Software Obfuscation

Quid Novi ?

Pascal Junod
Julien Rinaldini
Grégory Ruch
~
HEIG-VD

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Outline

- Software Security
- Protection Techniques
- Hacking with LLVM

Software Security

Adversaries

- Let's have a quick look at how cryptographers classify adversaries...

Crypto Black-Box Adversaries

- Model my algorithm/protocol/system as a set of oracles
- Interact with those oracles
  - Ciphertext-only
  - Known plaintext-ciphertext
  - Chosen (adaptively or not) plaintexts and/or ciphertexts

Crypto Black-Box Adversaries

- Prove (mathematically) that your algorithm/protocol/system is secure if the underlying cryptographic primitives are secure.
- Examples:
  - RSA-OAEP
  - RSA-PSS

SW Black-Box Adversaries

- Interact with software, but nicely, i.e. using the common I/O ports :-)
  - Code injection
  - Memory corruption attack
  - Design weaknesses
    - Weak authentication
    - ...

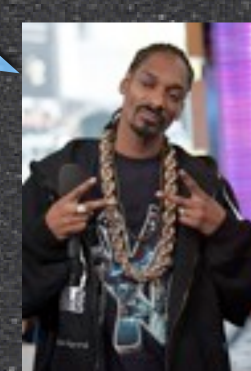
SW Black-Box Adversaries

- Ensure that your software is **as secure as possible**:
  - Threat analysis, definition of the attack surface, security requirements
  - Apply coding best practices
  - Design review, code audits, pen-testing
  - ...

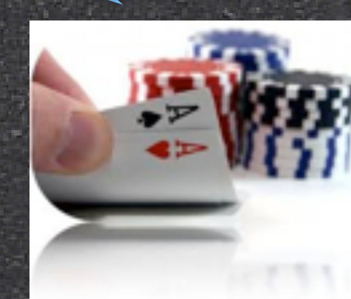
Crypto Grey-Box Adversaries

- Adversaries that were NOT foreseen by the theoretical cryptographers...
- Can interact with the cryptographic primitives, but might have (just) a bit more information about the computations, like:
  - Timings
  - Physical leakage
  - Faults

side-channel information



«tell»



SW Grey-Box Adversaries

- Can interact with the software, but might have (just) a bit more information about the computations, like:
  - Timings
  - Physical leakage
  - Faults
  - Error messages
  - ...

SW Grey-Box Adversaries

- Examples of software attacks by grey-box adversaries:
  - Padding oracles
    - Canvel et al attack against SSL, ASP.NET hack, BEAST 

  - Reverse-engineering of SW code on secured HW with help of a physical leakage channel and templates

Crypto/SW Grey-Box Adversaries

- Ensure that your software is even more secure than possible:
  - Constant-time coding
  - Masking/blinding
  - Use of redundant code
  - ...

Crypto White-Box Adversaries

- Adversaries that most cryptographers just do not want to hear about...

- Can do EVERYTHING they want !!
  - Complete reverse-engineering of SW/HW
  - Read/write all memories, including secure ones (containing keys)
  - Perturb all computations

SW White-Box Adversaries

- Examples of software attacks by black-box adversaries have a $\aleph_0$ cardinality:
  - SW complete reverse-engineering
  - SW cracking
    - Bypass of copy protection systems
    - Bypass of DRM systems
    - Bypass of licensing schemes
    - ...

Attacks based on Faults

- Consider the following piece of code that could valid the RSA signature during the secure boot of a trusted device:

```
if (RSA_verify (signature) ==
RSA_VALID_SIGNATURE) {

  // Perform some critical operation
  } else {
   return NOT_AUTHENTICATED
  }
```

Attacks based on Faults

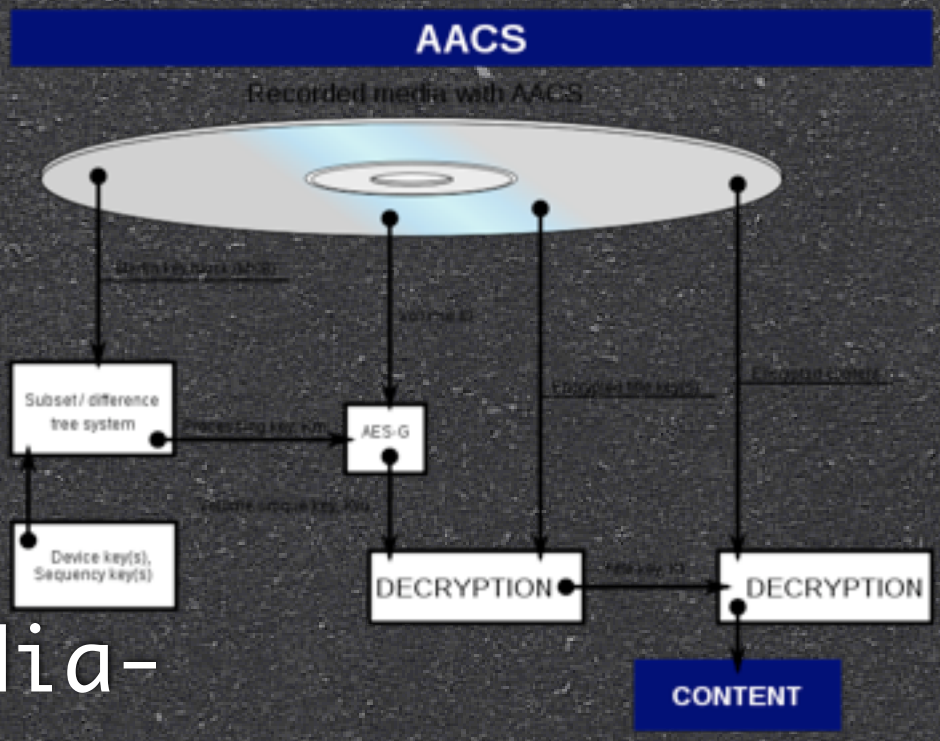
- This could translate into the following:

```
...
cmp     $0x0, %ebx
je      0x64FE89A1
...
```

The whole RSA signature verification mechanism security relies on whether this instruction will be executed or not...

White-Box Adversaries

- One example among many others: the AACS hack
  - DRM protection scheme for Blu-Ray
  - State-of-the art crypto
  - SW player broken
    - Just read the last media-encrypting key in memory :-o

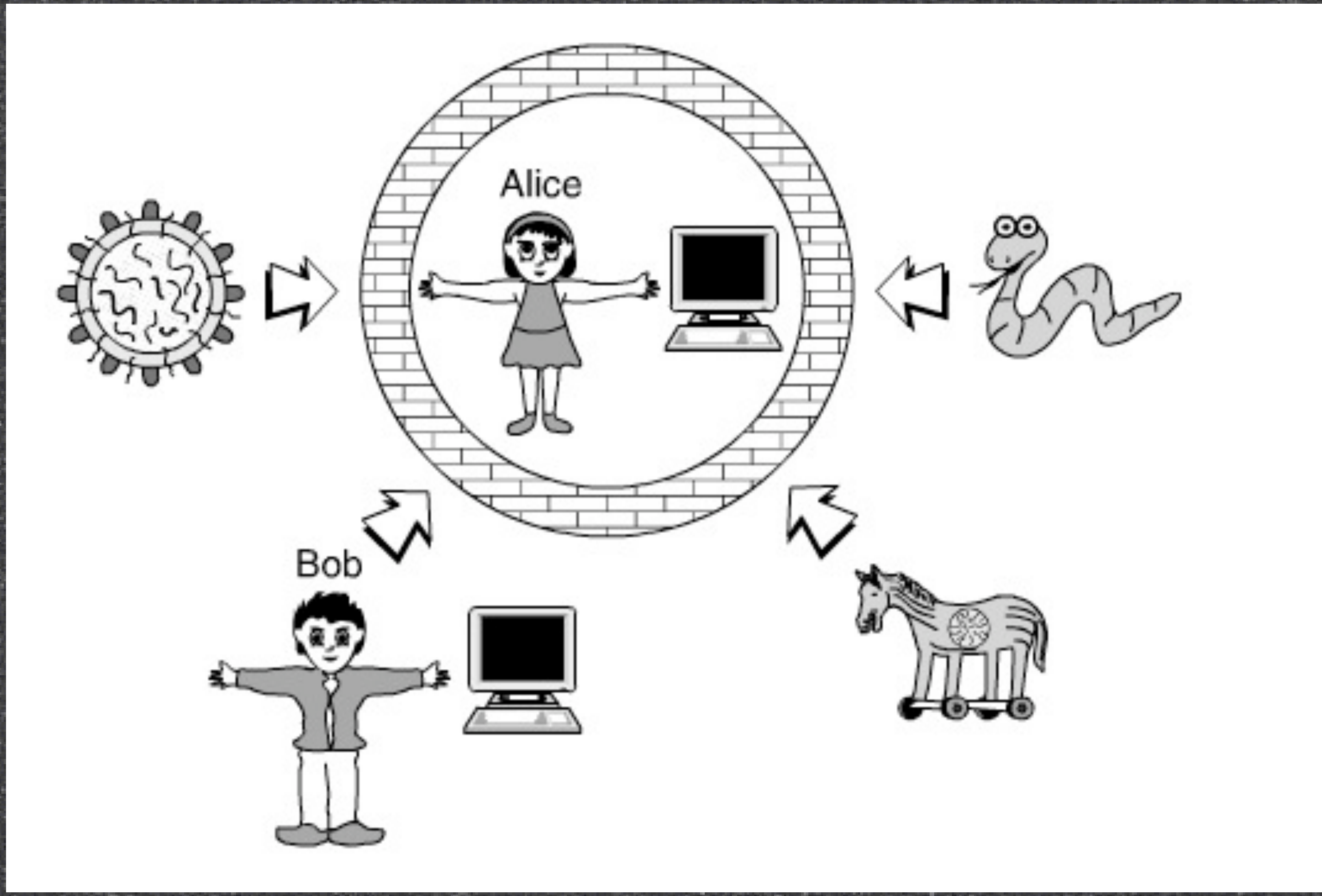


Protection Techniques

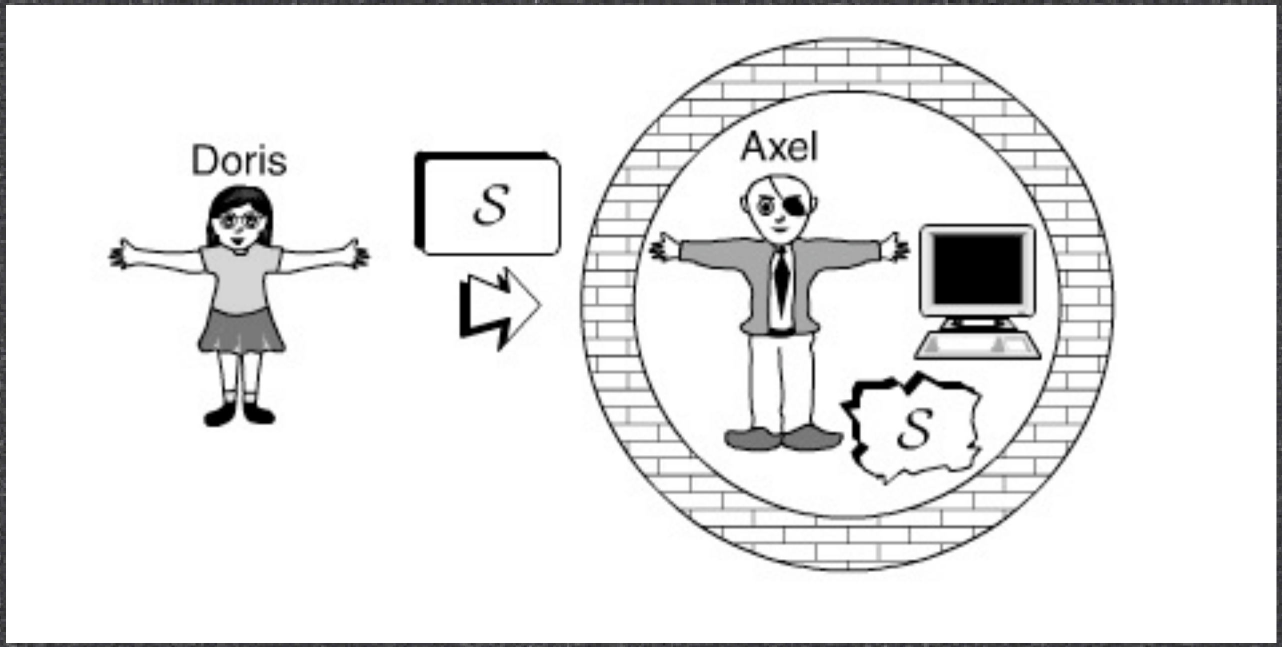
Software Protection

Back to basics ...

Software Protection

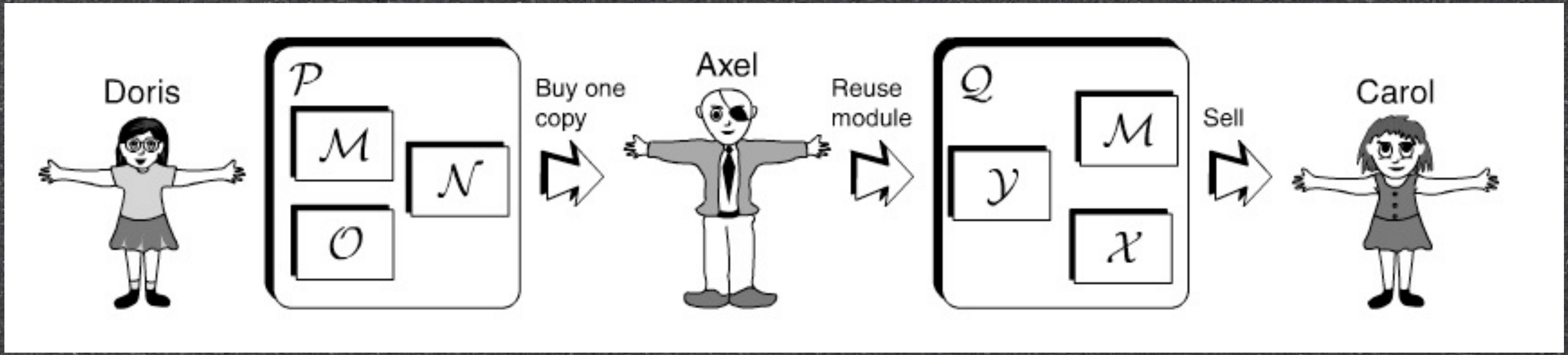


Software Protection



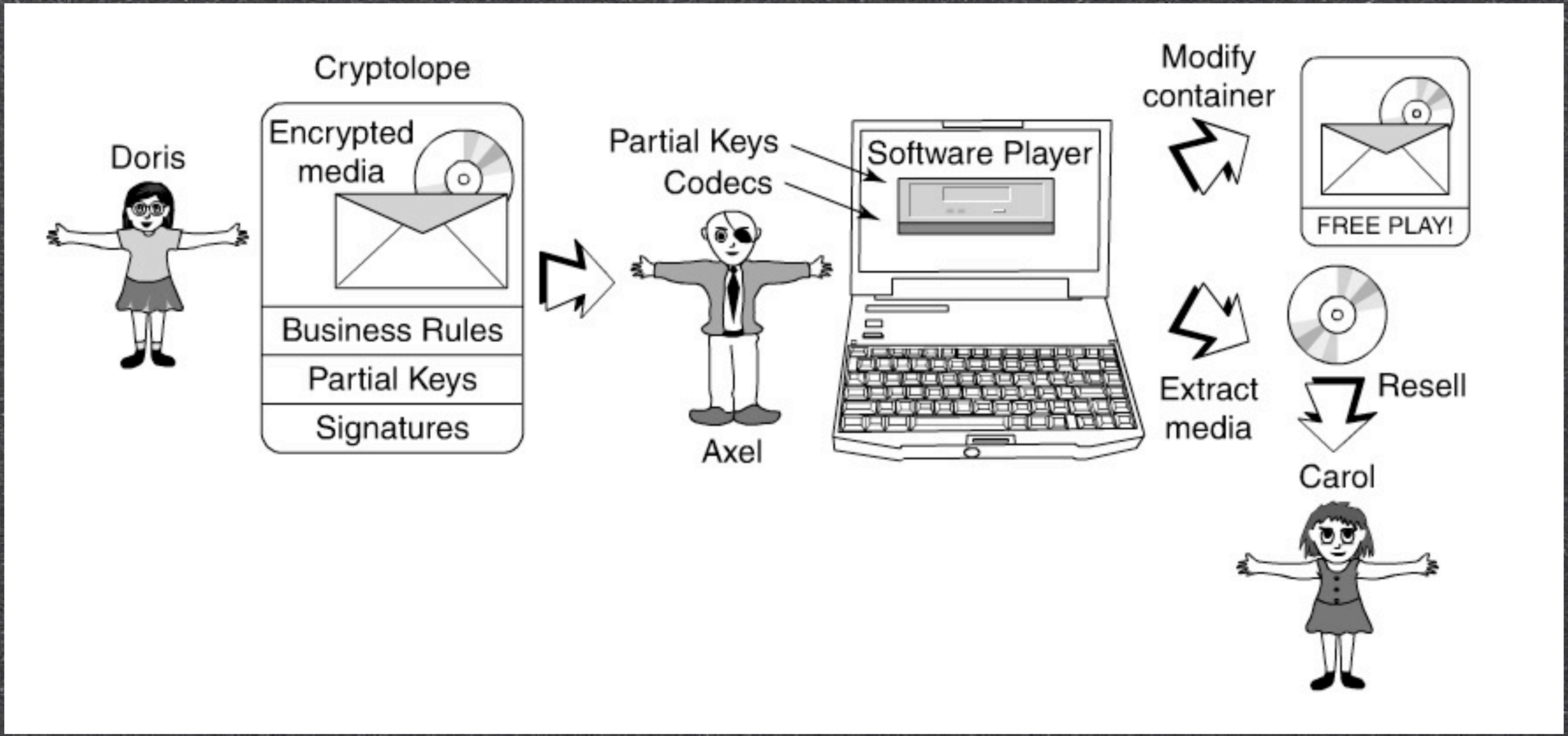
Pictures credit: Collberg, Nagra, «Surreptitious Software», Addison-Wesley, 2009

Malicious Reverse Engineering

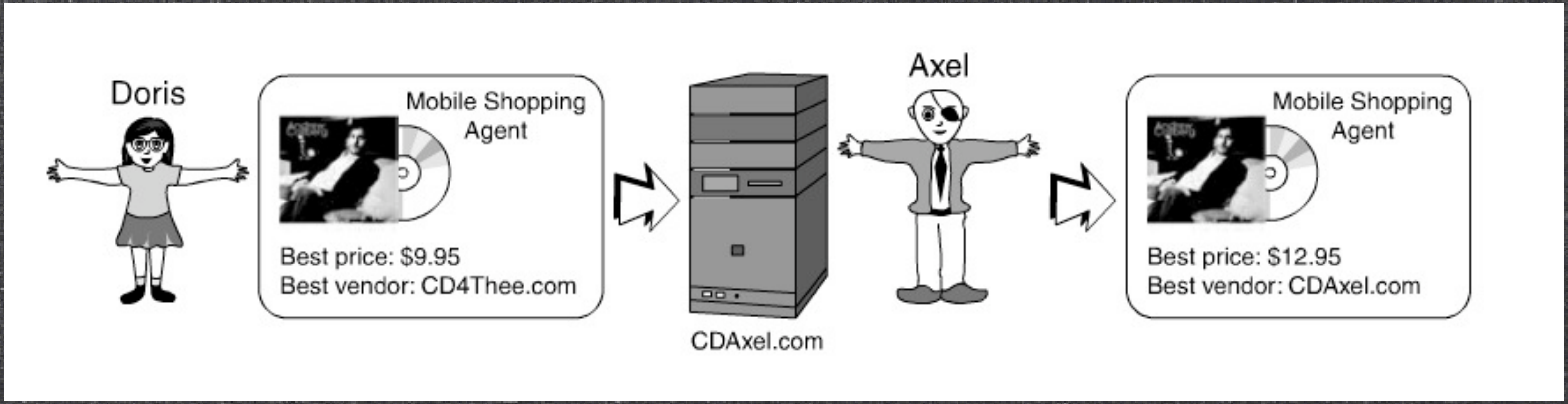


Pictures credit: Collberg, Nagra, «Surreptitious Software», Addison-Wesley, 2009

DRM

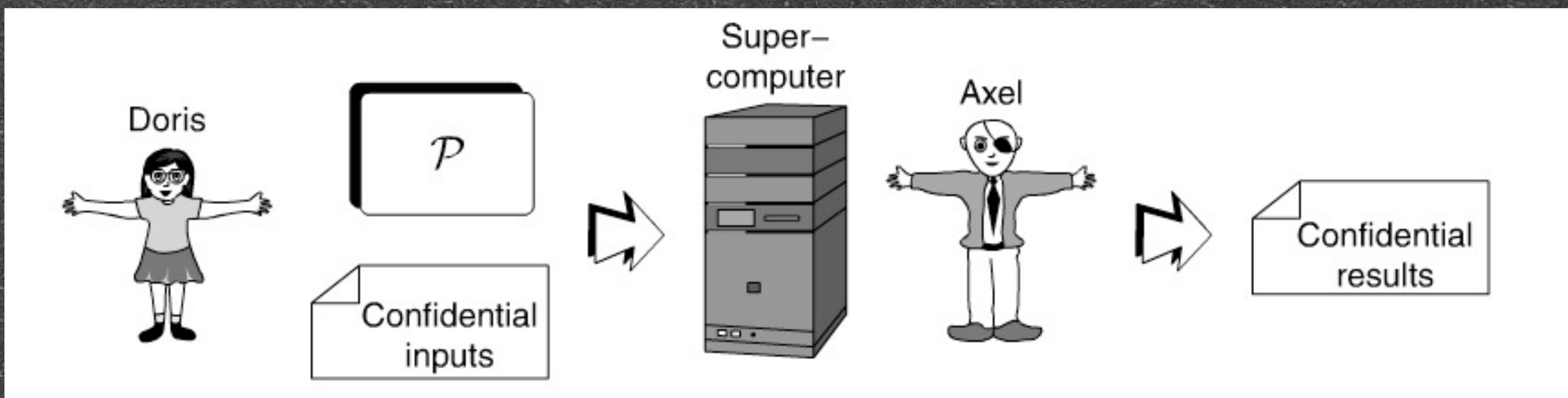

Cryptolope
Doris
Encrypted media
Business Rules
Partial Keys
Signatures
Partial Keys
Codecs
Axel
Software Player
Modify container
FREE PLAY!
Extract media
Resell
Carol

Mobile Computing



Pictures credit: Collberg, Nagra, «Surreptitious Software», Addison-Wesley, 2009

Cloud Computing



Pictures credit: Collberg, Nagra, «Surreptitious Software», Addison-Wesley, 2009

Software Protection

- A typical attack can be decomposed in three phases:
  - Program analysis, algorithm, design and secrets extraction
  - Code modification
    - e.g., removal of license check
  - Distribution
    - Violation of intellectual property right

Software Protection

- Defense possibilities
  - Add confusion to Doris' code, to render the analysis by Axel more costly
  - Add tamper-protection to Doris' code, to prevent him from modifying the code
  - Mark Doris' code

Code obfuscation

SW/ HW-based protections

SW watermarking, fingerprinting

Software Protection

Obfuscation Techniques

Software Protection

- According to Wikipedia, obfuscated code is source or machine code that has been made difficult to understand.

Software Protection

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.

Software Protection

- Even if it is impossible in theory, let's try it in practice :-)
  - Weaker security notion ?
    - Not necessarily resisting all adversaries, but forcing an adversary to work several hours/days/months to break my SW

Software Protection

```
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;
```

Source: Wikipedia

Software Protection

```
@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
```

This tiny Perl program displays the text «Just another Perl/Unix hacker», multiple characters at a time, with delays.

Source: Wikipedia

Software Protection

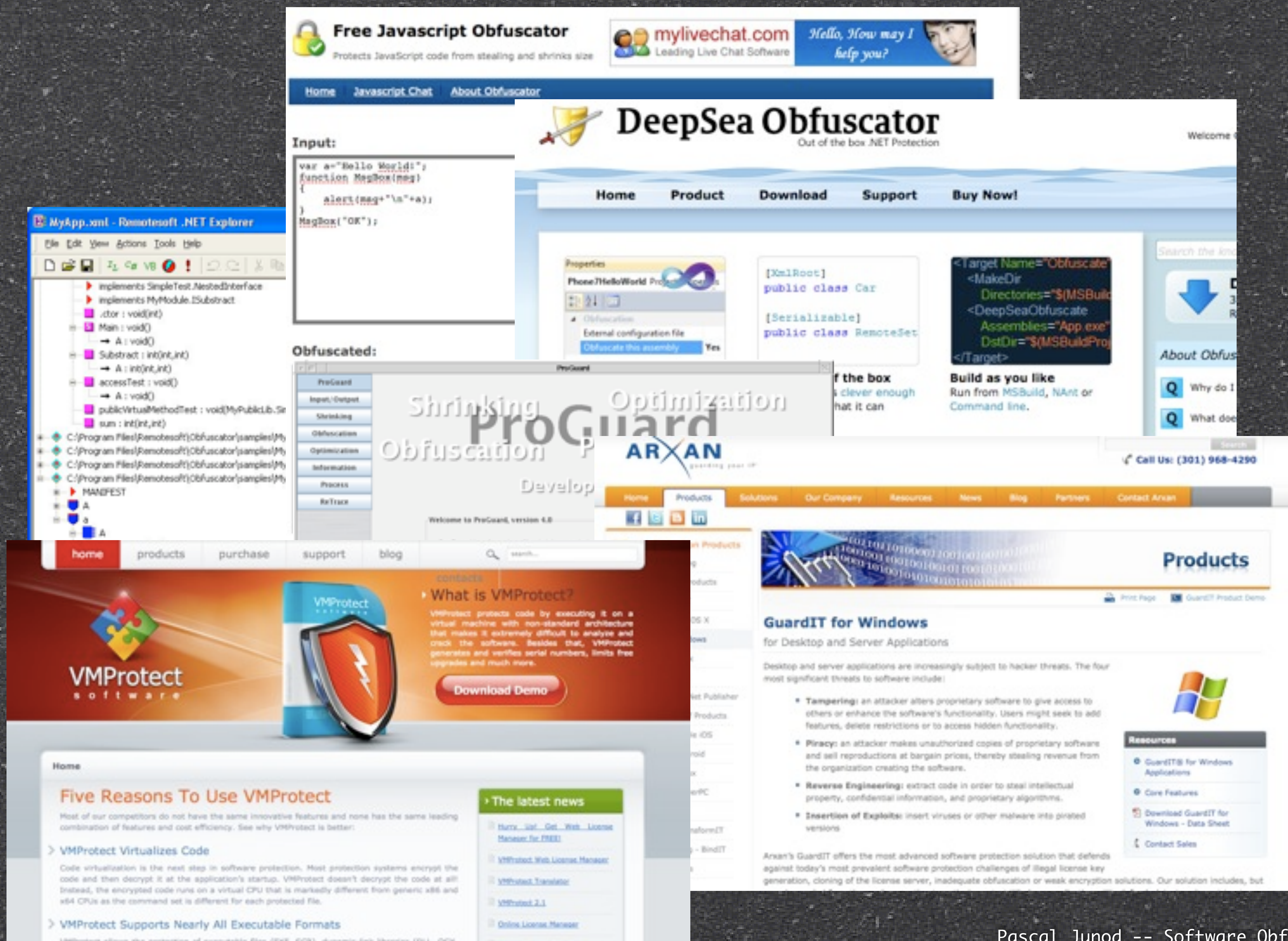
```
void primes(int cap) {
  int i, j, composite;
  for(i = 2; i < cap; ++i) {
    composite = 0;
    for(j = 2; j * j <= i; ++j)
      composite += !(i % j);
    if(!composite)
      printf("%d\t", i);
  }
}

int main(void) {
  primes(100);
}
```

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

This simple C program implements Erastothene’s Sieve.

Source: Wikipedia

Software Protection



Software Protection

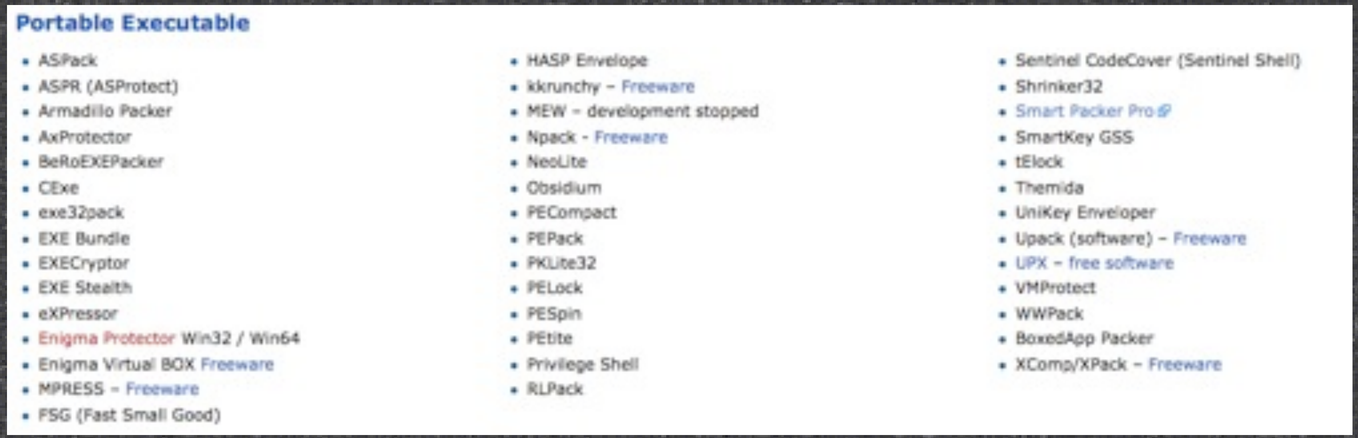
- Very different capabilities
  - Code source vs. binary obfuscation
  - Supported languages
    - .NET, C#, Java, Javascript, C/C++
  - Costs of obfuscation
    - Code size, code speed
  - Resistance to RE

Software Protection

Advanced Obfuscation Techniques

Packing

- Packers (executable compression/ encryption) have been used to thwart RE

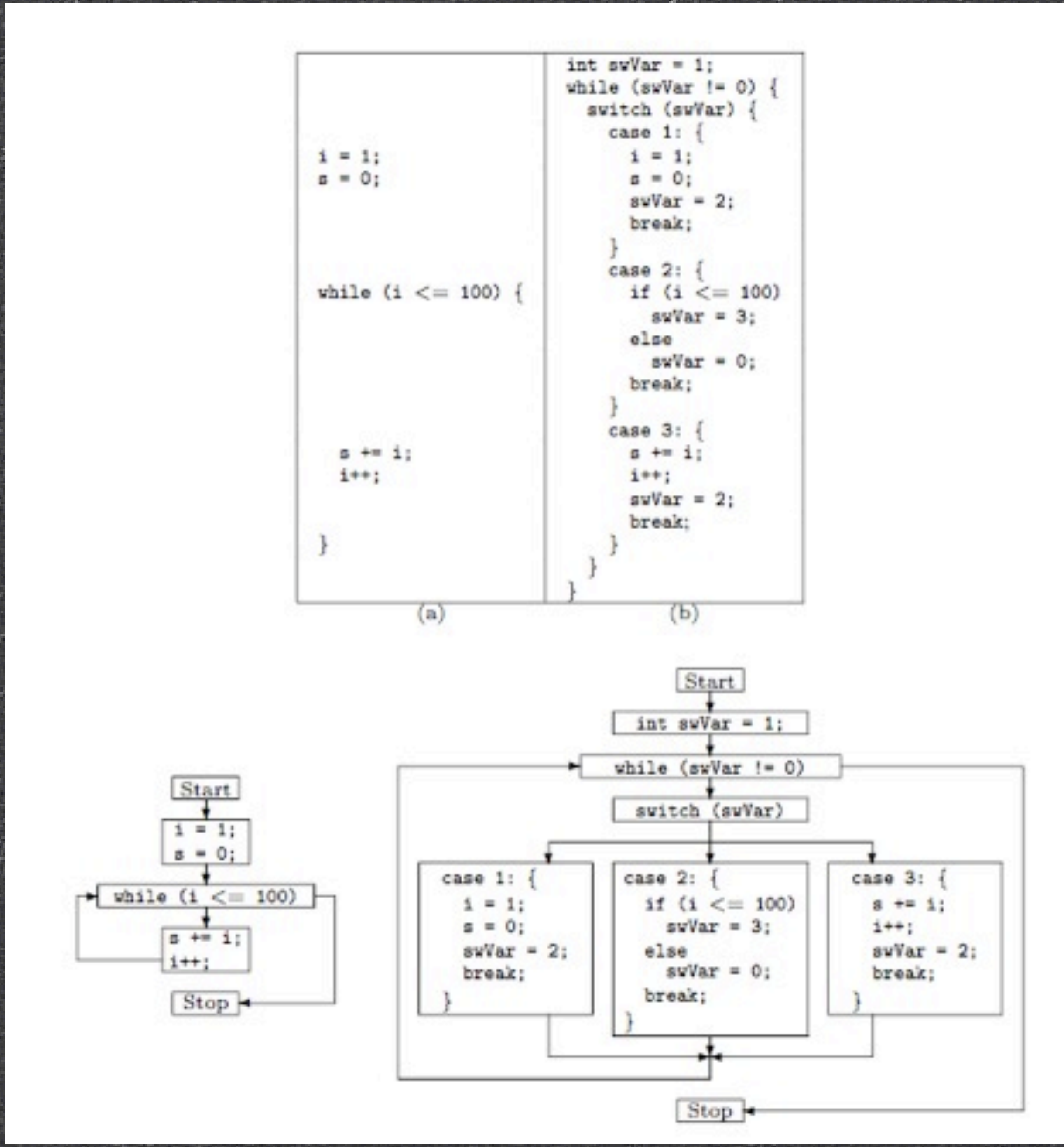


Portable Executable

- ASPack
- ASPR (ASProtect)
- Armadillo Packer
- AxProtector
- BeRoEXEPacker
- CExe
- exe32pack
- EXE Bundle
- EXECryptor
- EXE Stealth
- eXPressor
- Enigma Protector Win32 / Win64
- Enigma Virtual BOX Freeware
- MPRESS – Freeware
- FSG (Fast Small Good)
- HASP Envelope
- kkrunchy – Freeware
- MEW – development stopped
- Npack - Freeware
- NeoLite
- Obsidium
- PECompact
- PEPack
- PKLite32
- PELock
- PESpin
- PEtite
- Privilege Shell
- RLPack
- Sentinel CodeCover (Sentinel Shell)
- Shrinker32
- Smart Packer Pro
- SmartKey GSS
- tElock
- Themida
- UniKey Enveloper
- Upack (software) – Freeware
- UPX – free software
- VMProtect
- WWPack
- BoxedApp Packer
- XComp/XPack – Freeware

Code Flattening

- Code transformations
  - Break loops, conditional structures
  - Transform code in «spaghetti code»
  - ...



Opaque Predicates

- Opaque predicates is a (Boolean) expression for which
  - Outcome a priori known by the programmer
  - Reverse engineer **has to execute it** to know the answer
- Force moving from static analysis to **dynamic analysis/debugging/ emulation**.
- Can be evil if using real crypto :-)

Table 1: Examples of number-theoretical true opaque predicates

$\forall x, y \in \mathbf{Z}$	:	$7y^2 - 1 \neq x^2$
$\forall x \in \mathbf{Z}$	:	$3 \mid (x^3 - x)$
$\forall x \in \mathbf{N}$	:	$14 \mid (3 \cdot 7^{4x+2} + 5 \cdot 4^{2x-1} - 5)$
$\forall x \in \mathbf{Z}$	:	$2 \mid x \vee 8 \mid (x^2 - 1)$
$\forall x \in \mathbf{Z}$	:	$\sum_{i=1, 2 \nmid i}^{2x-1} i = x^2$
$\forall x \in \mathbf{N}$	:	$2 \mid \lfloor \frac{x^2}{2} \rfloor$

Code Interleaving

- Idea: systematically identify portions of code that can be run in parallel (i.e., which do not depend on each others)
  - Interleave those code portions
  - Go down to the instruction level, if possible
  - Maybe add additional junk code

Virtualisation

- Push the concept of packer a bit further, and rewrite SW as (custom) byte-code aimed for a (custom) VM
- Idea can be pushed further by iterating the process, but beware of performances...
- Use tricky, esoteric, exotic, but still Turing-complete computer architectures:
  - brainfuck (8 instr. with no operands)
  - subleq (1 instr. with 3 operands)
  - ...

Playing with Networks

- When you have a (fast network) at your disposal, you can leverage this !
  - Sending tamper-proofing challenges with 1s answer delay
  - Sending sensitive code encrypted and on-demand
- Techniques mostly used by the online gaming industry to avoid cheating.

Hacking with LLVM

Why ?

- No satisfactory open-source tool able to obfuscate C/C++
- Cool new research playground !
- Have already worked on the subject in other lives
- RE is hard, protecting against RE in an efficient way is even harder !

Obfuscator Project

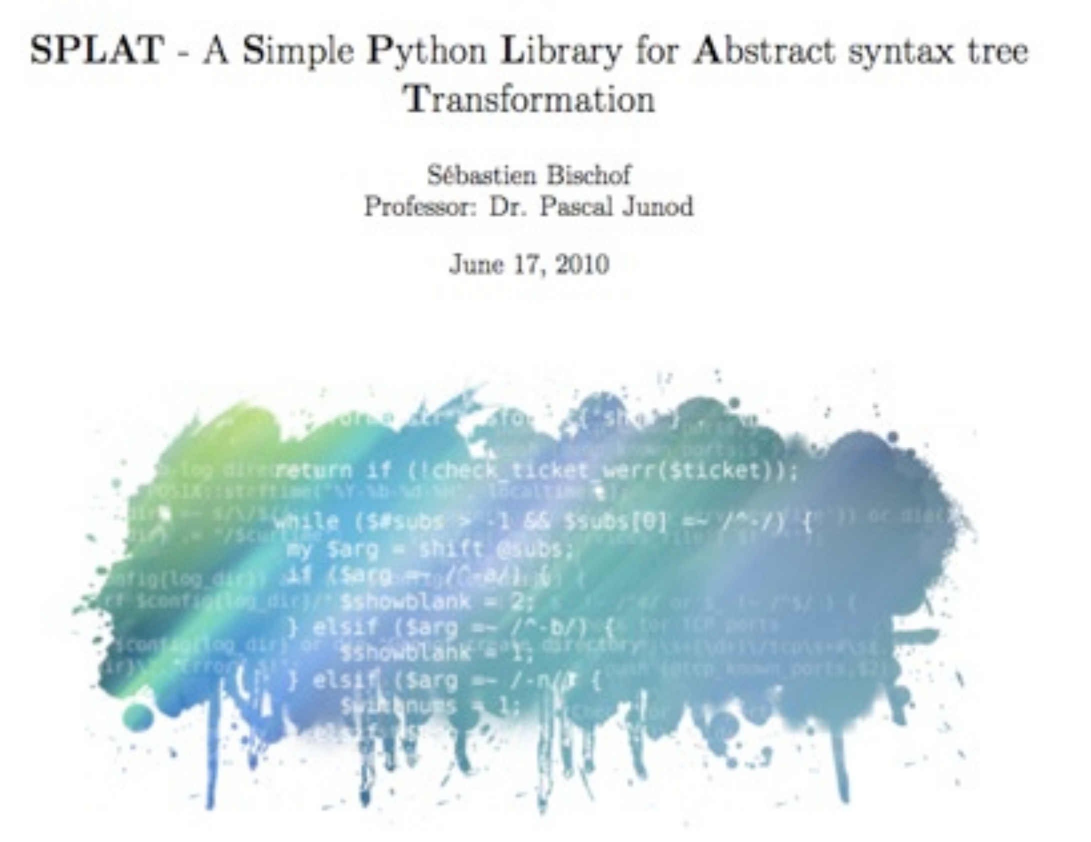
- HES-SO-funded 1-year project
  - HEIG-VD
    - PJ, Greg Ruch, Julien Rinaldini + master students, with focus on source code obfuscation
  - EIA-FR
    - Jean-Roland Schuler + research assistants, with focus on binary code obfuscation

Obfuscator Project

- Goals of «Obfuscator»:
  - Create knowledge & know-how on the subject at HES-SO
  - Develop prototype tools

Obfuscator Project

- Explored and abandoned paths:
  - C language parser and code modifier written in Python (Sébastien Bischof)
    - Code flattening implemented
    - Too much time spent on the parser :-/


SPLAT - A Simple Python Library for Abstract syntax tree Transformation

Sébastien Bischof
Professor: Dr. Pascal Junod

June 17, 2010

Obfuscator Project

- Explored and abandoned paths:
  - Extending the GNU compiler suite
    - Almost no documentation

GCC, the GNU Compiler Collection

The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgcj,...). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom.

We strive to provide regular, high quality releases, which we want to work well on a variety of native and cross targets (including GNU/Linux), and encourage everyone to contribute changes or help testing GCC. Our sources are readily and freely available via SVN and weekly snapshots.

Obfuscator Project

- Explored and abandoned paths:
  - Extending the LLVM CLang front-end (Grégory Ruch)
    - APIs not very stable, and not really adapted for this task

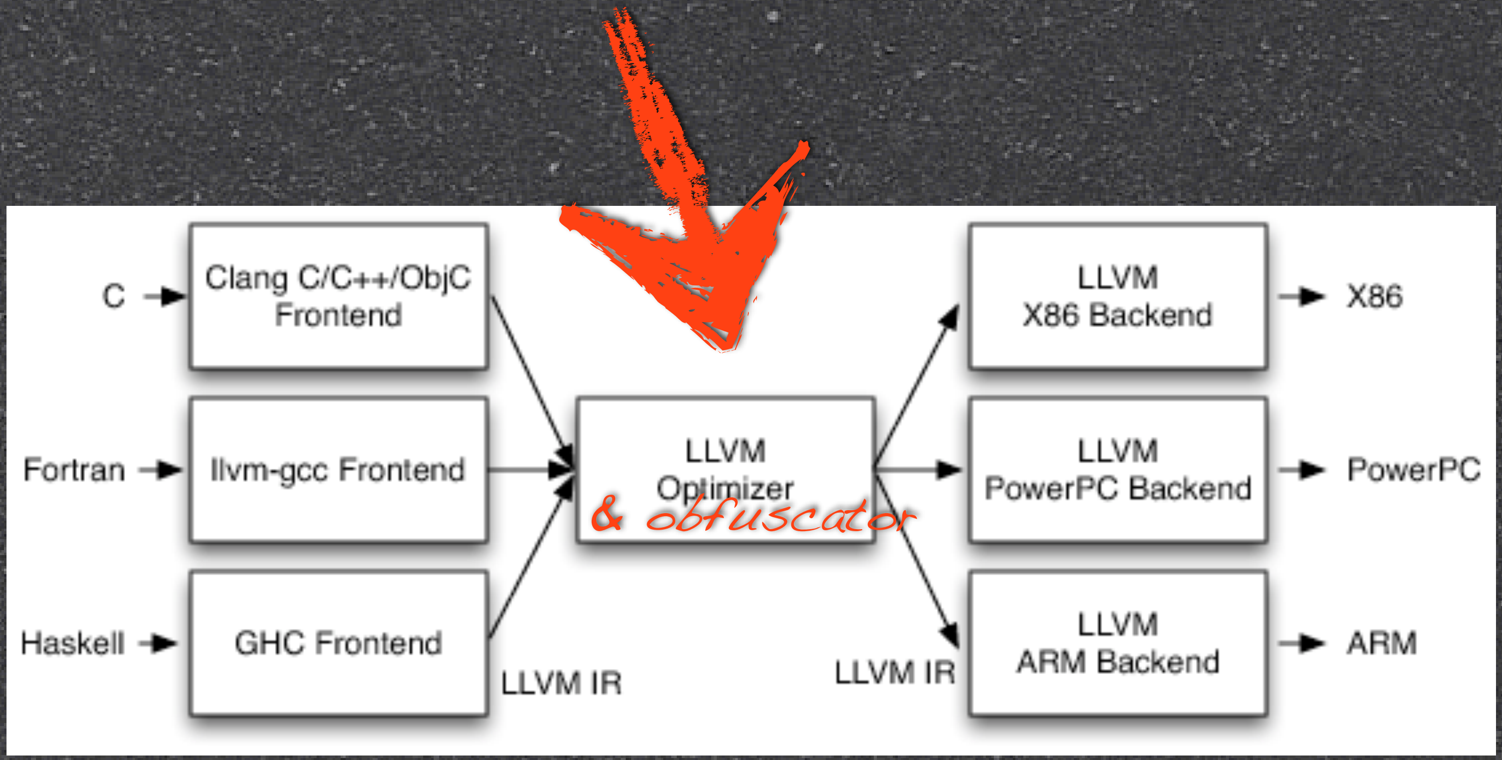
Obfuscator - Abstract syntax tree Transformation

Grégory Ruch
Professeur : Dr. Pascal Junod

29 avril 2011

Obfuscator Project

- Why focusing on C/C++ ?!
- Why not using the excellent design of LLVM and be (a bit) more ambitious ?



Obfuscator Project

- Let us work on the LLVM intermediate representation (IR) !
  - Clean and stable APIs
  - Fully documented

```
define i32 @add1(i32 %a, i32 %b) {
entry:
  %tmp1 = add i32 %a, %b
  ret i32 %tmp1
}

define i32 @add2(i32 %a, i32 %b) {
entry:
  %tmp1 = icmp eq i32 %a, 0
  br i1 %tmp1, label %done, label %recurse

recurse:
  %tmp2 = sub i32 %a, 1
  %tmp3 = add i32 %b, 1
  %tmp4 = call i32 @add2(i32 %tmp2, i32 %tmp3)
  ret i32 %tmp4

done:
  ret i32 %b
}
```

Obfuscator Project

- LLVM interesting features
  - Front-ends for C, C++, Objective-C, Fortran, Java, Ada, ...
  - Back-ends for X86, X86-64, PowerPC, PowerPC-64, ARM, Thumb, SPARC, Alpha, CellSPU, MIPS, MSP430, SystemZ, and XCore
  - «Optimizations» passes order can be fine-tuned

Obfuscator Project

- Current concrete results, besides failures:
  - Developed a test-bed (!)
  - Developed several simple obfuscations passes
    - Code substitution
  - Development of a code-flattening pass on-going
- But still a long, hard way to go !!

Obfuscator Project

```
// For each basicblock
        for (Function::iterator i = F.begin(), e = F.end(); i != e; ++i) {
            BasicBlock *blk = i;
            // For each instruction
            for (BasicBlock::iterator j = blk->begin(), e = blk->end(); j != e; ++j) {
                llvm::Instruction *inst = j;
                // If it's a binary operation
                if(inst->isBinaryOp()) {
                    unsigned opcode = inst->getOpcode();
                    int operand;
                    // Go to right opcode
                    switch(opcode) {
                        case Instruction::Add:
                        case Instruction::FAdd:
                            // Choose one substitution randomly
                            switch(rand()%6) {
                                // Implementation of a = b - (-c)
                                case 0:
                                    // Create a neg value
                                    varName += "_";
                                    var = new Twine(varName);

                                    // Create sub
                                    if(opcode == Instruction::FAdd) {
                                        op = BinaryOperator::CreateFNeg(cast<Value>(inst-
>getOperand(1)),*var,inst);
                                        finalOp = BinaryOperator::Create(Instruction::FSub,
cast<Value>(inst->getOperand(0)),cast<Value>(op),inst->getNameStr(),inst);
                                    }
                                    else {
                                        op = BinaryOperator::CreateNeg(cast<Value>(inst-
>getOperand(1)),*var,inst);
                                        finalOp = BinaryOperator::Create(Instruction::Sub,
cast<Value>(inst->getOperand(0)),cast<Value>(op),inst->getNameStr(),inst);
                                    }

                                    // Check signed wrap
```

Conclusion

Conclusion

- Obfuscation is a fascinating research playground
- Obfuscation needs will increase in the future (personal prediction)
  - Mobile platforms
  - Embedded SW platforms

Contact Information

- Website http://crypto.junod.info
- Twitter @cryptopathe
- E-mail pascal@junod.info