Obfuscator reloaded

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

Please see this as obfuscated software!
In a Nutshell ...

vs.

Doris

Axel
In a Nutshell ...

Malicious Reverse-Engineering
In a Nutshell ...
In a Nutshell ...

Cloud Computing

[Diagram showing a process involving Doris, a Supercomputer, and Axel, leading to Confidential results.]
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
In a Nutshell ...
In a Nutshell ...

@P=split//,",",URRUU
\c8R";@d=split//,\n
reP 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
$_}keys%p;p;p;p;p;map{$p{$_}=~/
^[P.]/&&
close$_}p;wait until$?;map{/^r/&&<
$_>}p;$_=$d[$q];sleep rand(2)if/
\$/;print
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);
}
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
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 ;-) 

- 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
    - \( A \ ^{\ Land} \ B = (A \ & \ \sim B) \ | \ (\sim A \ & \ B) \)
    - \( A \ + \ B = A - (\sim B) \)
    - ...
LLVM-based Obfuscation

- Available obfuscation passes
  - Insertion of fake branches with opaque predicates
    (bachelor thesis of Julie Michielen)
LLVM-based Obfuscation

- Available obfuscation passes
  - Code flattening
LLVM-based Obfuscation

- Code flattening: case of IF-THEN-ELSE

```
entry:
  %retval = alloca i32, align 4
  %argc.addr = alloca i32*, align 4
  %argv.addr = alloca i8**, align 8
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  store i32 0, i32* %retval
  store i32 %argc, i32* %argc.addr, align 4
  store i8** %argv, i8*** %argv.addr, align 8
  %0 = load i8** %argv.addr, align 8
  %arrayidx = getelementptr inbounds i8**, %0, i64 1
  %1 = load i8** %arrayidx, align 8
  %call = call i32 @foo(i8** %1) nounwind readonly
  store i32 %call, i32* %a, align 4
  %2 = load i32* %a, align 4
  %cmp = icmp eq i32 %2, 2,
  br i1 %cmp, label %if.then, label %if.else

if.then:
  %3 = load i32* %a, align 4
  store i32 %3, i32* %b, align 4
  br label %if.end

if.else:
  store i32 0, i32* %b, align 4
  br label %if.end

if.end:
  ret i32 0
```

CFG for 'main' function
LLVM-based Obfuscation

- Code flattening: case of a FOR loop
LLVM-based Obfuscation

- Code flattening: case of a SWITCH

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

  def 1

sw.default:
  store i32 0, i32* %b, align 4
  br label %sw.epilog

sw.bb:
  store i32 1, i32* %b, align 4
  br label %sw.epilog

sw.epilog:
  ret i32 0

CFG for 'main' function
```
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

RECYCLABLE
PAPER
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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileSiz</th>
<th>MemSiz</th>
<th>Flg</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHDR</td>
<td>0x000034</td>
<td>0x00008034</td>
<td>0x00008034</td>
<td>0x000c0</td>
<td>0x000c0</td>
<td>R E</td>
<td>0x4</td>
</tr>
<tr>
<td>INTERP</td>
<td>0x0000f4</td>
<td>0x000080f4</td>
<td>0x000080f4</td>
<td>0x00014</td>
<td>0x00014</td>
<td>R</td>
<td>0x1</td>
</tr>
</tbody>
</table>

[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...

<table>
<thead>
<tr>
<th>Segment</th>
<th>Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>.interp</td>
</tr>
<tr>
<td>01</td>
<td>.interp .hash .dynsym .dynstr .rel.plt .init .plt .text .fini .rodata .eh_frame</td>
</tr>
<tr>
<td>02</td>
<td>.init_array .fini_array .jcr .dynamic .got .data .bss</td>
</tr>
<tr>
<td>03</td>
<td>.dynamic</td>
</tr>
</tbody>
</table>
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:
  - \((X^2 + X) \mod 2 == 0\) is always true
  - \((3X^3 + 2X^2 + X) \mod 6 == 0\) 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
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