A Software Protection Method Based on Instruction Camouflage

2nd COE Postdoctoral and Doctoral Researchers Technical Presentation
May 26 , 2005

Yuichiro Kanzaki

Software Engineering Laboratory,
Graduate School of Information Science,
Nara Institute of Science and Technology

Background

Software cracking has posed a serious problem for copyright protection of the software.

Example

- An attacker analyzes a digital contents distribution system and obtains a secret key[1].
- An attacker analyzes a program embedded in a set-top box and steals a device key[2].

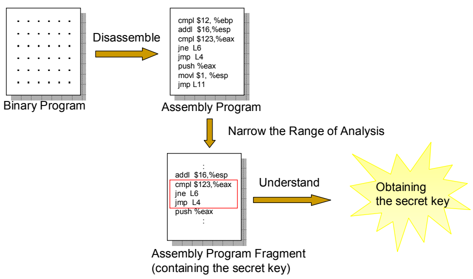
Attacker : an individual who illegally analyzes software, and uses the outcome for other purposes.

We need a method for protecting software to create a safe ubiquitous computing environment.

[1] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot: A white-box DES implementation for DRM applications, Proc. 2nd ACM Workshop on Digital Rights Management, pp.1-15, Nov. 2002.
[2] The United Kingdom Parliament, ``The mobile telephones (re-programming) bill,'' House of Commons Library Research Paper no.02/47, July 2002.

How is software attacked ?

A scenario of obtaining a secret key in a program



An effective solution to protect software against illegal code analysis is to increase costs for understanding the program.

Previous work and Goal

Methods to increase cost for understanding a program

- Program Obfuscation
  Making expressions and procedures in a program more complex than the original[1].
- Program Encryption
  Making a program harder to understand with encryption[2].

Previous methods are still impractical :

- difficult to automate
- easy to nullify

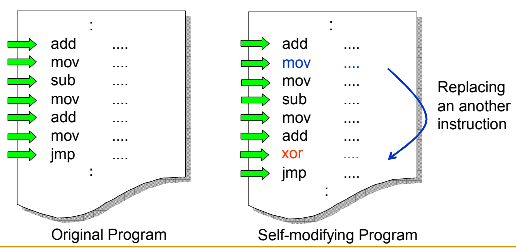
We propose a practical method for protecting software and develop a system.

[1] C. Collberg, C. Thomborson, ``Watermarking, Tamper-Proofing, and Obfuscation -- Tools for Software Protection,'' IEEE Trans. Software Eng., vol.28, no.8, pp.735-746, June 2002.
[2] F. B. Cohen,"Operating system protection through program evolution," Computers and Security, vol.12, Issue 6, pp. 565-584, 1993.

Approach - Self-modification mechanism

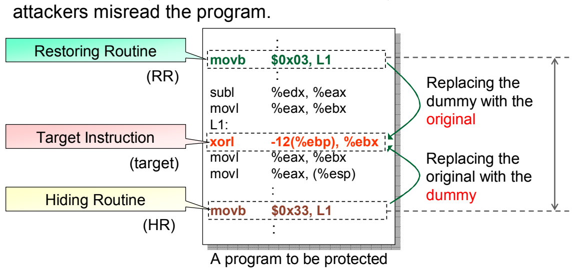
We add a self-modification mechanism to a program, to increase the cost for understanding a program.

self-modification: An instruction in the program replaces another instruction in the same program at run-time.



Camouflaging an instruction

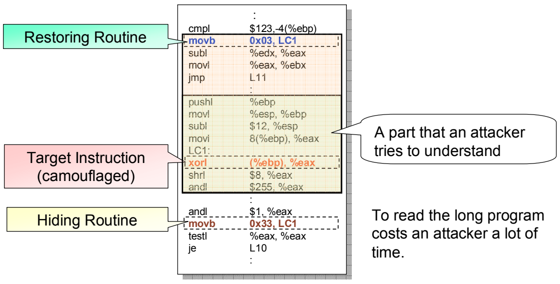
Overwrite an original instruction with a dummy, which makes attackers misread the program.



1. We overwrite a target instruction with a dummy instruction.
2. We add self-modification routines that replace the dummy instruction with the original one within a certain period of execution.

Extending a range of analysis

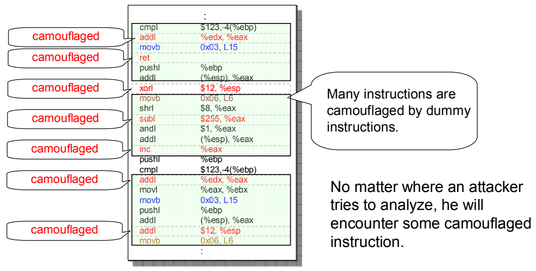
Camouflaged instructions force attackers into extending the range of analysis.



A program to be protected

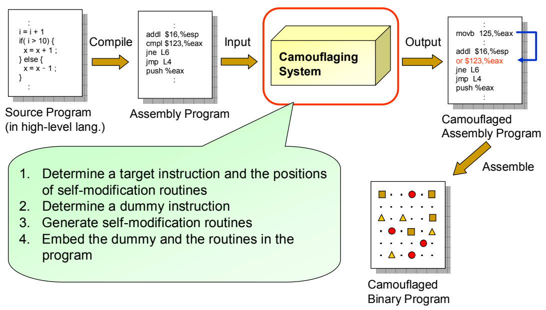
Multiple camouflaging

We camouflage many of the original instructions by dummy instructions and add routines.



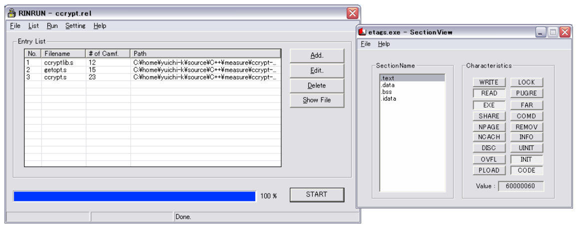
A program to be protected

Outline of our system



RINRUN

We have implemented a system that automates the construction of camouflaged programs[1].



RINRUN outputs *camouflaged* Windows Executable from C source (GCC)

[1] http://se.naist.jp/rinrun/

Discussion – Overview

We discuss the effectiveness of the proposed method.

Discussion Items (based on [1])

- obscurity — cost for understanding a part of program
- resilience — difficulty of constructing an automatic tool to undo
- stealth — difficulty of finding protection mechanism embedded
- overhead — the execution time/space penalty incurred

Example Program

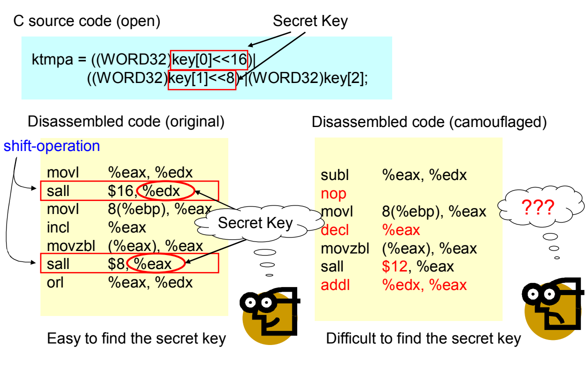
A simple data decryption program using C2

C2(Cryptomeria Cipher) :

- A Feistel network-based block cipher designed for use in the area of digital entertainment content protection
- Algorithm is open to the public[2]
- *Secret Key* is security-sensitive

[1] C. Collberg, C. Thomborson, ``Watermarking, Tamper-Proofing, and Obfuscation — Tools for Software Protection,'' IEEE Trans. Software Eng., vol.28, no.8, pp.735–746, June 2002.
[2] 4C Entity, "Content protection for recordable media specification – Introduction and common cryptographic elements," rev. 1.0, 31 pp.1.0, Jan. 2003.

Discussion – Obscurity(1/2)



Discussion – Obscurity(2/2)

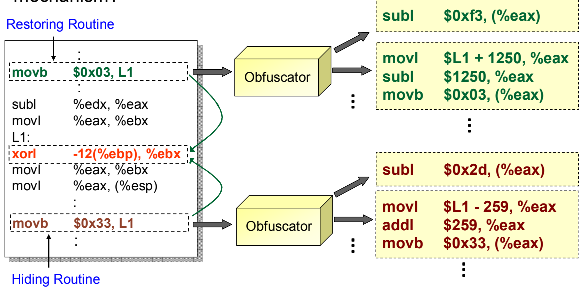
C source code — Secret Key

`secretconst[] = {0x20, 0x35, 0x4f …}`

Disassembled code (original)	Disassembled code (camouflaged)
:	:
movb $20, -24(%ebp)	movb $08, -24(%ebp)
movb $35, -23(%ebp)	movb $31, -23(%ebp)
movb $4f, -22(%ebp)	movb $55, -22(%ebp)
:	:
Secret Key	***Camouflaged* Secret Key**
Easy to obtain the secret key	Difficult to obtain the secret key

Discussion - Resilience

Is it difficult for attackers to automatically nullify the protection mechanism?



Discussion - Stealth

Is it difficult for attackers to find the protection mechanism embedded?

The self-modification routines mainly consist of *common* instructions (e.g. ,*mov* , *add* ..)

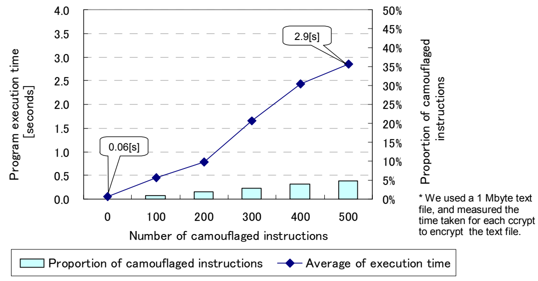
Instruction Distribution of the sample program

instruction	proportion
mov	55%
shift	9%
add/sub	8%
lea	6%
or	6%
jmp/jg/jle	4%
other	12%

It is not easy to notice that the mechanism is embedded.

Discussion – Performance Overhead

Target program : *ccrypt* (well-known GNU utility for encrypting files)



When 500 instructions are camouflaged, the average execution time is about 2.9 seconds, which is about 47 times as long as the original (0.06 seconds).

Conclusion and future plan

Conclusion

- We presented a systematic method for protecting software against the code analysis, by camouflaging instructions.
- We discuss the effectiveness of the proposed method.
  - Costs for understanding the program seems to be drastically increased.
  - The more we camouflage the instructions, the more expensive program overhead becomes.

Future Plan

- Improving our system in consideration of architectural aspects to reduce performance overhead.