Hybrid Static-Dynamic Attacks against Software Protection Mechanisms

Matias Madou, Bertrand Anckaert, Bjorn De Sutter and Koen De Bosschere

Ghent University, Belgium
Motivation

• Advances in reverse engineering and program analysis: Are the techniques to protect software still secure?
  - Techniques: level of security against static-only attacks
  - Easy to break with static-dynamic and dynamic-only attacks

• Goal:
  - Discuss attack models
  - Raise the bar for future techniques
Contributions

- Classification of attacks against protection techniques
- Provide a realistic attack model
- How to obtain program understanding

To illustrate this:

A realistic attack on a recently proposed algorithm for software watermarking
Case Study: Watermarking

Method:
Introducing: branch function (f)
Manipulates the return address

Dynamic path-based software watermarking, Collberg et al. (PLDI04)
Madou et al., Hybrid Static-Dynamic Attacks against Software Protection Mechanisms
Case Study: Watermarking

Encoding bits:
forward jump: 1
backward jump: 0

Tamper resistance:
introducing side-effects in the branch function

Dynamic path-based software watermarking, Collberg et al.(PLDI04)

Madou et al., Hybrid Static-Dynamic Attacks against Software Protection Mechanisms
Could the assembly code be the output of a regular compiler?

- **Dynamic**: call to a specific function doesn't return to the proper return address... tail call optimization? No
- **Static**: a lot of calls to that function (branch function). Number of call-sites to the function: gcc: 15336 call-sites
Branch function: observe behavior

- **observe:**
  - input-output behavior
  - side-effects
Static-Dynamic attack

- **Static**: Identify calls to the branch function
- **Dynamic**: Execute the branch function under control of GDB
- **Finally**:
  - modify calls to direct jumps
  - make sure side-effects are correct
Madou et al., Hybrid Static-Dynamic Attacks against Software Protection Mechanisms

---

**Static**: branch function: 0x809ddb7

**Identify calls**

**Dynamic**:  
- Breakpoint  
- Execute

---

```
80487ee: call 0x809ddb7
80487f3: lea 0xfffffffff4(%ebp),%esp
...
80487fb: sub $0xc,%esp
80487fe: push $0x80a1940
8048803: call 0x804e86a
8048808: add $0x10,%esp
804880b: call 0x809ddb7
8048810: call 0x809ddb7
8048815: sub $0xc,%esp
8048818: push $0x80a1955
804881d: call 0x804e86a
8048822: add $0x10,%esp
8048825: call 0x809ddb7
804882a: nop
...
```

---

```
809ddb7: pushf
809ddb8: push %edx
...
809de03: popf
809de04: ret
```
Madou et al., Hybrid Static-Dynamic Attacks against Software Protection Mechanisms
Collecting Information

Software

Hardware

Machine code instrumentation

source code instrumentation

Library instrumentation

Hardware instrumentation
Machine code instrumentation

Difficulties:

• Code and data

• relocating code and data after insertion (self-modifying code, tamper resistance, ...)

Solution: Diota (http://www.elis.ugent.be/diota)

• Running a program unaltered

• Generate instrumentation code on the fly somewhere else
(1) Collecting Instructions

Problem: 8b 44 24 04 03 44 24 0c ....
Disassemble!

Static: linear sweep and recursive traversal
Data in code, overlap of instructions, insertion of junk bytes,...

\texttt{mov 4(\%esp), \%eax}
(1) Collecting Instructions

Problem: 8b 44 24 04 03 44 24 0c ....
Disassemble!

Static: linear sweep and recursive traversal
Data in code, overlap of instructions, insertion of junk bytes,...

\[
\begin{align*}
  8b & \text{ mov 4(\%esp),\%eax} \\
  44 & \text{ inc \%esp} \\
  24 & 04 \\
  04 & 03 \\
  44 & 24 \\
  24 & 0c
\end{align*}
\]
(1) Collecting Instructions

Problem: 8b 44 24 04 03 44 24 0c ....
Disassemble!

Static: linear sweep and recursive traversal
Data in code, overlap of instructions, insertion of junk bytes,...

```
8b  mov  4(%esp),%eax
44  inc  %esp
24  add $4,%al
04
03
44
24
0c
```

Dynamic: only executed code (coverage 80%)
Madou et al., Hybrid Static-Dynamic Attacks against Software Protection Mechanisms
(1) Collecting Instructions

(1) Dynamic execution: skeleton of the program

(2) Static disassembler: fills in the missing holes

jne 0x200

0x200:
(2) Collecting Control Transfers

- **Needed:** Instructions + Control transfer
- **Detecting control transfers:** a large number of tools are confronted with this problem
- **Problems:** Indirect jumps; obfuscators opaque predicates
(2) Collecting Control Transfers

(1) Dynamic execution: skeleton of the program
   (Instructions and control transfers)

(2) Static disassembler: fills in the missing holes

(3) Static control transfer

Assemble no overlap
Assume opaque predicates

Madou et al., Hybrid Static-Dynamic Attacks against Software Protection Mechanisms
Attacks

• Static/Dynamic
• Conservative/Approximative
• Sound/Unsound
• Stand-alone/Environment-dependent

Realistic:
Static and Dynamic information/approximative and practically sound
Summary

• Discussed the power of a hybrid static-dynamic attack
• We hope this leads to stronger software protection mechanisms
Questions?

Presentation: http://www.madou.net