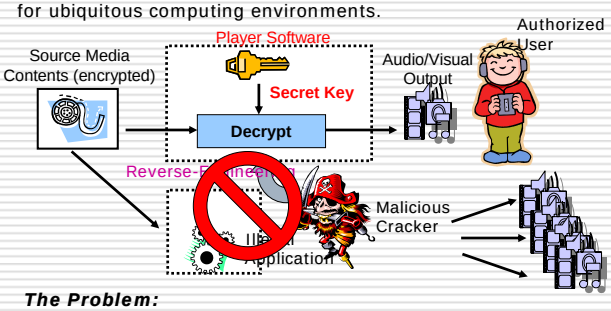
Applying obfuscation techniques to cipher programs using goal-oriented analysis

The 11th COE Postdoctoral and Doctoral Researchers Technical Presentation
Feb. 24, 2006.

Software Engineering Laboratory
Hiroki Yamauchi

Background

Digital Rights Management (DRM) software (such as iTunes and Windows Media Player) became an important application for ubiquitous computing environments.



The Problem:
Protect the authorized software (esp. Secret Key) from the reverse-engineering by malicious crackers.

Program Obfuscation

A technique of inhibiting the reverse-engineering, transforming a program into an equivalent one that is difficult to understand.

```
int fact=1, upper=16
int  i;
for(i=1;i<=upper;i++){
  fact *= i;
}
printf("%d",fact);
```

Original program [Computing 16!]

obfuscate

```
int a[] = {1,50};
int i=1; LOOP:
if(i!=(a[1]-2)/3)){ a[0]
*= (++i); goto LOOP;
}printf("%d",a[0]);
```

Obfuscated program [Computing 16!]

Effects:

- The obfuscated program preserves the specification of the original program.
- Cracking the obfuscated program requires an expensive cost.

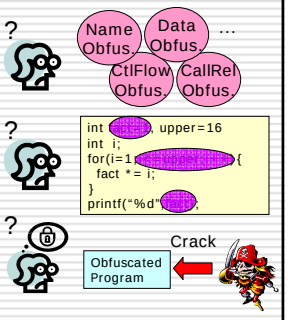
Existing Techniques:

- Name obfuscation
- Control flow obfuscation
- Data obfuscation (Type, Value, Structure)
- Operation obfuscation
- Inter module call relation obfuscation
- Overload induction obfuscation

Problem

There is no systematic method on how to apply obfuscation techniques appropriately.

- Which obfuscation technique should be used?
- Which part of the program should be obfuscated?
- How much effects of obfuscation can be expected?



These problems are caused because the conventional techniques do not count the purpose and target of the cracker.

Research Objective

Establish a goal-oriented analysis framework for proper use of the existing obfuscation techniques.

Key idea

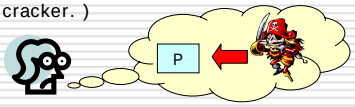
- Assume an imaginary cracker with his purpose and target (i.e., goal).
- Break down the goal into pieces, each of which an appropriate obfuscation is applied to.

Approach

Step1. Determine a capability of an imaginary cracker.
Step2. Identify a cracker's goal.
Step3. Conduct a goal-oriented analysis.
Step4. For every terminal sub-goal, select an obfuscation.
Step5. Apply the selected obfuscations to the program.

Step1. Determine an Imaginary Cracker.

Define a capability model of an imaginary cracker from three viewpoints: knowledge, observation, control (presuming a very skillful cracker.)



Knowledge : What does the cracker know beforehand?
- The cracker has full knowledge of the principles and external specification of a program.

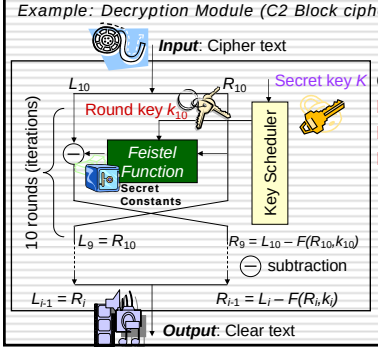
Observation : What can the cracker see in the program?
- The cracker owns a disassembled code of a program.
- The cracker can watch the program states and execution traces.

Control : What can the cracker do for the program?
- The cracker can execute the program with an arbitrary input.
- The cracker can modify the program in any desired way.

Step2. Identify Cracker's Goal

For the given program *P*, define specific goals, for which the cracker reverse-engineer *P*.

Example: Decryption Module (C2 Block cipher [1]*)*



Cracker's Goals (candiates)

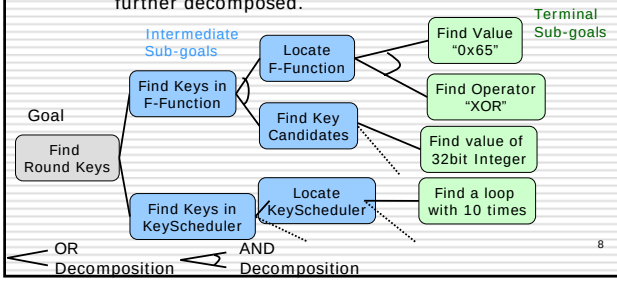
- ☐ **Find Round Keys**
- ☐ Find Secret Key
- ☐ Extract Secret Constants from Feistel Function

[1] 4C Entity, "Content protection for recordable media specification – Introduction and common cryptographic elements," rev. 1.0, 31 pp.1.0, Jan. 2003.

Step3. Conduct a Goal-Oriented Analysis

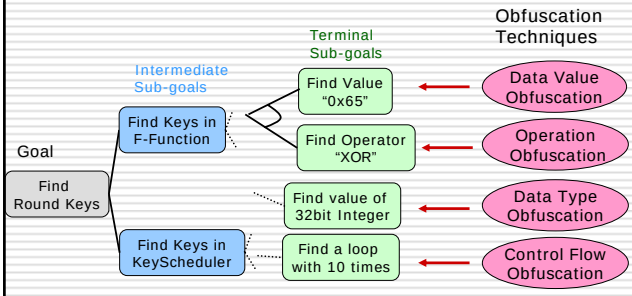
For each goal decided in Step2, decompose and refine the goal into more specific sub-goals.

- Make an AND-OR graph based on Cracker's Capability Model defined in Step1.
- Repeat the decomposition until no sub-goal can be further decomposed.



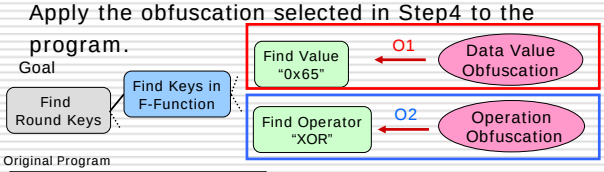
Step4. Select an appropriate obfuscation.

For each terminal sub-goal, select an appropriate obfuscation that can inhibit the sub-goal.



Step5. Apply Selected Obfuscations

Apply the obfuscation selected in Step4 to the program.



Original Program

```
public static int F(int data, int key) {
  :
  u = (byte)(v[0] ^ 0x65);
  :
}
```

O1: Data Value Obfuscation
Replace value "0x65" using homo-morphism f(x) = 2x+1

```
public static int F(int data, int key) {
  :
  int d = 203;
  :
  u = (byte)(v[0] ^ ((d-1)/2));
  :
}
```

```
public static int F(int data, int key) {
  :
  int d = 203;
  :
  u = (byte)(~v[0] & ((d-1)/2)) |
      (byte)((v[0] & ~((d-1)/2))
  :
}
```

O2: Operation Obfuscation
Replace XOR with NOT, AND, OR.

Case study

We have applied the proposed framework to a practical cipher program.

Target program

- A data decryption program using the C2 block cipher[1].
- Comprising 140 lines of Java code.

Cracker's Capability Model

Knowledge: Know C2 algorithm, but does not know keys.
Observation: Watch memory, stack and execution trace.
Control: Use all debugger operations, binary editor, disassembler, decompiler.

Goal

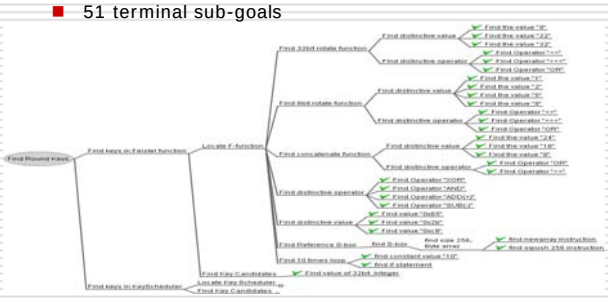
- Find the round key.

[1] 4C Entity, "Content protection for recordable media specification – Introduction and common cryptographic elements," rev. 1.0, 31 pp.1.0, Jan. 2003.

Result of Goal-Oriented Analysis

82 sub-goals have been identified.

- 31 intermediate sub-goals
- 51 terminal sub-goals



We confirmed all the terminal sub-goals can be inhibited by certain obfuscation techniques.

Summary

We have proposed a goal-oriented framework of applying the existing obfuscation techniques.

- Determine the capability model and goal of cracker.
- Using the goal-oriented analysis, decompose the goal into pieces, corresponding to an appropriate obfuscation.

Future Work

- Evaluate the proposed framework with other programs.
- Investigate optimal obfuscation.
  - Dependency analysis among sub-goals.
  - Dependency analysis among obfuscation techniques.

Thank you, That's ALL.