# Dynamic, Metamorphic (and opensource) Virtual Machines

A. Desnos

ESIEA - Operational Cryptology and Virology Laboratory (CVO)
38 rue des Dr Calmette et Guérin, 53 000 Laval, France
desnos@esiea.fr

Hack.lu 2010


# Current section

1 Introduction

2 Obfuscation

3 Virtual Machines

4 Android/Java appplications

5 Conclusion


# Introduction

## New techniques to enable efficient software obfuscation and protection

- Innovative
- Reusable
- Opensource

# Current section

1 Introduction

2 Obfuscation

3 Virtual Machines

4 Android/Java appplications

5 Conclusion


# Obfuscation

## Impossible?

- On the (Im)possibility of Obfuscating Programs, CRYPTO 2001 (B. Barak, O. Goldreich R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan and K. Yang)
  - Creating an obfuscator is impossible
  - But you can play with the time and the result

# Obfuscation

## T-Obfuscation

- On the possibility of practically obfuscating programs - Towards a unified perspective of code protection (Philippe Beaucamps, Eric Filiol)
  - You have to estimate the time ($\tau$) required to break your protection
    - Window of time
  - But this mainly relates to malwares or cyber attacks

# Obfuscation

## Definition of Obfuscation in our context

- We are not in a context of cyber attacks,
- We must try to protect a software against evil guys to steal the apps (or part of them ) and to resell them into the market by basic decompilation, and (un)obfuscation,
- We must use multiple technics, and not only basic packing.

# Obfuscation

## Using Virtual Machines?

- Actually it is one of the most difficult problems for malware analysts
- But it is not a full VM like Qemu, Bochs, Vmware
- VMProtect, Themida use VM
- Of course, it is just one step for the software protection

# Current section

1 Introduction

2 Obfuscation

3 Virtual Machines

4 Android/Java appplications

5 Conclusion

# Virtual Machines

## What's ?

- Simple code which interprets another one
  - Easy to use and modify
  - Dynamic
  - Metamorphic
  - Fast

## Steps

- Take the original instruction code (ASM, Bytecodes ...)
- Transform it into the desired intermediate language (IL)
- Build the VM
- Run it!

# Virtual Machines

## What's ?

- Simple code which interprets another one
  - Easy to use and modify
  - Dynamic
  - Metamorphic
  - Fast

## Steps

- Take the original instruction code (ASM, Bytecodes ...)
- Transform it into the desired intermediate language (IL)
- Build the VM
- Run it!

# Virtual Machines

## Which IL?

- Plainty of IL ...
- But we can use anyone!

## REIL

- Zynamics
  - REIL: A platform-independent intermediate language of disassembled code for static code analysis
  - Thomas Dullien and Sebastian Porst
  - http://www.zynamics.com/downloads/csw09.pdf

# Virtual Machines

## Which IL?

- Plainty of IL ...
- But we can use anyone!

## REIL

- Zynamics
  - REIL: A platform-independent intermediate language of disassembled code for static code analysis
  - Thomas Dullien and Sebastian Porst
  - http://www.zynamics.com/downloads/csw09.pdf

# Virtual Machines

## REIL

- 17 instructions (ADD, AND, BISZ, BSH, DIV, JCC, LDM, MOD, MUL, NOP, OR, STM, STR, SUB, UNDEF, UNKN, XOR)
- 3 operands (but some instructions use 0 or 2 operands)
- Operand can be a:
  - REIL REGISTER (no limit about the number of registers),
  - REIL INTEGER,
  - REIL OFFSET.
- Each operand has a specific size and the third operand is classicaly the output operand

# Virtual Machines

## REIL Format

- INSTR (X, bX), (Y, bY), (Z, bZ)

## REIL example

- ADD (t0, b4), (0x90, b4), (t1, b4)


# Virtual Machines

## REIL Format

- INSTR (X, bX), (Y, bY), (Z, bZ)

## REIL example

- ADD (t0, b4), (0x90, b4), (t1, b4)

# Virtual Machines

## REIL example

- Assembly instruction : "push ebp"

$\Rightarrow$ SUB (esp, b4, 0, 0), (0x4, b4, 1, 0), (esp, b4, 0, 0)

$\Rightarrow$ STM (ebp, b4, 0, 0), , (esp, b4, 0, 0)

# Virtual Machines

## Transformation

- Each operand :
  - type
  - size
- Types, Sizes, OP_CODE, O1, O2, O3
- 1320229, 262148, 233, 3049, 0, 49

# Virtual Machines

## Dynamic bytecodes

- At each generation of a VM
  - the format is different
  - the encoding is different
  - opcodes (instructions + registers) are different

# Virtual Machines

## Dynamic functions

- Implicit by the format and opcodes
- But it is possible to find "static" patterns

$\Rightarrow$ We must generate more dynamic code for the VM

# Virtual Machines

## Dynamic functions

- Implicit by the format and opcodes
- But it is possible to find "static" patterns

$\Rightarrow$ We must generate more dynamic code for the VM

# Virtual Machines

## Metamorphism

- Classical metamorphism transformation
  - On our bytecodes
  - On the original assembly code?

## Polymorphism ?

- It is impossible with classical VM

$\Rightarrow$ But we can provide such features with our bytecodes

# Virtual Machines

## Metamorphism

- Classical metamorphism transformation
  - On our bytecodes
  - On the original assembly code?

## Polymorphism ?

- It is impossible with classical VM

$\Rightarrow$ But we can provide such features with our bytecodes

# Virtual Machines

## Metamorphism

- Classical metamorphism transformation
  - On our bytecodes
  - On the original assembly code?

## Polymorphism ?

- It is impossible with classical VM

⇒ But we can provide such features with our bytecodes

# Virtual Machines

## Metamorphism

- Classical metamorphism transformation
  - On our bytecodes
  - On the original assembly code?

## Polymorphism ?

- It is impossible with classical VM

$\Rightarrow$ But we can provide such features with our bytecodes

# Current section

1 Introduction

2 Obfuscation

3 Virtual Machines

4 Android/Java appplications

5 Conclusion


# Mobiles ?

## why ?

- The number of Mobile Apps is growing quickly
- But there is no real protection/obfuscation on java bytecodes
- ie: the android developper blog recommends to use ProGuard

# Mobile Format

## Dex/Class ?

- JVM : .class format (classic java applications)
- DalvikVM : .dex format (classic android applications)
- .dex is obtained by the transformation of .class format

# Mobiles Format

## JVM

- stack-based
- mainly for Java language
- JIT with HotSpot

## DalvikVM

- register-based
- "an uncompressed .dex file is typically a few percent smaller in size than a compressed .jar (Java Archive) derived from the same .class files" wikipedia
- since Android 2.2, JIT !

# Mobile Format

## Which format?

- Both are interesting
- But it is more interesting to work at the level of the .class format

⇒ And we can work between the end of the compilation and the transformation (easy with Ant)

⇒ So it is not only for mobile devices

# Mobile Format

## Which format?

- Both are interesting
- But it is more interesting to work at the level of the .class format

⇒ And we can work between the end of the compilation and the transformation (easy with Ant)

⇒ So it is not only for mobile devices

# Mobile Format

## Which format?

- Both are interesting
- But it is more interesting to work at the level of the .class format

⇒ And we can work between the end of the compilation and the transformation (easy with Ant)

⇒ So it is not only for mobile devices

# Mobile Format

## JVM Format

- Header (magic, minor_version, major_version, constant_pool_count, constant_pool, access_flags, this_class, super_class, interfaces_count, interfaces, fields_count, fields, methods_count, methods, attributes_count)

# Mobile Format

## JVM Format - Constant Pool

- Description of classes, fields, methods, interfaces, strings, integers, floats ....
- Methodref_info(tag=10, class_index=33, name_and_type_index=51)
- – Class_info(tag=7, name_index=75)
- — Utf8_info(tag=1, length=16) Utf8_next(bytes='java/lang/Object')
- – NameAndType_info(tag=12, name_index=40, descriptor_index=41)
- — Utf8_info(tag=1, length=6) Utf8_next(bytes='<init>')
- — Utf8_info(tag=1, length=3) Utf8_next(bytes='()V')

# Mobile Format

## JVM Format - Constant Pool

- Description of classes, fields, methods, interfaces, strings, integers, floats ....
- Methodref_info(tag=10, class_index=33, name_and_type_index=51)
  - – Class_info(tag=7, name_index=75)
  - — Utf8_info(tag=1, length=16) Utf8_next(bytes='java/lang/Object')
  - – NameAndType_info(tag=12, name_index=40, descriptor_index=41)
  - — Utf8_info(tag=1, length=6) Utf8_next(bytes='<init>')
  - — Utf8_info(tag=1, length=3) Utf8_next(bytes='()V')

# Mobile Format

## JVM Format - Constant Pool

- Description of classes, fields, methods, interfaces, strings, integers, floats ....
- Methodref_info(tag=10, class_index=33, name_and_type_index=51)
- – Class_info(tag=7, name_index=75)
- — Utf8_info(tag=1, length=16) Utf8_next(bytes='java/lang/Object')
- – NameAndType_info(tag=12, name_index=40, descriptor_index=41)
- — Utf8_info(tag=1, length=6) Utf8_next(bytes='<init>')
- — Utf8_info(tag=1, length=3) Utf8_next(bytes='()V')

# Mobile Format

## JVM Format - Constant Pool

- Description of classes, fields, methods, interfaces, strings, integers, floats ....
- Methodref_info(tag=10, class_index=33, name_and_type_index=51)
- – Class_info(tag=7, name_index=75)
- — Utf8_info(tag=1, length=16) Utf8_next(bytes='java/lang/Object')
- – NameAndType_info(tag=12, name_index=40, descriptor_index=41)
- — Utf8_info(tag=1, length=6) Utf8_next(bytes='<init>')
- — Utf8_info(tag=1, length=3) Utf8_next(bytes='()V')

# Mobiles Format

## JVM Format - Constant Pool

- Description of classes, fields, methods, interfaces, strings, integers, floats ....
- String_info(tag=8, string_index=59)
- – Utf8_info(tag=1, length=4)
- – Utf8_next(bytes='IDX ')

# Mobiles Format

## JVM Format - Constant Pool

- Description of classes, fields, methods, interfaces, strings, integers, floats ....
- String_info(tag=8, string_index=59)
  - Utf8_info(tag=1, length=4)
  - Utf8_next(bytes='IDX ')

# Mobiles Format

## JVM Format - Constant Pool

- Description of classes, fields, methods, interfaces, strings, integers, floats ....
- String_info(tag=8, string_index=59)
- – Utf8_info(tag=1, length=4)
- – Utf8_next(bytes='IDX ')

# Mobile Format

## JVM Format - Field/Method Pool

- Described precisely a field/method
- MethodInfo(access_flags=0, name_index=40, descriptor_index=41, attributes_count=1) <init> ()V
- AttributeInfo(attribute_name_index=42, attribute_length=29) Code
- LOW(max_stack=1, max_locals=1, code_length=5)
- 0 0 aload_0
- 1 1 invokespecial ['java/lang/Object', '<init>', '()V']
- 2 4 return

# Mobile Format

## JVM Format - Field/Method Pool

- Described precisely a field/method
- MethodInfo(access_flags=0, name_index=40, descriptor_index=41, attributes_count=1) <init> ()V
- AttributeInfo(attribute_name_index=42, attribute_length=29) Code
- LOW(max_stack=1, max_locals=1, code_length=5)
- 0 0 aload_0
- 1 1 invokespecial ['java/lang/Object', '<init>', '()V']
- 2 4 return

# Mobile Format

## JVM Format - Field/Method Pool

- Described precisely a field/method
- MethodInfo(access_flags=0, name_index=40, descriptor_index=41, attributes_count=1) <init> ()V
- AttributeInfo(attribute_name_index=42, attribute_length=29) Code
- LOW(max_stack=1, max_locals=1, code_length=5)
- 0 0 aload_0
- 1 1 invokespecial ['java/lang/Object', '<init>', '()V']
- 2 4 return

# Mobile Format

## JVM Format - Field/Method Pool

- Described precisely a field/method
- MethodInfo(access_flags=0, name_index=40, descriptor_index=41, attributes_count=1) <init> ()V
- AttributeInfo(attribute_name_index=42, attribute_length=29) Code
- LOW(max_stack=1, max_locals=1, code_length=5)
- 0 0 aload_0
- 1 1 invokespecial ['java/lang/Object', '<init>', '()V']
- 2 4 return

# Modify .class format

## Add and remove string

- Insert a new CONSTANT_Utf8 into the Constant Pool
  - ('>BH', namedtuple( "CONSTANT_Utf8_info", "tag length" ) + bytes

## Modify the name of a field or a method

- FieldInfo or MethodInfo
  - Change bytes into CONSTANT_Utf8 at name_index

# Modify .class format

## Add and remove string

- Insert a new CONSTANT_Utf8 into the Constant Pool
  - (’>BH’, namedtuple( "CONSTANT_Utf8_info", "tag length" ) + bytes

## Modify the name of a field or a method

- FieldInfo or MethodInfo
  - Change bytes into CONSTANT_Utf8 at name_index

# Modify .class format

## Add/Remove instructions into MethodInfo

- Insert new instructions into a human readable format
- Apply relocations on specific instructions (goto*, if*, jsr*)
- Modify code_length in CodeAttribute and attribute_length in AttributeInfo

## Example

- j = jvm.JVMFormat( open(TEST).read() )
- code = j.get_method("test")[0].get_code()

⇒ code.insert_at( 13, [ "aload_0" ] )

⇒ code.insert_at( 14, [ "invokevirtual", "toto", "(I)I" ] )

# Modify .class format

## Add/Remove instructions into MethodInfo

- Insert new instructions into a human readable format
- Apply relocations on specific instructions (goto*, if*, jsr*)
- Modify code_length in CodeAttribute and attribute_length in AttributeInfo

## Example

- j = jvm.JVMFormat( open(TEST).read() )
- code = j.get_method("test")[0].get_code()

⇒ code.insert_at( 13, [ "aload_0" ] )

⇒ code.insert_at( 14, [ "invokevirtual", "toto", "(I)I" ] )

# Modify .class format

## Insert new "craft" method

- Create new objects :
  - MethodInfo (information + code)
  - MethodRef (class + name_type) + NameAndType (name + type)
- Add MethodInfo into the Method Pool

## Example

- j = jvm.JVMFormat( open(TEST).read() )
- j.insert_craft_method( "toto", [ "ACC_PUBLIC", "[B", "[B" ], [ [ "aconst_null" ], [ "areturn" ] ] )

# Modify .class format

## Insert new "craft" method

- Create new objects :
  - MethodInfo (information + code)
  - MethodRef (class + name_type) + NameAndType (name + type)
- Add MethodInfo into the Method Pool

## Example

- j = jvm.JVMFormat( open(TEST).read() )
- j.insert_craft_method( "toto", [ "ACC_PUBLIC", "[B", "[B" ], [ [ "aconst_null" ], [ "areturn" ] ] )

# Modify .class format

## Insert new "craft" method

- Create new objects :
  - MethodInfo (information + code)
  - MethodRef (class + name_type) + NameAndType (name + type)
- Add MethodInfo into the Method Pool

## But ...

⇒ Interesting but it’s very difficult to insert advanced instructions

# Modify .class format

## Insert the new method from another .class file

- Same as craft method
- Fix attributes
- Patch invokes*, ldc*, anewarray, getstatic, new, ...

## Example

- j = jvm.JVMFormat( open(TEST).read() )
- j2 = jvm.JVMFormat( open(TEST_REF).read() )

⇒ j.insert_direct_method( "toto2",
j2.get_method("test3")[0] )

# Modify .class format

## Insert the new method from another .class file

- Same as craft method
- Fix attributes
- Patch invokes*, ldc*, anewarray, getstatic, new, ...

## Example

- j = jvm.JVMFormat( open(TEST).read() )
- j2 = jvm.JVMFormat( open(TEST_REF).read() )

⇒ j.insert_direct_method( "toto2",
j2.get_method("test3")[0] )

# Modify .class format

## Insert the new method from another .class file

- Same as craft method
- Fix attributes
- Patch invokes*, ldc*, anewarray, getstatic, new, ...

## Example

- j = jvm.JVMFormat( open(TEST).read() )
- j2 = jvm.JVMFormat( open(TEST_REF).read() )

⇒ j.insert_direct_method( "toto2",
j2.get_method("test3")[0] )

# Transform simple integers into VM

## Transformation

- Get manipulation of basic constant integers, like :
  - bipush 0x10

⇒ Create the VM


# Transform simple integers into VM

## Transformation

- Get manipulation of basic constant integers, like :
  - bipush 0x10

$\Rightarrow$ Create the VM


# Transform simple integers into VM

## Mathematical formulas

- Transform a simple integer into a reversible mathematical formula
  - X0 = 16 ; X1 = X0 - I1 ; X2 = X1 + I2 ; X3 = I3 - X2
  - X3 = I4 ; X2 = I3 + X3 ; X1 = X2 - I2 ; X0 = X1 + I1

## PRNG : Linear congruential generator

- X0 = 16; 'A': 1, 'GERME': 0, 'C': 5, 'M': 29, 'ITER': 9
- GERME = (A * GERME + C) % M
- 5 10 15 20 25 1 6 11 16 : X0

## ...

- Use of SAT formulas ?

# Transform simple integers into VM

## Mathematical formulas

- Transform a simple integer into a reversible mathematical formula
- X0 = 16 ; X1 = X0 - I1 ; X2 = X1 + I2 ; X3 = I3 - X2
- X3 = I4 ; X2 = I3 + X3 ; X1 = X2 - I2 ; X0 = X1 + I1

## PRNG : Linear congruential generator

- X0 = 16; 'A': 1, 'GERME': 0, 'C': 5, 'M': 29, 'ITER': 9
- GERME = (A * GERME + C) % M
- 5 10 15 20 25 1 6 11 16 : X0

## ...

- Use of SAT formulas ?

# Transform simple integers into VM

## Mathematical formulas

- Transform a simple integer into a reversible mathematical formula
- X0 = 16 ; X1 = X0 - I1 ; X2 = X1 + I2 ; X3 = I3 - X2
- X3 = I4 ; X2 = I3 + X3 ; X1 = X2 - I2 ; X0 = X1 + I1

## PRNG : Linear congruential generator

- X0 = 16; 'A': 1, 'GERME': 0, 'C': 5, 'M': 29, 'ITER': 9
- GERME = (A * GERME + C) % M
- 5 10 15 20 25 1 6 11 16 : X0

## ...

- Use of SAT formulas ?

# Transform simple integers into VM

## Mathematical formulas

- Transform a simple integer into a reversible mathematical formula
- X0 = 16 ; X1 = X0 - l1 ; X2 = X1 + l2 ; X3 = l3 - X2
- X3 = l4 ; X2 = l3 + X3 ; X1 = X2 - l2 ; X0 = X1 + l1

## PRNG : Linear congruential generator

- X0 = 16; 'A': 1, 'GERME': 0, 'C': 5, 'M': 29, 'ITER': 9
- GERME = (A * GERME + C) % M
- 5 10 15 20 25 1 6 11 16 : X0

## ...

- Use of SAT formulas ?

# Transform simple integers into VM

## Mathematical formulas

- Transform a simple integer into a reversible mathematical formula
- X0 = 16 ; X1 = X0 - l1 ; X2 = X1 + l2 ; X3 = l3 - X2
- X3 = l4 ; X2 = l3 + X3 ; X1 = X2 - l2 ; X0 = X1 + l1

## PRNG : Linear congruential generator

- X0 = 16; ’A’: 1, ’GERME’: 0, ’C’: 5, ’M’: 29, ’ITER’: 9
- GERME = (A * GERME + C) % M
- 5 10 15 20 25 1 6 11 16 : X0

## ...

- Use of SAT formulas ?

# Transform simple integers into VM

## Transformation

- Transform the previous operation into REIL
- STR( l4, , x3)
- ADD( l3, X3, X2 )
- SUB( X2, l2, X1 )
- ADD( X1, l1, X0 )
- Apply metamorphism/polymorphism transformations
- Apply CFG obfuscation
- Return a specific register (X0) to have the result + specific parameters values

# Transform simple integers into VM

## Transformation

- Transform the previous operation into REIL
- STR( l4, , x3)
- ADD( l3, X3, X2 )
- SUB( X2, l2, X1 )
- ADD( X1, l1, X0 )
- Apply metamorphism/polymorphism transformations
- Apply CFG obfuscation
- Return a specific register (X0) to have the result + specific parameters values

# Transform simple integers into VM

## Transformation

- Transform the previous operation into REIL
- STR( l4, , x3)
- ADD( l3, X3, X2 )
- SUB( X2, l2, X1 )
- ADD( X1, l1, X0 )
- Apply metamorphism/polymorphism transformations
- Apply CFG obfuscation
- Return a specific register (X0) to have the result + specific parameters values

# Transform simple integers into VM

## Transformation

- Transform the previous operation into REIL
- STR( l4, , x3)
- ADD( l3, X3, X2 )
- SUB( X2, l2, X1 )
- ADD( X1, l1, X0 )
- Apply metamorphism/polymorphism transformations
- Apply CFG obfuscation
- Return a specific register (X0) to have the result + specific parameters values

# Transform simple integers into VM

## Transformation

- Transform the previous operation into REIL
- STR( l4, , x3)
- ADD( l3, X3, X2 )
- SUB( X2, l2, X1 )
- ADD( X1, l1, X0 )
- Apply metamorphism/polymorphism transformations
- Apply CFG obfuscation
- Return a specific register (X0) to have the result + specific parameters values

# Transform simple integers into VM

## Transformation

- Transform the REIL bytecodes into JAVA :
  - Format
  - Registers
  - Instructions
- Transform each REIL instructions into JAVA
- Simple loop which interprets each bytecode

# Transform simple integers into VM

## Transformation

- Transform the REIL bytecodes into JAVA :
  - Format
  - Registers
  - Instructions
- Transform each REIL instructions into JAVA
- Simple loop which interprets each bytecode

# Transform simple integers into VM

## Transformation

- Transform the REIL bytecodes into JAVA :
  - Format
  - Registers
  - Instructions
- Transform each REIL instructions into JAVA
- Simple loop which interprets each bytecode

# Insertion of the VM

## Insertion

- Replace the original instructions with a simple call :
  - ⇒ [ "aload_0" ] + [ "invokevirtual", "Test1", "vm", descriptor ]
- Insert the new method
- Save the new file

# Current section

1 Introduction

2 Obfuscation

3 Virtual Machines

4 Android/Java appplications

5 Conclusion


# Conclusion

## The end ...

- Framework + tools + demos available at http://code.google.com/p/androguard/
- Full python code
- It's mainly focus on software protection, but you can do other things ...
  - JVM/DalvikVM format access
  - Modification ...
  - ... Save !

## !

- Thanks to Hack.lu 2010
- Questions?

# Conclusion

## The end ...

- Framework + tools + demos available at http://code.google.com/p/androguard/
- Full python code
- It's mainly focus on software protection, but you can do other things ...
  - JVM/DalvikVM format access
  - Modification ...
  - ... Save !

## !

- Thanks to Hack.lu 2010
- Questions?

# Conclusion

## The end ...

- Framework + tools + demos available at http://code.google.com/p/androguard/
- Full python code
- It’s mainly focus on software protection, but you can do other things ...
  - JVM/DalvikVM format access
  - Modification ...
  - ... Save !

## !

- Thanks to Hack.lu 2010
- Questions?

# Conclusion

## The end ...

- Framework + tools + demos available at http://code.google.com/p/androguard/
- Full python code
- It's mainly focus on software protection, but you can do other things ...
  - JVM/DalvikVM format access
  - Modification ...
  - ... Save !

## !

- Thanks to Hack.lu 2010
- Questions?