Obfuscating Transformations

Yury Lifshits

Mathematics & Mechanics Faculty
Saint Petersburg State University

Spring 2005 – SETLab

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Outline

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Motivation: Java Virtual Machine

What is difference between Java and others?

Most programming languages:	Java:
Source Code	Java Source Code
↓	↓
Machine Code	Java-byte Code
↓	↓
Predefined Architecture	Virtual Processor = JVM
	OS & Hardware

Java Virtual Machine: implemented on huge number of hardware architectures / operating systems

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Notion of Obfuscator

Objectives:

- ⇨ Make code not readable by human
- ⇨ Make automated analysis difficult
- ⇨ Make code more complicated
- ⇨ Make decompilation & reverse engineering difficult

Anatomy of Obfuscator I

How real obfuscator works?

1. Prepares program to be obfuscated
2. Makes a single transformation
3. Repeats **step 2** until task completed or constraints exceeded

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Anatomy of Obfuscator II

How real obfuscator works (more precisely)?

The workflow is:

- ⇨ Parse input program
  - Makes a list of obfuscation candidates: classes, variables, methods
  - Constructs internal representation of the program (e.g. control flow and basic blocks)
  - Makes some appropriateness suggestions
- ⇨ Main while loop (until constraints are exceeded or quality is achieved)
  - Choose next (by priority) element of the program to be obfuscated
  - Implement appropriate obfuscating transformation (from obfuscator library)
  - Update internal representation

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Quality of Obfuscation

Slide from Lecture 1

So you supposed to explain it to me...

Strength can be measured by:

- ⇨ Potency
  $\frac{E(P')}{E(P)} - 1$
- ⇨ Resilience
  Trivial, weak, strong, full, one-way
- ⇨ Cost
  Free, cheap, costly, expensive
- ⇨ Stealthy

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Program Complexity Metrics

We want: make program complicated

So what is program code complexity?

- ⇨ Program length
  Number of operators and operands
- ⇨ Data flow complexity
  Number of inter-block variable references
- ⇨ Cyclomatic complexity
  Number of predicates in a function
- ⇨ Nesting complexity
  Number of nesting level of conditionals in a program
- ⇨ Data structure complexity
  Complexity of the static data structures in the program like variables, vectors, records
- ⇨ OO Metrics
  Level of inheritance, coupling, number of methods triggered by another method, non-cohesiveness

Cost Analysis

What do we pay for security?

- ⇨ Costs at creation time

  Obfuscation need time!
- ⇨ Costs at transmition time (resulting size)

  Inlining library functions may increase size enormously!
- ⇨ Cost at execution time

  Checking procedures, dummy code, inlining
- ⇨ Cost by not using efficiency enhancing mechanisms

  Caching is rarely possible; losing module structure

Statistics measurement

Other metrics?

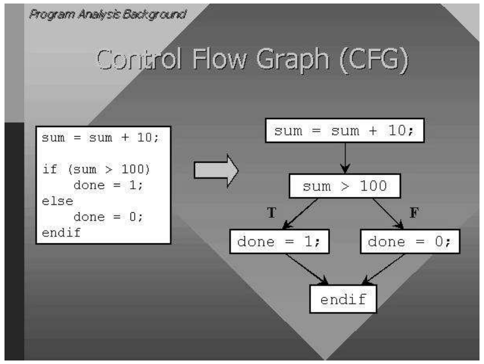
Statistics! What kind of?

- ⇨ Distribution of basic construstions
  - Use rare elements more often
- ⇨ Clustering of program elements. Uniform distribution – difficult to understand!
  - Usage of variables
  - Data processing
  - Control flow commands
- ⇨ Code patterns
  - Destroy long patterns in program

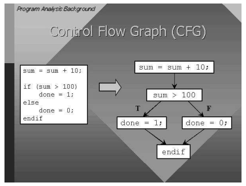

```
                                              $_='ev
                                           al('seek\040D
           ATA,0,                        0;');foreach(1..3)
       {<DATA>;}my                     @camel1hump;my$camel;
  my$Camel  ;while(                   <DATA>){$_=sprintf('%-6
 9s',$_);my@dromedary                1=split(//);if(defined($
_=<DATA>)){@camel1hum              p=split(//);}while(@dromeda
 ry1){my$camel1hump=0            ;my$CAMEL=3;if(defined($_=shif
        t(@dromedary1          ))&&/\S/){$camel1hump+=1<<$CAMEL;}
       $CAMEL--;if(d       efined($_=shift(@dromedary1))&&/\S/){
      $camel1hump+=1    <<$CAMEL;}$CAMEL--;if(defined($_=shift(
     @camel1hump))&&/\S/){$camel1hump+=1<<$CAMEL;}$CAMEL--;if(
     defined($_=shift(@camel1hump))&&/\S/){$camel1hump+=1<<$CAME
     L;;}$camel.=(split(//,'\040..m`{/J\047\134}L^7FX'))[$camel1h
      ump];}$camel.='\n';}@camel1hump=split(/\n/,$camel);foreach(@
      camel1hump){chomp;$Camel=$_;y/LJF7\173\175`\047/\061\062\063\
      064\065\066\067\070/;y/12345678/JL7F\175\173\047`/;$_=reverse;
       print'$_\040$Camel\n';}foreach(@camel1hump){chomp;$Camel=$_;y
        /LJF7\173\175`\047/12345678/;y/12345678/JL7F\175\173\0 47`/;
         $_=reverse;print'\040$_$Camel\n';}';;s/\s*//g;;eval;   eval
           ('seek\040DATA,0,0;');undef$/;$_=<DATA>;s/\s*//g;(   );;s
             ;^.*_;;;map{eval'print\'$_\'';}/.{4}/g; __DATA__   \124
               \1   50\145\040\165\163\145\040\157\1 46\040\1  41\0
                    40\143\141  \155\145\1 54\040\1   51\155\  141
                    \147\145\0  40\151\156 \040\141    \163\16 3\
                     157\143\   151\141\16  4\151\1     57\156
                     \040\167  \151\164\1   50\040\      120\1
                     45\162\   154\040\15    1\163\      040\14
                     1\040\1   64\162\1      41\144       \145\
                     155\14    1\162\        153\04       0\157
                      \146\     040\11      7\047\        122\1
                      45\15      1\154\1  54\171          \040
                      \046\         012\101\16            3\16
                      3\15           7\143\15             1\14
                      1\16            4\145\163           \054
                     \040            \111\156\14          3\056
                    \040\          125\163\145\14        4\040\
                    167\1          51\164\1  50\0       40\160\
                  145\162                              \155\151
                \163\163                                \151\1
              57\156\056
```


Control Flow Model I

What is *program* from our point of view?



Control Flow Model II



Compilers and program optimization theory represent programs by control flow graph (CFG)

- Each node in the graph represents a basic block
- Basic block: straight-line piece of code without any jumps or jump targets
- Directed edges are used to represent jumps in the control flow
- Jump targets start a block; jumps end a block

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Simple transformations

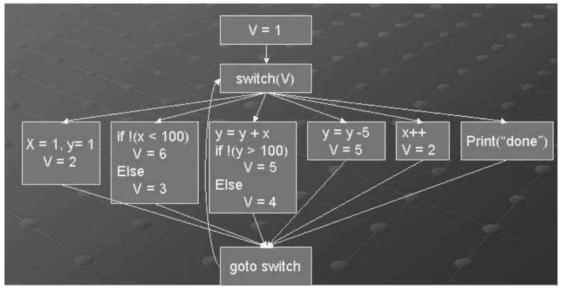
So what transformations do you can suggest?

- Data Obfuscation
  - Variable splitting
  - Scalar/object conversion
  - Static data to procedure
  - Change variable lifetime
  - Split/fold/merge arrays
  - Change encoding
  - Merge scalar variables
- Control Obfuscation
  - Break basic blocks
  - Inline methods
  - Outline statements
  - Unroll loops
  - Reorder statements
  - Reorder loops

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Control Flow Flattening

How to destroy control flow graph?



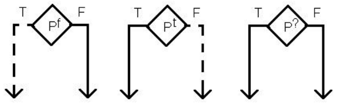
Step by step:

- Write down a list of all basic blocks
- Split and merge some of them
- Enumerate them
- Replace all calls by indirect pointing
- Write a single dispatcher to maintain all control flow

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Opaque predicates

How can we use IF operator for obfuscation?



Opaque predicates: every time the same value
Difficult to discover by automatical static analysis

Examples:

$$\boxed{((q + q^2)\ mod\ 2) \ = \ 0}$$

$$\boxed{((q^4)\ mod\ 16) \ = \ 0 \text{ OR } ((q^4)\ mod\ 16) \ = \ 1}$$

Functions Unifying

How can we make program procedures indistinguishable?

Procedures relations are expressed by Program Call Graph: procedures are nodes and procedure calls are directed edges

Idea: merge functions and call universal function with additional parameter
Difficulty: different signatures (input-output specifications)

Solution: unify signatures (in groups)

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Even more transformations

Question: Can you invent more?

- ⇨ Reuse identifiers
- ⇨ Introduce misleading comments :-)
- ⇨ Modify inheritance relations
- ⇨ Convert static data to procedural data
- ⇨ Store part of the program as a text and interpret it only during runtime
- ⇨ Remove library calls

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Current Techniques: Pro and Contra

Advantages

- ✔ Easy to implement
- ✔ Universal
- ✔ Good against static analysis

Disadvantages

- ✘ No guaranteed security
- ✘ Even no hope for that
- ✘ Weak against dynamic attacks

Obfuscating Transformations
Yury Lifshits
What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics
Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations
Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research
Summary

Classical Program Analysis

What's about program analysis?

Static analysis: only read code
Dynamic analysis: execute code

Usual tasks:

- ⇨ May be aliased
- ⇨ Must be aliased
- ⇨ May be modified
- ⇨ Must be constant
- ⇨ Must be killed
- ⇨ Must be available
- ⇨ May be used before kill
- ⇨ May be referenced

Is Deobfuscation Hard?

Can we prove the difficulty of deobfuscation?

Not yet. But...

We can prove program analysis to be hard for obfuscated programs:

Alias analysis of obfuscated programs is NP-hard!

Obfuscating Transformations
Yury Lifshits

What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics

Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations

Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research

Summary

Approaches to Deobfuscation

Almost all obfuscating transformations have a efficient deobfuscating method...

Do you believe?

Deobfuscator can use:

- ⇨ Functions parameters catching
- ⇨ Program slicing
- ⇨ Statistical analysis
- ⇨ Data flow analysis
- ⇨ Pattern matching

Obfuscating Transformations
Yury Lifshits

What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics

Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations

Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research

Summary

Further Research

What can we do here?

- ⇨ Just new transformations
- ⇨ Preventive transformations
- ⇨ Protection against recompilation
- ⇨ Introducing more deobufuscation hardness results

Good Luck with this stuff!

Obfuscating Transformations
Yury Lifshits

What is Obfuscator?
Notion of Obfuscator
Anatomy of Obfuscator
Obfuscator Characteristics

Obfuscation Library
Program Representation
Data & Control : Basic Tricks
Control Flow Obfuscation
Even more transformations

Obfuscation vs. Deobfuscation
Classical Program Analysis
Deobfuscation Hardness
Further Research

Summary

Summary

- ⇨ Obfuscator analyse and modify program by series of transformations
- ⇨ Obfuscating transformations consist of layout, data and control tricks
- ⇨ Hardness of deobfuscation is not proved

Question Time!

Not covered by the talk

Obfuscation vs. watermarking
Obfuscation for watermarking
Making disassembling hard.
Decompiled – uncompilable for Java
General idea – make program dictionary as short as possible
Preventive obfuscation
Profiling in the obfuscator
Reducible and non-reducible graphs
Are obfuscating transformations comparable, e.g. one OT is every time better than another OT?
Program Analysis classification

For Further Reading

Collberg - Thomborson - Low
Series of papers
http://www.cs.arizona.edu/~collberg/research/publications/