126
minver	129	130	131	130	129
nbody	119	119	120	120	119
nettle-aes	101	101	102	102	101
nettle-sha256	105	98	96	97	99
nsichneu	91	92	91	92	90
picojpeg	183	181	185	182	184
primecount	125	124	125	124	124
sglibcombined	242	254	263	260	249
slre	141	138	136	136	140
st	124	132	129	127	126
statemate	133	130	129	131	133
tarfind	123	124	125	125	124
ud	103	100	94	97	101
verifypin_0	109	104	103	104	105
wikisort	177	175	173	174	177

Table 4: Number of indirect jump instructions that correspond to a function return

Option/Program	-O1	-O2	-O3	-Os	-fno-jump-tables
crc32	27	27	27	27	27
cubic	29	29	29	29	29
edn	16	16	16	16	16
fibonacci	25	25	25	25	25
huffbench	27	27	27	27	27
matmult-int	16	16	16	16	16
md5sum	27	27	27	27	27
minver	18	18	18	18	18
nbody	17	17	17	17	17
nettle-aes	16	16	16	16	16
nettle-sha256	18	18	16	18	18
nsichneu	16	16	16	16	16
picojpeg	34	34	34	34	34
primecount	27	27	27	27	27
sglibcombined	32	32	32	32	32
slre	27	27	27	27	27
st	17	17	17	17	17
statemate	16	16	16	16	16
tarfind	27	27	27	27	27
ud	16	16	16	16	16
verifypin_0	25	25	25	25	25
wikisort	58	58	58	58	58

Table 5: Number of indirect jump instructions that doesn't correspond to a function return