Software Code Protection Through Software Obfuscation

Presented by:

Sabu Emmanuel, PhD
School of Computer Engineering
Nanyang Technological University, Singapore
E-mail: asemmanuel@ntu.edu.sg

20, Mar, 2009 – IIT, Guwahati
What is Ahead

- Motivation
- Software obfuscation techniques
  - Obfuscation space
  - Design obfuscation
  - Layout obfuscation
  - Data flow obfuscation
  - Control flow obfuscation
  - Diablo & PLTO
  - IDA Pro
  - Demos
### Software Piracy

#### Study Highlights:
**Fifth Annual Global Software Piracy Study**
May 2008

**2007 Worldwide PC Software Piracy Figures**
- Global piracy rate: 38%
- Total packaged PC software losses: nearly $48 billion (USD)
- Changes from 2006: 3% rise in global piracy rate; losses increased 20%

<table>
<thead>
<tr>
<th>Countries with Highest Piracy Rates</th>
<th>Countries with Lowest Piracy Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armenia</td>
<td>United States</td>
</tr>
<tr>
<td>Bangladesh</td>
<td>Luxembourg</td>
</tr>
<tr>
<td>Azerbaijan</td>
<td>New Zealand</td>
</tr>
<tr>
<td>Moldova</td>
<td>Japan</td>
</tr>
<tr>
<td>Zimbabwe</td>
<td>Australia</td>
</tr>
<tr>
<td>Yemen</td>
<td>Denmark</td>
</tr>
<tr>
<td>Libya</td>
<td>Finland</td>
</tr>
<tr>
<td>Venezuela</td>
<td>Sweden</td>
</tr>
<tr>
<td>Vietnam</td>
<td>Switzerland</td>
</tr>
<tr>
<td>Iraq</td>
<td>United Kingdom</td>
</tr>
</tbody>
</table>

**Highlights**
- Of the 108 countries included in the study, PC software piracy dropped in sixty-seven countries and rose in only eight.
- Worldwide, for every two dollars’ worth of software purchased legitimately, one dollar’s worth was obtained illegally. In countries with 75% piracy or higher, for every one dollar spent on PC hardware, less than seven cents was spent on legitimate software.
- Russia led the way with a one-year PC software piracy drop of seven points to 73%, and a five-year drop of 14 points.
- Piracy rates dropped slightly in many low piracy countries where rates have been stagnant for several years, including the United States (-1%), United Kingdom (-1%), and Austria (-1%). Many other developed economies experienced a continuing gradual decline, including Australia, Belgium, Ireland, Japan, Singapore, South Africa, Sweden, and Taiwan.
- Because the worldwide PC market grew fastest in high piracy emerging markets, the worldwide PC software rate increased by three percentage points to 38%, and worldwide losses rose by $8 billion to nearly $48 billion worldwide. PC shipments in Brazil, Russia, India, & China (BRIC) grew 26% last year, compared to 13% in North America, Western Europe, and Japan. The combined BRIC countries are now as large a PC market as the United States.
## Software Piracy

### 2007 Global Software Piracy Study

Released May 2008

<table>
<thead>
<tr>
<th>Region</th>
<th>Piracy Rates</th>
<th>Losses ($M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIA-PACIFIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brazil</td>
<td></td>
<td></td>
</tr>
<tr>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hong Kong</td>
<td></td>
<td></td>
</tr>
<tr>
<td>India</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indonesia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Korea</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Zealand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nigeria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Philippines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singapore</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thailand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vietnam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Asia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL ASIA-PACIFIC</td>
<td>$2,500</td>
<td>$2,500</td>
</tr>
<tr>
<td>CENTRAL &amp; EASTERN EUROPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belgium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bulgaria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Czech Republic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denmark</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estonia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>France</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hungary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latvia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lithuania</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netherlands</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Poland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portugal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Romania</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Russia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slovakia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slovenia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Europe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL CENTRAL &amp; EASTERN EUROPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LATIN AMERICA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argentina</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bolivia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brazil</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colombia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Costa Rica</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominican Republic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ecuador</td>
<td></td>
<td></td>
</tr>
<tr>
<td>El Salvador</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guatemala</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Honduras</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nicaragua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panama</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paraguay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peru</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uruguay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Venezuela</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Latin America</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL LATIN AMERICA</td>
<td>$5,000</td>
<td>$5,000</td>
</tr>
</tbody>
</table>

3/21/2009
Software Protection

Reverse engineering
  Software piracy
  Malicious modifications
  Discovering vulnerabilities

Company A
  Algorithm G
  $O(n)$

Company B
  Algorithm G'
  $O(n \log(n))$
Software Protection

Encryption Vs. Obfuscation

- Execution time overhead is higher for encryption
- Protection level is higher for encryption
- Decryption need to be applied before executing the encrypted program
- Encryption is high cost approach
- Run-time decryption is more costly

Encryption ➔ Protection of software; Obfuscation ➔ Reverse engineering too expensive
Software Protection

Obfuscation

• To introduce confusion on the understanding of data in program
• To confuse the cracker on the execution path of program
• To increase the complexity of design of automatic reverse engineering tools
• To prevent attackers from dynamic tracing
Compilation & Reverse Engineering Process Flow

- Source code
  - parsing
  - Syntax tree
  - Intermediate code gen.
  - Control flow graph
  - Final code gen.
  - Assembly code
  - assembly
  - Machine code
  - disassembly
  - decompilation
  - Reverse engineering
Obfuscation Space

- Source code level
- Assembly language level
- Byte code level (java)
- Binary level
- Compile time
- Link time
Reverse Engineering

Reverse Engineering: is the process of recovering higher-level structure and semantics from a machine code program.

- Disassembly: machine code → assembly language code
- Decompilation: assembly language code → higher level structure and semantics

- Disassembly
  - Static
    - Linear sweep
    - Recursive traversal
  - Dynamic
Static Disassembly - Linear Sweep

```plaintext
global strt_Addr, end_Addr;
proc DisasmLinear(addr)
    begin
        while (strt_Addr < addr < end_Addr) do
            I := decode instruction at address addr; addr += length(I);
        od
    end

proc main()
    begin
        strt_Addr := address of the first executable byte;
        endAddr := strt_Addr + text section size;
        DisasmLinear(ep);
    end
```

GNU Utility - Objdump
Static Disassembly - Recursive Traversal

global 

proc DisasmRec(addr)
  begin
    while (strt_Addr \leq addr < end_Addr) do
      if (addr has been visited already) return;
      \( I := \) decode instruction at address \( addr \); mark \( addr \) as visited;
      if (\( I \) is a branch or function call)
        for each possible target \( t \) of \( I \) do
          DisasmRec(\( t \));
        od
      end
      return;
    od
    else addr += length(\( I \));
  end

proc main()
  begin
    strt_Addr := program entry point;
    end_Addr := strt_Addr + text section size;
    DisasmRec(strt_Addr);
  end
Evaluation Metric

- Obfuscations are evaluated with respect to:
  - **Potency** - To what degree is a human reader confused
  - **Resilience** - How well are automatic deobfuscation attacks resisted
  - **Cost** - How much time/space overhead is added
  - **Stealth** - How well does obfuscated code blend in with the original code
Evaluation Metric

- To be a potent obfuscation:
  - Increase overall program size, introduce new classes and methods
  - Introduce new predicates, increase the nesting level of conditional and looping constructs
  - Increase the number of method arguments and inter-class instance variable dependencies
  - Increase the height of inheritance tree
  - Increase long-range variable dependencies
Evaluation Metric

- Resiliency of obfuscation:
Obfuscation Classes

- Design obfuscation
- Layout obfuscation
- Data flow obfuscation
- Control flow obfuscation
Design Obfuscation

- Design obfuscation: which will obscure the design intent of object-oriented software.
  - Class merging
  - Class splitting
  - Type hiding
- Class merging obfuscation transforms a program into another one by merging two or more classes in the program into a single class.
- Class splitting obfuscation splits a single class in a program into several classes.
- Type hiding obfuscation uses Java interfaces to obscure the types of objects manipulated by the program.

M. Sosonkin et al.
Design Obfuscation – Class Merging

Original classes:

```java
import java.lang.*;

public class A {
    private int i;
    public A() {
        i = 5;
    }
    public boolean m() {
        return i < 0;
    }
}

class B extends A {
    public B() {
        super();
        i = 10;
    }
    public boolean m() {
        return i < 0;
    }
}

class C {
    void m() {
        A a;
        if (...) {
            a = new A();
        } else {
            a = new B();
        }
        a.m();
    }
}
```

Obfuscated classes:

```java
import java.lang.*;

public class AB {
    private int i;
    private boolean isA;
    public AB() {
        i = 5;
        isA = true;
    }
    public AB(int i) {
        this();
        i = 10;
        isA = false;
    }
    public boolean m() {
        if (isA) {
            return i < 0;
        } else {
            return i < 10;
        }
    }
}

class C {
    void m() {
        AB a;
        if (...) {
            a = new AB();
        } else {
            a = new AB(38);
        }
        a.m();
    }
}
```

M. Sosonkin et al.
Design Obfuscation – Class Splitting

Original classes:

```java
class C {
    private int i;
    private double d;
    protected Object o;
    public C() {
        i = 5;
        d = 1.0;
        o = new Object();
    }
    public C(int iarg, double darg) {
        i = iarg;
        d = darg;
        o = new Object();
    }
    public boolean m1() {
        return i < 0;
    }
    public void m2() {
        d = 3.0;
        m3(3);
    }
    protected void m3(int iarg) {
        i = iarg;
        m4(new Object());
    }
    public void m4(Object obj) {
        o = obj.getClass();
    }
}
class D {
    void n() {
        C c = new C();
        if (c.m1) {...}
        c.m2;
        c.m4;
    }
}
```

Obfuscated classes:

```java
class C1 {
    private int i;
    private double d;
    public C1() {
        i = 5;
        d = 1.0;
    }
    public C1(int iarg, double darg) {
        i = iarg;
        d = darg;
    }
    public boolean m1() {
        return i < 0;
    }
    protected void m3(int iarg) {
        i = iarg;
        m4(new Object());
    }
    public void m4(Object obj) {
        o = obj.getClass();
    }
}
class C2 extends C1 {
    protected Object c;
    public C2() {
        super();
        o = new Object();
    }
    public C1(int iarg, double darg) {
        super(iarg, darg);
        o = new Object();
    }
    public void m2() {
        d = 3.0;
        m3(3);
    }
    public void m4(Object obj) {
        o = obj;
    }
}
class D {
    void n() {
        C2 c = new C2();
        if (c.m1) {...}
        c.m2;
        c.m4;
    }
}
```

M.Sosonkin et al.
Layout Obfuscation

- Source code formatting
- Comments removal
- Scrambling identifiers
  - These obfuscations does not affect the execution time and space overheads
Data Flow Obfuscation

- Change data encoding
- Promote scalar to object
- Split variable - one variable is expressed in two or more variables – one 8-bit variable obtained from two 4-bit variables
- Merge variable - two or more variables are expressed in one variable – two 4-bit variables merged to obtain an 8-bit variable
- Flatten array - uses a lower-dimensional array variable for a higher-dimensional array
- Fold array - expresses a lower-dimensional array variable in a higher-dimensional array
- Change variable lifetime - converts a global variable to a local variable.
- Convert static data to procedure – This procedure would produce the static data when called
Change Data Encoding

Int l = 1;
while (l < 10)
{
    .....l++;
}

Int l = 5;
while (l < 23)
{
    .....l+=2;
}

I = 2*l+3
Promote Scalar to Object

int k = 0;
While (k<10)
{
    k++;
    .....;
}

Int k = new Int(0);
While (k.value<10)
{
    k.value++;
    .....;
}

Promote scalar to object
Array Restructuring

A:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
A_0 & A_1 & A_2 & A_3 & A_4 & A_5 & A_6 & A_7 & A_8 & A_9 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

B:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots & 19 \\
B_0 & B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_8 & B_9 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots & 19 \\
\end{array}
\]

C:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots & 19 \\
C_0 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & \cdots & C_{19} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots & 19 \\
\end{array}
\]

D:

\[
\begin{array}{cccccccc}
0 & 1 & 2 \\
D_0 & D_1 & D_2 & D_3 & D_4 & D_5 & D_6 & D_7 & D_8 & D_9 \\
0 & 1 & 2 \\
\end{array}
\]

E:

\[
\begin{array}{ccc}
0 & E_{0,0} & E_{0,1} & E_{0,2} \\
1 & E_{1,0} & E_{1,1} & E_{1,2} \\
2 & E_{2,0} & E_{2,1} & E_{2,2} \\
\end{array}
\]
Control Flow Obfuscation

- **Opaque Variables**: A variable V is opaque at a point p in a program, if V has a property q at p, which is known at obfuscation time.

- **Opaque Predicates**: A predicate P is opaque at p if its outcome, either True or False is known at obfuscation time.

```c
{ int v, a=2; b=8;
  V = a + b; /* v is 10 here. */
  if (b>9)  → False
  if (random(1,7) < 8) → True
}
```
Control Flow Obfuscation

Opaque Predicates (cntd.):
- Insert dead or irrelevant code
- Extending loop conditions
  - While(i<10){…} => While (i< 10) && (3>2) && (1<2){…}
- All these use opaque predicates.

- Dynamic opaque predicates - predicates constant over a single program run but varied over different program runs.
Control Flow Obfuscation

- Computation transformations
  - Insert dead or irrelevant code
  - Extend loop conditions - add opaque predicate in a loop to change control flow graph but to keep the semantics of the program
  - Include a pseudo-code interpreter - choose a section of code in the unobfuscated program, then create a virtual process not existing in the host language and add a pseudo-code interpreter for such a virtual process into the obfuscated program
  - Remove library calls
  - Add redundant operands
  - Parallelize code
Control Flow Obfuscation

- Aggregation transforms
  - Inlining of procedures
  - Outlining of procedures
  - Interleave procedures
  - Clone procedures

- Ordering transformations
  - Changes the locality of terms within expressions, statements with basic blocks, basic blocks within procedures, procedures within classes and classes within files.

- Signal flow approaches
- Self-modifying codes
Control Flow Flatten


Control Flow Flatten

- *If-then-goto* constructs + target address of *goto* are determined dynamically
- In C the *goto* statement is implemented using *switch* statement
- Control flow is made data dependent, which makes control flow analysis and data flow analysis complex

Chenxi Wang et al.
int a, b;
a = 1; b = 2;
while (a < 10)
{
    b = a + b;
    if (b > 10)
        b--;
    a++;
}
use (b);

Chenxi Wang et al.
Control Flow Flatten

Flattened control flow

Chenxi Wang et al.
Control Flow Flatten

Chenxi Wang et al.
Control Flow Flatten

Control flow flatten with

- Global arrays for dispatch variables and
- Dummy basic blocks and pointers

Udupa et al.
Control Flow Flatten

A = Random(1)
swvar{A} = 2
swvar{A+1} = 1
...........
Call(a,b)

A = Random(2)
swvar{A} = 2
swvar{A+1} = 1
...........
Call(a,b)

Switch(swvar)

goto

L1:
...........
swvar{A+1};

L2:
...........
If ...... swvar{A+8};
else
swvar{A+1};

L3:
...........
If ...... swvar{A+2};
else
swvar{A+8};

L4:
...........
swvar{A+5};

L5:
...........
swvar{A+1};

L6:
...............

L9:
Dummy
If ...... swvar{A+9};
else
swvar{A+1};

Udupa et al.

A = Random(1)
sWvar{A} = 2
sWvar{A+1} = 1
...........
Call(a,b)

A = Random(2)
sWvar{A} = 2
sWvar{A+1} = 1
...........
Call(a,b)
Signal-based Obfuscation

Signal-based Obfuscation

- Signal-based obfuscation
  - Signal, or software interrupt, is a message mechanism between different processes.
  - Three types of signal
    - SIGINT 2 keyboard interrupt (such as "break" is pressed)
    - SIGILL 4 illegal instruction
    - SIGSEGV 11 illegal memory usage
  - List all signals with command “kill -l”, list usage with command “man 7 signal”

Igor V. Popov et al.
Signal-based Obfuscation

- Signal-based obfuscation

signal install instructions → ...

Code-before

Trap instruction

Code-after

Install user’s signal handler with signal() function

Kernel trap handler

User’s signal handler

Kernel restore function

Igor V. Popov et al.
**Signal-based Obfuscation**

1. User’s signal handler installation instruction is placed somewhere in _start before the call to __libc_start_main
2. User’s signal handler is installed with signal() function

1. Set flag to inform our signal handler that this trap is raised by us, rather than system
2. Save machine state
3. Reserve stack to save trap instruction address

1. Overwrite the kernel restore function’s return address with our restore function’s address
2. Save the address of trap instruction in stack to hash the target address in restore function

1. Restore machine state when trap happened
2. Hash the target address with trap instruction address, push the address into stack and transfer control to target Subfun address by ‘return’ instruction

---

**Igor V. Popov et al.**

3/21/2009
Signal-based Obfuscation

- Signal-based obfuscation

```
file.c
GCC
file.O3
PLTO

gcc file.c -o file.O3

file.obf
PLTO

file.prof

file.counts

Relocatable file
Profiled file
Binary executable
Counter file
containing information on how many times of BBLs and edges have been executed
```
Self-Modifying Code Techniques

- Y. Kanzaki, A. Monden, M. Nakamura, and K. ichi Matsumoto; Exploiting self modification mechanism for program protection.

- Matias Madou, Bertrand Anckaert, Patrick Moseley, Saumya Debray, Bjorn De Sutter, and Koen De Bosschere; Software Protection Through Dynamic Code Mutation.
Exploiting Self-Modification Mechanism for Program Protection

Yichiro Kanzaki et al.

RR $i$ – Restores the dummy instruction ‘Dummy $i$’ to target instruction.

HR $i$ – Hides the target instruction (to dummy instruction).
Software Protection Through Dynamic Code Mutation

Matias Madou et.al.
Uses Edit Engine + Edit Script
One pass protection
Cluster protection
References


References


Obfuscation Prototypes and Demo

- Diablo – link time binary rewriting system
- PLTO – link time binary rewriting system
- IDA Pro – Professional disassembler tool
- Demo
Thank You!!