Analyse and binary transformation

Guillaume Bouffard

Analyse and binary transformation

Guillaume Bouffard

Université de Limoges

technicolor

Outline

1. **Introduction**

2. **Profiling step**

3. **Translation step**

4. **Binary Modification**

5. **Proof Of Concept**

6. **Conclusion**

Outline

1. **Introduction**
   - Technicolor
   - My Internship
2. Profiling step
3. Translation step
4. Binary Modification
5. Proof Of Concept
6. Conclusion

Technicolor Security and Content Protection Labs

Technicolor

- Creating, managing and delivering video
- For the Communication, Media and Entertainment industries.

Their works

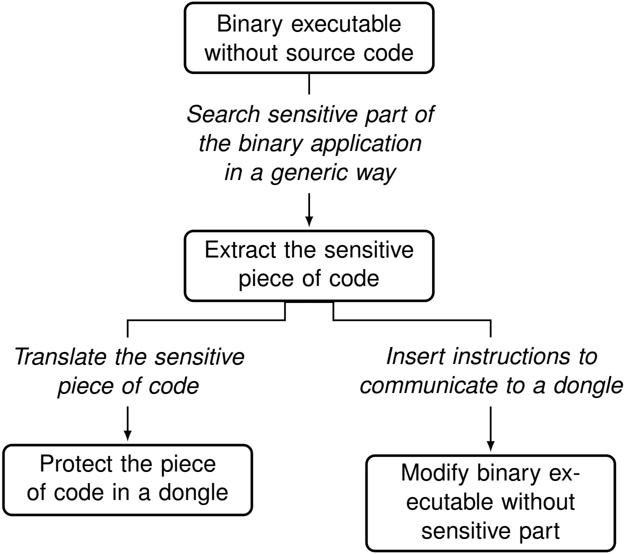
- Cryptography
- Signal processing for security
- Content protection (DRM)
- Network security
- Tamper resistance

Context

The Internship Context

- Illegal software duplication and intellectual property theft
- Software protection VS hardware protection
- Hardware protection?

Subject



Binary executable without source code

Search sensitive part of the binary application in a generic way

Extract the sensitive piece of code

Translate the sensitive piece of code

Protect the piece of code in a dongle

Insert instructions to communicate to a dongle

Modify binary executable without sensitive part

Motivation

What was my motivation?

- A blend of compilation and smart card problems
- Discover the computer science underground
- Think on a research subject

Outline

1 Introduction

2 Profiling step

3 Translation step

4 Binary Modification

5 Proof Of Concept

6 Conclusion

Application Profiling

What do you want to find?

- Each executed binary piece of code
- Found the **sensitive** parts

What can tools do that?

- OProfile
- Valgrind

Outline

1 Introduction

2 Profiling step

3 Translation step

4 Binary Modification

5 Proof Of Concept

6 Conclusion

Translation step

The Goal

- Protect the sensitive pieces of code in a dongle
- These pieces of code are executed by the dongle

$=>$ **A solution: UQBT**

Outline

1 Introduction

2 Profiling step

3 Translation step

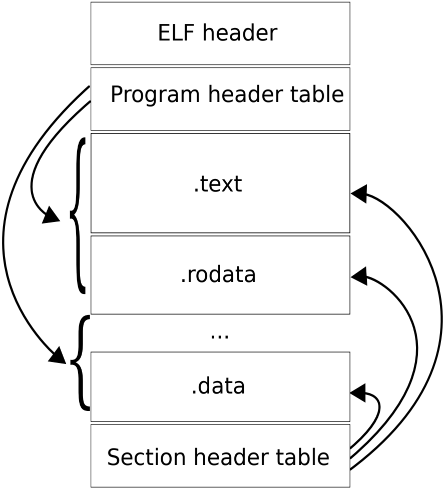
4 **Binary Modification**

- ELF Format
- Diablo
- Samples

5 Proof Of Concept

6 Conclusion

Executable and Linkable Format



ELF header

Program header table

.text

.rodata

...

.data

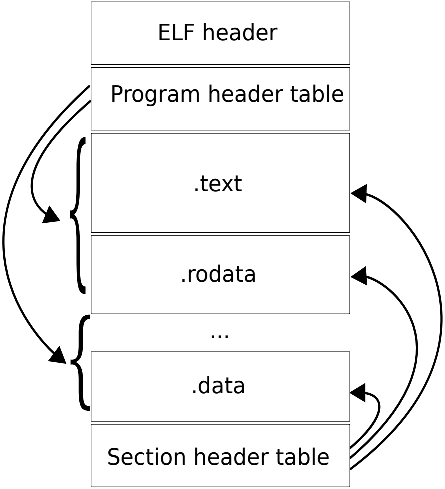
Section header table

Executable and Linkable Format

- Used by Unices & GNU/Linux
- Each section are linked

How can I modify this file format?

Executable and Linkable Format

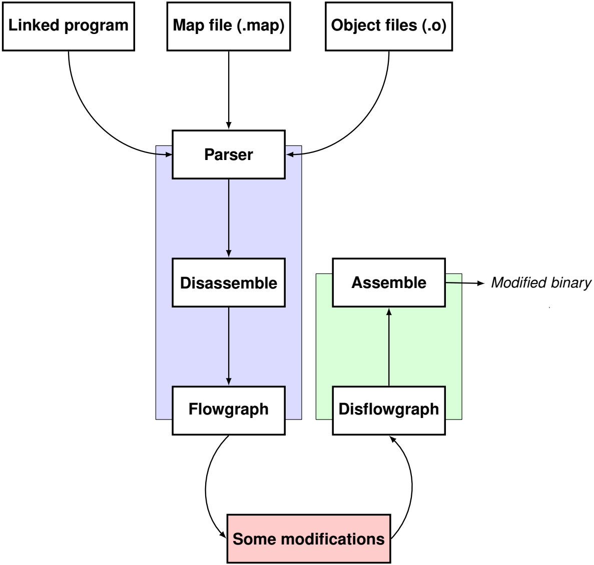


Executable and Linkable Format

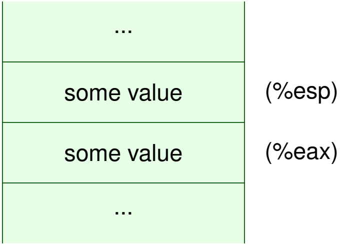
- Used by Unices & GNU/Linux
- Each section are linked

How can I modify this file format?

Diablo


Linked program
Map file (.map)
Object files (.o)
Parser
Disassemble
Flowgraph
Assemble
Modified binary
Disflowgraph
Some modifications

Brief overview of assembler



...

some value (%esp)

some value (%eax)

...

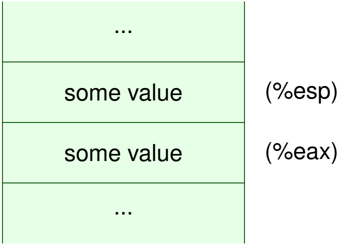
```
$ ./hello_world
hello world
```

```
#include <stdio.h>

int main ( void ) {
	printf("hello world\n");
	return EXIT_SUCCESS ;
	}
```

```
<main>:
	mov DWORD PTR [esp],0x8096188
	call 80486c0 <_IO_printf>
	mov eax,0x0
	leave
	ret
```

Brief overview of assembler


...

some value (%esp)

some value (%eax)

...

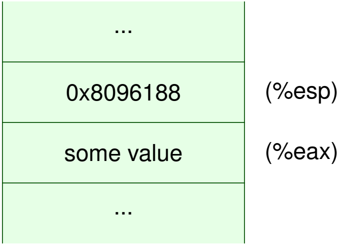
```
#include <stdio.h>

int main ( void ) {
⇒printf("hello world\n");
⇒return EXIT_SUCCESS ;
⇒}
```

```
<main>:
⇒mov DWORD PTR [esp],0x8096188
⇒call 80486c0 <_IO_printf>
⇒mov eax,0x0
⇒leave
⇒ret
```

```
$ ./hello_world
hello world
```

Brief overview of assembler



...

0x8096188 (%esp)

some value (%eax)

...

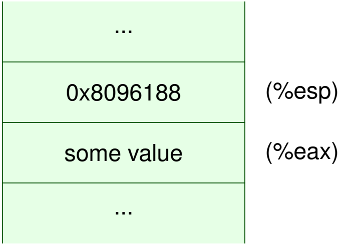
```
#include <stdio.h>

int main ( void ) {
⇒printf("hello world\n");
 return EXIT_SUCCESS ;
 }
```

```
<main>:
⇒mov DWORD PTR [esp],0x8096188
 call 80486c0 <_IO_printf>
 mov eax,0x0
 leave
 ret
```

```
$ ./hello_world
hello world
```

Brief overview of assembler



...

0x8096188 (%esp)

some value (%eax)

...

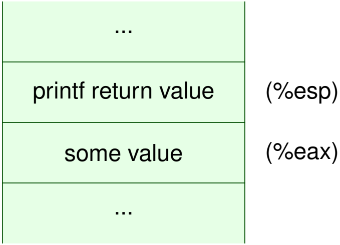
```
$ ./hello_world
hello world
```

```
#include <stdio.h>

int main ( void ) {
⇒printf("hello world\n");
⇒return EXIT_SUCCESS ;
⇒}
```

```
<main>:
⇒mov DWORD PTR [esp],0x8096188
⇒call 80486c0 <_IO_printf>
⇒mov eax,0x0
⇒leave
⇒ret
```

Brief overview of assembler



...

printf return value (%esp)

some value (%eax)

...

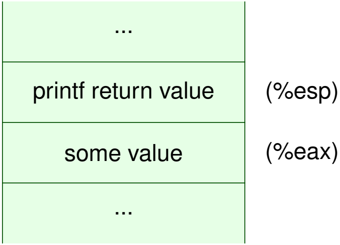
```
$ ./hello_world
hello world
```

```
#include <stdio.h>

int main ( void ) {
⇒printf("hello world\n");
  return EXIT_SUCCESS ;
  }
```

```
<main>:
  mov DWORD PTR [esp],0x8096188
⇒call 80486c0 <_IO_printf>
  mov eax,0x0
  leave
  ret
```

Brief overview of assembler



...

printf return value (%esp)

some value (%eax)

...

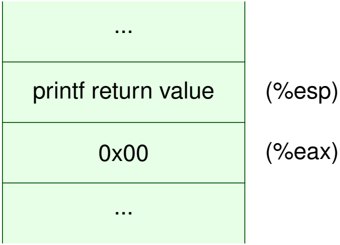
```
#include <stdio.h>

int main ( void ) {
  printf("hello world\n");
⇒return EXIT_SUCCESS ;
  }
```

```
<main>:
  mov DWORD PTR [esp],0x8096188
  call 80486c0 <_IO_printf>
⇒mov eax,0x0
  leave
  ret
```

```
$ ./hello_world
hello world
```

Brief overview of assembler



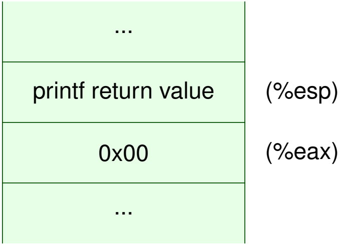
```
#include <stdio.h>

int main ( void ) {
  printf("hello world\n");
⇒return EXIT_SUCCESS ;
  }
```

```
<main>:
  mov DWORD PTR [esp],0x8096188
  call 80486c0 <_IO_printf>
⇒mov eax,0x0
  leave
  ret
```

```
$ ./hello_world
hello world
```

Brief overview of assembler



...

printf return value (%esp)

0x00 (%eax)

...

```
$ ./hello_world
hello world
```

```
#include <stdio.h>

int main ( void ) {
  printf("hello world\n");
  return EXIT_SUCCESS ;
⇒}
```

```
<main>:
  mov DWORD PTR [esp],0x8096188
  call 80486c0 <_IO_printf>
  mov eax,0x0
⇒leave
  ret
```

Brief overview of assembler

...	
printf return value	(%esp)
0x00	(%eax)
...	

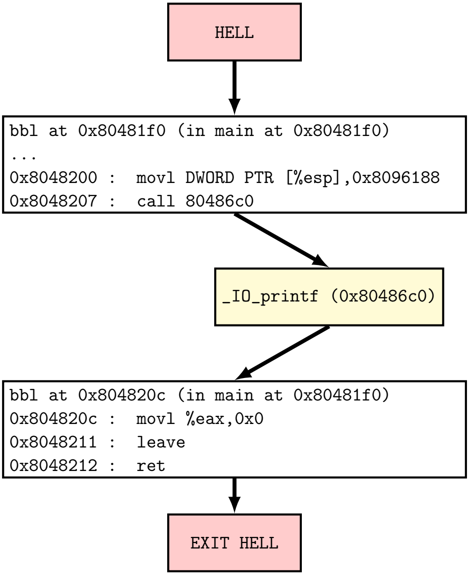
```
#include <stdio.h>

int main ( void ) {
  printf("hello world\n");
  return EXIT_SUCCESS ;
⇒}
```

```
$ ./hello_world
hello world
```

```
<main>:
  mov DWORD PTR [esp],0x8096188
  call 80486c0 <_IO_printf>
  mov eax,0x0
  leave
⇒ret
```

Hello World



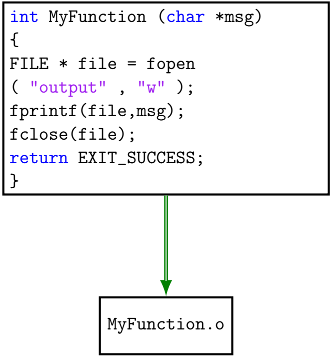
HELL

```
bbl at 0x80481f0 (in main at 0x80481f0)
...
0x8048200 :  movl DWORD PTR [%esp],0x8096188
0x8048207 :  call 80486c0
```

_IO_printf (0x80486c0)

```
bbl at 0x804820c (in main at 0x80481f0)
0x804820c :  movl %eax,0x0
0x8048211 :  leave
0x8048212 :  ret
```

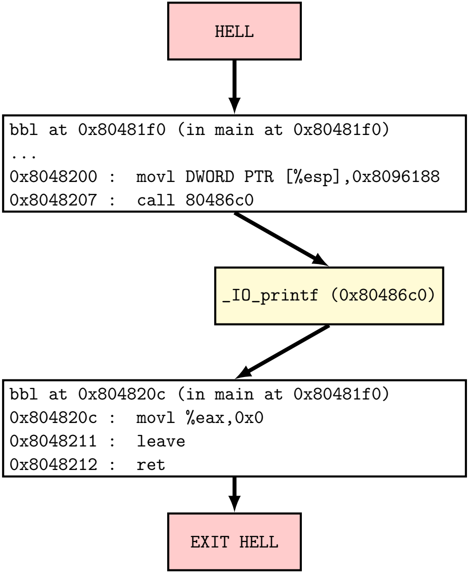
EXIT HELL



```
int MyFunction (char *msg)
{
FILE * file = fopen
( "output" , "w" );
fprintf(file,msg);
fclose(file);
return EXIT_SUCCESS;
}
```

MyFunction.o

Hello World

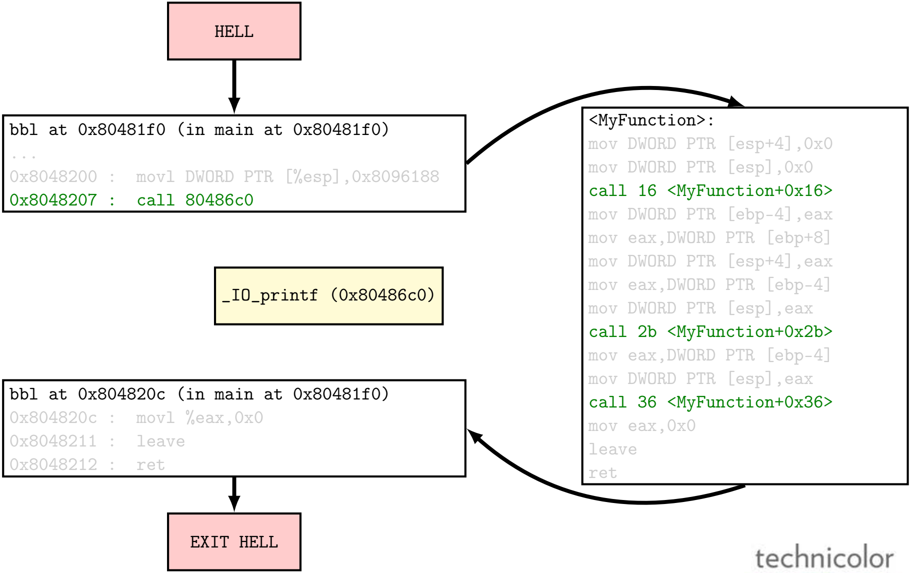


```
bbl at 0x80481f0 (in main at 0x80481f0)
...
0x8048200 :  movl DWORD PTR [%esp],0x8096188
0x8048207 :  call 80486c0
```

```
bbl at 0x804820c (in main at 0x80481f0)
0x804820c :  movl %eax,0x0
0x8048211 :  leave
0x8048212 :  ret
```

```
<MyFunction>:
mov DWORD PTR [esp+4],0x0
mov DWORD PTR [esp],0x0
call 16 <MyFunction+0x16>
mov DWORD PTR [ebp-4],eax
mov eax,DWORD PTR [ebp+8]
mov DWORD PTR [esp+4],eax
mov eax,DWORD PTR [ebp-4]
mov DWORD PTR [esp],eax
call 2b <MyFunction+0x2b>
mov eax,DWORD PTR [ebp-4]
mov DWORD PTR [esp],eax
call 36 <MyFunction+0x36>
mov eax,0x0
leave
ret
```

Hello World



HELL

```
bbl at 0x80481f0 (in main at 0x80481f0)
...
0x8048200 :  movl DWORD PTR [%esp],0x8096188
0x8048207 :  call 80486c0
```

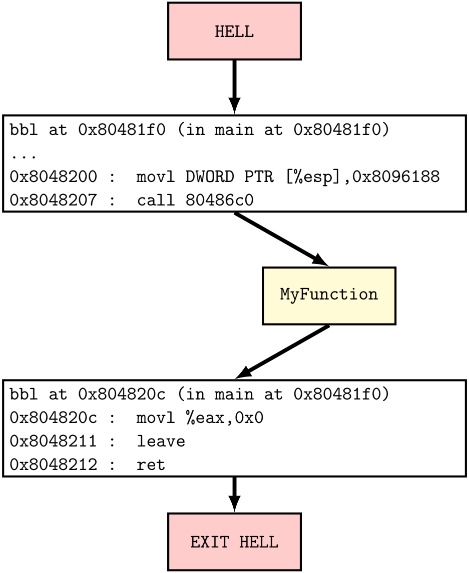
_IO_printf (0x80486c0)

```
bbl at 0x804820c (in main at 0x80481f0)
0x804820c :  movl %eax,0x0
0x8048211 :  leave
0x8048212 :  ret
```

EXIT HELL

```
<MyFunction>:
mov DWORD PTR [esp+4],0x0
mov DWORD PTR [esp],0x0
call 16 <MyFunction+0x16>
mov DWORD PTR [ebp-4],eax
mov eax,DWORD PTR [ebp+8]
mov DWORD PTR [esp+4],eax
mov eax,DWORD PTR [ebp-4]
mov DWORD PTR [esp],eax
call 2b <MyFunction+0x2b>
mov eax,DWORD PTR [ebp-4]
mov DWORD PTR [esp],eax
call 36 <MyFunction+0x36>
mov eax,0x0
leave
ret
```

CouCou World



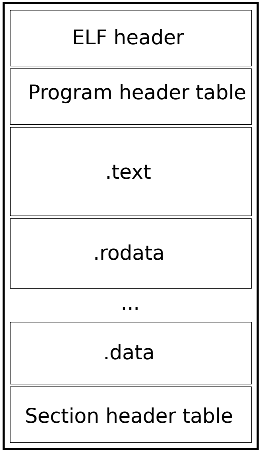
HELL

```
bbl at 0x80481f0 (in main at 0x80481f0)
...
0x8048200 :  movl DWORD PTR [%esp],0x8096188
0x8048207 :  call 80486c0
```

MyFunction

```
bbl at 0x804820c (in main at 0x80481f0)
0x804820c :  movl %eax,0x0
0x8048211 :  leave
0x8048212 :  ret
```

EXIT HELL



ELF header

Program header table

.text

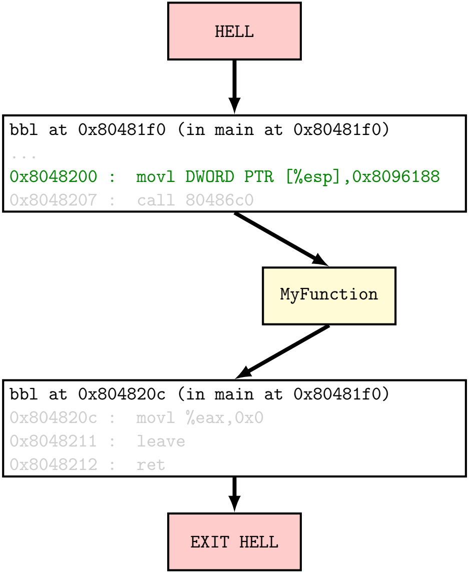
.rodata

...

.data

Section header table

CouCou World



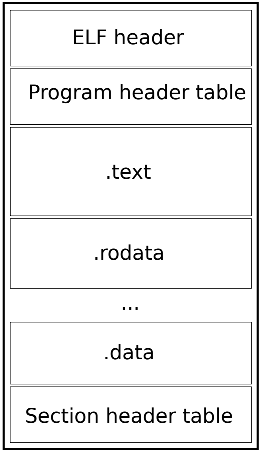
HELL

```
bbl at 0x80481f0 (in main at 0x80481f0)
...
0x8048200 :  movl DWORD PTR [%esp],0x8096188
0x8048207 :  call 80486c0
```

MyFunction

```
bbl at 0x804820c (in main at 0x80481f0)
0x804820c :  movl %eax,0x0
0x8048211 :  leave
0x8048212 :  ret
```

EXIT HELL



ELF header

Program header table

.text

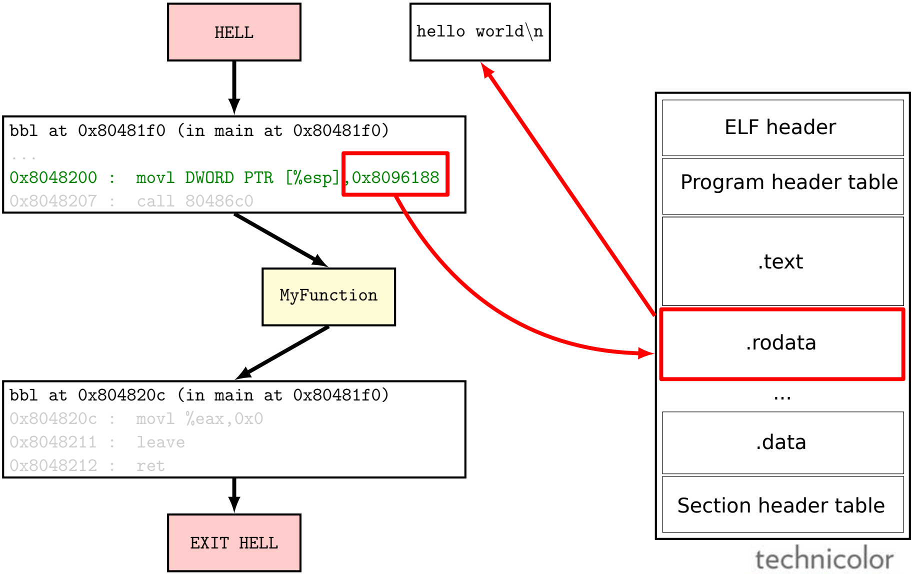
.rodata

...

.data

Section header table

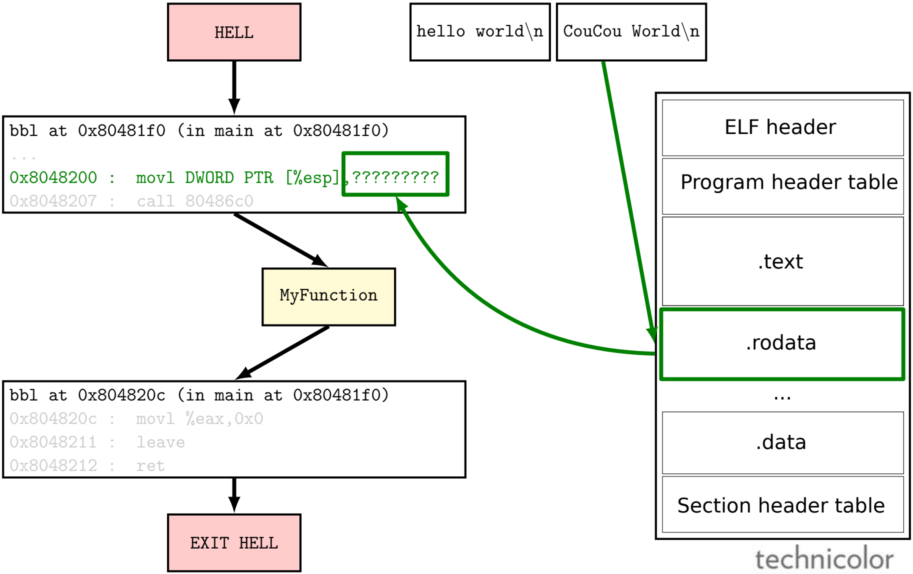
CouCou World



```
bbl at 0x80481f0 (in main at 0x80481f0)
...
0x8048200 :  movl DWORD PTR [%esp],0x8096188
0x8048207 :  call 80486c0
```

```
bbl at 0x804820c (in main at 0x80481f0)
0x804820c :  movl %eax,0x0
0x8048211 :  leave
0x8048212 :  ret
```

CouCou World



HELL

hello world\n

CouCou World\n

```
bbl at 0x80481f0 (in main at 0x80481f0)
...
0x8048200 :  movl DWORD PTR [%esp],?????????
0x8048207 :  call 80486c0
```

MyFunction

```
bbl at 0x804820c (in main at 0x80481f0)
0x804820c :  movl %eax,0x0
0x8048211 :  leave
0x8048212 :  ret
```

EXIT HELL

ELF header

Program header table

.text

.rodata

...

.data

Section header table

Outline

1 Introduction

2 Profiling step

3 Translation step

4 Binary Modification

5 **Proof Of Concept**

- Java Card Side
- Communication Binary Application ⇔ the Smart Card
- Binary Modification

6 Conclusion

Integers Multiplication

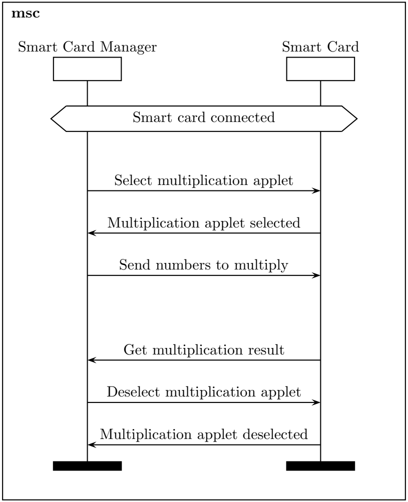
Main Idea

- Use a simple product matrix
- Make each multiplication operation on a smart card
- Search & replace each multiplication instruction

An Integers Multiplication on a Java Card

- Java Card cannot make a 32-bit number multiplication

Communication Binary Application ⇔ the Smart Card

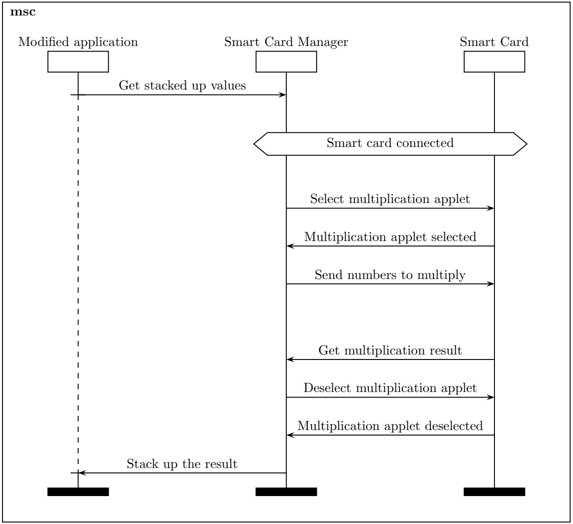


msc
Smart Card Manager
Smart Card
Smart card connected
Select multiplication applet
Multiplication applet selected
Send numbers to multiply
Get multiplication result
Deselect multiplication applet
Multiplication applet deselected

Implementation

- Using a framework made by laboratory members
- Override `libpcsc-lite` to add some features
- Just a little bit complex. . .

The Last Binary Modification with Diablo


msc
Modified application
Smart Card Manager
Smart Card
Get stacked up values
Smart card connected
Select multiplication applet
Multiplication applet selected
Send numbers to multiply
Get multiplication result
Deselect multiplication applet
Multiplication applet deselected
Stack up the result

Problems

- Diablo cannot parse the C++ framework…
- …and it cannot parse `libpcsc-lite`

Outline

1 Introduction

2 Profiling step

3 Translation step

4 Binary Modification

5 Proof Of Concept

6 **Conclusion**

Internship conclusion

Objectives accomplished

- Can found each executed instruction without source code
- Modify binary executable with Diablo

To Do list

- Realize the translation step
- Make a complete proof of concept
- Don’t use Java Card!
- Obfuscate the APDU request
- Upgrade Diablo toolchain

technicolor

Personal Conclusion

Personal impact

- Discover a private laboratory
- With a research project

The End. . .

Thank you for your attention!
Any questions?