Software Mining and Re-engineering

About Code Obfuscation

Master M2 MoSiG (AISSE)

Academic Year 2018 - 2019
Code Obfuscation

→ Protecting a code against reverse-engineering techniques allowing to inspect and/or tamper a software (man at the end attacks !)
Code Obfuscation

→ Protecting a code against reverse-engineering techniques allowing to inspect and/or tamper a software (man at the end attacks !)

Typical applications domains:

▶ intellectual property of some algorithms
▶ data confidentiality
▶ white-box cryptography
▶ digital rights managements (DRM)
▶ ...
▶ and malware implementation !
Code Obfuscation

→ Protecting a code against reverse-engineering techniques allowing to inspect and/or tamper a software (man at the end attacks !)

Typical applications domains:

▷ intellectual property of some algorithms
▷ data confidentiality
▷ white-box cryptography
▷ digital rights managements (DRM)
▷ ...
▷ and malware implementation !

obfuscation may target various reverse-engineering approaches

▷ from source code vs from binary code
▷ manual vs tool-assisted
▷ static (i.e., code inspection) vs dymanic (i.e., code execution) techniques
▷ etc

⇒ a large spectrum of obfuscation techniques ...
Some examples of code obfuscation techniques

Kinds of obfuscation for each target information
Outline

Basic transformations

Examples of Data Obfuscation

Examples of Code Obfuscation

Some other obfuscation techniques
Example: source-level obfuscation against manual RE (1/3)

Example: From Stunnix

- Actual code:
  - function foo( arg1)
  - {
  -   var myVar1 = "some string"; //first comment
  -   var intVar = 24 * 3600;
  -   //second comment
  -   /* here is
  -   a long
  -   multi-line comment blah */
  -   document. write( "vars are:" + myVar1 + "" + intVar + "" + arg1);
  - }

- Obfuscated code:
  - function z001c775808( z3833986e2c) { var
  - z0d8bd8ba25= "\x73\x6f\x6d\x65\x20\x73\x74\x72\x69\x6e\x67"; var
  - z0ed9bcbcc2= (0x90b+785-0xc04) * (0x1136+6437-0x1c4b); document. write( "\x76\x61\x72\x73\x61\x61\x72\x65\x3a"+
  - z0d8bd8ba25+ "\x20"+
  - z0ed9bcbcc2+ "\x20"+
  - z3833986e2c);};}
Step by step examination

- The Stunnix obfuscator targets at obfuscating only the layout of the JavaScript code
- As the obfuscator parses the code, it removes spaces, comments and new line feeds
- While doing so, as it encounters user defined names, it replaces them with some random string
- It replaces print strings with their hexadecimal values
- It replaces integer values with complex equations
In the sample code that was obfuscated, the following can be observed:

- User defined variables:
  - foo replaced with z001c775808
  - arg1 replaced with z3833986e2c
  - myvar1 replaced with z0d8bd8ba25
  - intvar replaced with z0ed9bcbcc2

- Integers:
  - 20 replaced with (0x90b+785-0xc04)
  - 3600 replaced with (0x1136+6437-0x1c4b)

- Print strings:
  - “vars are” replaced with \x76\x61\x72\x73\x20\x61\x72\x65\x3a
  - Space replaced with \x20
Example: source-level obfuscation against manual RE (3/3)

- In the sample code that was obfuscated, the following can be observed

- User defined variables:
  - `foo` replaced with `z001c775808`
  - `arg1` replaced with `z3833986e2c`
  - `myvar1` replaced with `z0d8bd8ba25`
  - `intvar` replaced with `z0ed9bcbcc2`

- Integers:
  - `20` replaced with `(0x90b+785-0xc04)`
  - `3600` replaced with `(0x1136+6437-0x1c4b)`

- Print strings:
  - “vars are” replaced with `\x76\x61\x72\x73\x20\x61\x72\x65\x3a`
  - Space replaced with `\x20`
Outline

Basic transformations

Examples of Data Obfuscation

Examples of Code Obfuscation

Some other obfuscation techniques
Data re-encoding

Replace variables by complex expressions, e.g.,

```c
int a = arg1;
int b = arg2;
int x = a*b;
printf("x=%i\n",x);
```

replaced by

```c
a = 1789355803 * arg1 + 1391591831;
b = 1789355803 * arg2 + 1391591831;
x = ((3537017619 * (a * b) - 3670706997 * a) -
    3670706997 * b) + 3171898074;
printf("x=%i\n", -757949677 * x - 3670706997);
```
Data re-encoding

Replace **variables** by complex expressions, e.g.,

```c
int a = arg1;
int b = arg2;
int x = a*b;
printf("x=%i\n",x);
```

replaced by

```c
a = 1789355803 * arg1 + 1391591831;
b = 1789355803 * arg2 + 1391591831;
x = ((3537017619 * (a * b) - 3670706997 * a) -
     3670706997 * b) + 3171898074;
printf("x=%i\n", -757949677 * x - 3670706997);
```

Replace **standard arithmetic operations** by more complex ones, e.g.,

\[ z = x + y + w \]

replaced by:

```c
z = (((x ^ y) + ((x & y) << 1)) | w) +
    (((x ^ y) + ((x & y) << 1)) & w)
```
Data re-encoding

Replace variables by complex expressions, e.g.,

```c
int a = arg1;
int b = arg2;
int x = a*b;
printf("x=%i\n",x);
```

replaced by

```c
a = 1789355803 * arg1 + 1391591831;
b = 1789355803 * arg2 + 1391591831;
x = ((3537017619 * (a * b) - 3670706997 * a) -
    3670706997 * b) + 3171898074;
printf("x=%i\n", -757949677 * x - 3670706997);
```

Replace standard arithmetic operations by more complex ones, e.g.,

```
z = x + y + w
```

replaced by:

```
z = (((x ^ y) + ((x & y) << 1)) | w) +
    (((x ^ y) + ((x & y) << 1)) & w)
```

⇒ obfuscate the data operations performed in the code
Data split, fold or merge

- **Split** some variables of type $T_1$ into sets of variables of type $T_2$, e.g.:

  ```
  int a
  split into
  struct {char a1; char a2; char a3; char a4} a
  ```

- **Merge** some variables of type $T_1$, $T_2$ into a variable of type $T$, e.g.:

  ```
  int a; char b
  merged into
  long ab
  ```

- **Fold or Flatten** arrays into higher/lower dimensionnal arrays

- **Convert** static data into procedural data (“table look-up”, see next slide)

  → needs **alias computations** and **encoding/decoding** functions
Converting Static Data to Procedural Data

main() {
    String S1, S2, S3, S4;
    S1 = "AAA";
    S2 = "BAAA";
    S3 = "CCB";
    S4 = "CCB";
}

main() {
    String S1, S2, S3, S4, S5;
    S1 = G(1);
    S2 = G(2);
    S3 = G(3);
    S4 = G(5);
    if \( P^n \) S5 = G(9);
}

static String G (int n) {
    int i=0;
    int k;
    char[] S = new char[20];
    while (true) {
        \( L1: \) if (n==1) \{ S[i++]=='A'; k=0; goto L6 \};
        \( L2: \) if (n==2) \{ S[i++]=='B'; k=-2; goto L6 \};
        \( L3: \) if (n==3) \{ S[i++]=='C'; goto L9 \};
        \( L4: \) if (n==4) \{ S[i++]=='X'; goto L9 \};
        \( L5: \) if (n==5) \{ S[i++]=='C'; goto L11 \};
        \( \) if (n>12) goto L1;
        \( L6: \) if (k++<=2) \{ S[i++]=='A'; goto L6 \}
            else goto L8;
        \( L8: \) return String.valueOf(S);
        \( L9: \) S[i++]=='C'; goto L10;
        \( L10: \) S[i++]=='B'; goto L8;
        \( L11: \) S[i++]=='C'; goto L12;
        \( L12: \) goto L10;
    }
}
Outline

Basic transformations

Examples of Data Obfuscation

Examples of Code Obfuscation

Some other obfuscation techniques
Opaque predicates

Transform the control-flow graph (CFG) by inserting spurious conditions (evaluating always to \textbf{true})
The condition is given as complex predicate, those value is hard to predict at compile-time, i.e.:

- not removed by the optimizer
- not detected by static code analyser

\footnote{http://tigress.cs.arizona.edu/transformPage/docs/addOpaque/index.html}
Opaque predicates

Transform the control-flow graph (CFG) by inserting spurious conditions (evaluating always to true)
The condition is given as complex predicate, those value is hard to predict at compile-time, i.e.:

- not removed by the optimizer
- not detected by static code analyser

Some applications

- if expr=false then
call to random existing function
- if expr=false then
call to non-existing function
- if expr=true then
existing statement
else
buggified version of the statement

http://tigress.cs.arizona.edu/transformPage/docs/addOpaque/index.html
Virtualization

Turns a function into an interpreter by:
- generating a dedicated (bytecode) instruction set
- a bytecode array, a virtual program counter (VPC) and a virtual stack pointer (VSP)
- a dispatch unit, and the bytecode instruction handlers

```c
void foo()
{
    ...
}
```
Outline

Basic transformations

Examples of Data Obfuscation

Examples of Code Obfuscation

Some other obfuscation techniques
Anti-Disassembling

Code Obfuscation in Disassembly Phase

- Thwarting disassembly
- Junk Insertion
- Thwarting Linear Sweep
- Thwarting Recursive Traversal
  - Branch functions
  - Call conversion
  - Opaque predicates
  - Jump Table Spoofing
Anti-Dynamic analysis

Prevent a program to be analyzed under a debugger, an emulator, a virtual machine . . .

▶ use process control primitives to prevent debugging
e.g., `ptrace` on Linux,

▶ try to access regular peripherals (network, printer, filesystem, etc.)

▶ monitor the execution time

▶ etc.

*Rk:* (highly) used by malwares . . .!
Conclusion

Many other transformations proposed so far . . .

Expected properties of an obfuscator

▶ correctness: should preserve the code semantics
▶ resilience: should prevent (basic/advanced ?) reverse-engineering
▶ cost: should not “explode” the code complexity (time, memory, etc.)
Conclusion

Many other transformations proposed so far . . .

Expected properties of an obfuscator

- correctness: should preserve the code semantics
- resilience: should prevent (basic/advanced ?) reverse-engineering
- cost: should not “explode” the code complexity (time, memory, etc.)

However . . .

- no chance to build an universal obfuscator (i.e., able to obfuscate any input program)
- de-obfuscation tools are guided by existing obfuscation techniques . . . (keep your obfuscator secret !)
Conclusion

Many other transformations proposed so far . . .

Expected properties of an obfuscator

▶ correctness: should preserve the code semantics
▶ resilience: should prevent (basic/advanced ?) reverse-engineering
▶ cost: should not “explode” the code complexity (time, memory, etc.)

However . . .

▶ no chance to build an universal obfuscator
(i.e., able to obfuscate any input program)
▶ de-obfuscation tools are guided by existing obfuscation techniques . . .
(keep your obfuscator secret !)

Credits

▶ https://fr.slideshare.net/bijondesai/code-obfuscation
▶ https://fr.slideshare.net/amolkamble16121/code-obfuscation-40283580
▶ Christian Collberg web page: http://tigress.cs.arizona.edu/index.html