



Obfuscator reloaded

Pascal Junod – HEIG-VD

Julien Rinaldini – HEIG-VD

Marc Romanens - EIA-FR

Jean-Roland Schuler – EIA-FR



Application Security Forum - 2012
Western Switzerland

7-8 novembre 2012 - Y-Parc / Yverdon-les-Bains
<https://www.appsec-forum.ch>

Agenda

- Context of the « Obfuscator » project
- LLVM-based obfuscation
- Playing with ARM binaries



In a Nutshell ...



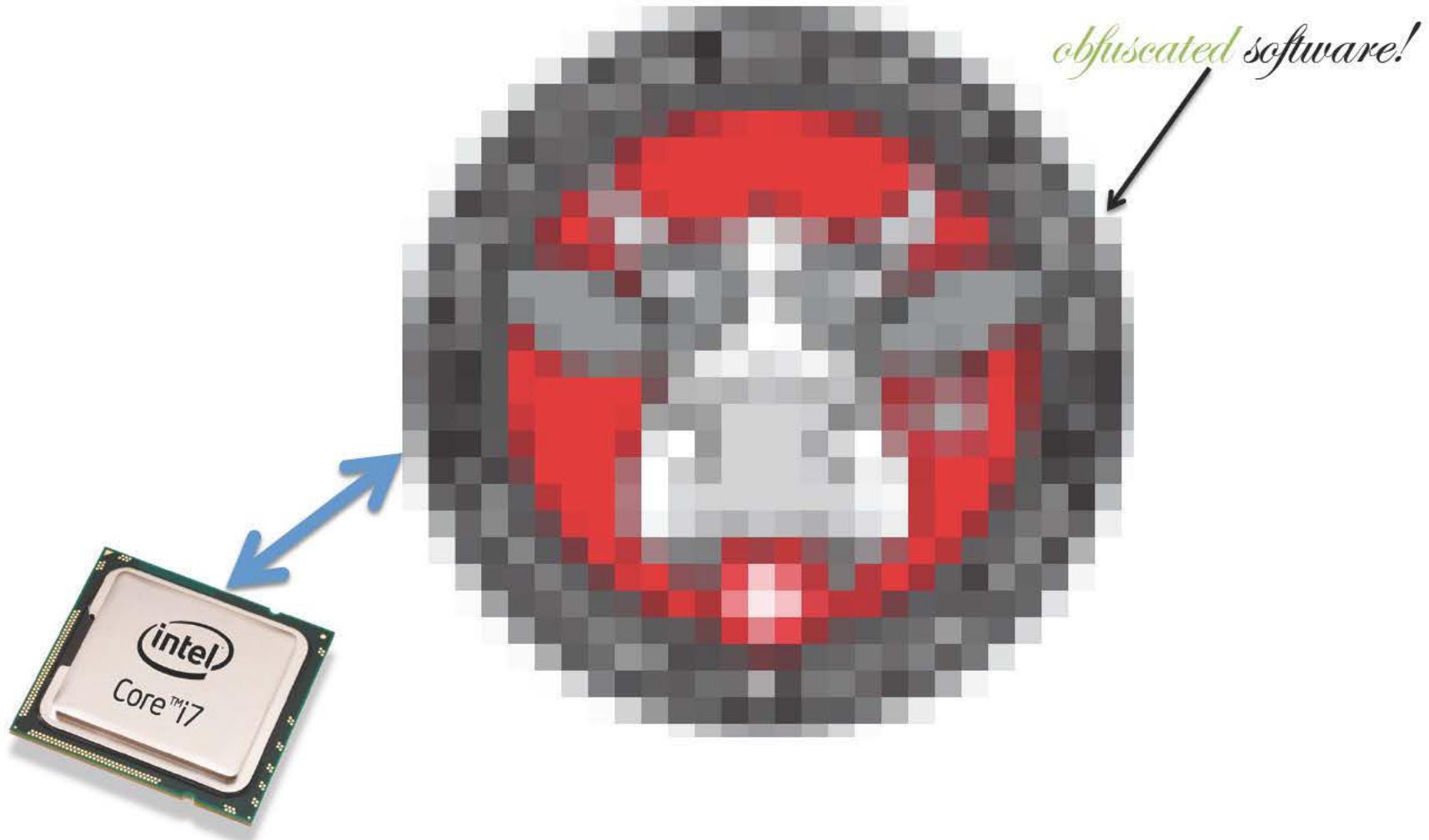
*Please see this as
unprotected software!*



In a Nutshell ...

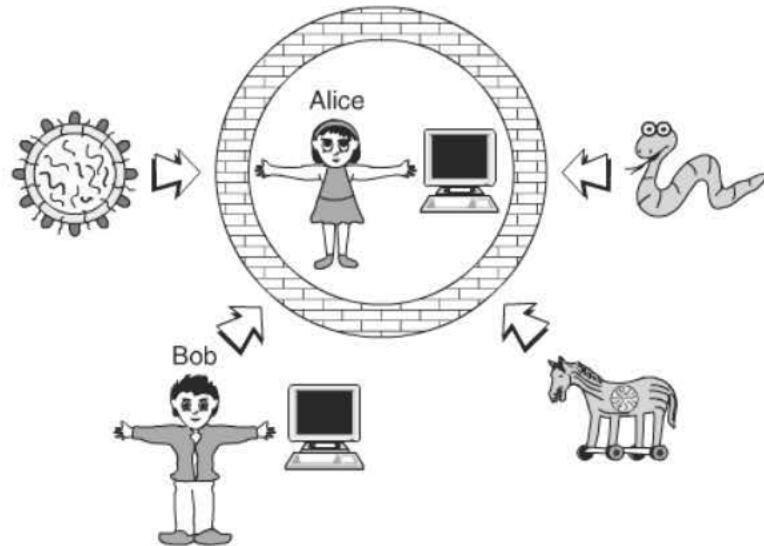


In a Nutshell ...

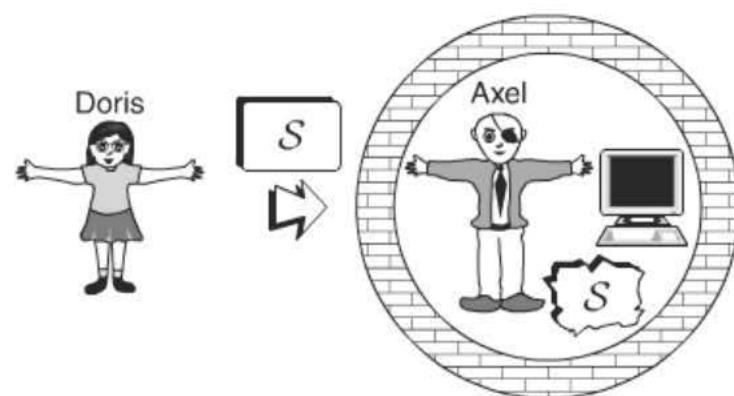




In a Nutshell ...

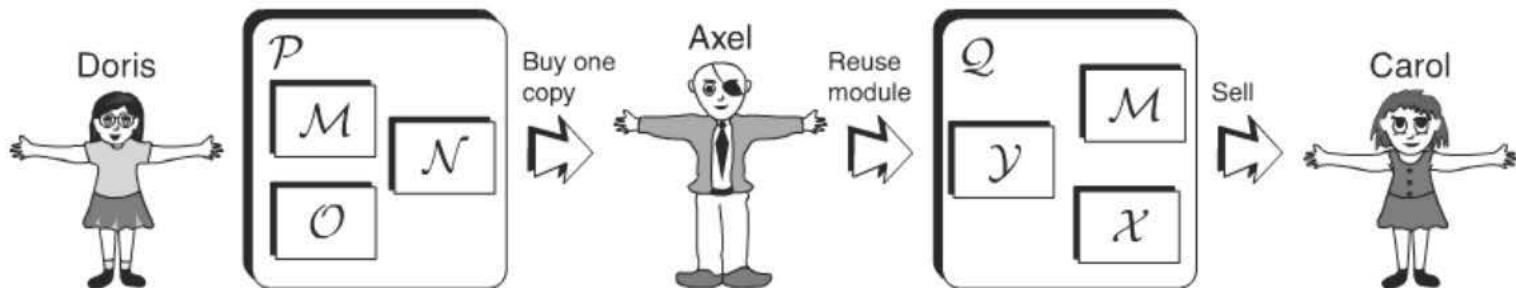


vs.



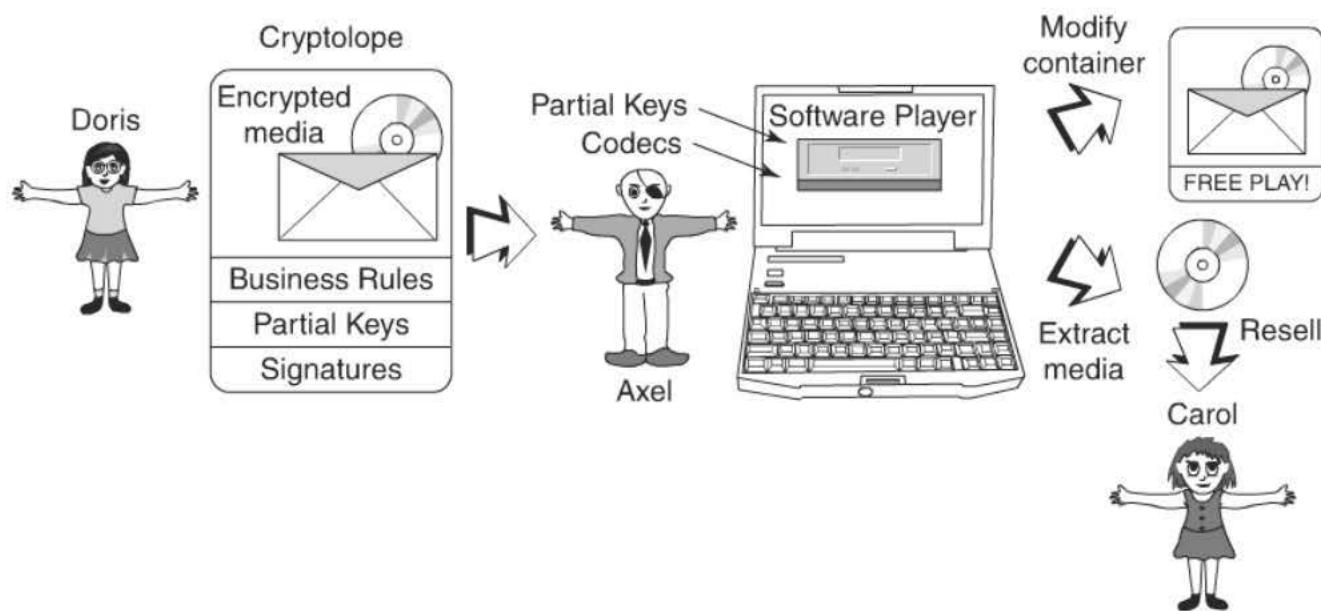
In a Nutshell ...

Malicious Reverse-Engineering



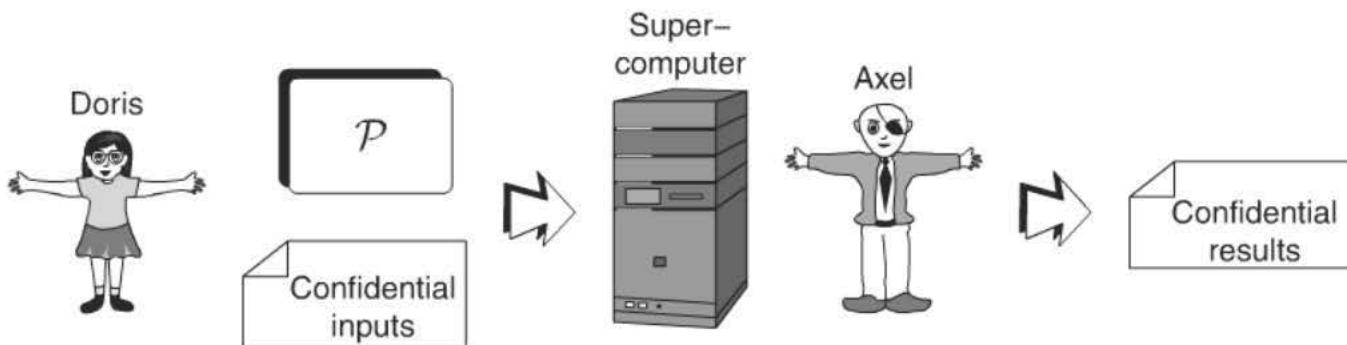
In a Nutshell ...

Digital Rights Management



In a Nutshell ...

Cloud Computing





In a Nutshell ...

- A typical attack can be decomposed in three phases:
 - Program analysis
 - Extraction of an algorithm, secret, design
 - Code modification
 - Removal of license check mechanisms
 - Distribution
 - Violation of intellectual property



In a Nutshell ...

- Defense possibilities (aka « software protection »)
 - Confusion
 - Obfuscation
 - Tamper-resistance
 - SW/HW-based protections, node-locking, ...
 - Watermark
 - SW watermarking, fingerprinting, ...



In a Nutshell ...

- According to Wikipedia, «*obfuscated code is source of machine code that has been made difficult to understand* »
- «difficult» ...
 - = costly
 - = time-consuming
 - not necessarily impossible

On the (Im)possibility of Obfuscating Programs*

Boaz Barak[†] Oded Goldreich[‡] Russell Impagliazzo[§] Steven Rudich[¶]
Amit Sahai^{||} Salil Vadhan^{**} Ke Yang^{††}

July 29, 2010

Abstract

Informally, an *obfuscator* \mathcal{O} is an (efficient, probabilistic) “compiler” that takes as input a program (or circuit) P and produces a new program $\mathcal{O}(P)$ that has the same functionality as P yet is “unintelligible” in some sense. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice’s theorem. Most of these applications are based on an interpretation of the “unintelligibility” condition in obfuscation as meaning that $\mathcal{O}(P)$ is a “virtual black box,” in the sense that anything one can efficiently compute given $\mathcal{O}(P)$, one could also efficiently compute given oracle access to P .

In this work, we initiate a theoretical investigation of obfuscation. Our main result is that, even under very weak formalizations of the above intuition, obfuscation is impossible. We prove this by constructing a family of efficient programs \mathcal{P} that are *unobfuscatable* in the sense that (a) given *any* efficient program P' that computes the same function as a program $P \in \mathcal{P}$, the “source code” P can be efficiently reconstructed, yet (b) given *oracle access* to a (randomly selected) program $P \in \mathcal{P}$, no efficient algorithm can reconstruct P (or even distinguish a certain bit in the code from random) except with negligible probability.

We extend our impossibility result in a number of ways, including even obfuscators that (a) are not necessarily computable in polynomial time, (b) only approximately preserve the functionality, and (c) only need to work for very restricted models of computation (TC⁰). We also rule out several potential applications of obfuscators, by constructing “unobfuscatable” signature schemes, encryption schemes, and pseudorandom function families.



In a Nutshell ...

```
return (int) (((((x - 2) * (x - 3) * (x - 4) * (x - 5) * (x - 6) *
    (x - 7) * (x - 8) * (x - 9) * (x - 10) * (x - 11) *
    (x - 12) * 31) /
    (((x - 2) * (x - 3) * (x - 4) * (x - 5) * (x - 6) *
    (x - 7) * (x - 8) * (x - 9) * (x - 10) * (x - 11) *
    (x - 12) + .00001)) +
    (((x - 3) * (x - 4) * (x - 5) * (x - 6) * (x - 7) *
    (x - 8) * (x - 9) * (x - 10) * (x - 11) * (x - 12) *
    (28 + z)) /
    (((x - 3) * (x - 4) * (x - 5) * (x - 6) * (x - 7) *
    (x - 8) * (x - 9) * (x - 10) * (x - 11) * (x - 12) +
    .00001)) +
    (((x - 4) * (x - 5) * (x - 6) * (x - 7) * (x - 8) *
    (x - 9) * (x - 10) * (x - 11) * (x - 12) * 31) /
    (((x - 4) * (x - 5) * (x - 6) * (x - 7) * (x - 8) *
    (x - 9) * (x - 10) * (x - 11) * (x - 12) + .00001)) +
    (((x - 5) * (x - 6) * (x - 7) * (x - 8) * (x - 9) *
    (x - 10) * (x - 11) * (x - 12) * 30) /
    (((x - 5) * (x - 6) * (x - 7) * (x - 8) * (x - 9) *
    (x - 10) * (x - 11) * (x - 12) + .00001)) +
    (((x - 6) * (x - 7) * (x - 8) * (x - 9) * (x - 10) *
    (x - 11) * (x - 12) * 31) /
    (((x - 6) * (x - 7) * (x - 8) * (x - 9) * (x - 10) *
    (x - 11) * (x - 12) + .00001)) +
    (((x - 7) * (x - 8) * (x - 9) * (x - 10) * (x - 11) *
    (x - 12) * 30) /
    (((x - 7) * (x - 8) * (x - 9) * (x - 10) * (x - 11) *
    (x - 12) + .00001)) +
    (((x - 8) * (x - 9) * (x - 10) * (x - 11) * (x - 12) *
    31) /
    (((x - 8) * (x - 9) * (x - 10) * (x - 11) * (x - 12) +
    .00001)) +
    (((x - 9) * (x - 10) * (x - 11) * (x - 12) * 31) /
    (((x - 9) * (x - 10) * (x - 11) * (x - 12) + .00001)) +
    (((x - 10) * (x - 11) * (x - 12) * 30) /
    (((x - 10) * (x - 11) * (x - 12) + .00001)) +
    (((x - 11) * (x - 12) * 31) /
    (((x - 11) * (x - 12) + .00001)) +
    (((x - 12) * 30) / ((x - 12) + .00001)) + 31 + .1) -
    y;
```



In a Nutshell ...

```
@P=split//,".URRUU
\c8R";@d=split//,"\\nrekcah xinU /
lreP rehtona tsuJ";sub p{
@p{"r$p","u$p"}=(P,P);pipe"r$p","u
$p";++$p;($q*=2)+=$f=!fork;map{$P=
$P[$f^ord
($p{$\_})&6];$p{$\_}=/ ^$P/ix?$P:close
$\_}keys%p}p;p;p;p;p;map{$p{$\_}=~/
^[P.]/&&
close$_%p;wait until$?;map{/^r/&&<
$_>}%p;$_=">$d[$q];sleep rand(2)if/
\S/;print
```



In a Nutshell ...

```
void primes(int cap) {  
    int i, j, composite;  
    for(i = 2; i < cap; ++i) {  
        composite = 0;  
        for(j = 2; j * j <= i; ++j)  
            composite += !(i % j);  
        if(!composite)  
            printf("%d\t", i);  
    }  
}  
  
int main(void) {  
    primes(100);  
}
```



```
_ ( __, __, __, __ ) { __ /  
__ <= __ ? __ ( __, __ ) : ! ( __ % __ ) ?  
+ __, __, __ ) : ! ( __ % __ ) ?  
_ ( __, __ + __, __ % __, __ ) : __  
% __ == __ /  
__ && ! __ ? ( printf ("%d\t", __ /  
__ ), __ ( __, __ + __, __, __ ) ) :  
( __ % __ > __ && __ % __ < __ / __ ) ?  
_ ( __, __ +  
__ , __, __ + ! ( __ / __ % ( __  
% __ ) ) : __ < __ * __ ? __ ( __, __  
+ __, __, __ ) : 0 ; } main ( void )  
{ __ ( 100, 0, 0, 1 ) ; }
```



In a Nutshell ...

Free Javascript Obfuscator
Protects JavaScript code from stealing and shrinks size

mylivechat.com Leading Live Chat Software
Hello, How may I help you?

DeepSea Obfuscator
Out of the box .NET Protection

Input:

```
var a="Hello World!";
function MsgBox(msg)
{
    alert(msg+"\n"+a);
}
MsgBox("OK");
```

Obfuscated:

```
<Target Name="Oblfuscate">
<MakeDir Directories="$(MSBuildProjectDirectory)">
<DeepSeaObfuscate Assemblies="App.exe" DstDir="$(MSBuildProjectDirectory)">
</Target>
```

ProGuard
Shrinking
Obfuscation
Optimization
Information
Process
ReTrace

Optimization
ProGuard
Shrinking
Obfuscation
Optimization
Information
Process
ReTrace

Develop

Welcome to ProGuard, version 4.8

ProGuard is a free class file shrinker, optimizer and obfuscator. With this GUI you can create, edit, modify and use your saved configuration.

With the ReTrace part of this GUI you can de-compile and analyze ProGuard and ReTrace are written and maintained by Arxan Technologies Ltd. Copyright (C) 2002-2007.

Products

GuardIT for Windows
for Desktop and Server Applications

Desktop and server applications are increasingly subject to hacker threats. The four most significant threats to software include:

- **Tampering:** an attacker alters proprietary software to give access to others or enhance the software's functionality. Users might seek to add features, delete restrictions or to access hidden functionality.
- **Piracy:** an attacker makes unauthorized copies of proprietary software and sell reproductions at bargain prices, thereby stealing revenue from the organization creating the software.
- **Reverse Engineering:** extract code in order to steal intellectual property, confidential information, and proprietary algorithms.
- **Insertion of Exploits:** insert viruses or other malware into pirated versions.

Resources

- GuardIT for Windows Applications
- Core Features
- Download GuardIT for Windows - Data Sheet
- Contact Sales

Arxan's GuardIT offers the most advanced software protection solution that defends against today's most prevalent software protection challenges of illegal license key generation, cloning of the license server, inadequate obfuscation or weak encryption solutions. Our solution includes, but

In a Nutshell ...

- Different capabilities:
 - Code source vs. binary
 - Supported languages
 - .NET, C#, Java, Javascript, C/C++, ...
 - Costs
 - Size, speed
 - Security towards RE



In a Nutshell ...

- Usual techniques
 - Packing
 - Addition of junk code
 - Code transformations
 - Opaque predicates
 - White-box cryptography
 - Anti-debugging tricks
 - Virtualization
 - ...

The collage includes:

- Digital River softwarePassport**: A screenshot of the website for Digital River's softwarePassport, featuring a CD-ROM and a lock icon.
- Ultimate Packer for Executables**: A logo consisting of a large 'U' and 'P' with 'x' below it, next to the text "Ultimate Packer for Executables".
- DotPacker 1.0**: A screenshot of the DotPacker software interface, showing a code editor with C# code and a detailed description of its features.
- Code Comparison Diagrams**: Two side-by-side diagrams comparing original and obfuscated C# code. The left diagram shows the original code with comments like "x = 2;" and "y = 0;". The right diagram shows the obfuscated code with many more lines and complex logic.
- Control Flow Graphs**: Two flowcharts illustrating the control flow of a program. The left one is a simple loop with a condition "t <= 100". The right one is a more complex multi-case switch statement with various conditions and jumps.
- Application Security Forum Logo**: A circular logo for the Application Security Forum, featuring a stylized animal head and the text "APPLICATION SECURITY FORUM" and "WESTERN SWITZERLAND".

Context of « Obfuscator »

- Project funded by HES-SO (about CHF 160K, time span 2010-2012)

– HEIG-VD

- Pascal Junod
- Gregory Ruch
- Julien Rinaldini

– EIA-FR

- Jean-Roland Schuler
- Marc Romanens
- Adrien Giner

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <arpa/inet.h>

void server1(portServ ports)
{
    int sockServ1, sockServ2, sockClient;
    struct sockaddr_in monAddr, addrClient, addrServ2;
    socklen_t lenAddrClient;
    if ((sockServ1 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Error socket");
        exit(1);
    }
    if ((sockServ2 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Error socket");
        exit(1);
    }

    bzero(&monAddr, sizeof(monAddr));
    monAddr.sin_family = AF_INET;
    monAddr.sin_port = htons(ports.port1);
    monAddr.sin_addr.s_addr = INADDR_ANY;
    bzero(&addrServ2, sizeof(addrServ2));
}
```

```
BB 04 05 0C 30 40 00  mov eax, ds:[GetModuleHandle]
FF D0  call eax ; GetModuleHandle
54  push esi
56  push edi
BB 04 05 70 30 40 00  mov eax, ds:[GetProcAddress]
FF D0  call eax ; GetProcAddress
56  push esi
BB 04 05 5C 30 40 00  mov eax, ds:[GetModuleHandle]
FF D0  call eax ; GetModuleHandle
```



Context of «Obfuscator»

- Goal:
 - create knowledge and know-how in the domain of software obfuscation
 - Develop prototype tools
 - HEIG-VD: focusing on source code obfuscation
 - EIA-FR: focusing on binary obfuscation



LLVM-based Obfuscation

- No satisfactory **open-source** tool able to obfuscate C/C++
- Cool project to play with!
- RE is hard, protecting against RE in an efficient way is even harder!

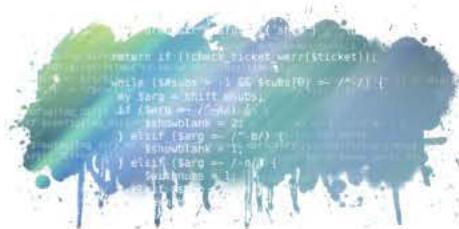


LLVM-based Obfuscation

- Several «false starts» (but that's research ;-)

SPLAT - A Simple Python Library for Abstract syntax tree Transformation

Sébastien Bischof
Professeur : Dr. Pascal Junod
June 17, 2010



Obfuscator - Abstract syntax tree Transformation

Grégory Ruch
Professeur : Dr. Pascal Junod
29 avril 2011



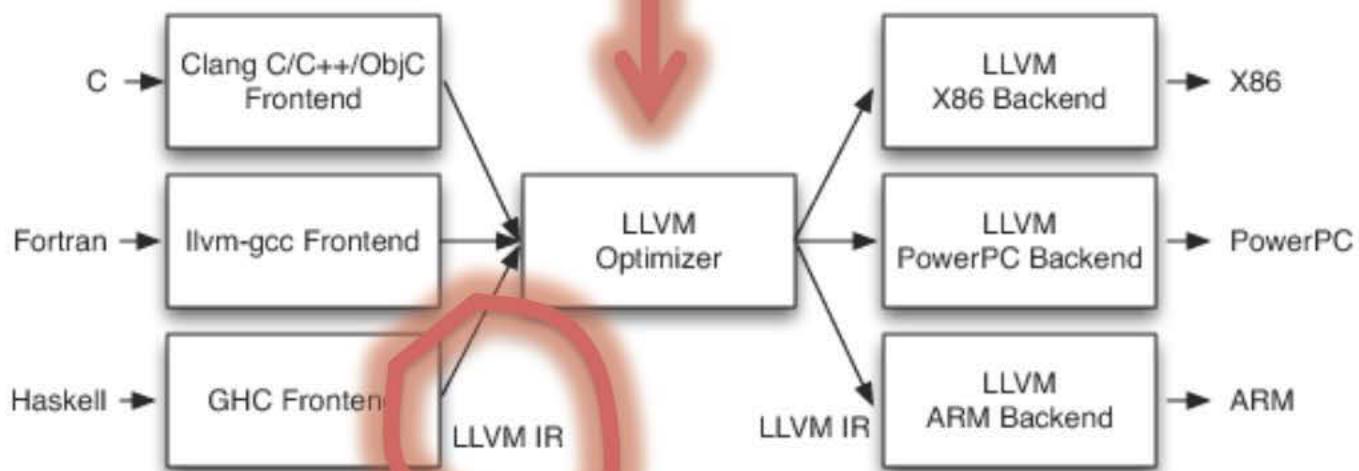
- Sébastien Bischof : parsing C with Python
- Grégory Ruch: hacking the LLVM Clang API

LLVM-based Obfuscation

- LLVM
 - Compiler infrastructure
 - Project initiated by the University of Illinois in 2000
 - Since 2005, pushed by Apple Inc.
 - Front-ends:
 - C/C++, Objective C, Fortran, Ada, Haskell, Python, Ruby, ...
 - Back-ends:
 - x86, x86-64, PowerPC, PowerPC-64, ARM, Thumb, Sparc, Alpha, CellSPU, MIPS, MSP430, SystemZ, XCore.



LLVM-based Obfuscation



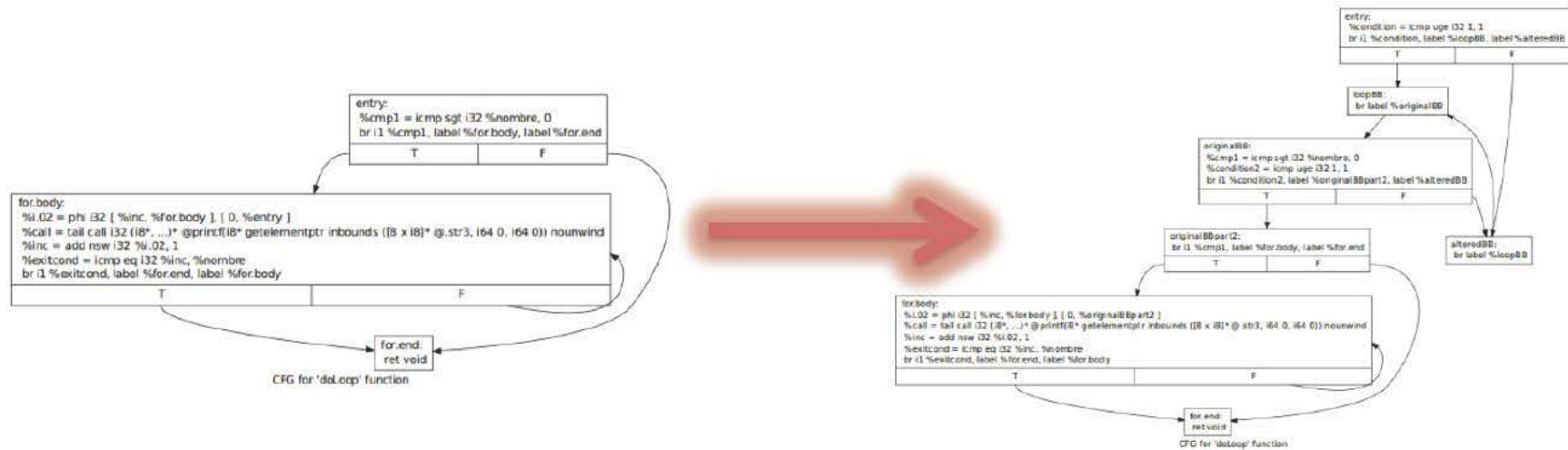
LLVM-based Obfuscation

- Available obfuscation passes
 - Code substitution
 - $A \wedge B = (A \& \sim B) \mid (\sim A \& B)$
 - $A + B = A - (-B)$
 - ...



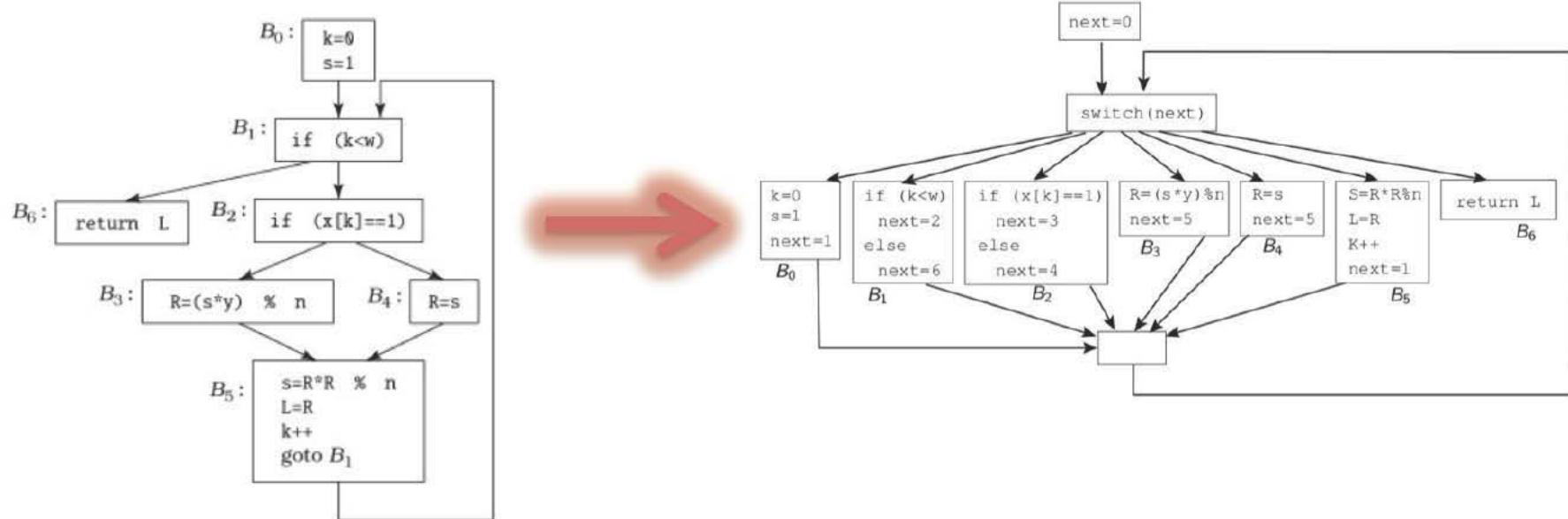
LLVM-based Obfuscation

- Available obfuscation passes
 - Insertion of fake branches with opaque predicates (bachelor thesis of Julie Michelin)



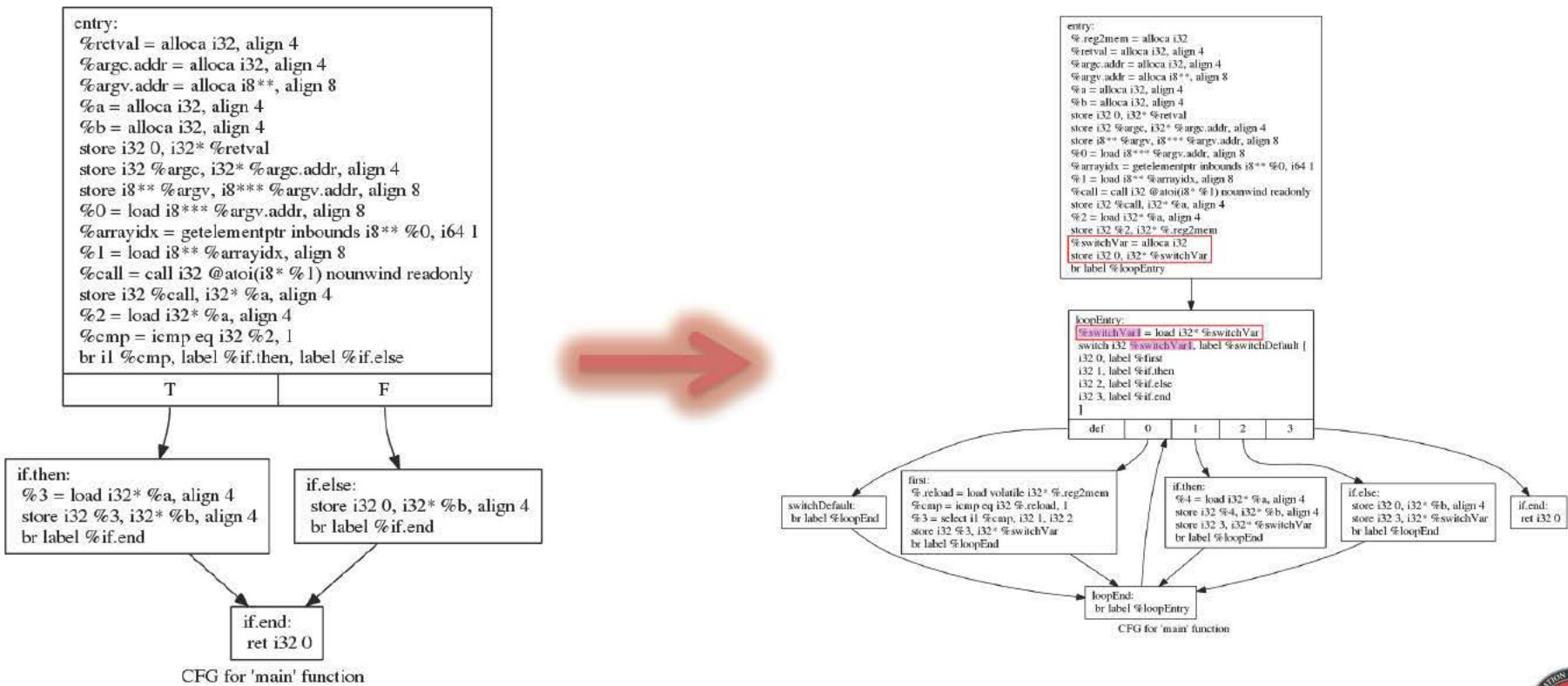
LLVM-based Obfuscation

- Available obfuscation passes
 - Code flattening



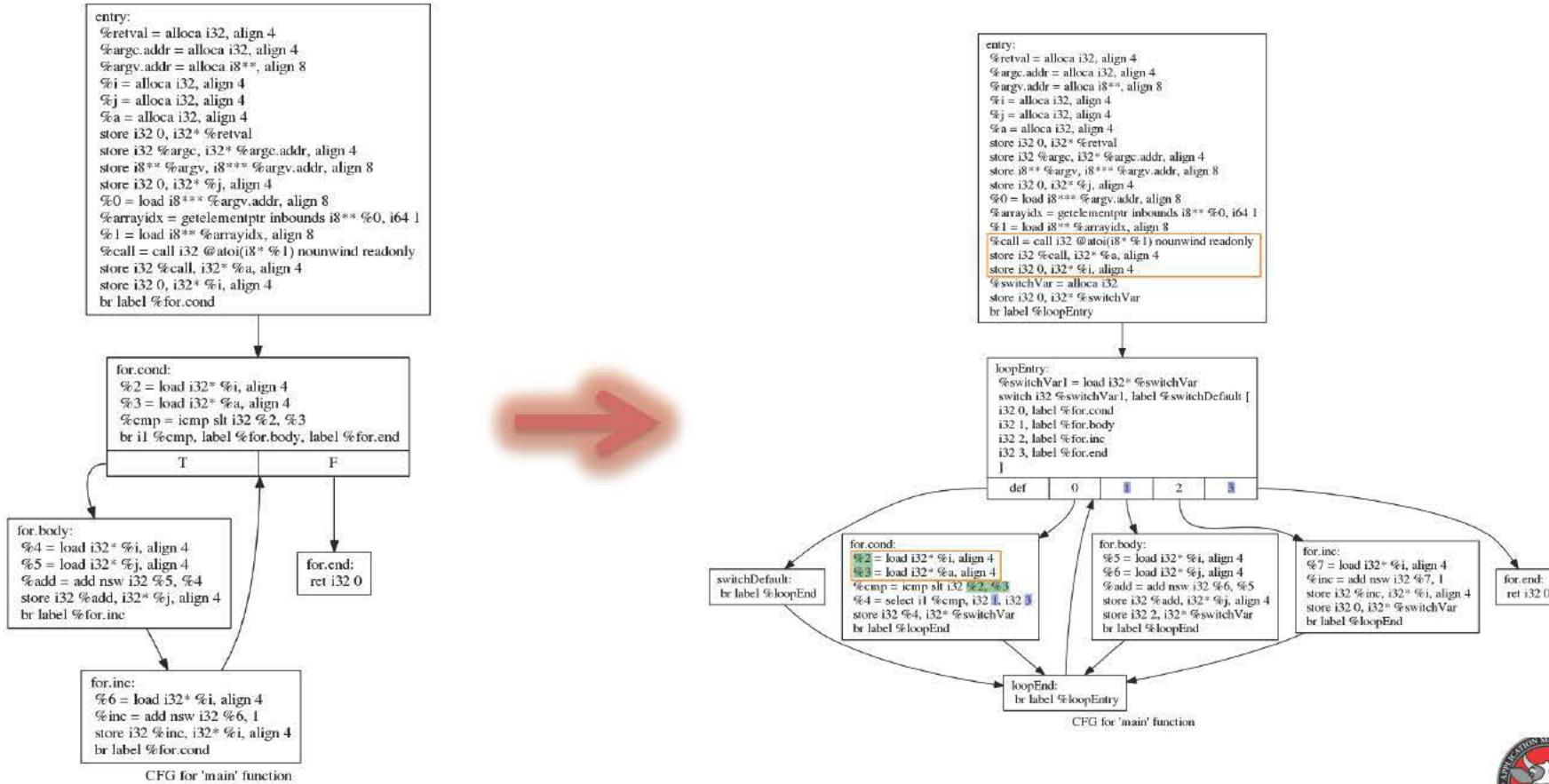
LLVM-based Obfuscation

■ Code flattening: case of IF-THEN-ELSE



LLVM-based Obfuscation

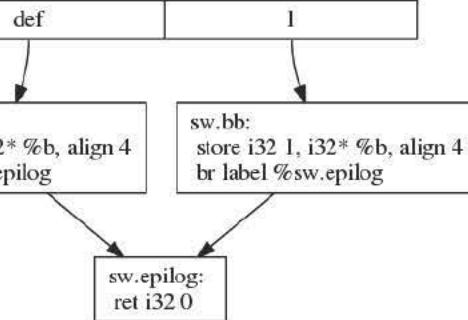
■ Code flattening: case of a FOR loop



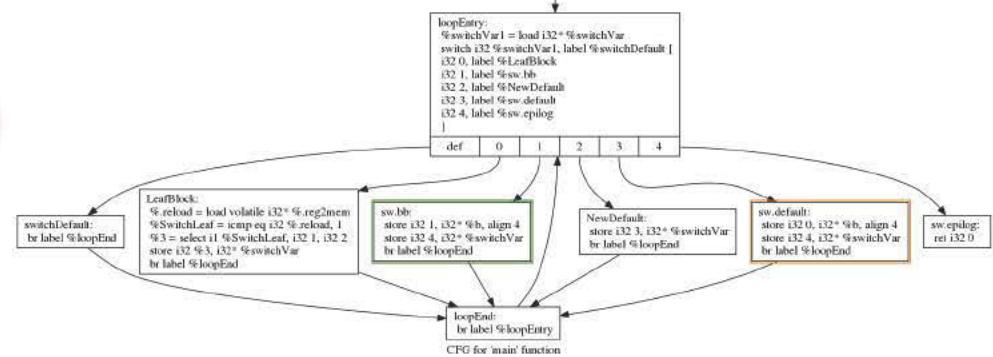
LLVM-based Obfuscation

■ Code flattening: case of a SWITCH

```
entry:
%retnval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%b = alloca i32, align 4
%a = alloca i32, align 4
store i32 0, i32* %retnval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
store i32 0, i32* %b, align 4
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8*** %0, i64 1
%1 = load i8*** %arrayidx, align 8
%call = call i32 @atoi(i8* %1) nounwind readonly
store i32 %call, i32* %a, align 4
%2 = load i32* %a, align 4
switch i32 %2, label %sw.default [
i32 1, label %sw.bb
]
```

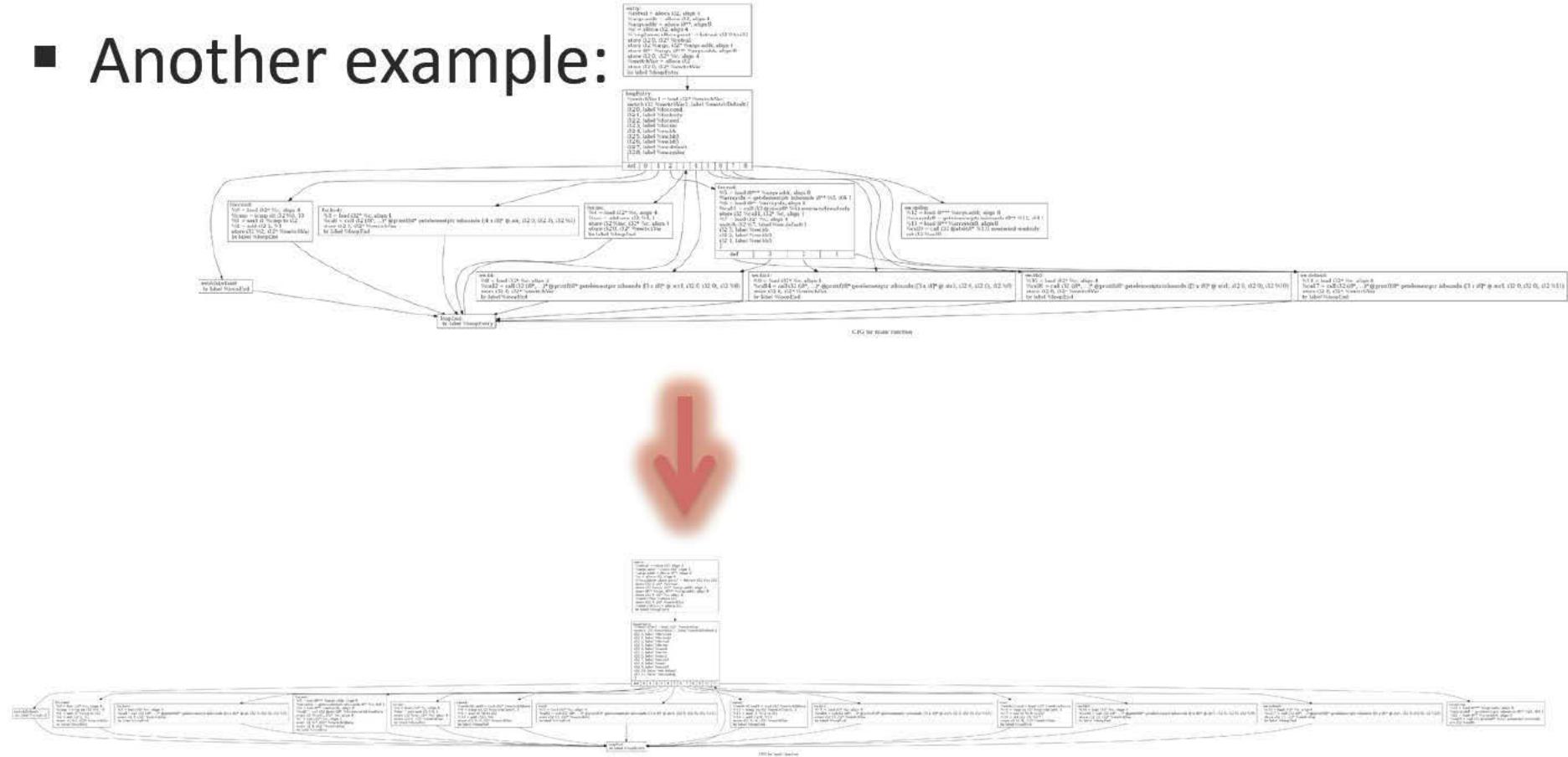


```
entry:
%reg2mem = alloca i32
%retnval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%b = alloca i32, align 4
%a = alloca i32, align 4
store i32 0, i32* %retnval
store i8** %argv, i8*** %argv.addr, align 8
store i32 0, i32* %b, align 4
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8*** %0, i64 1
%1 = load i8*** %arrayidx, align 8
%call = call i32 @atoi(i8* %1) nounwind readonly
store i32 %call, i32* %a, align 4
%2 = load i32* %a, align 4
store i32 %2, i32* %reg2mem
%switchVar = alloca i32
store i32 0, i32* %switchVar
br label %loopEntry
```



LLVM-based Obfuscation

- ## ■ Another example:



LLVM-based Obfuscation

- Test procedure
 - Run test suite of the cryptographic library libtomcrypt
 - Run test suite of the graphical library ImageMagick
 - Found a routine of 6000+ C lines in it !!
 - Run test suite of the MySQL database
 - A single test is still failing ☹
- Todo
 - Flatten try-catch constructs as well
 - Test, debug, test, debug, test !



LLVM-based Obfuscation

Of course, we used



Part III Arm obfuscation

- About me
 - Marc Romanens
 - Scientific collaborator at EIA-FR
 - Works on a “Exploration Project” on binary obfuscation for processor ARM.



Agenda

File format ELF

Code insertion into ELF file

ARM obfuscation

Opaque predicate

Junk code



File format ELF

Summary I

- 3 headers tables
 - Header file
 - Entry point
 - Position of others headers
 - Architecture
 - Program header (define the segments)
 - Flags (RWE)
 - Offset / virtual addresses
 - Section header (define the sections (optional))



File Format Elf

Summary II

There are no section groups in this file.

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
PHDR	0x000034	0x00008034	0x00008034	0x000c0	0x000c0	R E	0x4
INTERP	0x0000f4	0x000080f4	0x000080f4	0x00014	0x00014	R	0x1
[Requesting program interpreter: /lib/ld-uClibc.so.0]							
LOAD	0x000000	0x00008000	0x00008000	0x0052c	0x0052c	R E	0x8000
LOAD	0x00052c	0x0001052c	0x0001052c	0x000f0	0x0010c	RW	0x8000
DYNAMIC	0x000538	0x00010538	0x00010538	0x000b8	0x000b8	RW	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RWE	0x4

Section to Segment mapping:

Segment Sections...

00	
01	.interp
02	.interp .hash .dynsym .dynstr .rel.plt .init .plt .text .fini .rodata .eh_frame
03	.init_array .fini_array .jcr .dynamic .got .data .bss
04	.dynamic
05	



Memory allocation of segment

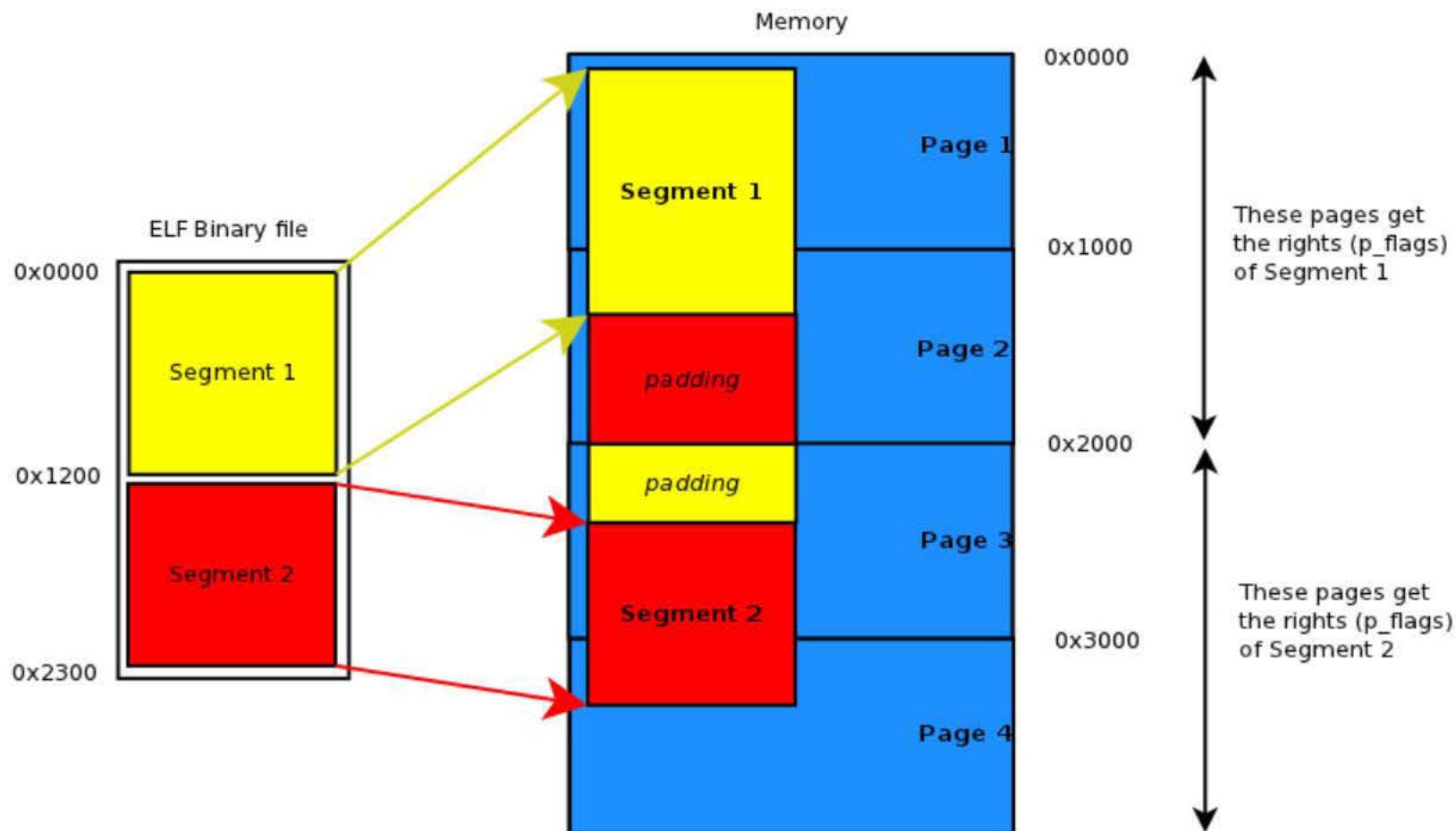


Image source: GNU Linux Magazine HS 32 Septembre Octobre 2007

Insertion of code

- Why ?
 - Obfuscation needs space
- How ?
 - Add a new segment
 - Increase size of old segment



Obfuscation technique :

Opaque predicate

- Example :
 - $(X^2 + X) \% 2 == 0$ is always true
 - $(3X^3 + 2 X^2 + X) \% 6 == 0$ is always true
- Opaque predicate aims at:
 - Creating false branch
 - Improving complexity of serial number
 - Watermarking



Obfuscation technique :

Junk code

- Junk code aims at:
 - Hiding the useful part of code
- Implementation :
 - Use condition code in ARM
 - Use opaque predicate for enforcing the stealth



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define SIZE 2
6
7 int main(char argc, char argv[])
8 {
9     int i;
10    printf("Enter your serial (%d digits)\n", SIZE);
11    char* serial = malloc(SIZE * sizeof(char));
12    memset(serial, 0, SIZE);
13    fgets(serial, SIZE, stdin);
14
15    int result = 0;
16    int junk = 0;
17
18    for (i = 0; i < SIZE; i++)
19    {
20        if ((serial[i] + serial[i] * serial[i]) % 2 == 0)
21        {
22            result += serial[i];
23        }
24        else
25        {
26            junk += serial[i];
27        }
28    }
29
30    if (100 - result <= 0)
31    {
32        printf("Good job you are authentificate\n");
33    }
34    else
35    {
36        printf("Go out hacker %d\n", result);
37    }
38 }
```



Opaque predicate

```

loc_8628
LDR    R3, [R11,#var_14]
LDR    R2, [R11,#$]
ADD    R3, R2, R3
LDRB   R3, [R3]
ADD    R2, R3, #1
LDR    R3, [R11,#var_14]
LDR    R1, [R11,#$]
ADD    R3, R1, R3
LDRB   R3, [R3]
MOU    R1, R3
MUL    R3, R1, R2
AND    R3, R3, #1
CMP    R3, #0
BNE    loc_8680

```

Junk code, this code is never execute

```

LDR    R3, [R11,#var_14]
LDR    R2, [R11,#$]
ADD    R3, R2, R3
LDRB   R3, [R3]
LDR    R2, [R11,#var_C]
ADD    R3, R2, R3
STR    R3, [R11,#var_C]
B     loc_869C

```

```

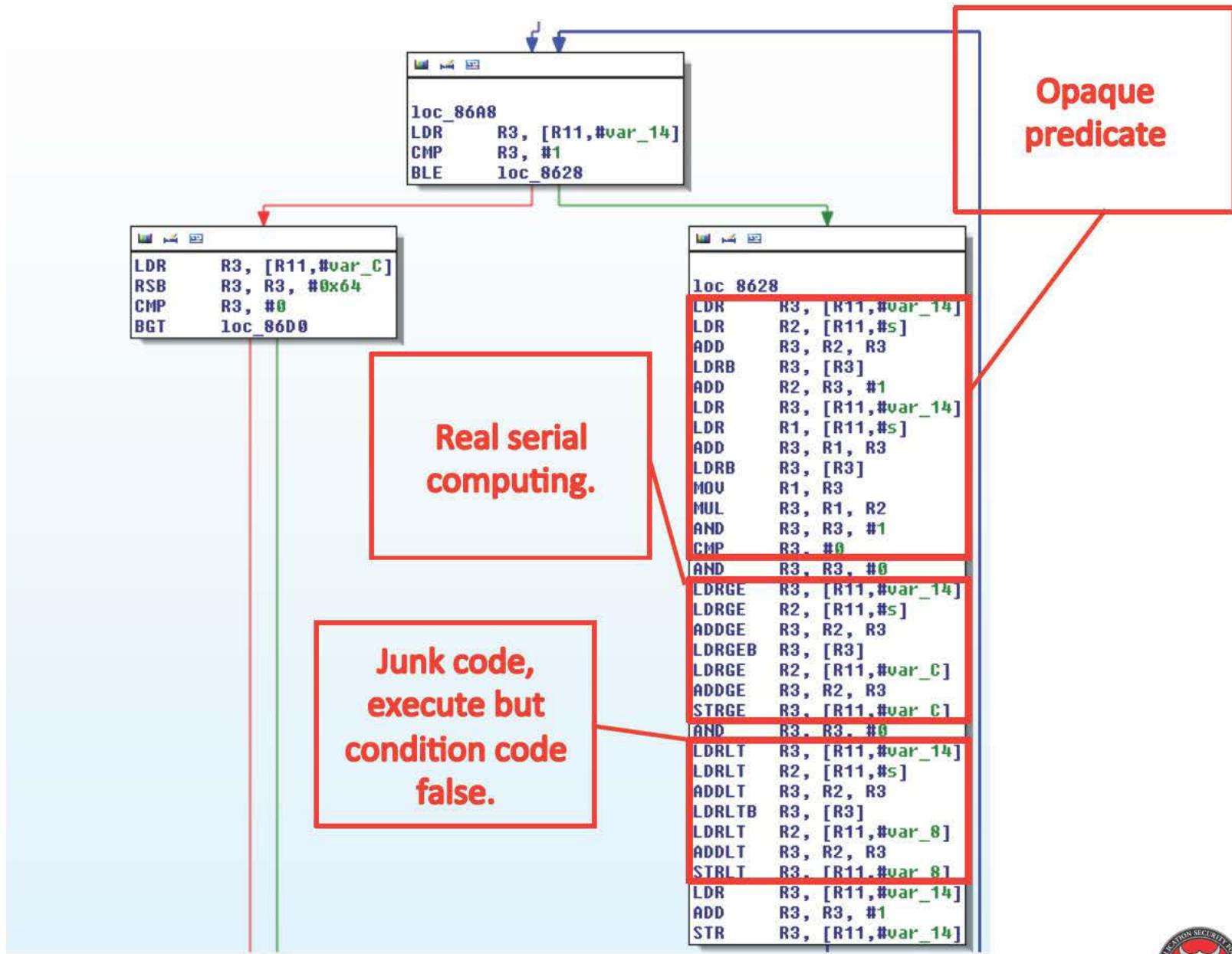
loc_8680
LDR    R3, [R11,#var_14]
LDR    R2, [R11,#$]
ADD    R3, R2, R3
LDRB   R3, [R3]
LDR    R2, [R11,#var_8]
ADD    R3, R2, R3
STR    R3, [R11,#var_8]

```

```

loc_869C
LDR    R3, [R11,#var_14]
ADD    R3, R3, #1
STR    R3, [R11,#var_14]

```

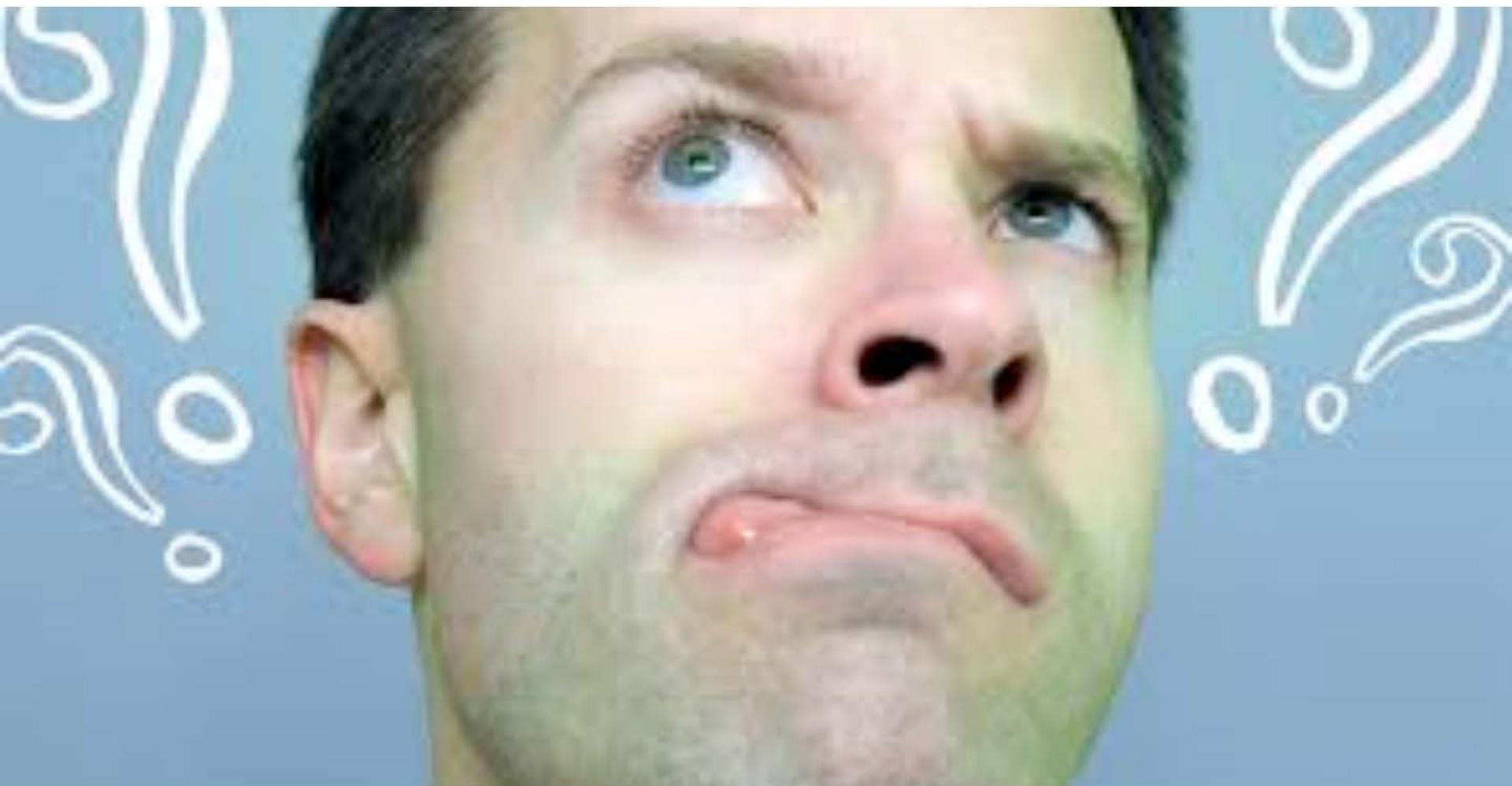


Junk code Optimisation

- Mix junk code and real code
- Change condition code with data processing instructions
- Add more predicates



Questions?



Merci/Thank you!

Contact:

pascal.junod@heig-vd.ch

[@cryptopathe](https://twitter.com/cryptopathe)

<http://crypto.junod.info>

romanens.m@gmail.com

Slides:

<http://slideshare.net/ASF-WS/presentations>

