Andrea Di Sorbo, PhD student
Sicurezza delle Reti e dei Sistemi Software
CdLM in Ingegneria Informatica
Università degli Studi del Sannio
(disorbo@unisannio.it)

Metamorphic Malware

Implementation of a Metamorphic Engine

Metamorphic Malware

- Metamorphic malware are malicious software programs that have the ability to change their code as they propagate, through a set of transformation techniques.
- Metamorphic malware is rewritten with each iteration so that the succeeding version of the code is different from the preceding one.
- The code changes make it difficult for signature-based antivirus software programs to recognize that different iterations are the same malicious program.
- Metamorphic malware modifies the code structure without affecting the business logic.

Metamorphic vs Polymorphic

- Polymorphic malware tries to evade the signature-based detection through the encryption.
- A polymorphic virus might have:
  - a virus decryption routine (VDR);
  - an encrypted virus program body (EVB).
- When an infected application launches, the VDR decrypts the EVB back to its original form so the virus can perform its intended function.
- Once executed, the virus is re-encrypted with a new encryption key and added to another vulnerable host application.

Metamorphic vs Polymorphic

- The main limitation of the polymorphic techniques is that the decrypted code is essentially the same in each case thus memory based signature detection is possible.
- To overcome this limitation metamorphic malware has the ability to automatically recode itself each time it propagates. Thus the code changes at each iteration.

Activities of a metamorphic engine

- To change the structure of a malicious code, a metamorphic engine performs 5 main actions:
  1. **Locate own code** (locate the virus code)
  2. **Decode** (de-obfuscate the virus code)
  3. **Analyze** (in order to collect some useful information)
  4. **Transform** (through a set of heuristics)
  5. **Attach** (enclose the new generation of the malicious code in a host file)

Obfuscation techniques

1. **Garbage Code Insertion**: To change the byte sequence of viral code the metamorphic engine inserts instructions that have no effects. Win32/Evol virus (july,2000) exploits this technique by inserting junk code among main instructions.

Version 1

Binary Opcode	Assembly Code
C7060F000055	mov [esi], 5500000Fh
C746048BEC5151	mov[esi+0004],5151EC8Bh
String Signature:	C7060F000055C746048BEC5151

JUNK CODE

Version 2

Binary Opcode	Assembly Code
BF0F000055	mov edi,5500000Fh
893E	mov [esi],edi
5F	pop edi
52	push edx
B640	mov dh,40
BA8BEC5151	mov edx,5151EC8Bh
53	push ebx
8BDA	mov ebx,edx
895E04	mov [esi+0004],ebx
String Signature:	
BF0F000055893E5F52B640BA8BEC5151538BDA895E04	

Junk instructions examples

- Junk instructions have no effects on the code's functionalities.
- Junk instructions:
  - Instructions that are semantically similar to nop
  - Instructions sequences which momentarily modify the machine state without affecting the business logic

Instruction Rule	Operation
add Reg, 0	Reg ← Reg + 0
mov Reg, Reg	Reg ← Reg
or Reg, 0	Reg ← Reg | 0
and Reg, -1	Reg ← Reg & -1

Garbage Instructions	Comments
push cx pop cx	Before any effects, it returns the value to the register from stack
inc ax sub ax, 1	Value of ax remain unchanged

Obfuscation techniques

- **Register usage exchange:** This technique generates different versions of the same virus, each one using the same code but with different registers. The Win95.Regswap virus (December,1998) used this technique to create different variants of the virus.

Version 1

Binary Opcode	Assembly Code
5A	pop edx
BF04000000	mov edi,0004h
8BF5	mov esi,ebp
B80C000000	mov eax,000Ch
81C288000000	add edx,0088h
8B1A	mov ebx,[edx]
899C8618110000	mov [esi+eax*4+00001118],ebx

String Signature:

5ABF040000008BF5B80C00000081C2880000008B1A899C8618110000

Version 2

Binary Opcode	Assembly Code
58	pop eax
BB04000000	mov ebx,0004h
8BD5	mov edx,ebp
BF0C000000	mov edi,000Ch
81C088000000	add eax,0088h
8B30	mov esi,[eax]
89B4BA1811000	mov [edx+edi*4+00001118],esi

String Signature:

58BB040000008BD5BF0C00000081C0880000008B30 89B4BA18110000

Obfuscation techniques

3. **Instruction Replacement:** This method actually substitutes some instructions with their equivalent instructions in newer copies. This method is like using different synonyms in human language. Win95.Bistro used this technique.

Binary Opcode	Assembly Code
55	push ebp
54	push esp
5D	pop ebp
8B7608	mov esi, dword ptr [ebp + 08]
09F6	or esi, esi
743B	je 401045
8B7E0C	mov edi, dword ptr [ebp + 0c]
85FF	test edi, edi
7434	je 401045
28D2	sub edx, edx

String Signature:
55545D8B760809F6743B8B7E0C85FF74342 8D2

Binary Opcode	Assembly Code
55	push ebp
8BEC	mov ebp, esp
8B7608	mov esi, dword ptr [ebp + 08]
85F6	test esi, esi
743B	je 401045
8B7E0C	mov edi, dword ptr [ebp + 0c]
09FF	or edi, edi
7434	je 401045
31D2	xor edx, edx

String Signature:
558BEC8B760885F6743B8B7E0C09FF743431D2

Examples of instruction replacements

- Some examples of readily realizable replacements:
  - Replace register moves with push/pop sequences

```
movl %eax, %ebx  →  pushl %eax
                    popl %ebx
```

  - xor/sub replacement

```
xorl %edx, %edx  →  subl %edx,%edx
```

  - or/test replacement

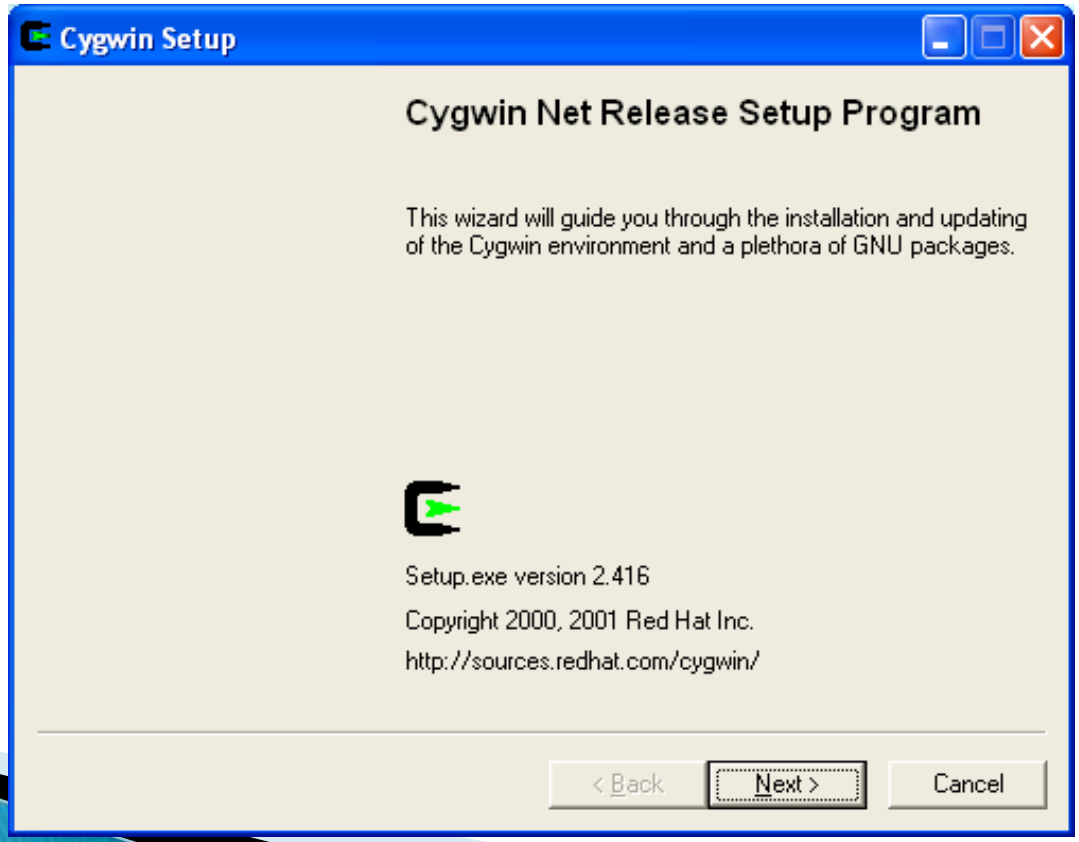
```
testl %eax, %eax  →  orl %eax,%eax
```

  - add/sub (with complement operand) replacement

```
addl $2, %eax  →  subl $-2, %eax
```

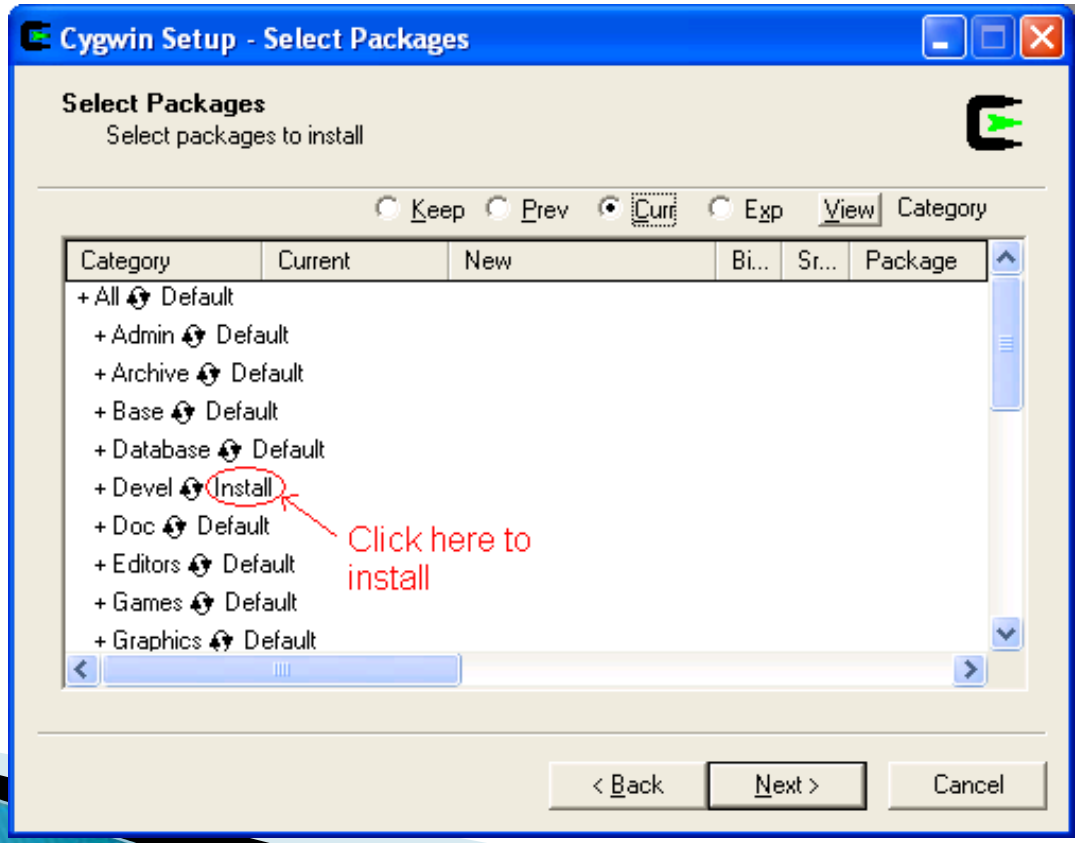
Preliminary operations

- If you are using Windows systems:
  1. Download and run setup.exe from: http://www.cygwin.com



Preliminary operations

2. Click "next" to reach the "Select Packages" screen
3. In this screen select "Install" for the entry "Devel"



Exercise 1

- Implement a metamorphic engine that:
  1. Takes in input a file in assembly code (hello.s)
  2. Returns in output a new variant (hello_mutation.s) of the input file obtained through operations of junk code insertion (Each execution may produce a different variant of the original file)
- Recompile the resulting file and verify that the two executions (hello.s and hello_mutation.s) are equivalent.

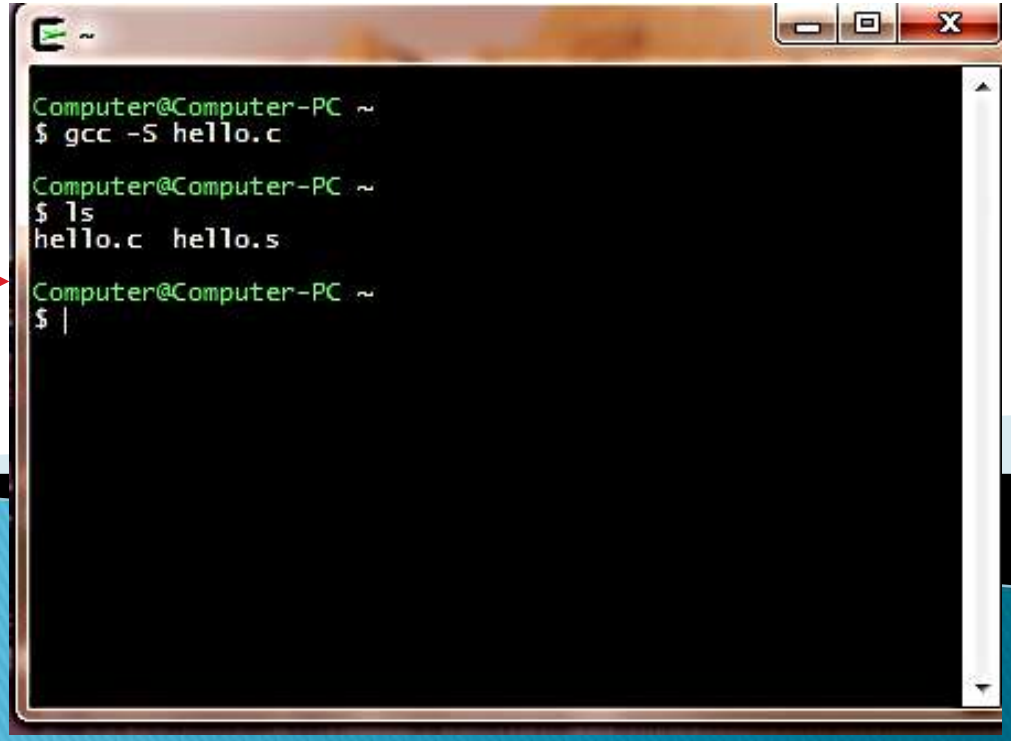
hello.c

```c
/* hello.c */
#include <stdio.h>

int main(){
    printf("Hello world!\n");

}
```

Cygwin



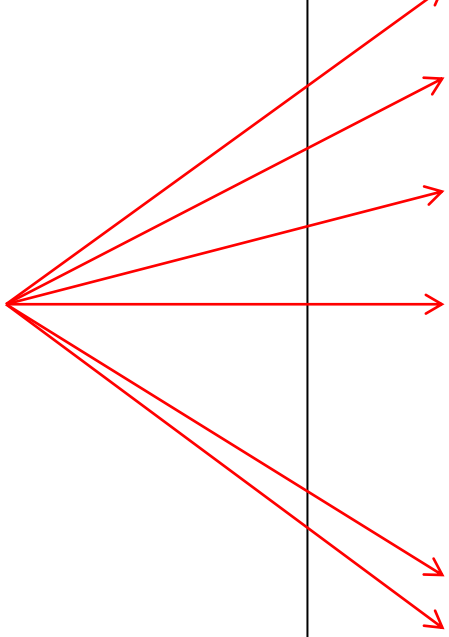
hello.s

```
        .file "hello.c"
        .def    ___main;    .scl    2;      .type   32;     .endef
        .section .rdata,"dr"
LC0:
        .ascii "Hello world!\0"
        .text
        .globl  _main
        .def    _main;      .scl    2;      .type   32;     .endef
_main:
LFB7:
        .cfi_startproc
        pushl %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $16, %esp
        call    ___main
        movl    $LC0, (%esp)
        call    _puts
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
LFE7:
        .ident      "GCC: (GNU) 4.8.3"
        .def    _puts;      .scl    2;      .type   32;     .endef
```

Target Code. In this code block we can apply obfuscation techniques

hello_mutation.s

Automatic generated code



```
_main:
LFB7:
        orl   %ecx, %ecx # garbage instruction
        andl  %eax, %eax # garbage instruction
        .cfi_startproc
        movl  %edi, %edi # garbage instruction
        andl  $1, %esi # garbage instruction
        andl  $1, %esi # garbage instruction
        pushl %ebp
        .cfi_def_cfa_offset 8
        movl  %eax, %eax # garbage instruction
        .cfi_offset 5, -8
        movl  %edi, %edi # garbage instruction
        subl  $0, %ecx # garbage instruction
        movl  %eax, %eax # garbage instruction
        movl  %esp, %ebp
        .cfi_def_cfa_register 5
        andl  $-16, %esp
        subl  $16, %esp
        call  ___main
        movl  $LC0, (%esp)
        call  _puts
        andl  %eax, %eax # garbage instruction
        leave
        movl  %edi, %edi # garbage instruction
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
LFE7:
        .ident      "GCC: (GNU) 4.8.3"
        .def  _puts;      .scl  2;     .type 32;    .endef
```

Recompiling and executing hello_mutation.s