Hes·so University of Applied Sciences Western Switzerland
heig-vd Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Obfuscator reloaded

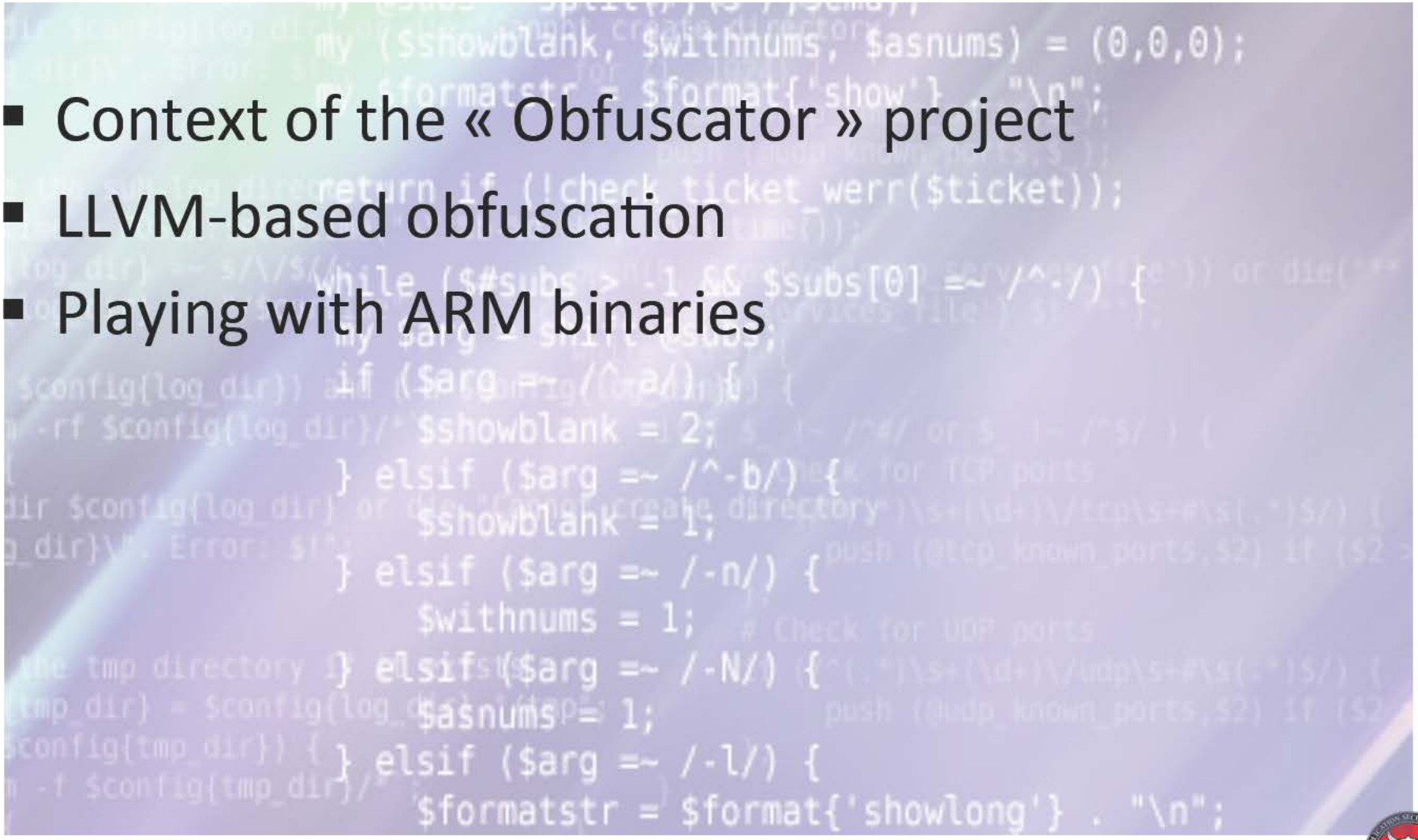
Pascal Junod – HEIG-VD
Julien Rinaldini – HEIG-VD
Marc Romanens - EIA-FR
Jean-Roland Schuler – EIA-FR

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



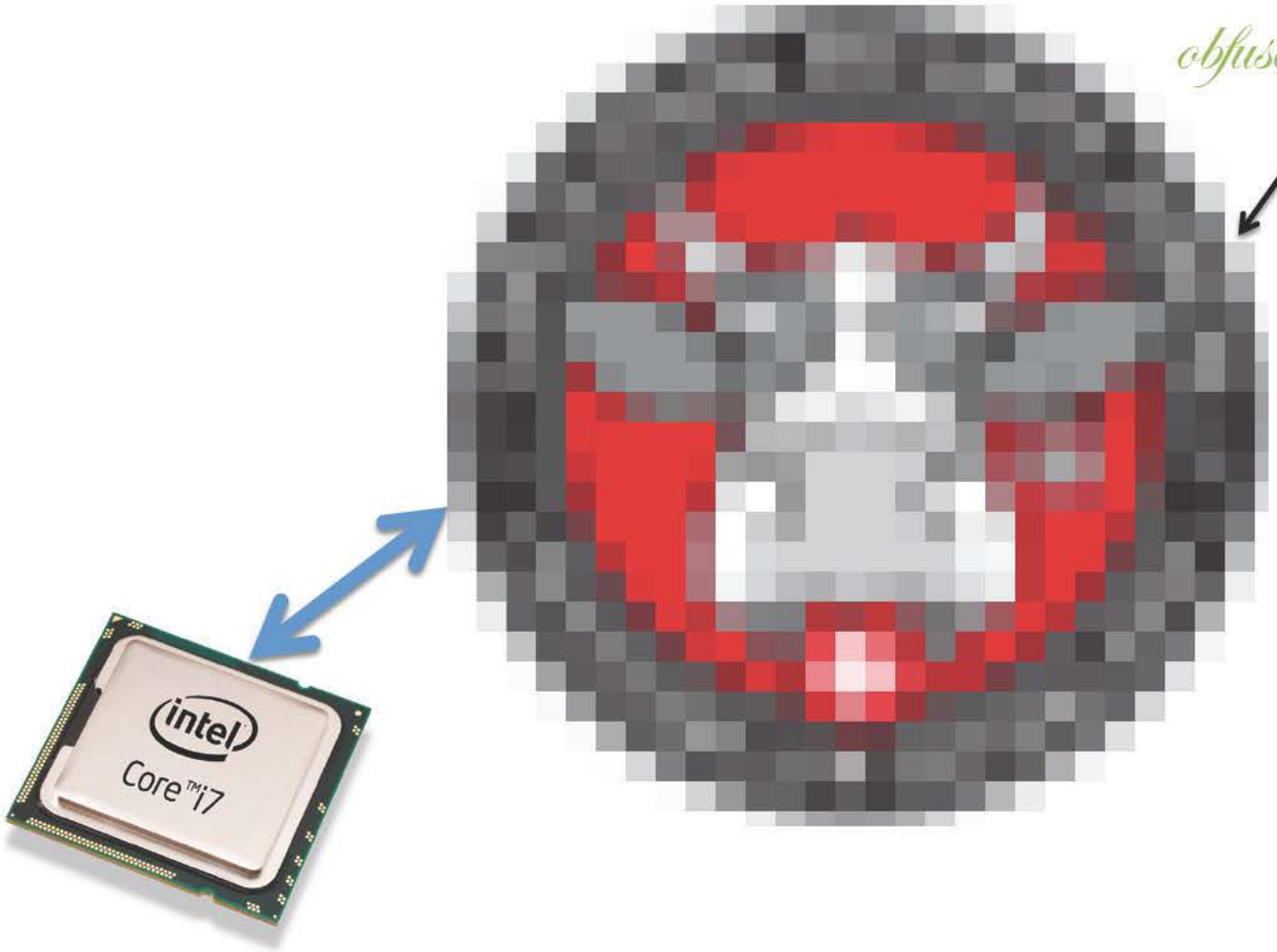
In a Nutshell …

Please see this as

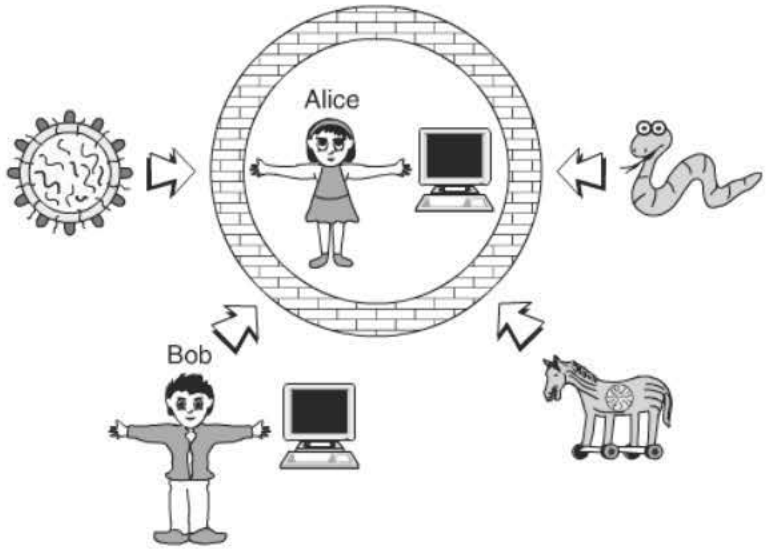
unprotected software!

In a Nutshell …

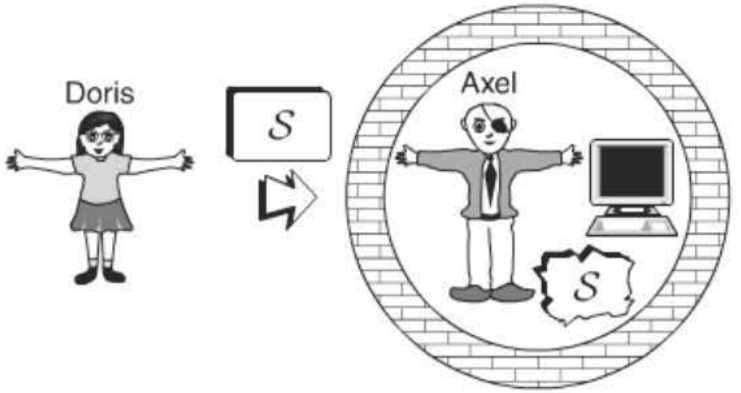
In a Nutshell …



In a Nutshell …

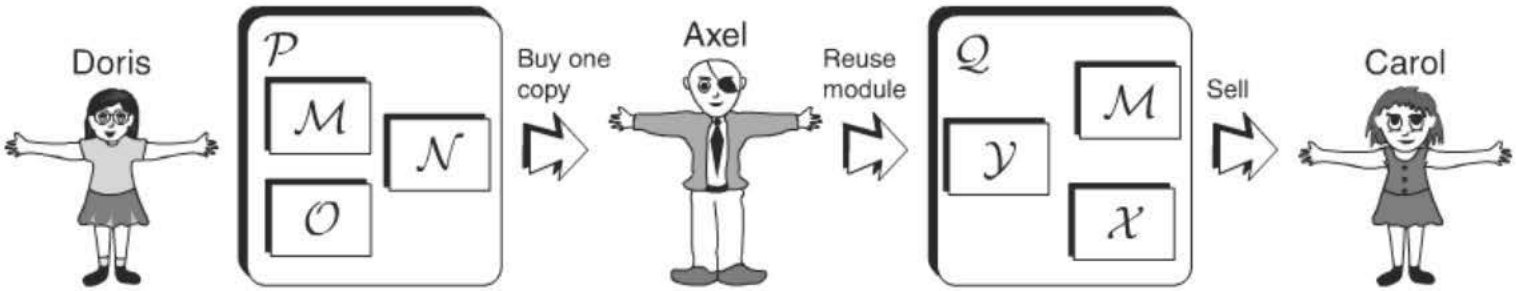


vs.



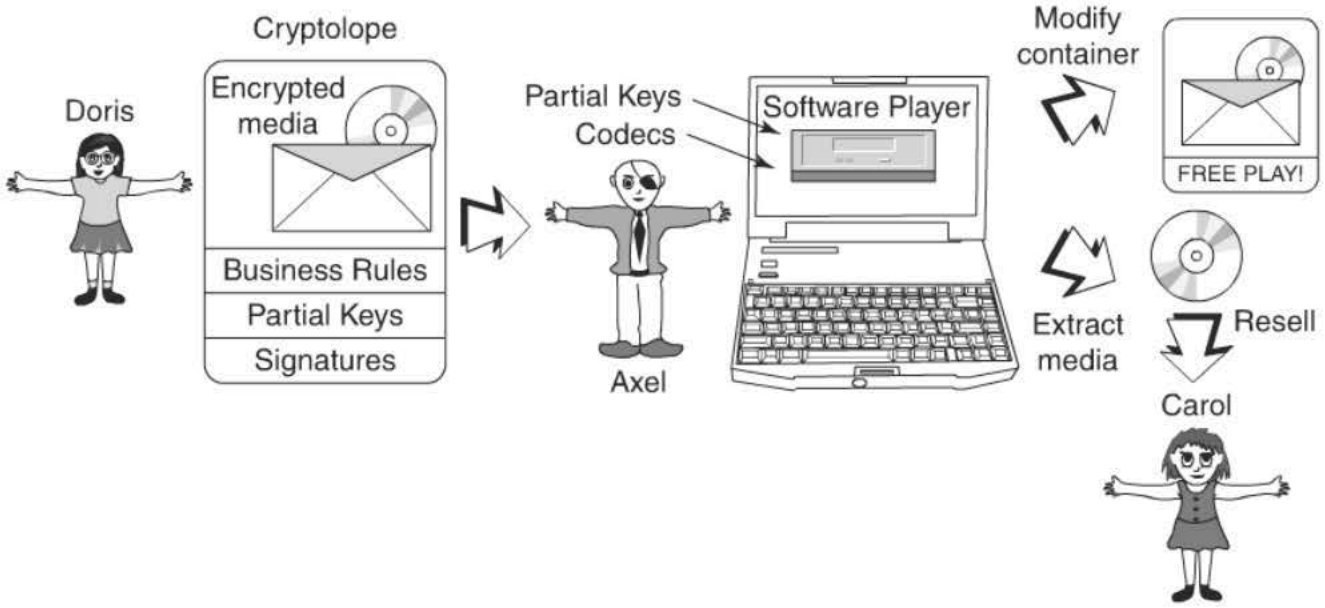
In a Nutshell …

Malicious Reverse-Engineering



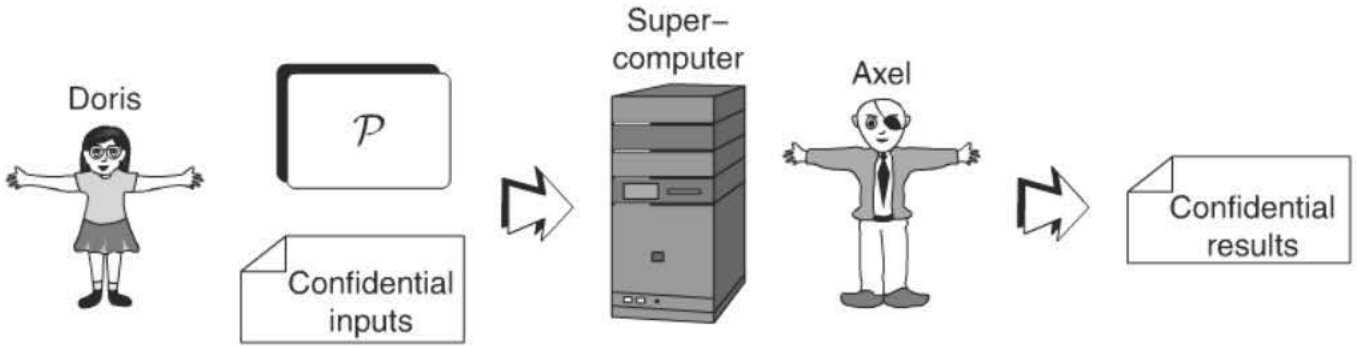
In a Nutshell …

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 ($\mathbf{TC}^0$). We also rule out several potential applications of obfuscators, by constructing "unobfuscatable" signature schemes, encryption schemes, and pseudorandom function families.

In a Nutshell …

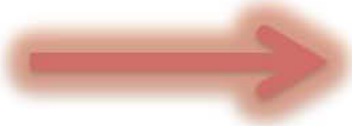
```
return (int) ((((x - 2) * (x - 3) * (x - 4) * (x - 5) * (x - 6) *
                (x - 7) * (x - 8) * (x - 9) * (x - 10) * (x - 11) *
                (x - 12) * 31) /
               ((x - 2) * (x - 3) * (x - 4) * (x - 5) * (x - 6) *
                (x - 7) * (x - 8) * (x - 9) * (x - 10) * (x - 11) *
                (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
```

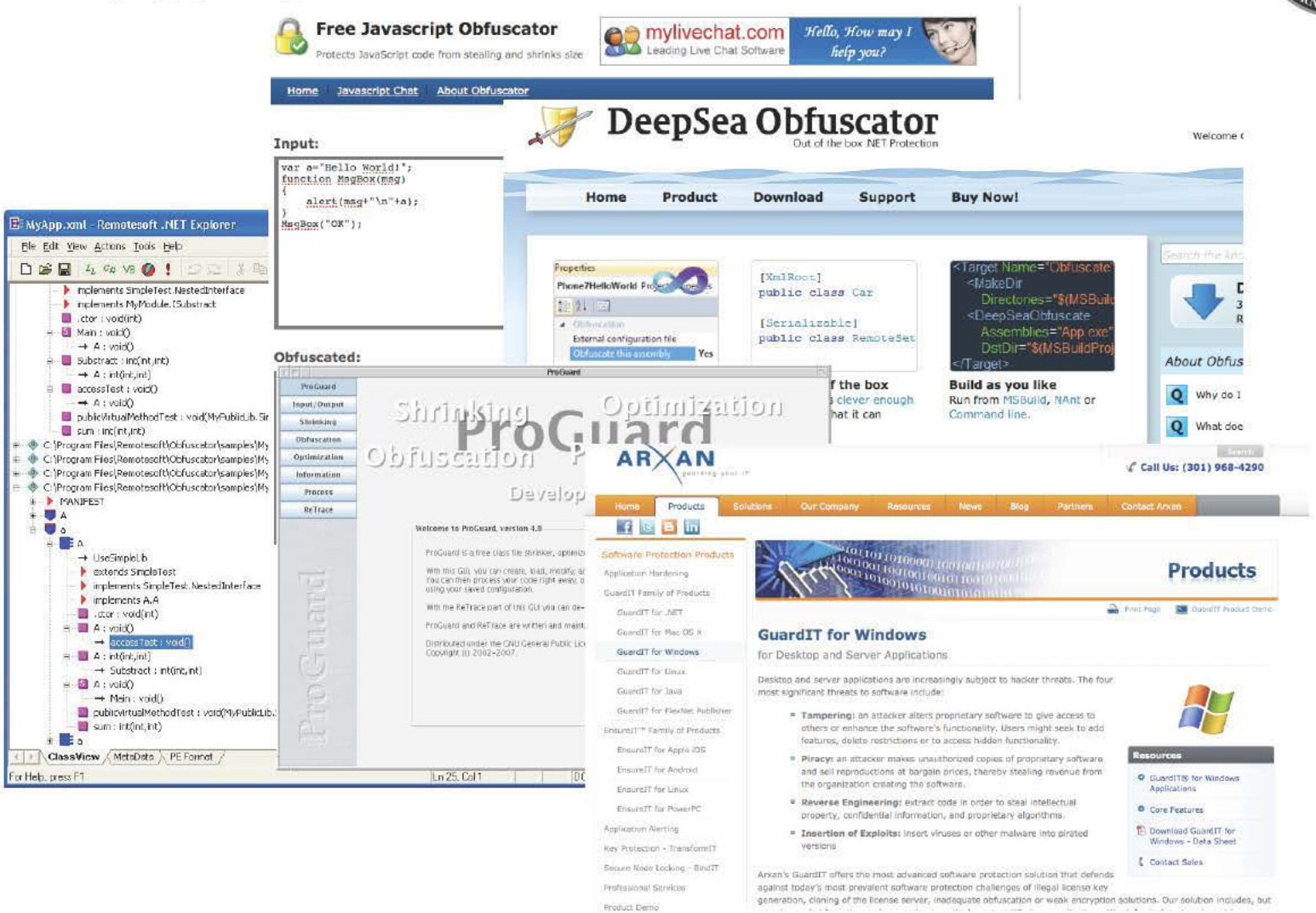
In a Nutshell …

```
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 …

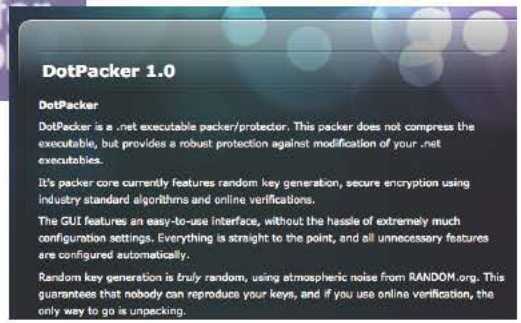


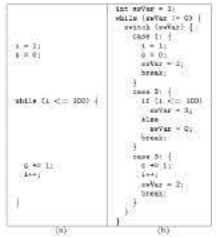
In a Nutshell …

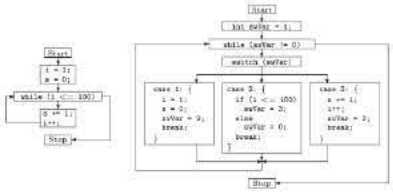
- Different capabilities:
  - Code source vs. binary
  - Supported languages
    - .NET, C#, Java, Javascript, C/C++, …
  - Costs
    - Size, speed
  - Security towards RE

In a Nutshell …

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

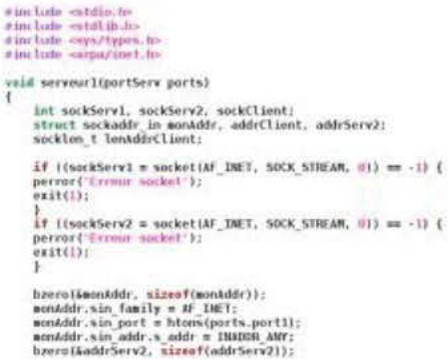




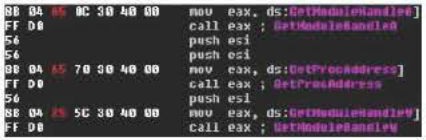


Context of « Obfuscator »

- 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 serveur1(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("Erreur socket");
    exit(1);
    }
    if ((sockServ2 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Erreur 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));
```



```
8B 0A A5 0C 30 40 00    mov  eax, ds:GetModuleHandleA]
FF D0                   call eax ; GetModuleHandleA
56                      push esi
56                      push esi
8B 0A A5 70 30 40 00    mov  eax, ds:GetProcAddress]
FF D0                   call eax ; GetProcAddress
56                      push esi
8B 0A A5 5C 30 40 00    mov  eax, ds:GetModuleHandleW]
FF D0                   call eax ; GetModuleHandleW
```

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 **S**imple **P**ython **L**ibrary for **A**bstract syntax tree **T**ransformation

Sébastien Bischof
Professor: Dr. Pascal Junod

June 17, 2010

Obfuscator - Abstract syntax tree Transformation

Grégory Ruch
Professeur : Dr. Pascal Junod

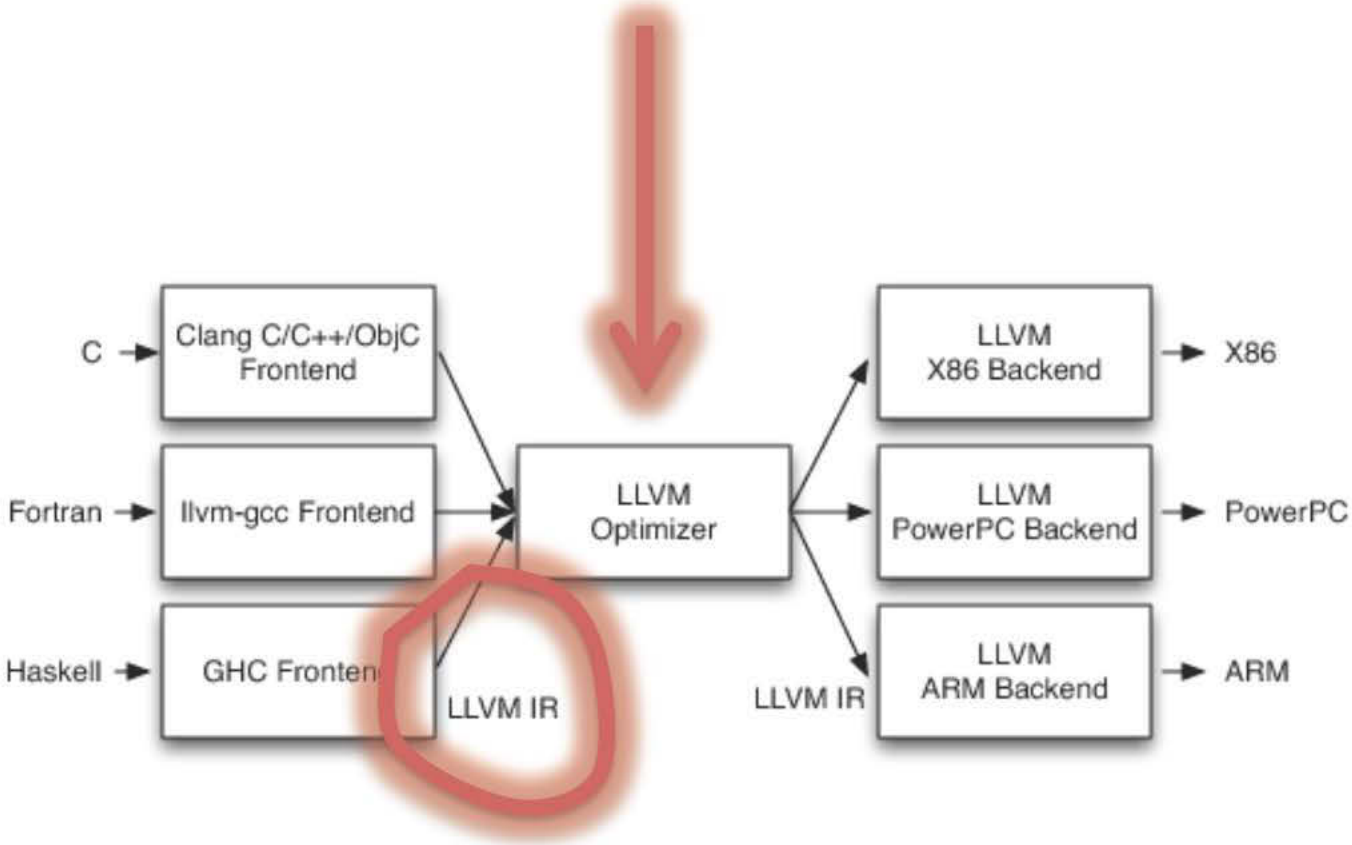
29 avril 2011

- 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

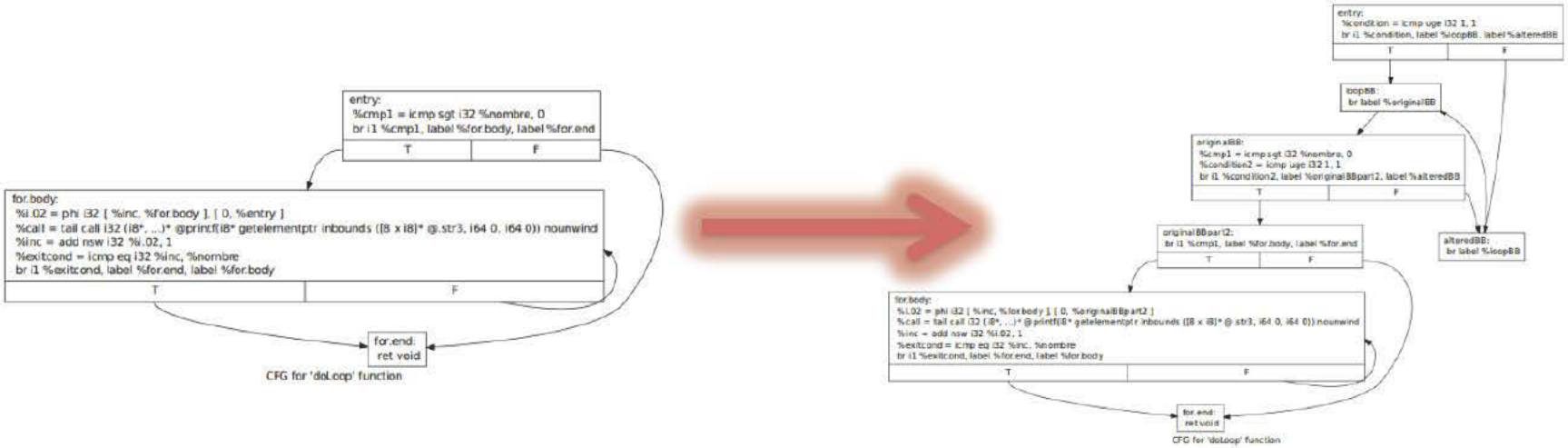


LLVM-based Obfuscation

- Available obfuscation passes
  - Code substitution
    - `A ^ B = (A & ~B) | (~A & B)`
    - `A + B = A - (-B)`
    - …

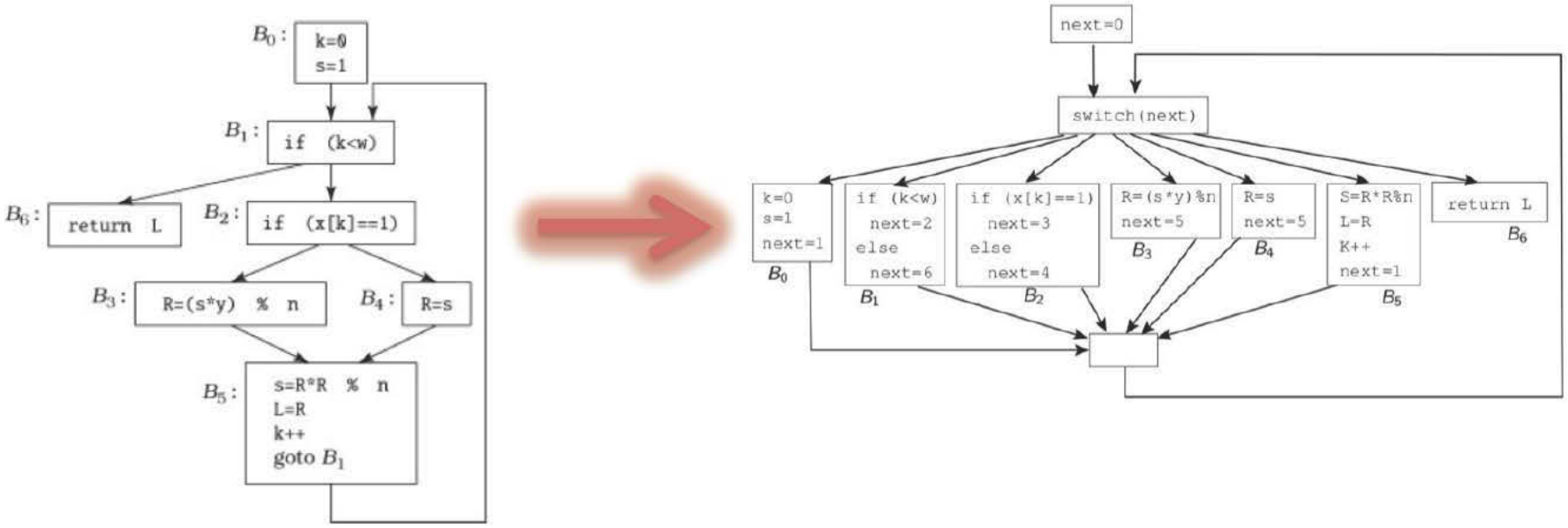
LLVM-based Obfuscation

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



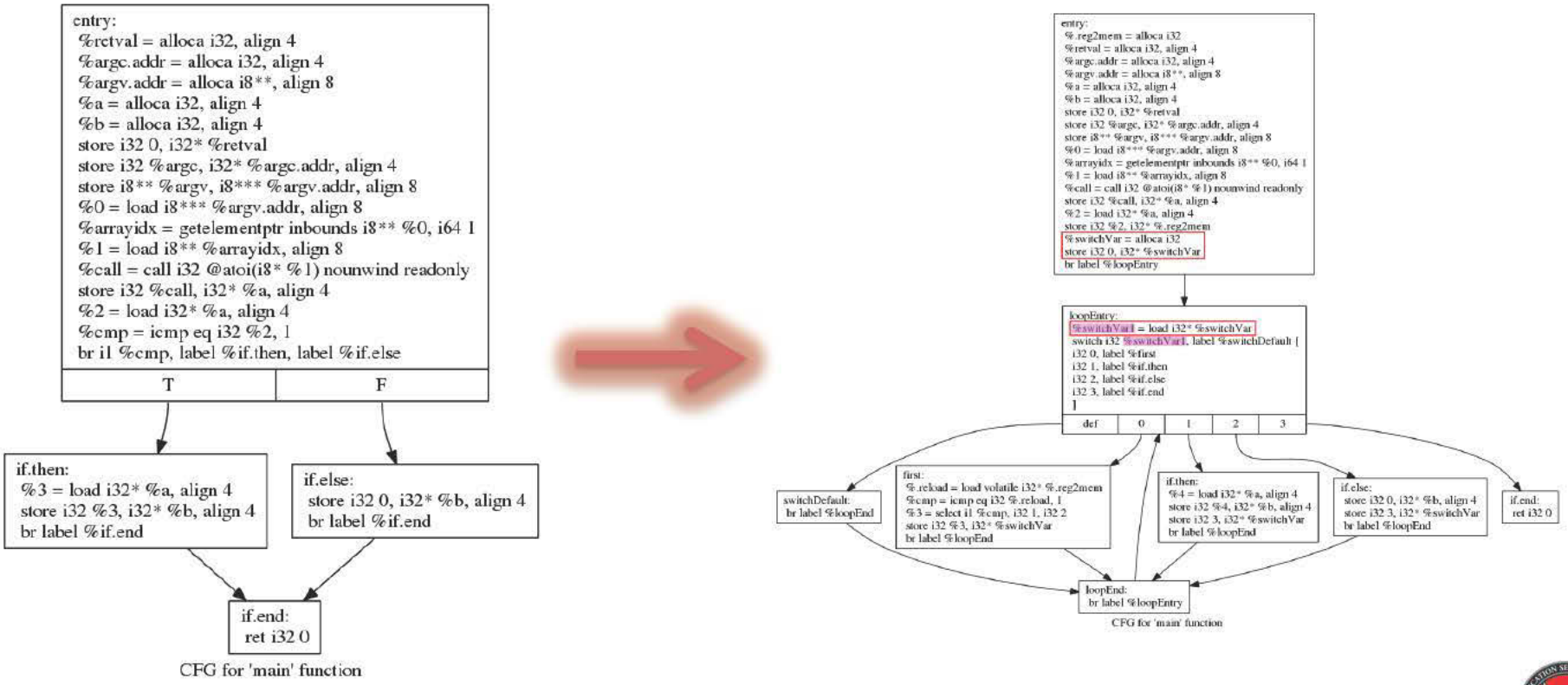
LLVM-based Obfuscation

- Available obfuscation passes
  - Code flattening



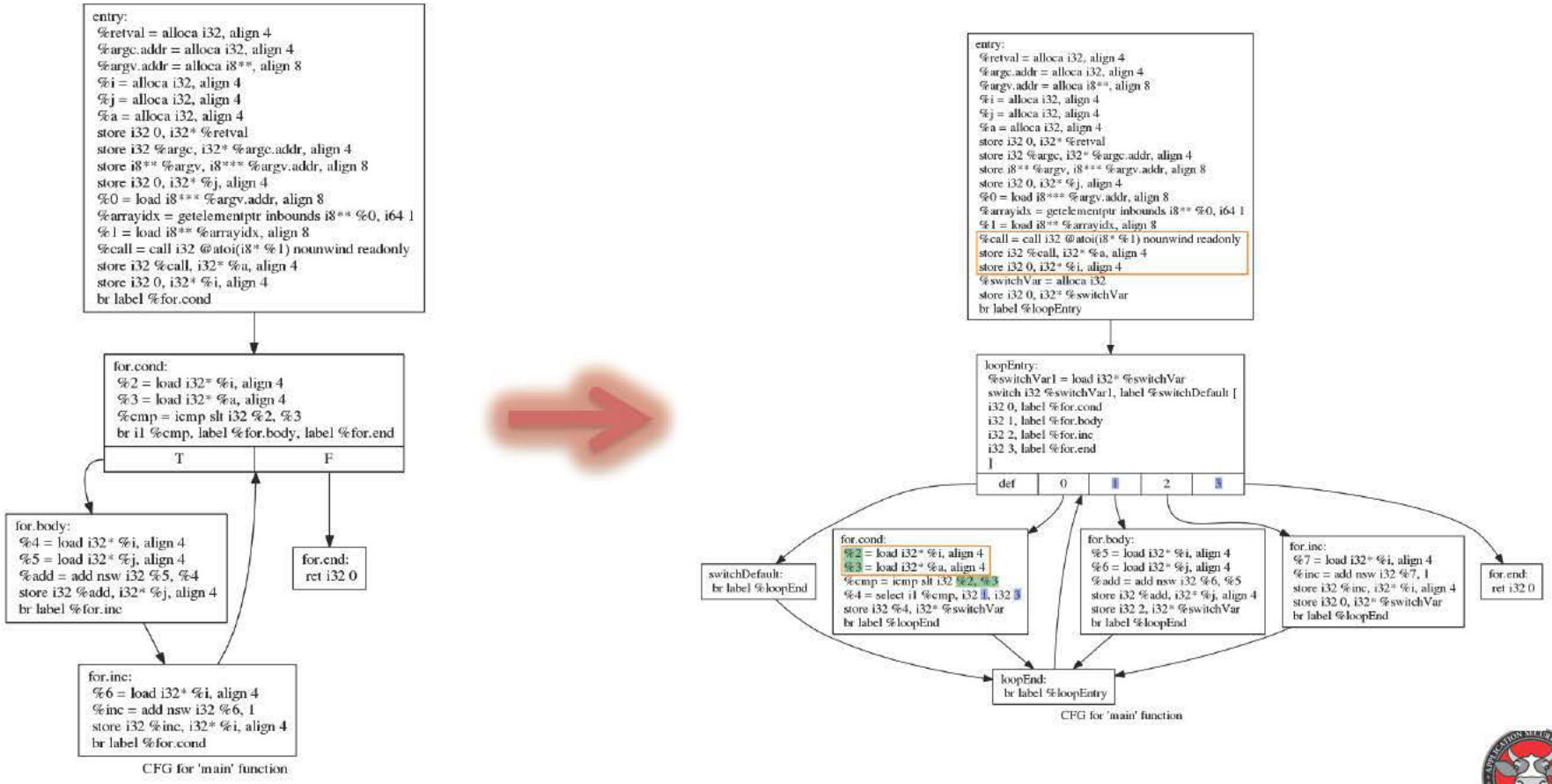
LLVM-based Obfuscation

- Code flattening: case of IF-THEN-ELSE



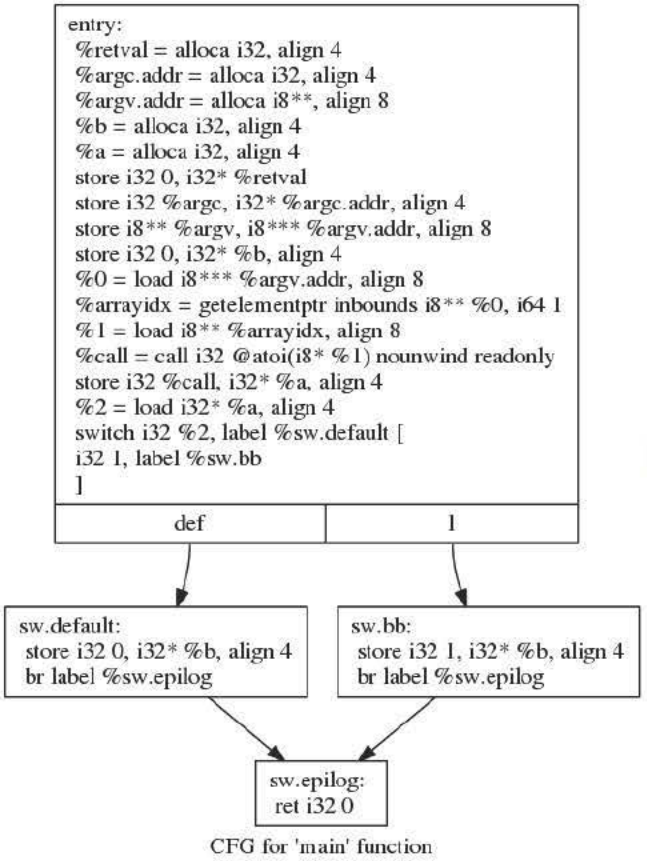
LLVM-based Obfuscation

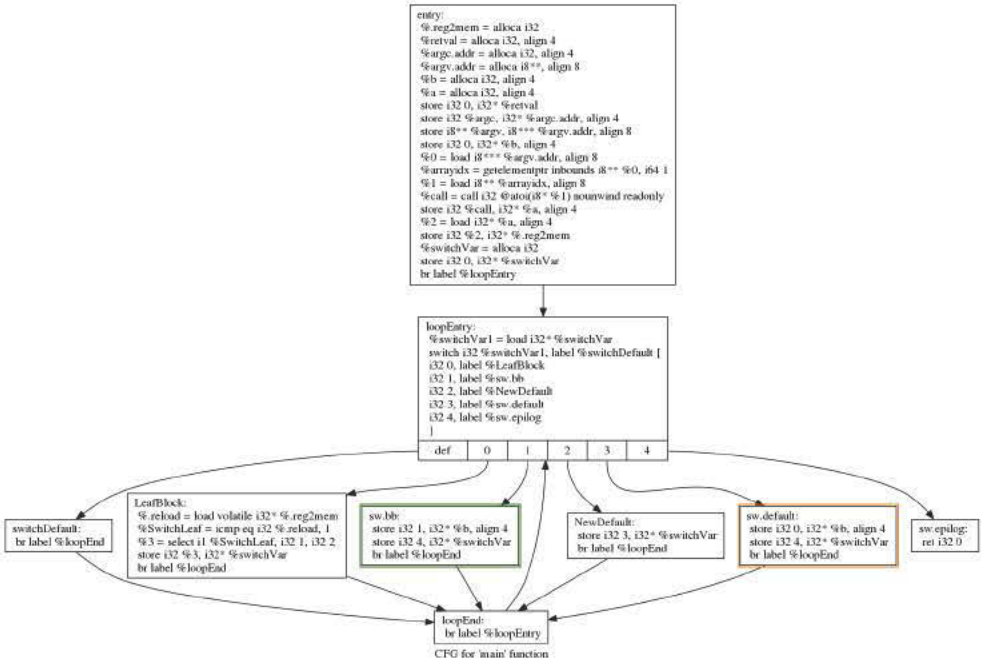
- Code flattening: case of a FOR loop



LLVM-based Obfuscation

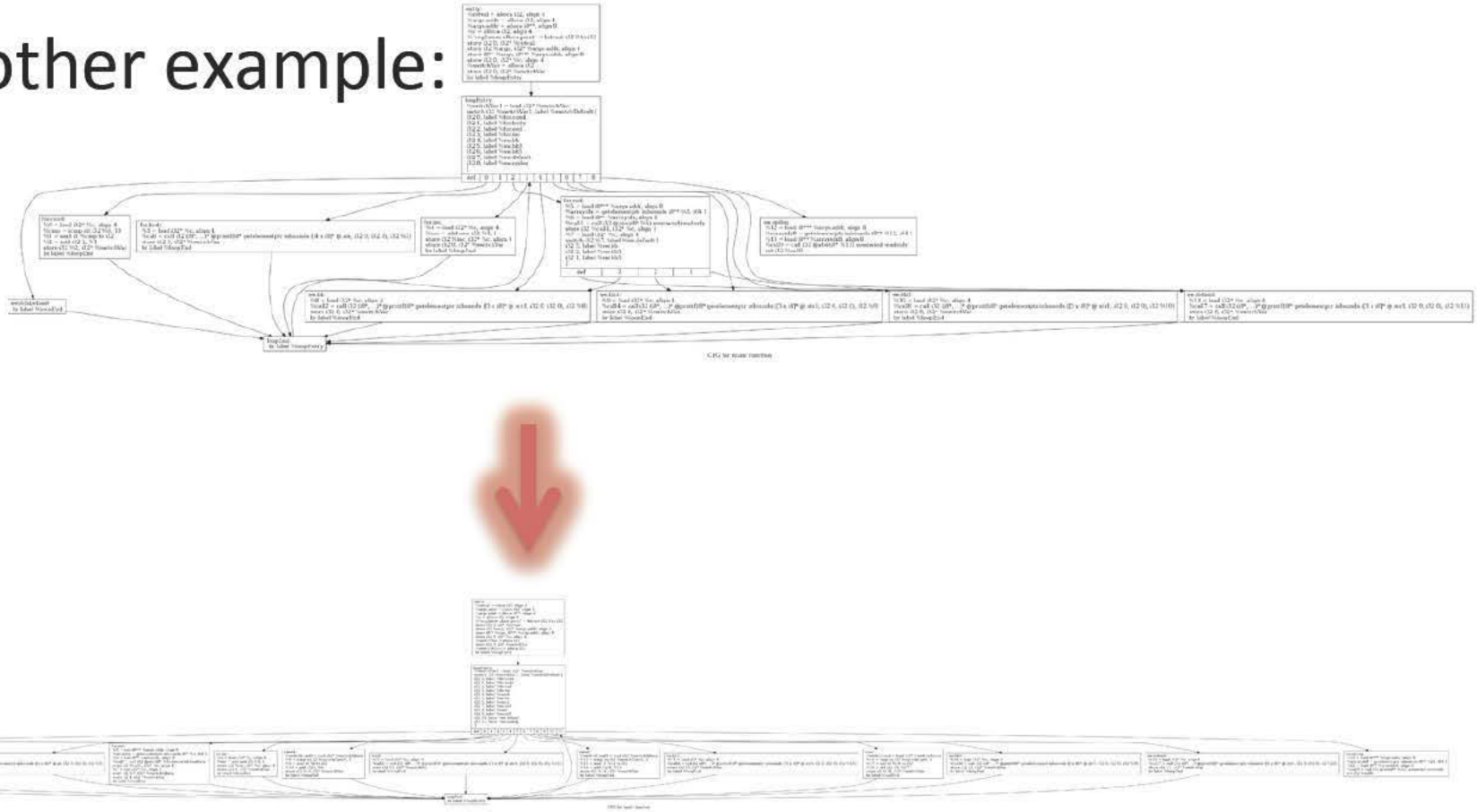
- Code flattening: case of a SWITCH





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

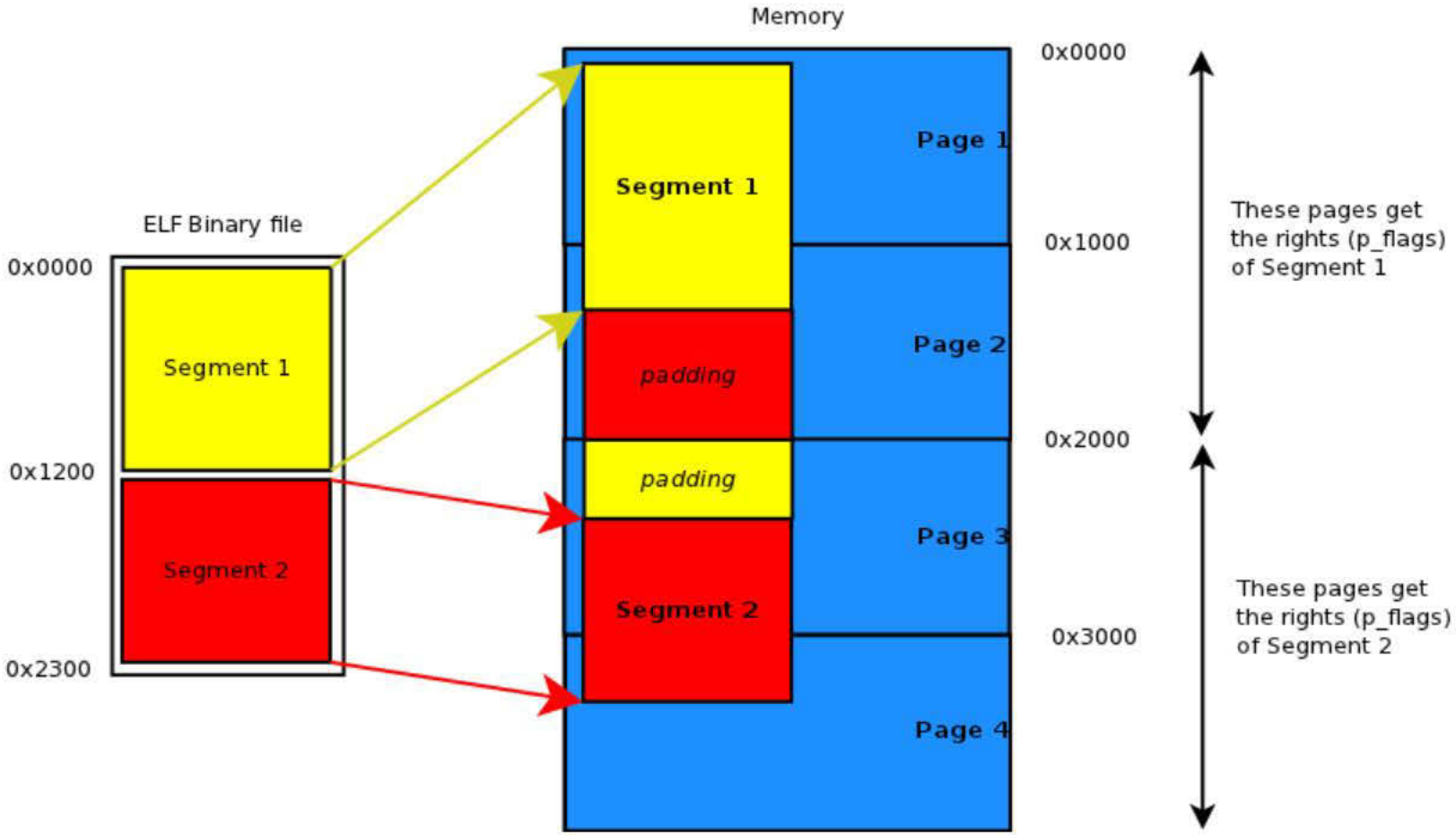


Image source: GNU Linux Magazine HS 32 Septembre Octobre 2007

Insertion of code

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

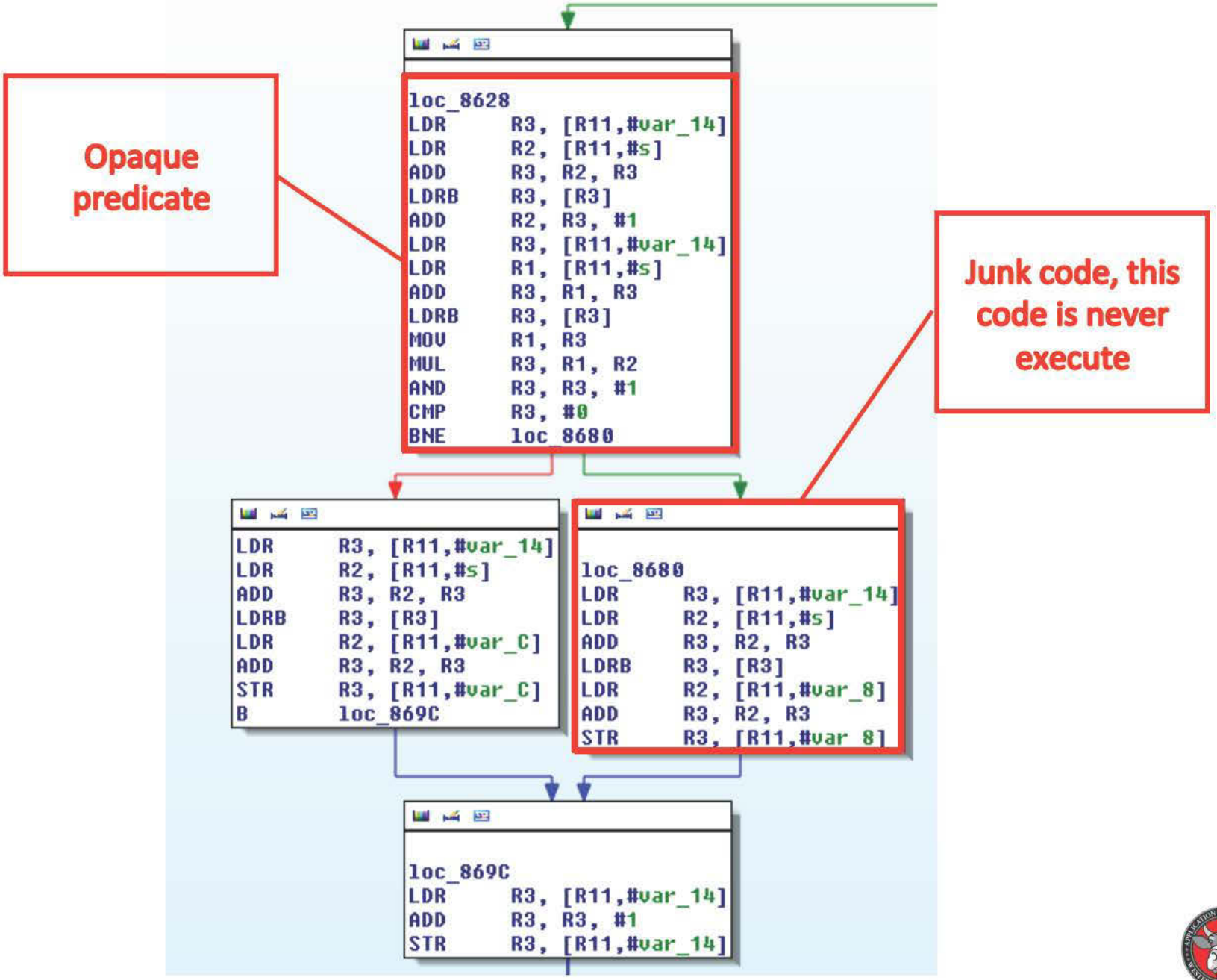
Obfuscation technique : Opaque predicate

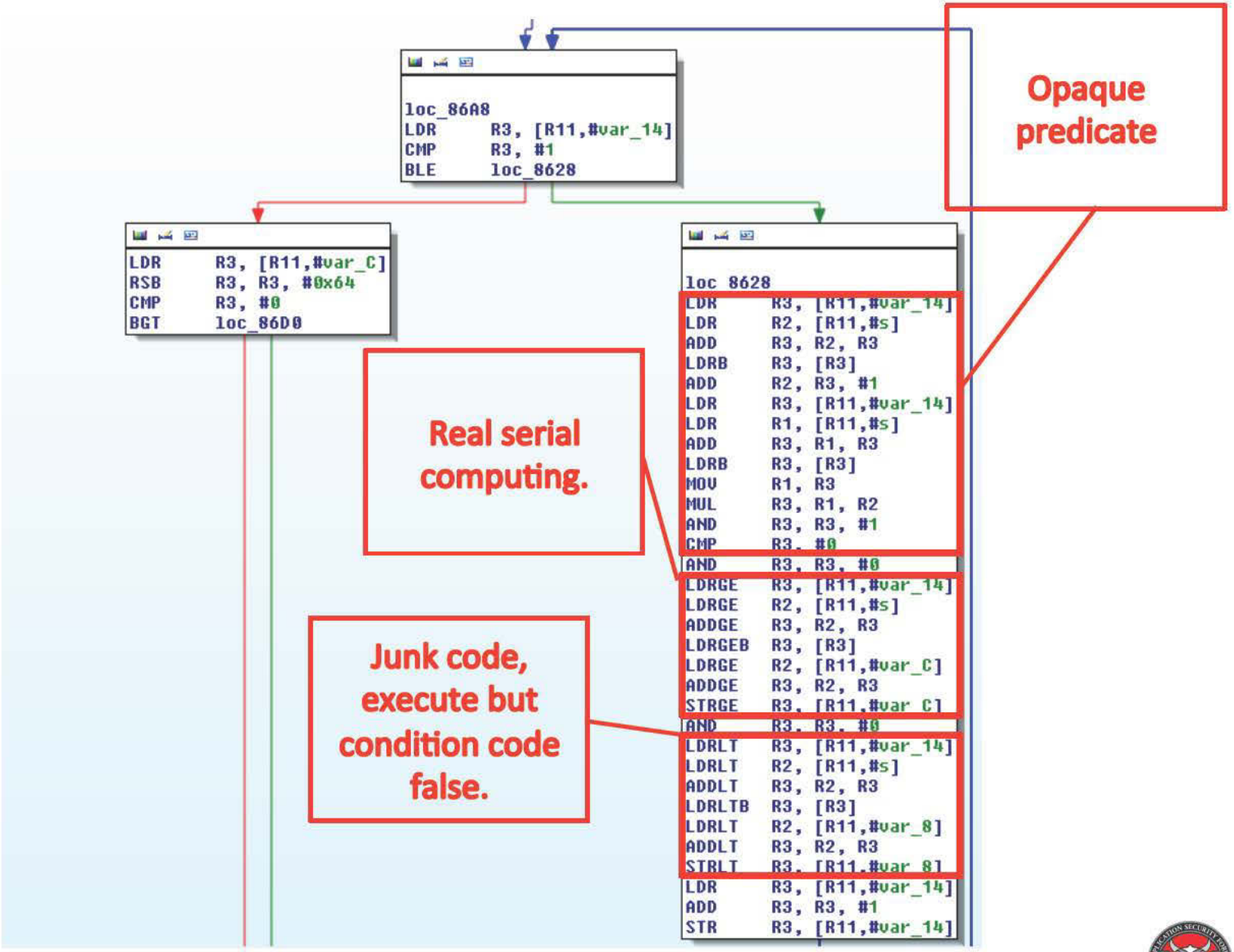
- Example :
  - $(X^2 + X) \% 2 == 0$ is always true
  - $(3X^3 + 2X^2 + X) \% 6 == 0$ is always true
- Opaque predicate aims at:
  - Creating false branch
  - Improving complexity of serial number
  - Watermarking

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 2

int main(char argc, char argv[])
{
    int i;
    printf("Enter your serial (%d digits)\n", SIZE);
    char* serial = malloc(SIZE * sizeof(char));
    memset(serial, 0, SIZE);
    fgets(serial, SIZE, stdin);

    int result = 0;
    int junk = 0;

    for (i = 0; i < SIZE; i++)
    {
        if ((serial[i] + serial[i] * serial[i]) % 2 == 0)
        {
            result += serial[i];
        }
        else
        {
            junk += serial[i];
        }
    }

    if (100 - result <= 0)
    {
        printf("Good job you are authentificate\n");
    }
    else
    {
        printf("Go out hacker %d\n", result);
    }
}
```


Opaque predicate
Junk code, this code is never execute
loc_8628
LDR R3, [R11,#var_14]
LDR R2, [R11,#s]
ADD R3, R2, R3
LDRB R3, [R3]
ADD R2, R3, #1
LDR R3, [R11,#var_14]
LDR R1, [R11,#s]
ADD R3, R1, R3
LDRB R3, [R3]
MOV R1, R3
MUL R3, R1, R2
AND R3, R3, #1
CMP R3, #0
BNE loc_8680
LDR R3, [R11,#var_14]
LDR R2, [R11,#s]
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,#s]
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]


loc_86A8
LDR R3, [R11,#var_14]
CMP R3, #1
BLE loc_8628
LDR R3, [R11,#var_C]
RSB R3, R3, #0x64
CMP R3, #0
BGT loc_86D0
loc_8628
LDR R3, [R11,#var_14]
LDR R2, [R11,#s]
ADD R3, R2, R3
LDRB R3, [R3]
ADD R2, R3, #1
LDR R3, [R11,#var_14]
LDR R1, [R11,#s]
ADD R3, R1, R3
LDRB R3, [R3]
MOV R1, R3
MUL R3, R1, R2
AND R3, R3, #1
CMP R3, #0
AND R3, R3, #0
LDRGE R3, [R11,#var_14]
LDRGE R2, [R11,#s]
ADDGE R3, R2, R3
LDRGEB R3, [R3]
LDRGE R2, [R11,#var_C]
ADDGE R3, R2, R3
STRGE R3, [R11,#var_C]
AND R3, R3, #0
LDRLT R3, [R11,#var_14]
LDRLT R2, [R11,#s]
ADDLT R3, R2, R3
LDRLTB R3, [R3]
LDRLT R2, [R11,#var_8]
ADDLT R3, R2, R3
STRLT R3, [R11,#var_8]
LDR R3, [R11,#var_14]
ADD R3, R3, #1
STR R3, [R11,#var_14]
Opaque predicate
Real serial computing.
Junk code, execute but condition code false.

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

http://crypto.junod.info

romanens.m@gmail.com

Slides:

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