Industrial Approach: Obfuscating Transformations

Yury Lifshits

Steklov Institute of Mathematics, St.Petersburg, Russia
yura@logic.pdmi.ras.ru

Tartu University
17/03/2006

Commercial Obfuscators:

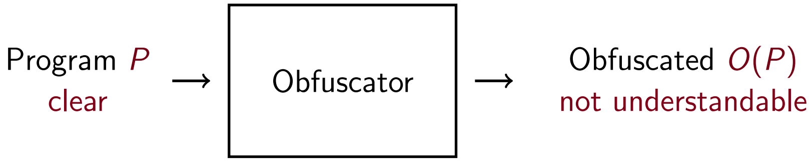
- Semantic Designs: Thicket^{tm} obfuscators
  `http://www.semanticdesigns.com/Products/Obfuscators/`
- Zelix Klassmaster^{tm} obfuscator
  `http://www.zelix.com/klassmaster/`
- PreEmptive: DotObfuscator^{tm}
  `http://www.preemptive.com/products/dotfuscator/`
- Only for Java: at least 26 obfuscators
  `http://dmoz.org/Computers/Programming/Languages/Java/`
  `Development_Tools/Obfuscators/`

Outline

Outline

Outline

Objectives of Obfuscator



Objectives:

- Make automated analysis difficult

Objectives of Obfuscator

$$\begin{array}{c} \text{Program } P \\ \text{clear} \end{array} \longrightarrow \boxed{\text{Obfuscator}} \longrightarrow \begin{array}{c} \text{Obfuscated } O(P) \\ \text{not understandable} \end{array}$$

Objectives:

- Make automated analysis difficult
- Make code more complicated

Objectives of Obfuscator

$$\begin{tikzcd} \underset{\text{clear}}{\text{Program } P} \arrow[r] & \boxed{\text{Obfuscator}} \arrow[r] & \underset{\text{not understandable}}{\text{Obfuscated } O(P)} \end{tikzcd}$$

Objectives:

- Make automated analysis difficult
- Make code more complicated
- Make decompilation & reverse engineering difficult

Objectives of Obfuscator

$$\begin{tikzcd} \begin{array}{c}\text{Program } P \\ \text{clear}\end{array} \arrow[r] & \boxed{\text{Obfuscator}} \arrow[r] & \begin{array}{c}\text{Obfuscated } O(P) \\ \text{not understandable}\end{array} \end{tikzcd}$$

Objectives:

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

Anatomy of Obfuscator (1)

How real obfuscator works?

Anatomy of Obfuscator (1)

How real obfuscator works?

1. Prepares program to be obfuscated

Anatomy of Obfuscator (1)

How real obfuscator works?

1. Prepares program to be obfuscated
2. Makes a single transformation

Anatomy of Obfuscator (1)

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

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- Parse input program

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- Parse input program
    - Makes a list of obfuscation candidates: classes, variables, methods

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- 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)

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- 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

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- 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)

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- 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

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- 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)

Anatomy of Obfuscator (2)

The workflow of obfuscator:

- 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

Quality of Obfuscation

How good is obfuscation? Measures:

- Potency
  $\frac{\text{Complexity}(\mathcal{O}(P))}{\text{Complexity}(P)}$
- Resilience (irreversibility)
  Weak, strong, one-way
- Cost
  Slowdown, increasing of code size and space requirements
- Stealth
  How similar are introduced obfuscated constructions to the rest of the code

Software Complexity Metrics

How do you define a program code complexity?

Software Complexity Metrics

How do you define a program code complexity?

- Program length

Software Complexity Metrics

How do you define a program code complexity?

- Program length

Number of operators and operands

Software Complexity Metrics

How do you define a program code complexity?

- Program length

  Number of operators and operands

- Data flow complexity

Software Complexity Metrics

How do you define a program code complexity?

- Program length

  Number of operators and operands

- Data flow complexity

  Number of inter-block variable references

Software Complexity Metrics

How do you define a program code complexity?

- Program length

  Number of operators and operands

- Data flow complexity

  Number of inter-block variable references

- Cyclomatic complexity

Software Complexity Metrics

How do you define a 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

Software Complexity Metrics

How do you define a 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

Software Complexity Metrics

How do you define a 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

Software Complexity Metrics

How do you define a 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

Software Complexity Metrics

How do you define a 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

Software Complexity Metrics

How do you define a 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

Software Complexity Metrics

How do you define a 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

Statistical Metrics

Measuring chaos:

- Distribution of opcodes (and any elements of program)
    - Rare elements contain iformation. Replace them by basic instructions
- Clustering (usage of variables, control flow commands)
    - Best of all: no clastering, uniform distribution
- Code patterns
    - Destroy long repeating patterns in program

Cost Analysis

What do we pay for security?

- Costs at creation time

Obfuscation need time!

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 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 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

Outline

Top Three Methods

- Renaming
  variables/procedures/classes/methods
- Deleting comments and spaces
  (destroying layout)
- Inserting dead code

Data Obfuscation

Any ideas for data obfuscation?

Data Obfuscation

Any ideas for data obfuscation?

- Variable splitting

Data Obfuscation

Any ideas for data obfuscation?

- Variable splitting
- Scalar/object conversion

Data Obfuscation

Any ideas for data obfuscation?

- Variable splitting
- Scalar/object conversion
- Static data to procedure

Data Obfuscation

Any ideas for data obfuscation?

- Variable splitting
- Scalar/object conversion
- Static data to procedure
- Change variable lifetime

Data Obfuscation

Any ideas for data obfuscation?

- Variable splitting
- Scalar/object conversion
- Static data to procedure
- Change variable lifetime
- Split/fold/merge arrays

Data Obfuscation

Any ideas for data obfuscation?

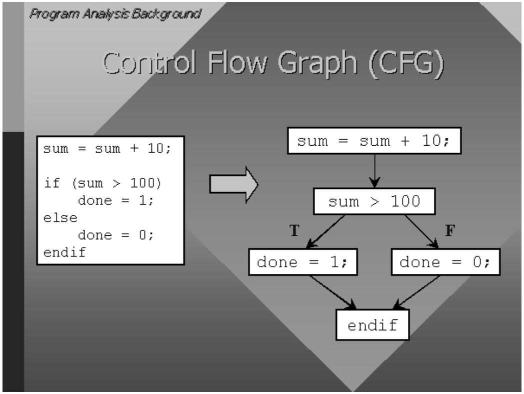
- Variable splitting
- Scalar/object conversion
- Static data to procedure
- Change variable lifetime
- Split/fold/merge arrays
- Change encoding

Data Obfuscation

Any ideas for data obfuscation?

- Variable splitting
- Scalar/object conversion
- Static data to procedure
- Change variable lifetime
- Split/fold/merge arrays
- Change encoding
- Merge scalar variables

Control Flow (1)



Program Analysis Background

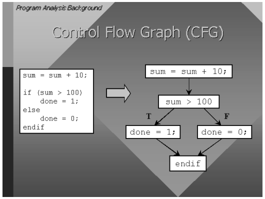
Control Flow Graph (CFG)

```
sum = sum + 10;

if (sum > 100)
     done = 1;
else
     done = 0;
endif
```

$$\begin{tikzcd}
 & \boxed{\texttt{sum = sum + 10;}} \arrow[d] & \\
 & \boxed{\texttt{sum > 100}} \arrow[dl, "T"'] \arrow[dr, "F"] & \\
\boxed{\texttt{done = 1;}} \arrow[dr] & & \boxed{\texttt{done = 0;}} \arrow[dl] \\
 & \boxed{\texttt{endif}} &
\end{tikzcd}$$

Control Flow (2)



```
sum = sum + 10;

if (sum > 100)
    done = 1;
else
    done = 0;
endif
```

Compiler theory: program = control flow graph (CFG)

- Node = basic block = straight-line piece of code without any jumps or jump targets
- Directed edges = jumps in the control flow
- Every block: starts from jump target, ends by jump command

Control Flow: Basic Tricks

Any ideas for control flow obfuscation?

Control Flow: Basic Tricks

Any ideas for control flow obfuscation?

- Break basic blocks

Control Flow: Basic Tricks

Any ideas for control flow obfuscation?

- Break basic blocks
- Inline methods

Control Flow: Basic Tricks

Any ideas for control flow obfuscation?

- Break basic blocks
- Inline methods
- Outline statements

Control Flow: Basic Tricks

Any ideas for control flow obfuscation?

- Break basic blocks
- Inline methods
- Outline statements
- Unroll loops

Control Flow: Basic Tricks

Any ideas for control flow obfuscation?

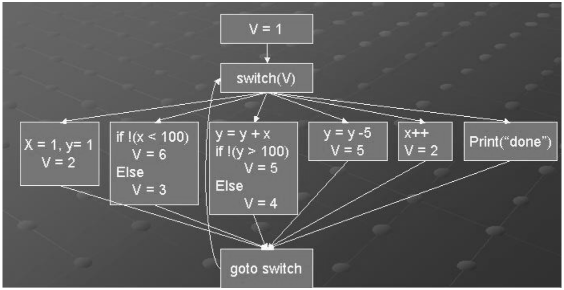
- Break basic blocks
- Inline methods
- Outline statements
- Unroll loops
- Reorder statements

Control Flow: Basic Tricks

Any ideas for control flow obfuscation?

- Break basic blocks
- Inline methods
- Outline statements
- Unroll loops
- Reorder statements
- Reorder loops

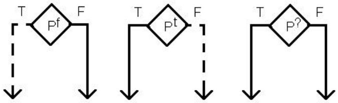
How to Destroy a Control Flow Graph?



1. Write down a list of all basic blocks
2. Split and merge some of them
3. Enumerate them
4. Replace all calls by indirect pointing
5. Write a single dispatcher to maintain all control flow

Opaque Predicates

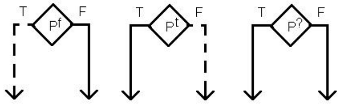
How can we use IF operator for obfuscation?



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

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?

Functions Unifying

How can we make program procedures indistinguishable?

Idea:

- Merge all functions to one
- Call universal function with additional parameter

Functions Unifying

How can we make program procedures indistinguishable?

Idea:

- Merge all functions to one
- Call universal function with additional parameter

Difficulty: different signatures (input-output specifications)

Functions Unifying

How can we make program procedures indistinguishable?

Idea:

- Merge all functions to one
- Call universal function with additional parameter

Difficulty: different signatures (input-output specifications)

Solution: unify signatures (in groups)

Even more transformations

Question: Can you invent more?

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
- Protection aginst specific decompiling tools

Current Techniques: Pro and Contra

Advantages:

Current Techniques: Pro and Contra

Advantages:

- ✓ Easy to implement

Current Techniques: Pro and Contra

Advantages:

- ✓ Easy to implement
- ✓ Universal

Current Techniques: Pro and Contra

Advantages:

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

Current Techniques: Pro and Contra

Advantages:

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

Current Techniques: Pro and Contra

Advantages:

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

Disadvantages:

Current Techniques: Pro and Contra

Advantages:

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

Disadvantages:

- ✘ No guaranteed security

Current Techniques: Pro and Contra

Advantages:

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

Disadvantages:

- ✘ No guaranteed security
- ✘ Even no hope for that

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

Summary

Main points:

- Obfuscator workflow: parse the program; apply transformations until the cost is exceeded

Summary

Main points:

- Obfuscator workflow: parse the program; apply transformations until the cost is exceeded
- Obfuscating transformations consist of layout, data and control tricks

Summary

Main points:

- Obfuscator workflow: parse the program; apply transformations until the cost is exceeded
- Obfuscating transformations consist of layout, data and control tricks
- Hardness of deobfuscation is not proved

Course Conclusion

Why programming people like code obfuscation so much?

Course Conclusion

Why programming people like code obfuscation so much?

Programming: CONSTRUCTIVE process

Obfuscation: DESTRUCTIVE process

Reading List

- C. Collberg, C. Thomborson, D. Low
  A taxonomy of obfuscating transformations, 1997.
  http://www.cs.arizona.edu/people/collberg/Research/Publications/CollbergThomborsonLow97a/A4.ps.

- C. Collberg, C. Thomborson, D. Low
  Breaking abstractions and unstructuring data structures, 1998.
  http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborsonLow98b/LETTER.ps.

- S. Chow, Y. Gu, H. Johnson, V. Zakharov
  An approach to the obfuscation of control-flow of sequential computer programs, 1998.
  http://www.ispras.ru/groups/dma/downloads/Malaga2.zip.

- M. Mambo, T. Murayama, E. Okamoto
  A tentative approach to constructing tamper-resistant software, 1998.
  http://web.yl.is.s.u-tokyo.ac.jp/~cocoa/reading/archive/p23-mambo.pdf.

- C. Linn, S. Debray
  Obfuscation of executable code to improve resistance to static disassembly, 2003.
  http://www.cs.arizona.edu/~linnc/research/CCS2003.pdf.

Thanks for attention. Questions?

Course Feedback

1. Comments/suggestions on contents:
   - Choice of topics? Ratio of theoretical/practical?
2. Comments/suggestions on presentation aspects:
   - Your opinion on slides? Black board explanation? Language mistakes?
3. Comments/suggestions on technical aspects:
   - Timetable of the course? Webpage? Room? Announcement?
4. Main advantage of the course (if any)?
   - Best lecture in your opinion?
5. Disatvantages. What and how can be improved?