
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%**

Countries with Highest Piracy Rates

	2007	2006
Armenia	93%	95%
Bangladesh	92%	92%
Azerbaijan	92%	94%
Moldova	92%	94%
Zimbabwe	91%	91%
Sri Lanka	90%	90%
Yemen	89%	—
Libya	88%	—
Venezuela	87%	86%
Vietnam	85%	88%
Iraq	85%	—

Countries with Lowest Piracy Rates

	2007	2006
United States	20%	21%
Luxembourg	21%	—
New Zealand	22%	22%
Japan	23%	25%
Austria	25%	26%
Belgium	25%	27%
Denmark	25%	25%
Finland	25%	27%
Sweden	25%	26%
Switzerland	25%	26%
United Kingdom	26%	27%

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



	Piracy Rates					Losses (\$M)				
	2007	2006	2005	2004	2003	2007	2006	2005	2004	2003
ASIA-PACIFIC										
Australia	28%	29%	31%	32%	31%	\$492	\$515	\$361	\$409	\$341
Bangladesh	92%	92%	-	-	-	\$92	\$90	-	-	-
China	82%	82%	86%	90%	92%	\$6,664	\$5,429	\$3,887	\$3,565	\$3,823
Hong Kong	51%	53%	54%	52%	52%	\$224	\$180	\$112	\$116	\$102
India	69%	71%	72%	74%	73%	\$2,025	\$1,275	\$966	\$519	\$367
Indonesia	84%	85%	87%	87%	88%	\$411	\$350	\$280	\$183	\$158
Japan	23%	25%	28%	28%	29%	\$1,791	\$1,781	\$1,621	\$1,787	\$1,633
Malaysia	59%	60%	61%	61%	62%	\$311	\$288	\$148	\$134	\$129
New Zealand	23%	22%	23%	23%	23%	\$55	\$49	\$30	\$25	\$21
Pakistan	84%	86%	90%	82%	83%	\$125	\$143	\$48	\$26	\$16
Philippines	63%	71%	71%	71%	72%	\$147	\$119	\$75	\$69	\$55
Singapore	37%	39%	40%	42%	43%	\$159	\$125	\$86	\$96	\$90
South Korea	43%	45%	46%	46%	48%	\$549	\$440	\$400	\$506	\$462
Sri Lanka	90%	90%	-	-	-	\$33	\$26	-	-	-
Taiwan	40%	41%	43%	43%	43%	\$215	\$182	\$111	\$161	\$139
Thailand	78%	80%	80%	79%	80%	\$468	\$421	\$259	\$183	\$141
Vietnam	85%	88%	92%	92%	92%	\$200	\$206	\$38	\$55	\$41
Other AP	91%	86%	82%	76%	76%	\$69	\$148	\$29	\$63	\$37
TOTAL AP	59%	55%	54%	53%	53%	\$14,090	\$11,718	\$8,050	\$7,897	\$7,555
CENTRAL & EASTERN EUROPE										
Albania	78%	77%	76%	77%	-	\$11	\$11	\$9	\$7	-
Armenia	93%	95%	95%	-	-	\$8	\$8	\$7	-	-
Azerbaijan	92%	94%	94%	-	-	\$50	\$51	\$40	-	-
Bosnia	68%	68%	67%	70%	-	\$13	\$14	\$12	\$12	-
Bulgaria	68%	69%	71%	71%	71%	\$68	\$50	\$41	\$33	\$26
Croatia	54%	55%	57%	58%	59%	\$68	\$62	\$51	\$50	\$45
Czech Republic	39%	39%	40%	41%	40%	\$161	\$147	\$121	\$132	\$106
Estonia	51%	52%	54%	55%	54%	\$20	\$16	\$18	\$17	\$14
Hungary	42%	42%	42%	44%	42%	\$125	\$111	\$106	\$126	\$96
Kazakhstan	79%	81%	85%	85%	85%	\$110	\$85	\$69	\$57	-
Latvia	56%	56%	57%	58%	57%	\$29	\$25	\$20	\$19	\$16
Lithuania	56%	57%	57%	58%	-	\$37	\$31	\$25	\$21	\$17
Poland	68%	69%	70%	72%	-	\$11	\$10	\$9	\$8	-
Moldova	52%	54%	56%	-	-	\$43	\$56	\$44	-	-
Montenegro	83%	82%	83%	83%	-	\$7	\$6	\$9	\$8	-
Romania	57%	57%	58%	59%	-	\$580	\$484	\$388	\$379	\$301
Russia	68%	69%	72%	74%	73%	\$151	\$114	\$111	\$62	\$49
Russia	73%	80%	83%	87%	87%	\$4,123	\$2,197	\$1,625	\$1,362	\$1,104
Serbia	76%	78%	80%	80%	-	\$72	\$72	\$65	\$65	-
Slovakia	45%	45%	47%	48%	50%	\$54	\$47	\$44	\$48	\$40
Slovenia	48%	48%	50%	51%	52%	\$39	\$33	\$37	\$37	\$32
Ukraine	83%	84%	85%	91%	91%	\$403	\$337	\$239	\$107	\$92
Other CEE	88%	90%	92%	88%	83%	\$173	\$166	\$145	\$112	\$173
TOTAL CEE	68%	69%	69%	71%	71%	\$6,371	\$4,124	\$3,262	\$2,662	\$2,111
LATIN AMERICA										
Argentina	74%	75%	77%	75%	71%	\$370	\$303	\$182	\$108	\$69
Bolivia	82%	82%	83%	80%	78%	\$19	\$15	\$10	\$9	\$11
Brazil	59%	60%	64%	64%	63%	\$1,617	\$1,148	\$766	\$659	\$519
Chile	66%	66%	66%	64%	63%	\$167	\$167	\$109	\$67	\$68
Colombia	58%	59%	57%	55%	53%	\$127	\$111	\$90	\$81	\$61
Costa Rica	61%	64%	66%	67%	68%	\$22	\$27	\$19	\$16	\$17
Dominican Republic	73%	73%	73%	73%	73%	\$39	\$39	\$24	\$24	\$24
Ecuador	65%	67%	69%	70%	69%	\$39	\$30	\$17	\$13	\$11
El Salvador	81%	81%	81%	78%	75%	\$28	\$28	\$18	\$14	\$14
Guatemala	80%	81%	81%	78%	77%	\$41	\$26	\$14	\$10	\$9
Honduras	74%	75%	75%	75%	73%	\$8	\$7	\$4	\$3	\$3
Mexico	61%	63%	65%	65%	63%	\$836	\$748	\$465	\$407	\$369
Nicaragua	80%	80%	80%	80%	79%	\$4	\$4	\$2	\$1	\$1
Panama	74%	74%	71%	70%	69%	\$22	\$18	\$8	\$4	\$4
Paraguay	83%	83%	83%	83%	83%	\$13	\$10	\$10	\$11	\$9
Peru	71%	71%	73%	73%	68%	\$75	\$59	\$40	\$39	\$31
Uruguay	69%	70%	70%	71%	67%	\$23	\$16	\$9	\$12	\$10
Venezuela	87%	83%	83%	73%	73%	\$464	\$467	\$73	\$71	\$71
Other LA	83%	83%	82%	79%	81%	\$195	\$96	\$32	\$6	\$7
TOTAL LA	65%	66%	68%	66%	63%	\$4,123	\$3,125	\$2,026	\$1,546	\$1,263

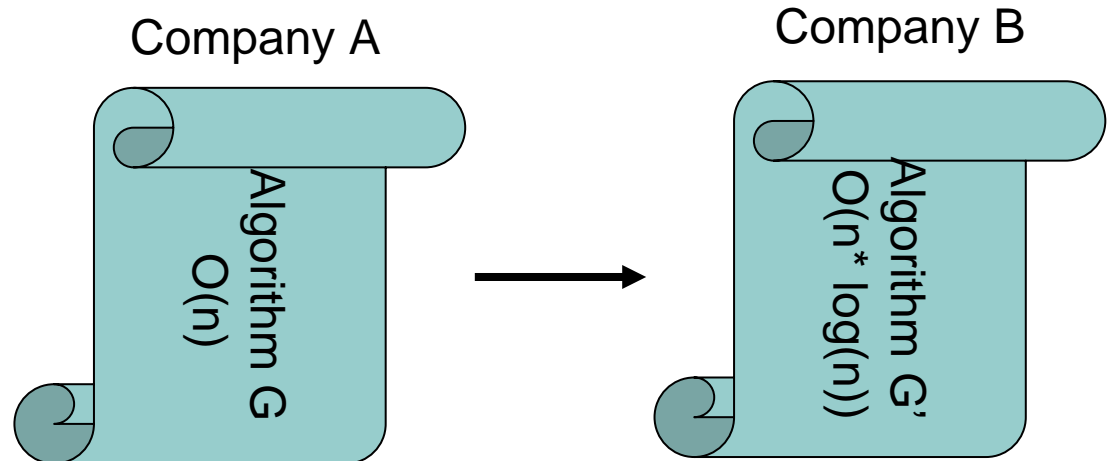
Software Protection

Reverse engineering

Software piracy

Malicious modifications

Discovering vulnerabilities



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

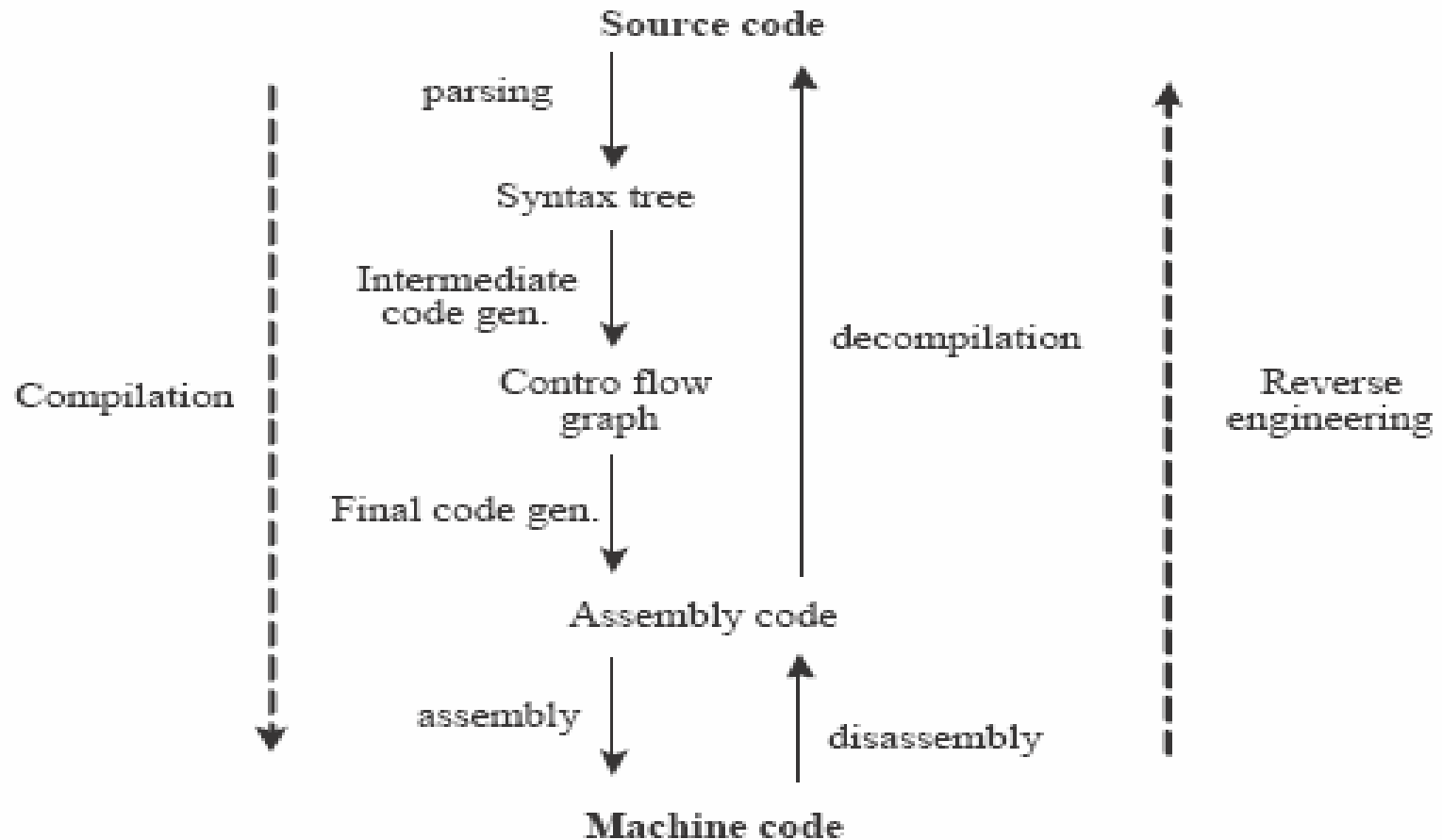
Encryption → Protection of software; Obfuscation → Reverse engineering too expensive

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



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

```
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
```

GNU Utility - Objdump

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

Static Disassembly - Recursive Traversal

```
global strt_Addr, end_Addr;
```

```
proc DisasmRec(addr)
```

```
  begin
```

```
    while (strt_Addr ≤ 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
```

```
        return;
```

```
      else addr += length(I);
```

```
      od
```

```
end
```

IDA Pro

```
proc main()
```

```
  begin
```

```
    strt_Addr := program entry point;
```

```
    end_Addr := strt_Addr + text section size;
```

```
    DisasmRec(strt_Addr);
```

```
  end
```

Evaluation Metric

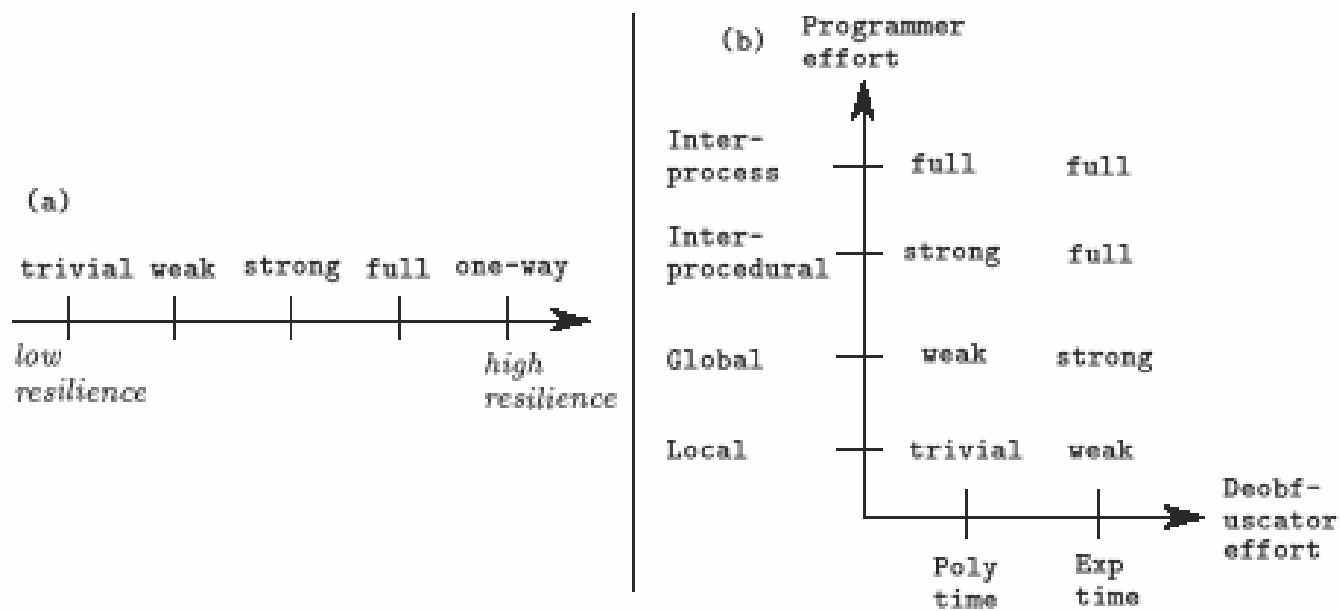
- Obfuscations are evaluated with respect to:
 - **Potency** - To what degree is a human reader confused
 - **Resilience** - How well are automatic deobfuscation attacks resisted
 - **Cost** - How much time/space overhead is added
 - **Stealth** - How well does obfuscated code blend in with the original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

```
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 < 10;
    }
}

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

Obfuscation



Obfuscated classes:

```
class AB {
    private int i;
    private boolean isA;

    public AB() {
        i = 5;
        isA = true;
    }

    public AB(int j) {
        this();
        i = 10;
        isA = false;
    }

    public boolean m() {
        if (isA) {
            return i < 0;
        } else {
            return i < 10;
        }
    }
}

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

M.Sosonkin et al.

Design Obfuscation – Class Splitting

Original classes:

```
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;
    }
}

class D {
    void n() {
        C c = new C();
        if (c.m1) {...}
        c.m2;
        c.m4;
    }
}
```

Obfuscation

Obfuscated classes:

```
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 o;

    public C2() {
        super();
        o = new Object();
    }

    public C(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;  
}
```

$l = 2 * l + 3$

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

	0	1	2	3	4	5	6	7	8	9
A:	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
B:	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	...	19
C:	C ₀	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	...	C ₁₉

	0	1	2	3	4	5	6	7	8	9
D:	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	D ₈	D ₉

	0	1	2	
E:	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}



	0	1	2	3	4
A1:	A ₀	A ₂	A ₄	A ₆	A ₈

	0	1	2	3	4
A2:	A ₁	A ₃	A ₅	A ₇	A ₉

	0	1	2	3	4	5	6	7	...	29
BC:	B ₀	C ₀	C ₁	B ₁	C ₂	C ₃	B ₂	C ₄	...	C ₁₉

	0	1	2	3	4	
D1:	0	D ₀	D ₁	D ₂	D ₃	D ₄
	1	D ₅	D ₆	D ₇	D ₈	D ₉

	0	1	2	3	4	5	6	7	8
E1:	E _{0,0}	E _{0,1}	E _{0,2}	E _{1,0}	E _{1,1}	E _{1,2}	E _{2,0}	E _{2,1}	E _{2,2}

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.

```
{ 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

- C. Wang, J. Hill, J. Knight, and J. Davidson, “Software tamper resistance: Obstructing static analysis of programs,” University of Virginia, Charlottesville, VA, 2000.
- SK Udupa, SK Debray, and M. Madou, “Deobfuscation: Reverse Engineering Obfuscated Code,” in Reverse Engineering, 12th Working Conference on, 2005, pp. 45–54.

Control Flow Flatten

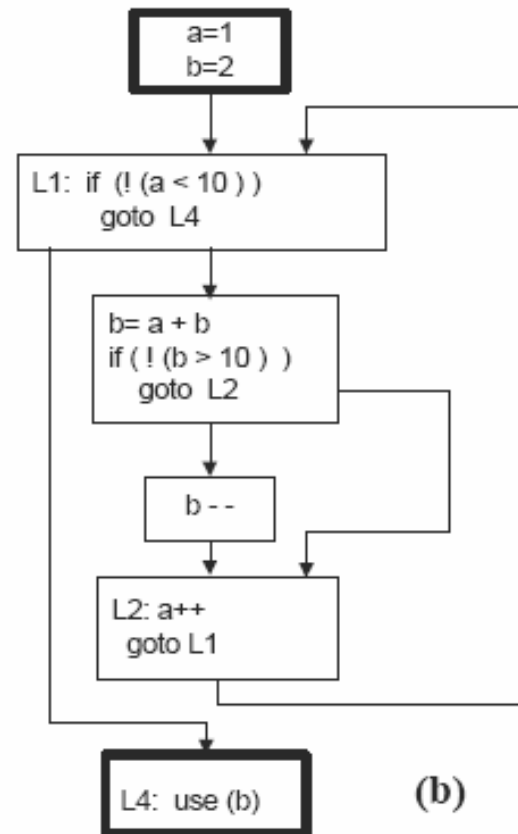
- ***If-then-goto*** constructs + target address of ***gotos*** are determined dynamically
- In C the ***goto*** statement is implemented using ***switch*** statement
- Control flow is made data dependent, which makes control flow analysis and data flow analysis complex

Chenxi Wang et al.

Control Flow Flatten

```
int a,b;  
a=1; b=2;  
while(a<10)  
{  
    b=a+b;  
    if(b>10)  
        b--;  
    a++;  
}  
use(b);
```

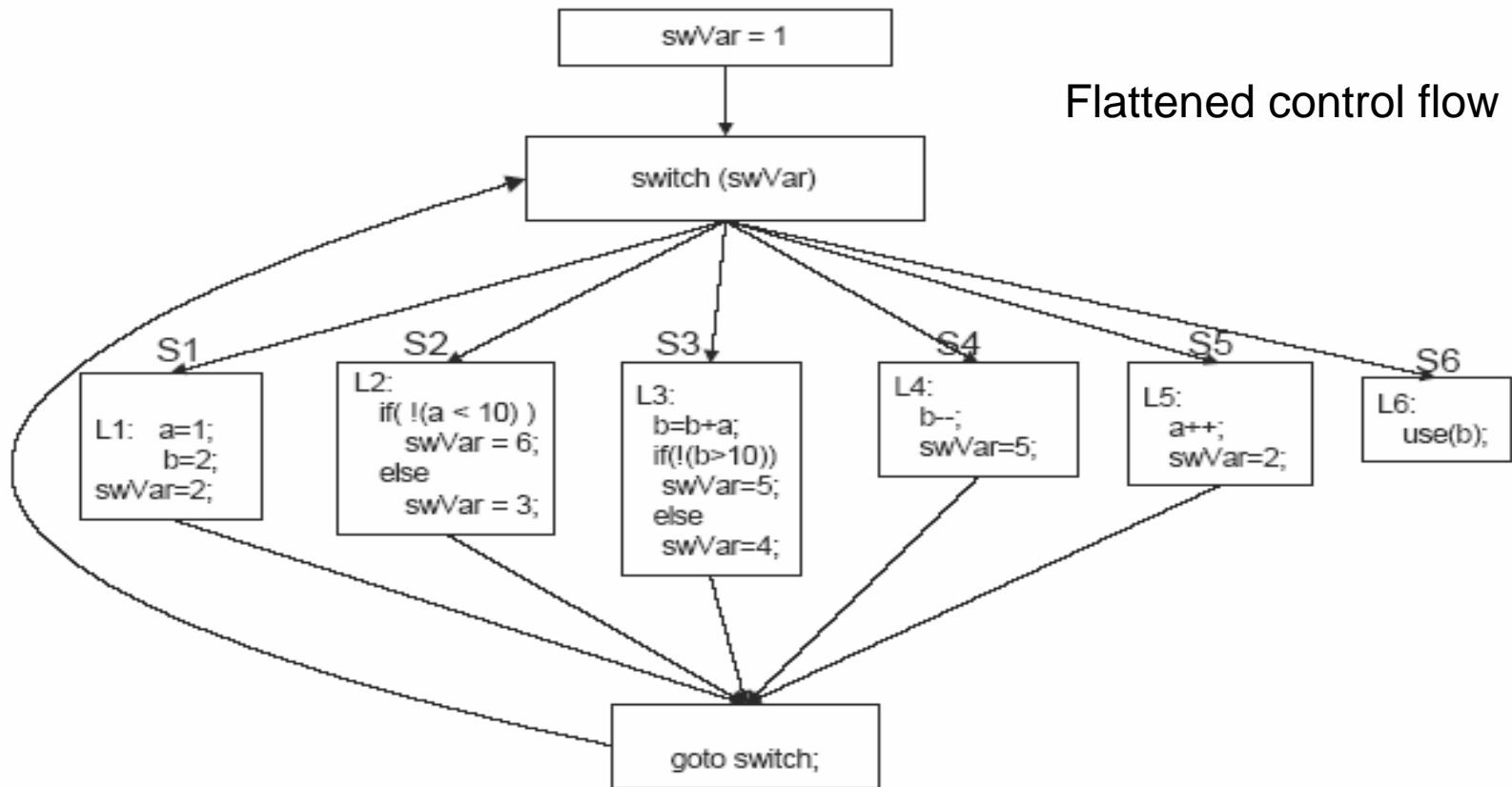
(a)



(b)

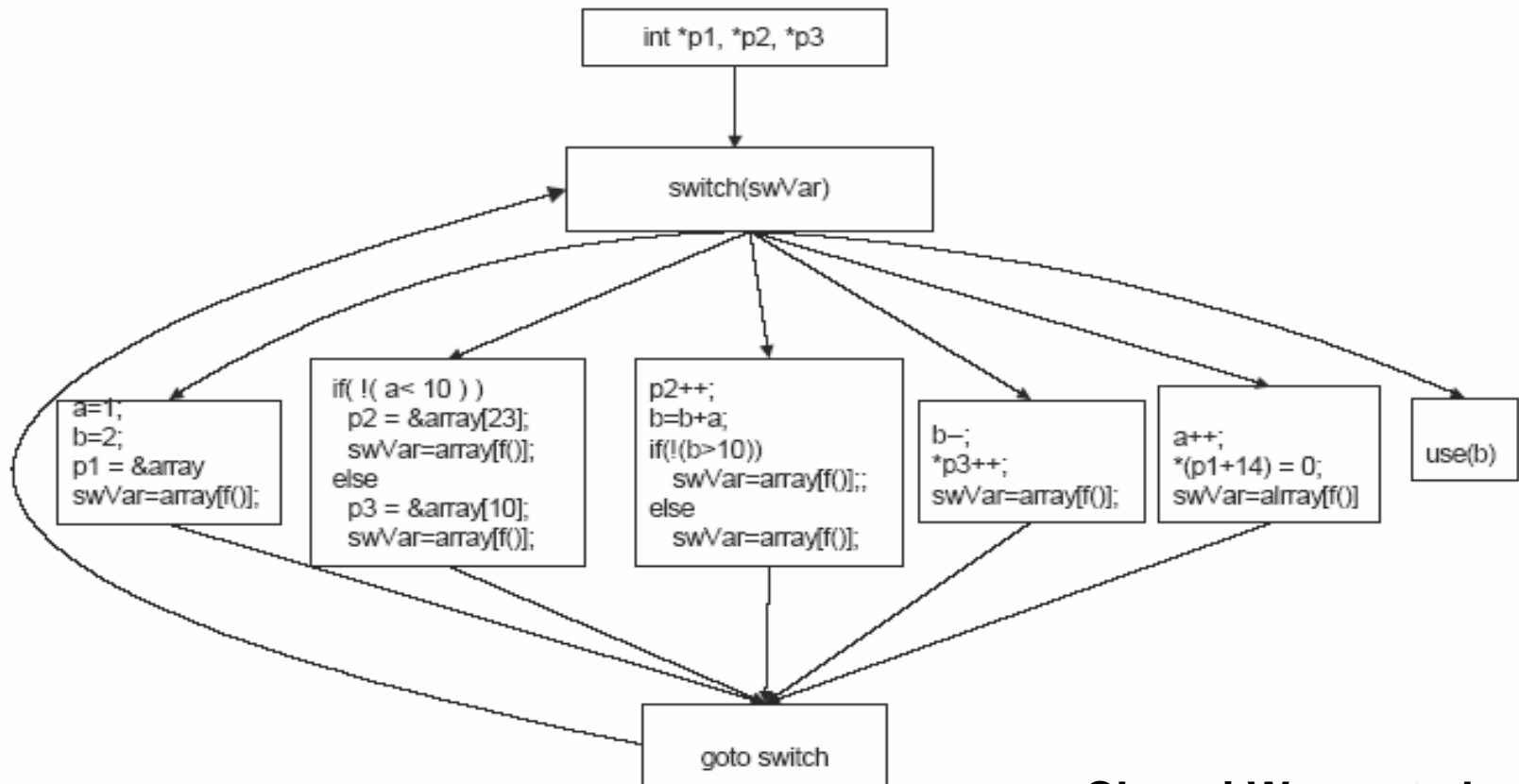
Chenxi Wang et al.

Control Flow Flatten



Chenxi Wang et al.

Control Flow Flatten



Chenxi Wang et al.

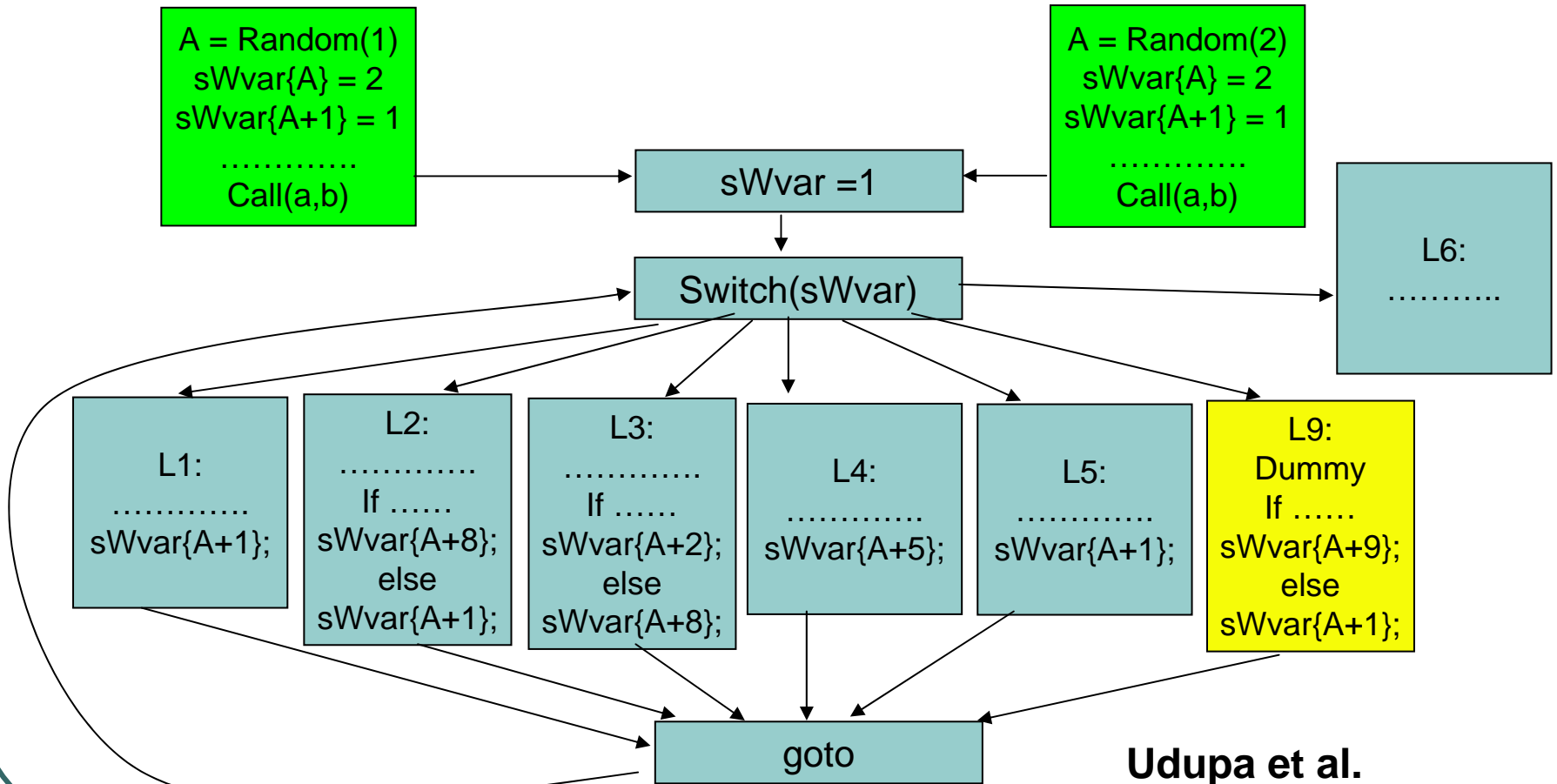
Control Flow Flatten

Control flow flatten with

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

Udupa et al.

Control Flow Flatten



Udupa et al.

Signal-based Obfuscation

- I.V. Popov, S.K. Debray, and G.R. Andrews. Binary obfuscation using signals. In Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium table of contents. USENIX Association Berkeley, CA, USA, 2007.

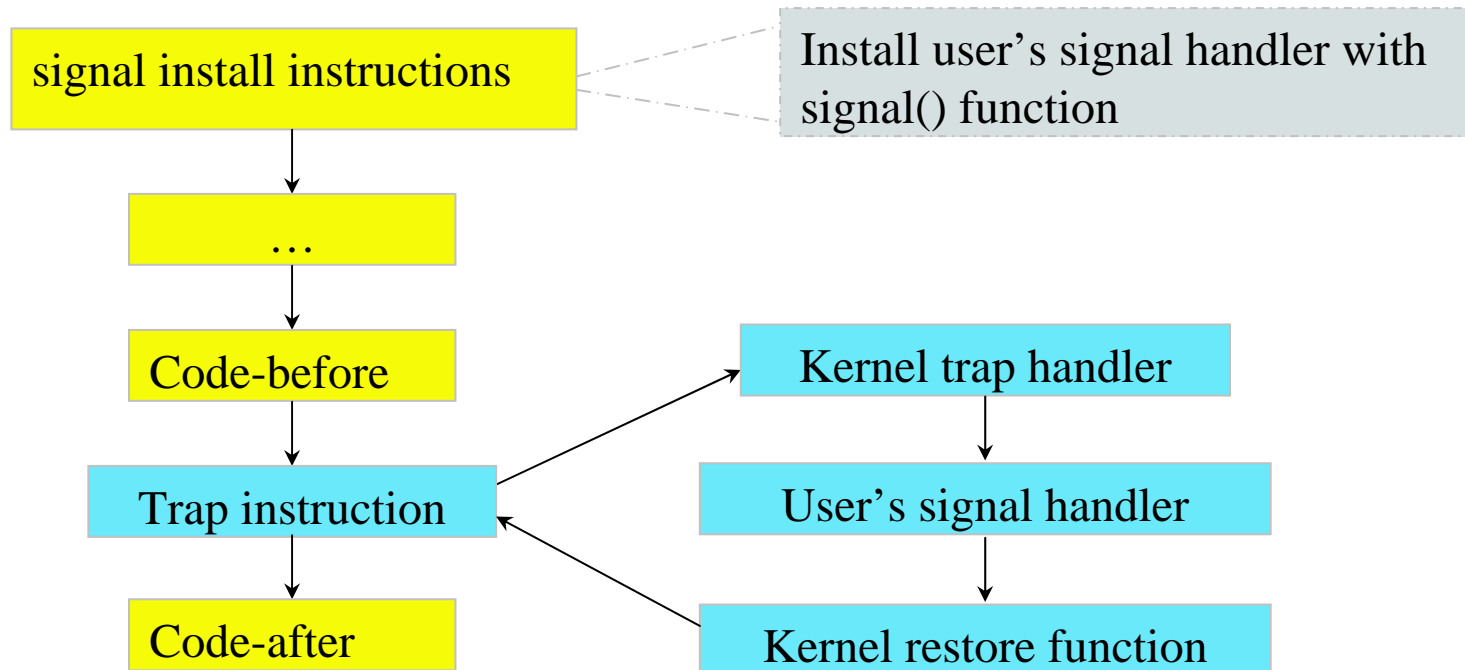
Signal-based Obfuscation

- 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.

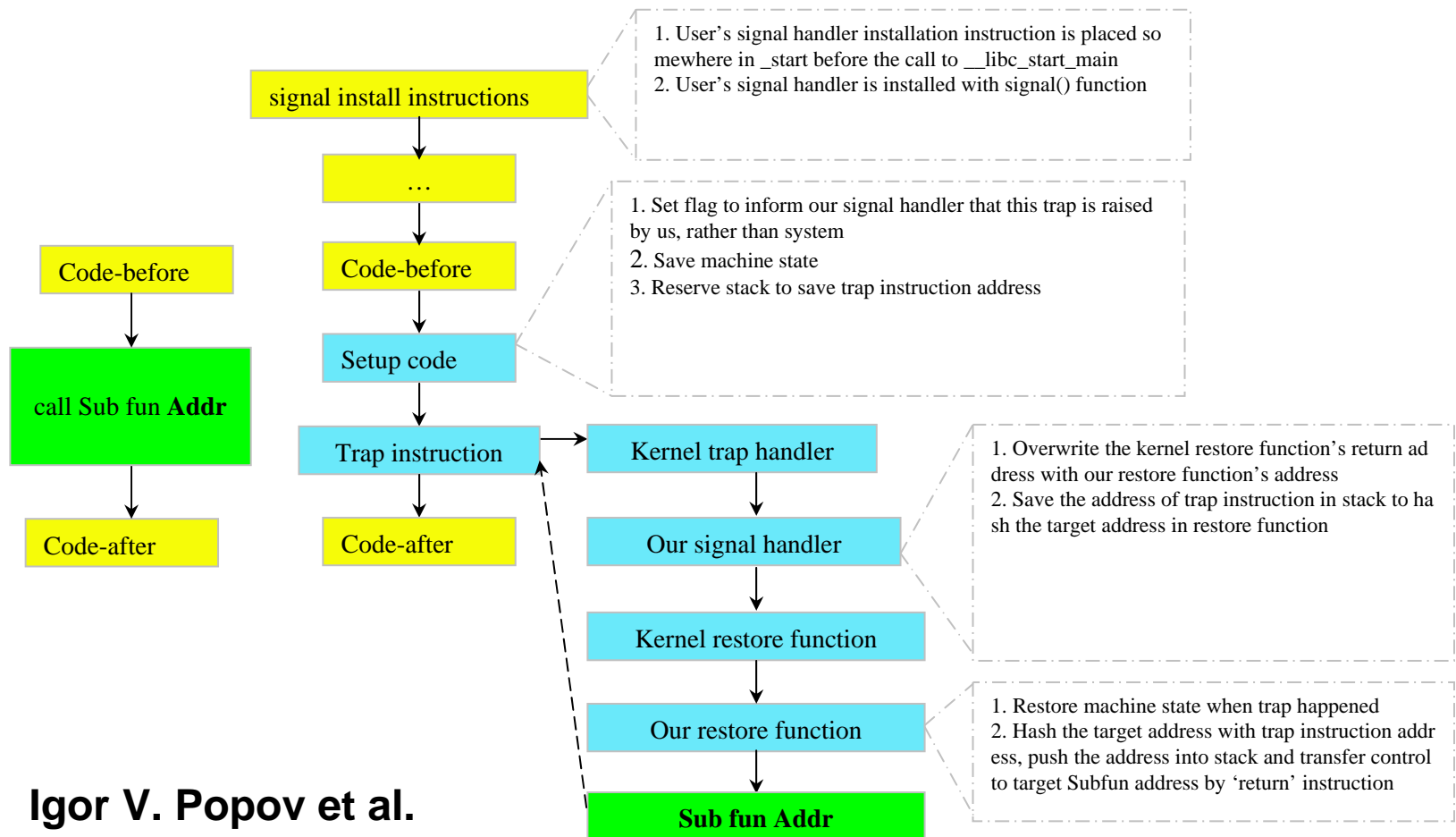
Signal-based Obfuscation

- Signal-based obfuscation



Igor V. Popov et al.

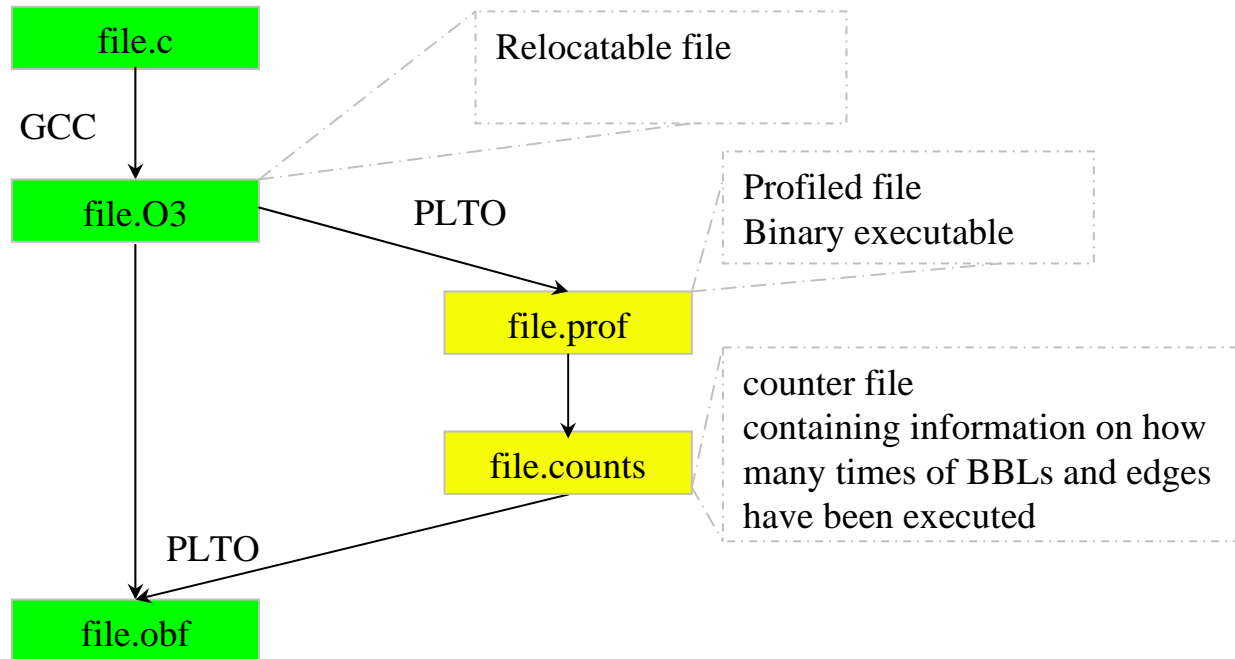
Signal-based Obfuscation



Igor V. Popov et al.

Signal-based Obfuscation

- Signal-based obfuscation



Self-Modifying Code Techniques

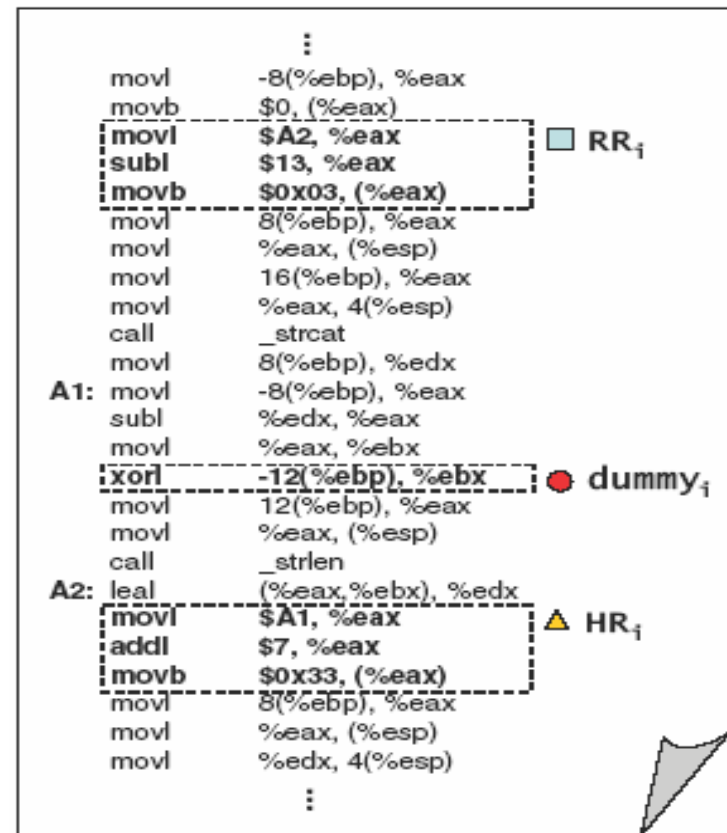
- Y. Kanzaki, A. Monden, M. Nakamura, and K. ichi Matsumoto; Exploiting self modification mechanism for program protection.
- Matias Madou, Bertrand Anckaert, Patrick Moseley, Saumya Debray, Bjorn De Sutter, and Koen De Bosschere; Software Protection Through Dynamic Code Mutation.

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).



References

- [1] C. Collberg, C. Thomborson, and D. Low, “Manufacturing cheap, resilient, and stealthy opaque constructs,” in Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM New York, NY, USA, 1998, pp. 184–196.
- [2] A. Majumdar and C. Thomborson, “Manufacturing opaque predicates in distributed systems for code obfuscation,” in Proceedings of the 29th Australasian Computer Science Conference-Volume 48. Australian Computer Society, Inc. Darlinghurst, Australia, Australia, 2006, pp. 187–196.
- [3] I.V. Popov, S.K. Debray, and G.R. Andrews, “Binary obfuscation using signals,” in Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium table of contents. USENIX Association Berkeley, CA, USA, 2007.
- [4] M. Madou, B. Anckaert, P. Moseley, S. Debray, B. De Sutter, and K. De Bosschere, “Software Protection Through Dynamic Code Mutation,” LECTURE NOTES IN COMPUTER SCIENCE, vol. 3786, pp. 194,2006.
- [5] J. Ge, S. Chaudhuri, and A. Tyagi, “Control flow based obfuscation,” in Proceedings of the 5th ACM workshop on Digital rights management. ACM New York, NY, USA, 2005, pp. 83–92.
- [6] Y. Kanzaki, A. Monden, M. Nakamura, and K. Matsumoto, “Exploiting self-modification mechanism for program protection,” in Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International, 2003, pp. 170–179.

References

- [7] C. Wang, J. Davidson, J. Hill, and J. Knight, “Protection of Software-based Survivability Mechanisms,” in Proc. International Conference of Dependable Systems and Networks, July 2001.
- [8] Chenxi Wang, “A Security Architecture for survivability Mechanisms,” in Phd thesis, Department of Computer Science, University of Virginia, October 2000.
- [9] S. Chow, Y. Gu, H. Johnson, and V.A. Zakharov, “An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs,” In Proc. 4th. Information Security Conference (ISC 2001), pp. 144–155, 2001.
- [10] C. Wang, J. Hill, J. Knight, and J. Davidson, “Software tamper resistance: Obstructing static analysis of programs,” University of Virginia, Charlottesville, VA, 2000.
- [11] B. Anckaert, M. Madou, and K. De Bosschere, “A Model for Self-Modifying Code,” LECTURE NOTES IN COMPUTER SCIENCE, vol. 3825, pp. 197–204, 2006.
- [12] M. Madou, B. Anckaert, B. De Sutter, and K. De Bosschere. Hybrid static-dynamic attacks against software protection mechanisms. Proceedings ACM workshop on Digital rights management, 2005.

Obfuscation Prototypes and Demo

- Diablo – link time binary rewriting system
- PLTO – link time binary rewriting system
- IDA Pro – Professional disassembler tool

- Demo

Thank You!!