ANTI-REVERSING

OXFORD BROOKES UNIVERSITY

COMP6016

Malware Analysis

Dr Muhammad Hilmi Kamarudin

OXFORD BROOKES UNIVERSITY

Learning outcomes

By the end of today's lecture and practical you should :-

- Understand the anti reversing techniques
- Understand the use of packers and obfuscation technique to avoid detection
- Understand how the anti debugger, anti disassembly and obfuscation work to delay malware analyst

OXFORD BROOKES UNIVERSITY

ANTI-REVERSING

- There are many cases where it is beneficial to create software that is immune to reversing.
- Some applications have a special need for anti-reversing measures. Eg: Copy protection/ DRM software.
- Some software development platforms really necessitate some form of anti-reversing measures.

BASIC APPROACHES TO ANTI-REVERSING (1)

OXFORD BROOKES UNIVERSITY

- **Eliminating Symbolic Information**
  - Eliminate any obvious textual information from the program.
- **Embedding Anti-debugger Code**
  - To prevent dynamic analysis
  - The idea is to have the program intentionally perform operations that would somehow damage or disable a debugger, if one is attached.
- **Obfuscating the Program**
  - Obfuscation is a generic name for a number of techniques that are aimed at reducing the program's vulnerability to any kind of static analysis

ELIMINATING SYMBOLIC INFORMATION

OXFORD BROOKES UNIVERSITY

- It is generally a nonissue in conventional compiler-based languages such as C and C++ because symbolic information is not usually included in release builds
- One area where even compiler-based programs can contain a little bit of symbolic information is the import and export tables.
- Big issue with most bytecode-based languages.

CODE ENCRYPTION

OXFORD BROOKES UNIVERSITY

- Encryption of program code is a common method for preventing static analysis.
- Executable compression is frequently used to deter reverse engineering or to obfuscate the contents of the executable
- Can make the process more costly.
- Additionally, the program must decrypt the code in runtime before it is executed

PE FILE

- No encryption

.RDATA SECTION

```
00443000 6A7570217465720045 6E746572207061 jup!ter.Enter pa
00443010 7373776F72643A2000 5B4F4B5D204163 ssword: .[OK] Ac
00443020 6365737320677261 6E7465642E005B45 cess granted..[E
00443030 72726F725D204163 6365737320646 56E rror] Access den
00443040 6965642E000000000 000000000000000 ied.............
```

- Encrypted with simple substitution

.RDATA SECTION

```
00445000 3532374238313732374632443744 3645 527B81727F2D7D6E
00445010 3830383038343743374637313437 3244 8080847C7F71472D
00445020 0037373832374432453831373237 4600 .77827D2E81727F.
00445030 3638354335383641324434453730 3730 685C586A2D4E7070
00445040 3732383038303244373437463645 3742 7280802D747F6E7B
00445050 3831373237313342000000003638 3532 8172713B....6852
00445060 3746374637433746364132443445 3730 7F7F7C7F6A2D4E70
00445070 3730373238303830324437313732 3742 707280802D71727B
00445080 3736373237313342000000000000 0000 7672713B........
```

PACKERS (1)

OXFORD BROOKES UNIVERSITY

- Packing programs, known as packers, have become extremely popular with malware writers
- They transform an executable to create a new executable that stores the transformed executable as data and contains an unpacking stub that is called by the OS
- With packed programs, the unpacking stub is loaded by the OS, and then the unpacking stub loads the original program.
- If you attempt to perform static analysis on the packed program, you will be analyzing the stub, not the original program.

PACKERS (2)

OXFORD BROOKES UNIVERSITY

- Sometimes, packed malware can be unpacked automatically by an existing program, but more often it must be unpacked manually

- OllyDump, a plug-in for OllyDbg, has two good features for unpacking: It can dump the memory of the current process, and it can search for the Original Entry Point (OEP) for a packed executable.

ANTI-DEBUGGING (1)- PMA-351

OXFORD BROOKES UNIVERSITY

- Breakpoints and Step-in/Stepover are key aspects of run time debugging
- Software breakpoint
  - int 3 instruction add to beginning of the line
- Hardware breakpoint
  - the processor simply knows to break when a specific memory address is accessed.
- Step
  - Stepping through code means that each instruction is executed individually and that control is returned to the debugger after each program instruction is executed.
  - interrupt number 1 - *singlestep* interrupt.

ANTI-DEBUGGING (2)

OXFORD BROOKES UNIVERSITY

IsDebuggerPresent API

- *IsDebuggerPresent* is a Windows API that can be used as a trivial tool for detecting user-mode debuggers such as OllyDbg or WinDbg
- The function accesses the current process's Process Environment Block (PEB) to determine whether a user-mode debugger is attached.
- A program can call this API and terminate if it returns TRUE
- However this call can be found at debugging and bypassed

```
mov     eax,fs:[00000018]
mov     eax,[eax+0x30]
cmp     byte ptr [eax+0x2], 0
je      RunProgram
; Inconspicuously terminate program here...
```

- Code retrieves offset +30 from the Thread Environment Block (TEB) data structure, which points to the current process's PEB.
- Then the sequence reads a byte at offset +2, which indicates whether a debugger is present or not.

ANTI-DEBUGGING (3)

OXFORD BROOKES UNIVERSITY

Some debuggers modify the code… This would catch said modifications!

Code Checksums

- Computing checksums on code fragments can make for a fairly powerful anti-debugging technique
- The general idea is to pre-calculate a checksum for functions within the program (this trick could be reserved for particularly sensitive functions

ANTI-DEBUGGING (4)

OXFORD BROOKES UNIVERSITY

NtQuerySystemInformation API

- The *NtQuerySystemInformation* native API can be used to determine if a kernel debugger is attached to the system. This function supports several different types of information requests.
- The *systemKernelDebuggerInformation* request code can obtain information from the kernel on whether a kernel debugger is currently attached to the system.

```
typedef struct _SYSTEM_KERNEL_DEBUGGER_INFORMATION {
    BOOLEAN DebuggerEnabled;
    BOOLEAN DebuggerNotPresent;
} SYSTEM_KERNEL_DEBUGGER_INFORMATION,
*PSYSTEM_KERNEL_DEBUGGER_INFORMATION;
```

- To determine whether a kernel debugger is attached to the system, the DebuggerEnabled should be checked.

ANTI-DISASSEMBLY (1)-PMA 327

OXFORD BROOKES UNIVERSITY

- Disassembling code is usually done using either **linear sweep** or **recursive traversal**.
- Linear sweep
  - Iterate over a block of code, disassembling one instruction at a time linearly
  - Decode blindly from start to end, ignores flow-control instructions that cause only a part of the buffer to execute
- Recursive traversal or Flow-oriented
  - takes control flow into account.

DISASSEMBLER/DEBUGGER NAME	DISSASEMBLY METHOD
OllyDbg	Recursive traversal
NuMega SoftICE	Linear sweep
Microsoft WinDbg	Linear sweep
IDA Pro	Recursive traversal
PEBrowse Professional (including the interactive version)	Recursive traversal

ANTI-DISASSEMBLY (2)

OXFORD BROOKES UNIVERSITY

```
                jmp     short near ptr loc_2+1
; ---------------------------------------------------------------------------

loc_2:                                  ; CODE XREF: seg000:00000000j
                call    near ptr 15FF2A71h ❶
                or      [ecx], dl
                inc     eax
; ---------------------------------------------------------------------------
                db    0
```

- This fragment of code was disassembled using the linear-disassembly technique, and the result is inaccurate.
- The target of the jmp instructions is invalid because it falls in the middle of the next instruction.

Fragment 1

```
                jmp     short loc_3
; ---------------------------------------------------------------------------
                db 0E8h
; ---------------------------------------------------------------------------

loc_3:                                  ; CODE XREF: seg000:00000000j
                push    2Ah
                call    Sleep ❶
```

- This fragment was disassembled with a flow-oriented disassembler,
- This fragment reveals a different sequence of assembly mnemonics, and it appears to be more informative.
- Here, we see a call to the API function Sleep.
- The target of the first jmp instruction is now properly represented, and we can see that it jumps to a push instruction followed by the call to Sleep.
- The byte on the third line of this example is 0xE8, but this byte is not executed by the program because the jmp instruction skips over it.

Fragment 2

ANTI-DISASSEMBLY (3)

OXFORD BROOKES UNIVERSITY

- Anti-disassembly techniques work by taking advantage of the assumptions and limitations of disassemblers.
- Eg: disassemblers can only represent each byte of a program as part of one instruction at a time. If the disassembler is tricked into disassembling at the wrong offset, a valid instruction could be hidden from view.
- In processor architectures that use variable-length instructions, such as IA-32 processors, it is possible to trick disassemblers into incorrectly treating invalid data as the beginning of an instruction.
- When you consider a recursive traversal disassembler, you can see that in order to confuse it into incorrectly disassembling data you'll need to feed it an opaque predicate.

ANTI-DISASSEMBLY TECHNIQUES (1)

OXFORD BROOKES UNIVERSITY

- Fake conditionals
  - Jump instructions with the Same Target: Back-to-back conditional jumps with the same target
    - if a jz loc_512 is followed by jnz loc_512,
    - the location loc_512 will always be jumped to.
    - The combination of jz with jnz is, in effect, an unconditional jmp, but the disassembler doesn't recognize it as such because it only disassembles one instruction at a time.
    - When the disassembler encounters the jnz, it continues disassembling the false branch of this instruction, despite the fact that it will never be executed in practice.

ANTI-DISASSEMBLY TECHNIQUES (2)

OXFORD BROOKES UNIVERSITY

- Jump instruction with a constant condition

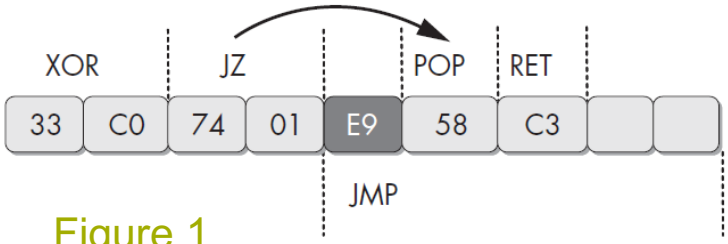


Figure 1

```
33 C0                   xor     eax, eax
74 01                   jz      short near ptr loc_4011C4+1
        loc_4011C4:                             ; CODE XREF: 004011C2j
                                                ; DATA XREF: .rdata:004020ACo
E9 58 C3 68 94          jmp     near ptr 94A8D521h
```

Figure 2

```
33 C0                   xor     eax, eax
74 01                   jz      short near ptr loc_4011C5
        ; ---------------------------------------------------------------------
E9                      db 0E9h
        ; ---------------------------------------------------------------------
        loc_4011C5:                             ; CODE XREF: 004011C2j
                                                ; DATA XREF: .rdata:004020AC
58                      pop     eax
C3                      retn
```

Figure 3

- This code begins with the instruction xor eax, eax.
- This instruction will set the EAX register to zero and, as a byproduct, set the zero flag.
- The next instruction is a conditional jump that will jump if the zero flag is set.
- In reality, this is not conditional at all, since we can guarantee that the zero flag will always be set at this point in the program.
- The disassembler will process the false branch first, which will produce conflicting code with the true branch, and since it processed the false branch first, it trusts that branch more.
- E9 is the opcode for a 5-byte jmp instruction,
- In each case, by tricking the disassembler into disassembling this location, the 4 bytes following this opcode are effectively hidden from view

OXFORD BROOKES UNIVERSITY

Code Obfuscations

WHAT IS OBFUSCATION?

OXFORD BROOKES UNIVERSITY

- **Literal Meaning of Obfuscation:**
  - Obfuscate – “to make obscure, unclear, or unintelligible.”
  - The word ‘obfuscation' refers to the concept of concealing the meaning of communication by making it more confusing and harder to interpret.

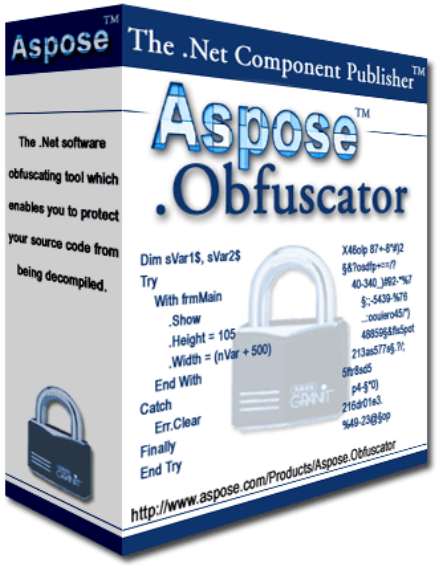
- **Code Obfuscation:**
  - Code obfuscation is the generation or alteration of source code and/or object code in such a way that it is easy for the computer to comprehend but considerably difficult to reverse engineer.
  - Alter code so as to confuse reverse engineer, but preserve functionality
  - Behavior preserving transformations on code that preserve function but reduce readability or understandability

OFF THE SHELF OBFUSCATORS

OXFORD BROOKES UNIVERSITY

dotfuscator

ProGuard



dasho

CODE ECLIPSE

HOW CAN OBFUSCATION HELP

OXFORD BROOKES UNIVERSITY

- Types of Obfuscation:
  - Code Structure Obfuscation
  - Data Obfuscation
  - Control Obfuscation
  - Preventive Obfuscation

- Effects of Obfuscation on Code:
  - Code logic doesn't change
  - Decreases footprint of code
  - Decreases performance (time)
  - Harder for developers during product cycle & possibly support

LIMITS TO OBFUSCATION

OXFORD BROOKES UNIVERSITY

- No obfuscation enough against extremely dedicated hackers
- Prevents against easy reverse engineering using tools
- Factor that prevent use of Obfuscation
  - Cost of Obfuscation
  - Execution time of code
  - High Program complexity

TYPES OF OBFUSCATIONS

OXFORD BROOKES UNIVERSITY

- Layout transformations
- Control transformations
- Aggregation transformations
- Data transformations

LAYOUT OBFUSCATION

OXFORD BROOKES UNIVERSITY

```
public class test1{
    private int term1;
    private int term2;
    private boolean areRelativelyPrime;

    public test1(int term1, int term2){
        this.term1=term1;
        this.term2=term2;
        areRelativelyPrime=areRelativelyPrime();
    }

    public static int gcd(int term1, int term2){
        int remainder;
        remainder=term1%term2;
        if (remainder==0){
            return term2;
        }
        else{
            return gcd(term2, remainder);
        }
    }

    private boolean areRelativelyPrime(){
        if (gcd(term1, term2)==1){
            return true;
        }
        else{
            return false;
        }
    }

    public static void main(String args[]) {
      test1 a=new test1(12, 19);
   }
}
```

```
public class a{
    private int a;
    private int b;
    private boolean c;

    public a(int a, int b){
        this.a=a;
        this.b=b;
        c=c();
    }

    public static int b(int a, int b){
        int c;
        c=a%b;
        if (c==0){
            return b;
        }
        else{
            return b(b, c);
        }
    }

    private boolean c(){
        if (b(a, b)==1)\{
            return true;
        }
        else{
            return false;
        }
    }

    public static void main(String args[]) {
      a b=new a(12, 19);
    }
}
```

CONTROL TRANSFORMATIONS

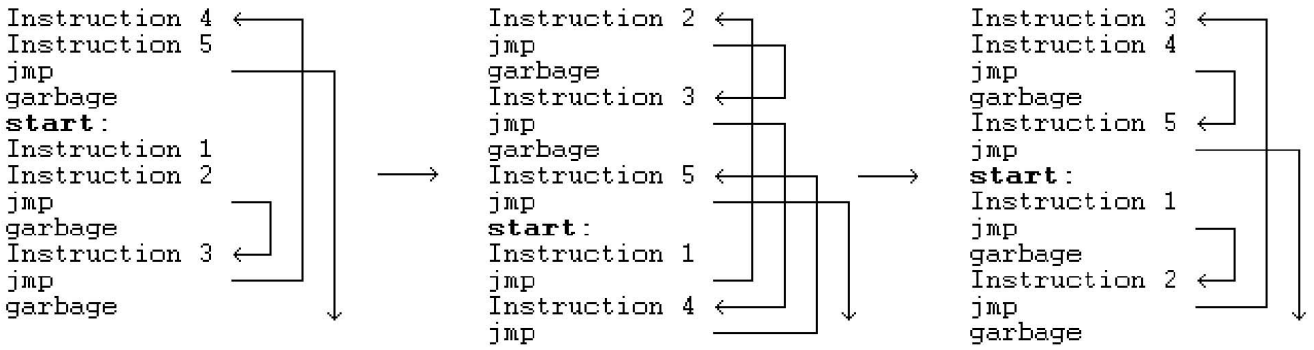
OXFORD BROOKES UNIVERSITY

- Insert dead or irrelevant code
- Extend loop conditions
- Convert a reducible to a non-reducible flow graph
- Redundant operands
- Parallelize code
- Replacing standard library routines by custom routines

CONTROL TRANSFORMATIONS

OXFORD BROOKES UNIVERSITY

- **Ordering:** This category performs reordering operations on statements, loops, and expressions to disturb the locality of related information.



- **Spurious Computations:** This type of obfuscation is done by modifying the real control-flow by adding spurious computation blocks.

OXFORD BROOKES UNIVERSITY

AGGREGATION TRANSFORMATIONS

- The original control-flow logic is disturbed by coalescing unrelated methods or splitting related methods.
- Inline and outline methods
- Interleave methods
- Clone methods

DATA OBFUSCATION

OXFORD BROOKES UNIVERSITY

- Change encoding
  - Pack variables into bigger variables
  - Pack variables into arrays
- Restructure arrays
- Convert static to procedural data
- Altering inheritance hierarchies
- Encryption of string literals
  - Strings are decrypted each time they are used using a bundled cipher

```
case __specifyUsername         : return  "Specify a username and press Enter: ";
case __specifyUsernameEdit     : return  "Specify a username and press Enter ({*}): ";
case __specifyVaultUsername    : return  "Specify vault username and press Enter: ";
case __specifyVaultPassword    : return  "Specify vault password and press Enter: ";
case __specifyVaultPassCurrent : return  "Specify current vault password and press Enter: ";
case __specifyVaultPassNew     : return  "Specify new vault password and press Enter: ";
```

```
case __specifyUsername         : DecryptMessageText("607D72707673862D6E2D8280727F7B6E7A722D6E7B712D7D7F7280
case __specifyUsernameEdit     : DecryptMessageText("607D72707673862D6E2D8280727F7B6E7A722D6E7B712D7D7F7280
case __specifyVaultUsername    : DecryptMessageText("607D72707673862D836E8279812D8280727F7B6E7A722D6E7B712D
case __specifyVaultPassword    : DecryptMessageText("607D72707673862D836E8279812D7D6E8080847C7F712D6E7B712D
case __specifyVaultPassCurrent : DecryptMessageText("607D72707673862D70827F7F727B812D836E8279812D7D6E808084
case __specifyVaultPassNew     : DecryptMessageText("607D72707673862D7B72842D836E8279812D7D6E8080847C7F712D
```

OXFORD BROOKES UNIVERSITY

Thank you