Engineering Code Obfuscation

EUROCRYPT 2016

Christian Collberg

Department of Computer Science
University of Arizona

http://collberg.cs.arizona.edu
collberg@gmail.com

Supported by NSF grants 1525820 and 1318955 and by the private foundation that shall not be named
Man-At-The-End Applications
Tools and Counter Tools
Obfuscation vs. Deobfuscation
Deploying Obfuscation
Evaluation
Discussion
Man-at-the-End Scenarios
snapchat() {
    after (8 seconds)
        remove_picture();
    if (screenshot())
        notify_sender();
    if (app_is_tampered() ||
        env_is_suspicious() ||
        bob_is_curious())
        punish_bob();
}
Security and Privacy Scientist
MATE attacks occur in any setting where an adversary has physical access to a device and compromises it by inspecting, reverse engineering, or tampering with its hardware or software.
Tamper → Clone → Keys → Code & Content
set_top_box() {
    if (bob_paid("ESPN"))
        allow_access();

    if (hw_is_tampered() ||
        sw_is_tampered() ||
        bob_is_curious() ||...
        punish_bob();
}

Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*
Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*
Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*
Tools vs. Counter Tools
Prog() {
}

Tool

Prog'
Prog() {

Assets
• Source
• Algorithms
• Keys
• Media

}
Protection?  
Overhead?

Prog' Code Transformations

Prog() {

Assets

• Source
• Algorithms
• Keys
• Media

}

Tool

Prog’
Protection?
Overhead?

Prog()

Obfuscation
Tamperproofing
Remote Attestation
Whitebox Cryptography
Environment Checking
Watermarking

Assets
• Source
• Algorithms
• Keys
• Media

Overhead?
Protection?

Tool

Prog'
Prog() {
Assets
• Source
• Algorithms
• Keys
• Media
}

Code Transformations
Obfuscation
Tamperproofing
Remote Attestation
Whitebox Cryptography
Environment Checking
Watermarking

Assets
• Source
• Algorithms
• Keys
• Media

Obfuscator-LLVM
VMProtect software
Aspire
Kudelski Group
Tigress
ARXAN
irdeto
Virtualizer
NAGRA
Code Analyses

Static analysis
Concolic analysis
Decomposition
Debugging
Dynamic analysis
Disassembly
Slicing
Emulation

Assets
- Source
- Algs
- Keys
- Data

Prog’ Tool
Code Analyses

Static analysis
Concolic analysis
 Decompilation
 Debugging
 Dynamic analysis
 Disassembly
 Slicing
 Emulation

Time?
Precision?

Prog'
Tool

Assets
- Source
- Algs
- Keys
- Data
What Matters?

Performance

Time-to-Crack

Stealth

Performance

Time-to-Crack

Stealth
<table>
<thead>
<tr>
<th>Metric</th>
<th>Program</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute time</td>
<td>application</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>relative</td>
<td>application</td>
<td>1.5x</td>
</tr>
<tr>
<td>relative</td>
<td>security kernel</td>
<td>100x-1000x</td>
</tr>
</tbody>
</table>
## Performance Matters?

<table>
<thead>
<tr>
<th>Metric</th>
<th>Program</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute time</td>
<td>application</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>relative</td>
<td>application</td>
<td>1.5x</td>
</tr>
<tr>
<td>relative</td>
<td>security kernel</td>
<td>100x-1000x</td>
</tr>
</tbody>
</table>

**Code virtualizer** | **ExeCryptor** | **VMProtect** | **Themida**  
100x                  | 700x             | 500x           | 1200x

Liem, Gu, Johnson: A compiler-based infrastructure for software-protection, PLAS’08
## Indistinguishability Obf.

<table>
<thead>
<tr>
<th>Program</th>
<th>Generate</th>
<th>Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-bit multiplier</td>
<td>1027 years</td>
<td>$10^8$ years</td>
</tr>
<tr>
<td>16-bit point function</td>
<td>7 hours, 25G</td>
<td>4 hours (later, 20 minutes)</td>
</tr>
</tbody>
</table>

Bernstein et al., Bad Directions in Cryptographic Hash Functions, IS&P’15
Apon, et al., Impl. Cryptographic Program Obfuscation, CRYPTO’14
# Time-to-Crack Matters

<table>
<thead>
<tr>
<th>Program</th>
<th>Adversary</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw+sw</td>
<td></td>
<td>many years</td>
</tr>
<tr>
<td>well protected</td>
<td>highly skilled, motivated</td>
<td>4-6 weeks</td>
</tr>
<tr>
<td>≈ VMProtect</td>
<td>experienced reverse engineer</td>
<td>≈ 12 months</td>
</tr>
<tr>
<td>mass market malware</td>
<td></td>
<td>minutes-hours</td>
</tr>
</tbody>
</table>
Obfuscation vs. Deobfuscation
$P_0 \rightarrow \text{Tigress}$
<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>add</td>
<td>push(pop()+pop())</td>
</tr>
<tr>
<td>1</td>
<td>store L</td>
<td>Mem[L]=pop()</td>
</tr>
<tr>
<td>2</td>
<td>breq L</td>
<td>if pop()==pop() goto L</td>
</tr>
</tbody>
</table>
# Virtual Instruction Set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>add</td>
<td>push(pop()+pop())</td>
</tr>
<tr>
<td>1</td>
<td>store L</td>
<td>Mem[L]=pop()</td>
</tr>
<tr>
<td>2</td>
<td>breq L</td>
<td>if pop()==pop() goto L</td>
</tr>
</tbody>
</table>

## DISPATCH

```c
void P1()
{
    VPC = 0;
    STACK = [];
}
```

## Tigrress

```
Tigress
```
Virtual Program Array

 Virtual Instruction Set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>add</td>
<td>push(pop() + pop())</td>
</tr>
<tr>
<td>1</td>
<td>store L</td>
<td>Mem[L] = pop()</td>
</tr>
<tr>
<td>2</td>
<td>breq L</td>
<td>if pop() = pop() goto L</td>
</tr>
</tbody>
</table>

void P_1()
{
    VPC = 0;
    STACK = [];
}

DISPATCH

HANDLER

HANDLER
void P1()
{
    VPC = 0;
    STACK = [];

    NEXTINSTR[VPC]

    add:{push(pop()+pop())}

    store:{Mem[L]=pop()}
}
\[
\text{SEED}
\]

\[
\text{P}_0
\]

\[
\text{Opcode} \quad \text{Mnemonic} \quad \text{Semantics}
\]

\[
\text{void} \ P_1()\{
\quad \text{VPC} = 0;
\quad \text{STACK} = [];
\quad \text{NEXTINSTR}[\text{VPC}]
\}
\]

\[
\text{add} : \{ \text{push}(\text{pop()} + \text{pop}) \}\}
\]

\[
\text{store} : \{ \text{Mem}[L] = \text{pop()} \}\}
\]
add:
  push(pop() + pop());
  VPC++;
}

store:
  Mem[L] = pop();
  VPC++=2;
}
add:
{
    push(pop() + pop());
    VPC++;
}

store:
{
    Mem[L] = pop();
    VPC += 2;
}
add:
  push(pop() + pop());
  VPC++;
}

store:
  Mem[L] = pop();
  VPC += 2;
}

NEXTINSTR[VPC]

VPC

add  store  L  ...

Manual Analysis

Rolles, Unpacking virtualization obfuscators, WOOT'09
Manual Analysis

Rolles, Unpacking virtualization obfuscators, WOOT'09
Manual Analysis

Manually reverse engineer instruction set

Virtual Instruction Set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Manually construct DISASSEMBLER

Rolles, Unpacking virtualization obfuscators, WOOT'09
Manual Analysis

Manually reverse engineer instruction set

Virtual Program Array

Virtual Instruction Set

Opcode  Mnemonic  Semantics

DISASSEMBLER

x86 machine code

Rolles, Unpacking virtualization obfuscators, WOOT'09
Manual Analysis

Rolling, Unpacking virtualization obfuscators, WOOT'09

Manual reverse engineer instruction set

Virtual Instruction Set
- Opcode
- Mnemonic
- Semantics

Manually construct

Virtual Program Array

DISASSEMBLER

OPTIMIZE + DECOMPILE

C source code

x86 machine code
• Superoperators
• Randomize operands
• Randomize opcodes
• Random dispatch

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>R[b]=L[a]; R[c]=M[R[d]]; R[f]=L[e]; M[R[g]]=R[h]; R[i]=L[j]; R[l]=L[k]; S[++sp]=R[m]; pc+=53;</td>
</tr>
</tbody>
</table>

```
pc++; regs[*(pc+4)]._vs=(void*)(locals++(pc));
regs[*(pc+8)]._int=*(regs[(*(pc+12))._vs);  
regs[*(pc+20)]._vs=(void*)(locals++*(pc+16));
*(regs[*(pc+24)]._vs)=regs[*(pc+28)]._int;  
regs[*(pc+32)]._vs=(void*)(locals++*(pc+36));  
regs[*(pc+44)]._vs=(void*)(locals++*(pc+40)); *
stack[sp+1]._int=*(regs[*(pc+48)]._vs);  
sp++; pc+=52; break;
```
Composition

$P_0$ → $T_1$ → $T_2$ → ...
Static Analysis

• Automatically reason about the program without executing it
• A sound analysis computes a valid over-approximation of the program semantics

Property that holds for all possible executions of the program

Kinder, Towards Static Analysis of Virtualization-Obfuscated Binaries, WCRE’12
vpc = 0

```
op = prog[vpc]

op == 03 (INC)
```

```
INC:

... vpc += 2;
```

```
op == 52 (MOV)
```

```
MOV:

... vpc += 3;
```

Abstract domain: interval of VPC indices
Abstract domain: interval of VPC indices
Abstract domain: interval of VPC indices
Abstract domain: interval of VPC indices
\( vpc \in [0, 0] \quad vpc = 0 \quad vpc \in [3, 3] \)

**op = prog[vpc]**

- **op == 03 (INC)**
  - \( vpc += 2; \)

- **op == 52 (MOV)**
  - \( vpc += 3; \)

**Abstract domain:** interval of VPC indices

**Program:**

- MOV x y INC x INC x
- MOV: { \( \ldots \) vpc += 3; }
- INC: { \( \ldots \) vpc += 2; }
Abstract domain: interval of VPC indices
\[ \text{vpc} \in [0,0] \cup [3,3] = [0,3] \]

\[ \text{vpc} = 0 \]

\[ \text{op} = \text{prog}[\text{vpc}] \]

\[ \text{op} == 03 \text{(INC)} \]

\[ \text{INC:} \{ \]
\[ \text{... } \text{vpc} += 2; \]
\[ \} \]

\[ \text{op} == 52 \text{(MOV)} \]

\[ \text{MOV:} \{ \]
\[ \text{... } \text{vpc} += 3; \]
\[ \} \]

Abstract domain: interval of VPC indices
\( \text{Abstract domain: interval of VPC indices} \)

\[ \text{Opcodes?} \]

\begin{align*}
\text{MOV} & : \{ \text{...} \quad x & \to & y \quad \text{INC} & : \{ \text{...} \quad vpc & += & 2; \} \\
& & & & + & = & 3; \} \\
\} \end{align*}
\[ \text{vpc} \in [0,0] \]
\[ \text{vpc} \in [0,0] \cup [3,3] = [0,3] \]

\[ \text{op} = \text{prog}[\text{vpc}] \]

\[ \text{op} == 03 (\text{INC}) \]
\[ \text{vpc} \in [0,3] \]

\[ \text{INC}: \{ \]
\[ \text{...} \]
\[ \text{vpc} += 2; \]
\[ \} \]

\[ \text{op} == 52 (\text{MOV}) \]

\[ \text{MOV}: \{ \]
\[ \text{...} \]
\[ \text{vpc} += 3; \]
\[ \} \]

Abstract domain:
interval of VPC indices
Abstract domain: interval of VPC indices

vpc ∈ [0, 0] ∪ [3, 3] = [0, 3]

op = prog[vpc]

op == 03 (INC)

INC: {
    ... vpc += 2;
}

op == 52 (MOV)

MOV: {
    ... vpc += 3;
}

vpc ∈ [2, 5]
\( vpc \in [0,0] \)
\( vpc \in [0,0] \cup [3,3] = [0,3] \)

\( \text{op} = \text{prog}[vpc] \)

\( \text{INC}: \{ \)
\( \quad \ldots \quad vpc += 2; \)
\( \} \)

\( \text{INC}: \{ \)
\( \quad \ldots \quad vpc += 2; \)
\( \} \)

\( \text{MOV}: \{ \)
\( \quad \ldots \quad vpc += 3; \)
\( \} \)

\( \text{op} = 03 \) (INC)

\( \text{op} = 52 \) (MOV)

Abstract domain:
interval of VPC indices
\[ vpc \in [0,0] \]
\[ vpc \in [0,0] \cup [3,3] = [0,3] \]
\[ vpc \in [0,3] \cup [2,5] = [0,5] \]

**op=prog[vpc]**

**op==03 (INC)**

**INC:**

\[
\begin{align*}
&\text{... } vpc+=2; \\
&\text{... }
\end{align*}
\]

**op==52 (MOV)**

**MOV:**

\[
\begin{align*}
&\text{... } vpc+=3; \\
&\text{... }
\end{align*}
\]

Abstract domain: interval of VPC indices
\[ vpc \in [0,0] \]
\[ vpc \in [0,0] \uplus [3,3] = [0,3] \]
\[ vpc \in [0,3] \uplus [2,5] = [0,5] \]

**op = prog[ vpc ]**

\[ op == 03 \text{ (INC)} \]
\[ vpc \in [3,5] \]

**INC:**

\[
\begin{align*}
\ldots & \quad vpc++=2; \\
\end{align*}
\]

**MOV:**

\[
\begin{align*}
\ldots & \quad vpc++=3; \\
\end{align*}
\]

Abstract domain:
interval of VPC indices
$vpc \in [0, 0]$

$vpc \in [0, 0] \cup [3, 3] = [0, 3]$

$vpc \in [0, 3] \cup [2, 5] = [0, 5]$

Abstract domain: interval of VPC indices
\[ vpc \in [0, 0] \]
\[ vpc \in [0, 0] \cup [3, 3] = [0, 3] \]
\[ vpc \in [0, 3] \cup [2, 5] = [0, 5] \]

\[ \text{op} = \text{prog}[vpc] \]

\[ \text{INC:} \{
    \ldots 
    vpc += 2;
\} \]

\[ \text{MOV:} \{
    \ldots 
    vpc += 3;
\} \]

Abstract domain: interval of VPC indices
\[
\text{vpc} \in [0,0]
\]
\[
\text{vpc} \in [0,0] \cup [3,3] = [0,3]
\]
\[
\text{vpc} \in [0,3] \cup [2,5] = [0,5]
\]

\[\text{op} = \text{prog}[\text{vpc}]\]

- \[\text{op} = 03 (\text{INC})\]
  - \[
  \text{INC}:\
  \{
  \quad \ldots \text{vpc} += 2; \\
  \}
  \]

- \[\text{op} = 52 (\text{MOV})\]
  - \[
  \text{MOV}:\
  \{
  \quad \ldots \text{vpc} += 3; \\
  \}
  \]

Abstract domain: interval of VPC indices
Virtualize+JIT

```
void P1()
{
}
```
P0 → Virtualize+JIT → 

```c
void P1()
{
    instrs = {
        "add...", "jump", ...
    };
}
```

→ P2 →
```c
void P1()
{
}

void P2()
{
    instrs=
        "{add..., "jump", ..."
    ;
    code=compile(instrs);
    goto *code;
}
```

```c
add   %cl,(%rax,%rax,1)
imul  %ecx,%ebx
ja    0x4242
```
1. Find the point where the code exists in cleartext
2. Print it
3. Statically analyze the cleartext code

```c
void P2()
{
    instrs=
    "add...", "jump", ...

    code=compile(instrs);
    goto *code;
}
```
Unpack+Print

1. Find the point where the code exists in cleartext
2. Print it
3. Statically analyze the cleartext code

```c
void P2(){
    instrs={
        "add...", "jump", ...
    };
    code=compile(instrs);
    goto *code;
}
```
Dynamic Obfuscation

- Keep the code in constant flux at runtime
- At no point should the entire code exist in cleartext
Dynamic Obfuscation

- Keep the code in constant flux at runtime
- At no point should the entire code exist in cleartext

```
void P1()
{
}
```

\( P_0 \) \rightarrow \text{tiger} \rightarrow \text{code block}
Cappaert, Preneel, et al. Towards Tamper Resistant Code Encryption P&E, ISPEC'08
Cappaert, Preneel, et al. Towards Tamper Resistant Code Encryption P&E, ISPEC'08
Madou, et al., Software protection through dynamic code mutation, WISA’05
Madou, et al., Software protection through dynamic code mutation, WISA’05
Madou, et al., Software protection through dynamic code mutation, WISA’05
Dynamic Analysis

main(argc, argv) {

}

Dynamic Analysis

Dynamic Analysis

Dynamic Analysis

Dynamic Analysis

Dynamic Analysis

INPUT

main(argc, argv) {

OUTPUT

• Huge traces
• Make traces even larger
• Trace may not cover all paths
• Prevent traces from being collected

void main(argc, argv) {
    VPC = 0;
    STACK = [];
    Virtual Program Array
    sub add call print
}
void main(argc, argv) {
    VPC = 0;
    STACK = [];
    
    Virtual Program Array
    
    sub   add   call   print
    
    Not input dependent!
}

void main(argc, argv) {
  VPC = 0;
  STACK = [];
  Virtual Program Array
  sub add call print
}

void main(argc, argv) {
    VPC =
    STACK =
    sub add call print =
    }

Anti-Taint Analysis
void main(argc, argv) {
    VPC = f(argv);
    STACK = g(argv);
    sub add call print = h(argv);
}

Make input dependent!
Anti-Taint Analysis

```c
void main(argc, argv){
    VPC = f(argv);
    STACK = g(argv);
    sub add call print = h(argv);
}
```

Make input dependent!
## Analysis Performance

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Program</th>
<th>Virtualization</th>
<th>Analysis Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Analysis</td>
<td>Fibonacci</td>
<td>Tigress</td>
<td>40s, 71MB</td>
</tr>
<tr>
<td>Bit-level taint</td>
<td>Huffman coding</td>
<td>VMProtect</td>
<td>449s, trace size 32M</td>
</tr>
<tr>
<td>Concolic analysis</td>
<td>14 line program</td>
<td>VMProtect</td>
<td>14,160s</td>
</tr>
</tbody>
</table>

Yadegari, Automatic Deobfuscation and Reverse Engineering of Obfuscated Code
Kinder, Towards Static Analysis of Virtualization-Obfuscated Binaries, WCRE’12
Time-Limited Protection

Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts
Time-Limited Protection

Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts
Obfuscation provides *time-limited protection*: an adversary will require greater-than-zero length of time to extract an asset from an obfuscated program.

Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts
Obfuscation provides *time-limited protection*: an adversary will require greater-than-zero length of time to extract an asset from an obfuscated program.

How can we get useful levels of protection from individual transformations that only provide time-limited protection?

Hohl, *Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts*
Deploying Obfuscation
Deploying Obfuscation
Deploying Obfuscation

Monitor adversarial communities
Deploying Obfuscation

Monitor adversarial communities

Be prepared with new technologies
Deploying Obfuscation

Monitor adversarial communities

Be prepared with new technologies

Give adversaries a diversity of targets

- Spatial diversity
- Temporal diversity
- Semantic diversity
Spatial Diversity

- Prevent collusion by giving each adversary a differently obfuscated program
Temporal Diversity

- Adversary sees a sequence of code variants over time
- Overwhelm his analytical abilities
- Small time window to execute an attack
- Known as “Planned Obsolescence”

London, Ending the Depression Through Planned Obsolescence, 1932
Temporal Diversity

- Adversary sees a sequence of code variants over time
- Overwhelm his analytical abilities
- Small time window to execute an attack
- Known as “Planned Obsolescence”

London, Ending the Depression Through Planned Obsolescence, 1932
Semantic Diversity

- Code variants are semantically incompatible
- Previously cracked code variants have no value
- Known as "Software Aging"

Semantic Diversity

- Code variants are semantically incompatible
- Previously cracked code variants have no value
- Known as “Software Aging”

Updatable Security

Trusted server

Untrusted clients

P

P
Updatable Security

Trusted server

Untrusted clients

Snapchat logo
Updatable Security

Trusted server

Untrusted clients

P
Continuous Replacement

Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC’12
Continuous Replacement

Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC’12
Continuous Replacement

\[ T_1 \ T_2 \ T_3 \]

- Function blocks
- Server code
- Diversity scheduler

Temporal Diversity

Function blocks

Client code

Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC’12
Continuous Replacement

\[ T_1, T_2, T_3 \]

Function blocks

Server code

Diversity scheduler

Function blocks

Client code

Temporal Diversity

Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC’12
Continuous Replacement

Collberg, et al., Distr. app. tamper detection via continuous softw. updates, ACSAC’12
Continuous Replacement

\[ T_1 \rightarrow T_2 \rightarrow T_3 \]

Server code

Client behaves?

Diversity scheduler

Client code
Continuous Replacement

$T_1 \ T_2 \ T_3$

Server code

Client behaves?

Diversity scheduler

Client code

HACK
Continuous Replacement

\( T_1 \ T_2 \ T_3 \)

Server code
Client behaves?

Diversity scheduler

Semantic Diversity

Client code

HACK 2
Continuous Replacement

Server code
Client behaves?
Diversity scheduler

Semantic Diversity

Client code

T_1 \ T_2 \ T_3
Continuous Replacement

$T_1 \ T_2 \ T_3$

Server code
Client behaves?

Diversity scheduler

RPC

Semantic Diversity

Client code

HACK
Continuous Replacement

$T_1 \ T_2 \ T_3$

Server code
Client behaves?

Diversity scheduler

Semantic Diversity

Client code

RPC
Continuous Replacement

T₁ T₂ T₃

Server code

Client behaves?

Diversity scheduler

RPC

Semantic Diversity

Client code
Our Story So Far...

1. Scenarios where obfuscation can be useful
2. Obfuscating transformations that give time-limited protection
3. Updatable security for longer-term protection

But, how do we know we’re doing anything good?
Evaluation
Evaluation in Academia

My transformation gets better security and performance than previous ones!

If we can’t do this, how can we make progress?
Evaluation in Industry

Professional red teams evaluate new transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>Broken in ‘09</td>
</tr>
<tr>
<td>$T_2$</td>
<td>Soon to be broken</td>
</tr>
<tr>
<td>$T_3$</td>
<td>Works for now</td>
</tr>
</tbody>
</table>

Experience from monitoring real world adversaries
Invent “stand-ins” for red team evaluation
Which metrics should we use?

* T₂ is better than T₁!
• Measure the time it takes for students to solve a task on the obfuscated code
• **Issues**: Inexperience, doesn’t scale, students get better over time

Ceccato et al., *The effectiveness of source code obfuscation: ..., ICPC’09*
Metric 2: SW Metrics

• Combine a few Software Complexity Metrics
• **Issues**: SCMs were not designed to measure code badness; 100s of SCMs - which ones should we use?

Anckaert, et al., Program Obfuscation: A Quantitative Approach
Metric 2: SW Metrics

<table>
<thead>
<tr>
<th>Complexity Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knot Count</td>
<td>Number of crossings of control flow arrows in a graph</td>
</tr>
<tr>
<td>Cyclomatic number</td>
<td>Number of decision points: #edges−#nodes+2*(#connected components)</td>
</tr>
</tbody>
</table>

Anckaert, et al., Program Obfuscation: A Quantitative Approach
Metric 2: SW Metrics

- Combine a few Software Complexity Metrics
- **Issues**: SCMs were not designed to measure code badness; 100s of SCMs - which ones should we use?

Anckaert, et al., Program Obfuscation: A Quantitative Approach
**Metric 3: Analysis Tools**

- Measure the runtime & precision of code analysis tool

int main(int argc, char* argv[]) {
    if (argv[1][0] == 97 &&
        argv[1][1] == 98 &&
        argv[1][2] == 99 &&
        argv[1][3] == 100 &&
        argv[1][4] == 101) {
        printf("win\n");
    } else {
        printf("lose\n");
    }
}

- Failure due to bugs, lack of performance tuning, or your transformation is good, ...
int main(int argc, char* argv[]) {
    if (argv[1][0] == 97 &&
        argv[1][1] == 98 &&
        argv[1][2] == 99 &&
        argv[1][3] == 100 &&
        argv[1][4] == 101) {
        printf("win\n");
    } else {
        printf("lose\n");
    }
}

Virtualize

Virtualize + Encode Program Array + Make Input Dependent

• Failure due to bugs, lack of performance tuning, or your transformation is good, ...
int main(int argc,
    char* argv[]) {
    if (argv[1][0] == 97 &&
        argv[1][1] == 98 &&
        argv[1][2] == 99 &&
        argv[1][3] == 100 &&
        argv[1][4] == 101) {
        printf("win\n");
    } else {
        printf("lose\n");
    }
}
Missing: Validation

1. Build model from the behavior of real hackers:
   - \textbf{Adversarial Model}
     - X is hard
     - Y is easy

2. Correlate with potential metrics:
   \begin{itemize}
     \item SCM\textsubscript{1}
     \item SCM\textsubscript{2}
     \item SCM\textsubscript{3}
   \end{itemize}
Adversarial Model Building
Adversarial Model Building

Code Analysis Tools

- S²E
- angr
- TRILON
- Hex-Rays
Adversarial Model Building

Challenges

P₀  P₁
P₂  P₃

Code Analysis Tools

S²E  angr
TRILION
Hex-Rays
Adversarial Model Building

Challenges

- $P_0$
- $P_1$
- $P_2$
- $P_3$

Code Analysis Tools

- S²E
- angr
- TRION
- Hex-Rays

VMware
Adversarial Model Building

Challenges

\[ P_0 \quad P_1 \]
\[ P_2 \quad P_3 \]

Code Analysis Tools

\[ S^2E \quad angr \]
\[ TRIL \quad \text{Klee} \]
\[ Hex-Rays \]

Model

- X is hard
- Y is easy
Generating Challenges

- Automatically generate many challenges
- Varying levels of complexity
Generating Challenges

- Automatically generate many challenges
- Varying levels of complexity
Challenges So Far...

- Easiest challenge broken by Google engineer in 8 hours.

http://tigress.cs.arizona.edu/challenges.html

Cash and/or book prizes!
Discussion
Meeting security criteria without meeting performance criteria is not a solution in a MATE scenario.
Meeting security criteria without meeting performance criteria is not a solution in a MATE scenario.

- Arbitrary levels of protection, at arbitrary levels of slowdown, is easy:
• Real programs are large, and analyses need to scale.
• Saying that an obfuscation falls against a particular analysis is meaningless without knowing the performance cost.

Meeting precision criteria without meeting performance criteria is not a solution for anti-MATE analyses.
Obfuscating transformations are primitives that provide time-limited protection. Updatable security can extend the protection they provide.

- All language-based obfuscations will break.
- Updatable security can increase the cost to the attacker.
To make progress in this field, the community must settle on rigorous evaluation procedures.

• Evaluation is a mess — we need to fix this.
• Help, anyone?
• Learn from public challenges.
### MATE Predictions?

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Security</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Language based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Crypto based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Updatable Security</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Which techniques will prevail?
- Will they coexist, but in different scenarios?
- Will we see combinations of techniques?
Questions?

collberg@gmail.com

Slides: tigress.cs.arizona.edu/eurocrypt.pdf