



Project UNIX

Pestilence

42 Staff pedago@staff.42.fr

Summary: AND HIS NAME IS



Version: 3

Contents

I	Preamble	2
II	Introduction	10
III	Objectives	11
IV	Mandatory part	12
V	Example of use	14
VI	Bonus part	17
VII	Submission and peer-evaluation	18

Chapter I

Preamble

Because a little culture never hurt no one, here is what the CTRL+C management looks like on MS-DOS 2.0:

```

;
; ^C status routines for MSDOS
;

INCLUDE DOSSEG.ASM

CODE    SEGMENT BYTE PUBLIC  'CODE'
        ASSUME  SS:DOSGROUP,CS:DOSGROUP

.xlist
.xcref
INCLUDE DOSSYM.ASM
INCLUDE DEVSYM.ASM
.cref
.list

        i_need  DevIOBuf,BYTE
        i_need  DidCTRLC,BYTE
        i_need  INDOS,BYTE
        i_need  DSKSTCOM,BYTE
        i_need  DSKSTCALL,BYTE
        i_need  DSKSTST,WORD
        i_need  BCON,DWORD
        i_need  DSKCHRET,BYTE
        i_need  DSKSTCNT,WORD
        i_need  IDLEINT,BYTE
        i_need  CONSWAP,BYTE
        i_need  user_SS,WORD
        i_need  user_SP,WORD
        i_need  ERRORMODE,BYTE
        i_need  ConC_spSave,WORD
        i_need  Exit_type,BYTE
        i_need  PFLAG,BYTE
        i_need  ExitHold,DWORD
        i_need  WPErr,BYTE
        i_need  ReadOp,BYTE
        i_need  CONTSTK,WORD
        i_need  Exit_Code,WORD
        i_need  CurrentPDB,WORD
        i_need  DIVMES,BYTE
        i_need  DivMesLen,BYTE

SUBTTL Checks for ^C in CON I/O
PAGE
ASSUME  DS:NOTHING,ES:NOTHING

        procedure  DSKSTATCHK,NEAR ; Check for ^C if only one level in
CMP      BYTE PTR [INDOS],1
retnz    ; Do NOTHING
PUSH     CX
PUSH     ES
PUSH     BX
PUSH     DS
PUSH     SI
PUSH     CS
POP      ES
PUSH     CS
POP      DS
ASSUME   DS:DOSGROUP
XOR      CX,CX
MOV      BYTE PTR [DSKSTCOM],DEV RDND
MOV      BYTE PTR [DSKSTCALL],DRDNDHL
MOV      [DSKSTST],CX
MOV      BX,OFFSET DOSGROUP:DSKSTCALL
LDS      SI,[BCON]
ASSUME   DS:NOTHING
invoke   DEVIOCALL2
TEST     [DSKSTST],STBUI
JNZ      ZRET ; No characters available
MOV      AL,BYTE PTR [DSKCHRET]

```

```

DSK1:
    CMP     AL,"C"- "@"
    JNZ     RET36
    MOV     BYTE PTR [DSKSTCOM],DEV RD
    MOV     BYTE PTR [DSKSTCALL],DRDWRHL
    MOV     BYTE PTR [DSKCHRET],CL
    MOV     [DSKSTST],CX
    INC     CX
    MOV     [DSKSTCNT],CX
    invoke  DEVIOCALL2           ; Eat the ^C
    POP     SI
    POP     DS
    POP     BX                   ; Clean stack
    POP     ES
    POP     CX
    JMP     SHORT CNTCHAND

ZRET:
    XOR     AL,AL               ; Set zero

RET36:
    POP     SI
    POP     DS
    POP     BX
    POP     ES
    POP     CX
    return

NOSTOP:
    CMP     AL,"P"- "@"
    JZ      INCHK

    IF      NOT TOGLPRN
    CMP     AL,"N"- "@"
    JZ      INCHK
    ENDIF

    CMP     AL,"C"- "@"
    JZ      INCHK
    return
DSKSTATCHK  ENDP

    procedure  SPOOLINT,NEAR
    PUSHF
    CMP     BYTE PTR [IDLEINT],0
    JZ      POPFRET
    CMP     BYTE PTR [ERRORMODE],0
    JNZ     POPFRET             ;No spool ints in error mode
    INT     int_spooler
POPFRET:
    POPF
RET18:  return
SPOOLINT  ENDP

    procedure  STATCHK,NEAR

    invoke  DSKSTATCHK          ; Allows ^C to be detected under
                                ; input redirection

    PUSH    BX
    XOR     BX,BX
    invoke  GET_IO_FCB
    POP     BX
    JC      RET18
    MOV     AH,1
    invoke  IOFUNC
    JZ      SPOOLINT
    CMP     AL,'S'- '@'
    JNZ     NOSTOP
    XOR     AH,AH
    invoke  IOFUNC              ; Eat Cntrl-S
    JMP     SHORT PAUSOSTRT

```

```

PRINTOFF:
PRINTON:
    NOT     BYTE PTR [PFLAG]
    return

PAUSOLP:
    CALL    SPOOLINT
PAUSOSTRT:
    MOV     AH,1
    invoke  IOFUNC
    JZ      PAUSOLP
INCHK:
    PUSH    BX
    XOR     BX,BX
    invoke  GET_IO_FCB
    POP     BX
    JC      RET18
    XOR     AH,AH
    invoke  IOFUNC
    CMP     AL,'P'-'@'
    JZ      PRINTON
    IF      NOT TOGLPRN
    CMP     AL,'N'-'@'
    JZ      PRINTOFF
    ENDIF
    CMP     AL,'C'-'@'
    retnz
STATCHK ENDP

    procedure    CNTCHAND,NEAR
; Ctrl-C handler.
; "C" and CR/LF is printed. Then the user registers are restored and
; the user CTRL-C handler is executed. At this point the top of the stack
; has 1) the interrupt return address should the user CTRL-C handler wish
; to allow processing to continue; 2) the original interrupt return address
; to the code that performed the function call in the first place. If
; the user CTRL-C handler wishes to continue, it must leave all registers
; unchanged and RET (not IRET) with carry CLEAR. If carry is SET then
; an terminate system call is simulated.
    MOV     AL,3                ; Display "C"
    invoke  BUFOUT
    invoke  CRLF
    PUSH    SS
    POP     DS
ASSUME DS:DOSGROUP
    CMP     BYTE PTR [CONSWAP],0
    JZ      NOSWAP
    invoke  SWAPBACK
NOSWAP:
    CLI                                ; Prepare to play with stack
    MOV     SP,[user_SP]
    MOV     SS,[user_SS]          ; User stack now restored
ASSUME SS:NOTHING
    invoke  restore_world        ; User registers now restored
ASSUME DS:NOTHING
    MOV     BYTE PTR [INDOS],0    ; Go to known state
    MOV     BYTE PTR [ERRORMODE],0
    MOV     [ConC_spsave],SP      ; save his SP
    INT     int_ctrl_c           ; Execute user Ctrl-C handler
    MOV     [user_SS],AX          ; save the AX
    PUSHF                          ; and the flags (maybe new call)
    POP     AX
    CMP     SP,[ConC_spsave]
    JNZ     ctrlc_try_new        ; new syscall maybe?
ctrlc_repeat:
    MOV     AX,[user_SS]          ; no...
    transfer COMMAND             ; Repeat command otherwise

ctrlc_try_new:
    SUB     [ConC_spsave],2       ; Are there flags on the stack?
    CMP     SP,[ConC_spsave]
    JZ      ctrlc_new            ; yes, new system call

```

```

ctrlc_abort:
    MOV     AX,(EXIT SHL 8) + 0
    MOV     BYTE PTR [DidCTRLC],OFFh

    transfer    COMMAND        ; give up by faking $EXIT

ctrlc_new:
    PUSH    AX
    POPF
    POP     [user_SS]
    JNC     ctrlc_repeat      ; repeat operation
    JMP     ctrlc_abort       ; indicate ^ced

CNTCHAND ENDP

SUBTTL DIVISION OVERFLOW INTERRUPT
PAGE
; Default handler for division overflow trap
procedure    DIVOV,NEAR
ASSUME DS:NOTHING,ES:NOTHING,SS:NOTHING
MOV     SI,OFFSET DOSGROUP:DIVMES
CALL    RealDivOv
JMP     ctrlc_abort          ; Use Ctrl-C abort on divide overflow
DIVOV    ENDP

;
; RealDivOv: perform actual divide overflow stuff.
; Inputs:    none
; Outputs:   message to BCON
;
    procedure    RealDivOv,NEAR ; Do divide overflow and clock process

    PUSH    CS                ; get ES addressability
    POP     ES

    PUSH    CS                ; get DS addressability
    POP     DS
ASSUME DS:DOSGROUP

    MOV     BYTE PTR [DskStCom],DevWrt
    MOV     BYTE PTR [DskStCall],DRdWrHL
    MOV     [DskSTST],0
    MOV     BL,[DivMesLen]
    XOR     BH,BH
    MOV     [DskStCnt],BX
    MOV     BX,OFFSET DOSGROUP:DskStCall
    MOV     WORD PTR [DskChRet+1],SI ; transfer address (need an EQU)
    LDS     SI,[BCON]
ASSUME DS:NOTHING
    invoke    DEVIOCALL2
    MOV     WORD PTR [DskChRet+1],OFFSET DOSGROUP:DevIOBuf
    MOV     [DskStCnt],1
    return
RealDivOv    ENDP

SUBTTL CHARHRD,HARDERR,ERROR -- HANDLE DISK ERRORS AND RETURN TO USER
PAGE
    procedure    CHARHARD,NEAR
ASSUME DS:NOTHING,ES:NOTHING,SS:DOSGROUP

; Character device error handler
; Same function as HARDERR

    MOV     WORD PTR [EXITHOLD+2],ES
    MOV     WORD PTR [EXITHOLD],BP
    PUSH    SI
    AND     DI,STECODE
    MOV     BP,DS                ;Device pointer is BP:SI
    CALL    FATALC
    POP     SI
    return
CHARHARD    ENDP

    procedure    HardErr,NEAR

```

```

ASSUME DS:NOTHING,ES:NOTHING
; Hard disk error handler. Entry conditions:
;
;   DS:BX = Original disk transfer address
;   DX = Original logical sector number
;   CX = Number of sectors to go (first one gave the error)
;   AX = Hardware error code
;   DI = Original sector transfer count
;   ES:BP = Base of drive parameters
;   [READOP] = 0 for read, 1 for write
;
;
XCHG    AX,DI                ; Error code in DI, count in AX
AND      DI,STECODE          ; And off status bits
CMP      DI,WRECODE          ; Write Protect Error?
JNZ      NOSETWRPERR
PUSH     AX
MOV      AL,ES:[BP.dpb_drive]
MOV      BYTE PTR [WPERR],AL    ; Flag drive with WP error
POP      AX
NOSETWRPERR:
SUB      AX,CX                ; Number of sectors successfully transferred
ADD      DX,AX                ; First sector number to retry
PUSH     DX
MUL      ES:[BP.dpb_sector_size] ; Number of bytes transferred
POP      DX
ADD      BX,AX                ; First address for retry
XOR      AH,AH                ; Flag disk section in error
CMP      DX,ES:[BP.dpb_first_FAT] ; In reserved area?
JB       ERRINT
INC      AH                    ; Flag for FAT
CMP      DX,ES:[BP.dpb_dir_sector] ; In FAT?
JB       ERRINT
INC      AH                    ; Must be in data area
ERRINT:
SHL      AH,1                ; Make room for read/write bit
OR       AH,BYTE PTR [READOP]
entry   FATAL
MOV      AL,ES:[BP.dpb_drive] ; Get drive number
entry   FATAL1
MOV      WORD PTR [EXITHOLD+2],ES
MOV      WORD PTR [EXITHOLD],BP ; The only things we preserve
LES      SI,ES:[BP.dpb_driver_addr]
MOV      BP,ES                ; BP:SI points to the device involved
FATALC:
CMP      BYTE PTR [ERRORMODE],0
JNZ      SETIGN                ; No INT 24s if already INT 24
MOV      [CONSTK],SP
PUSH     SS
POP      ES
ASSUME   ES:DOSGROUP
CLI      ; Prepare to play with stack
INC      BYTE PTR [ERRORMODE] ; Flag INT 24 in progress
DEC      BYTE PTR [INDOS]      ; INT 24 handler might not return
MOV      SS,[user_SS]
ASSUME   SS:NOTHING
MOV      SP,ES:[user_SP]      ; User stack pointer restored
INT      int_fatal_abort      ; Fatal error interrupt vector, must preserve ES
MOV      ES:[user_SP],SP      ; restore our stack
MOV      ES:[user_SS],SS
MOV      SP,ES
MOV      SS,SP
ASSUME   SS:DOSGROUP
MOV      SP,[CONSTK]
INC      BYTE PTR [INDOS]      ; Back in the DOS
MOV      BYTE PTR [ERRORMODE],0 ; Back from INT 24
STI
IGNRET:
LES      BP,[EXITHOLD]
ASSUME   ES:NOTHING
CMP      AL,2
JZ       error_abort
MOV      BYTE PTR [WPERR],-1    ; Forget about WP error
return

```



```

SETIGN:
    XOR     AL,AL                ;Flag ignore
    JMP     SHORT IGNRET

error_abort:
    PUSH    SS
    POP     DS
ASSUME DS:DOSGROUP
    CMP     BYTE PTR [CONSWAP],0
    JZ      NOSWAP2
    invoke  SWAPBACK
NOSWAP2:
    MOV     BYTE PTR [exit_Type],Exit_hard_error
    MOV     DS,[CurrentPDB]
ASSUME DS:NOTHING

;
; reset_environment checks the DS value against the CurrentPDB. If they
; are different, then an old-style return is performed. If they are
; the same, then we release jfns and restore to parent. We still use
; the PDB at DS:0 as the source of the terminate addresses.
;
; output:  none.
;
    entry   reset_environment
ASSUME DS:NOTHING,ES:NOTHING
    PUSH    DS                ; save PDB of process

    MOV     AL,int_Terminate
    invoke  $Get_interrupt_vector ; and who to go to
    MOV     WORD PTR [EXITHOLD+2],ES ; save return address
    MOV     WORD PTR [EXITHOLD],BX

    MOV     BX,[CurrentPDB]    ; get current process
    MOV     DS,BX             ;
    MOV     AX,DS:[PDB_Parent_PID] ; get parent to return to
    POP     CX

;
; AX = parentPDB, BX = CurrentPDB, CX = ThisPDB
; Only free handles if AX <> BX and BX = CX and [exit_code].upper is not
; Exit_keep_process
;
    CMP     AX,BX
    JZ      reset_return      ; parentPDB = CurrentPDB
    CMP     BX,CX
    JNZ     reset_return      ; CurrentPDB <> ThisPDB
    PUSH    AX                ; save parent
    CMP     BYTE PTR [exit_type],Exit_keep_process
    JZ      reset_to_parent    ; keeping this process

    invoke  arena_free_process

; reset environment at [CurrentPDB]; close those handles
    MOV     CX,FilPerProc

reset_free_jfn:
    MOV     BX,CX
    PUSH    CX
    DEC     BX                ; get jfn
    invoke  $CLOSE            ; close it, ignore return
    POP     CX
    LOOP    reset_free_jfn    ; and do 'em all

reset_to_parent:
    POP     [CurrentPDB]      ; set up process as parent

reset_return:
    ; come here for normal return
    PUSH    CS
    POP     DS
    ASSUME  DS:DOSGROUP
    MOV     AL,-1
    invoke  FLUSHBUF          ; make sure that everything is clean

    CLI
    MOV     BYTE PTR [INDOS],0                ;Go to known state
    MOV     BYTE PTR [WPERR],-1              ;Forget about WP error

```

```
; $
; Snake into multitasking... Get stack from CurrentPDB person
;
    MOV     DS,[CurrentPDB]
    ASSUME  DS:NOTHING
    MOV     SS,WORD PTR DS:[PDB_user_stack+2]
    MOV     SP,WORD PTR DS:[PDB_user_stack]

    ASSUME  SS:NOTHING
    invoke  restore_world
    ASSUME  ES:NOTHING
    POP     AX                      ; suck off CS:IP of interrupt...
    POP     AX
    POP     AX
    MOV     AX,0F202h              ; STI
    PUSH    AX
    PUSH    WORD PTR [EXITHOLD+2]
    PUSH    WORD PTR [EXITHOLD]
    STI
    IRET                      ; Long return back to user terminate address
HardErr ENDP

    ASSUME  SS:DOSGROUP

do_ext

CODE     ENDS
        END
```

Chapter II

Introduction

Obfuscation is a information management strategy that aims to cloud the meaning of a message. This strategy can be intentional or unintended.

This strategy can be used for the protection of privacy (personal date protection of digital reputation for instance), but it can also be a base for the message content, war tactics or confidentiality saving.

Chapter III

Objectives

Thanks to the Famine subject, you know what self-replicating programming means. You also know the subject difficulty to obtain a little "virus" useless in an updated system.

We are going to make this virus running method more complex bringing programming methods that you were not necessarily introduced to during your school years.

Your little Famine program is about to be upgraded. You will quickly understand that if you want to validate this project, you're going to have to make radical changes in your source code. When Famine just needed you to discreetly patch a binary, Pestilence will require a way more discreet binary forcing you to apply a "simple" obfuscation strategy within your code... when we say simple, keep in mind this is a very subjective matter...

Chapter IV

Mandatory part

Pestilence is a binary of your own design which, just like **Famine**, will apply a signature that will at least include your own logins on binaries located in a specific folder, following the same constraints as **Famine**. The signature might have this header for instance:

```
Pestilence version 1.0 (c)oded by <first-login> - <second-login>
```

However you will have to code **Pestilence** so that:

- the code won't execute if a specific process is active on the target machine.
- the code doesn't execute if you try to use a debugger to run it.
- when a manual modification on the virus in order to avoid the anti- debugging launching is made, the infection routine must be partly obfuscated.

In regards of the latest point difficulty, the goal will simply be to make the reading and understanding of your infection routine difficult for a physical person. Being an actual issue in the real world, the obfuscation level will have an influence on your final grade.

Here are the constraints for this subject:

- The executable will be named **Pestilence**.
- This executable is coded in assembler, C or C++. Nothing else.
- Your program will not display anything on the standard or error outputs.
- You **WILL HAVE** to work in a VM.
- You will choose the target OS type. However, you will have to set up a proper VM for the evaluation.
- Your program will have to act on the /tmp/test and /tmp/test2 folders or the counterpart on your chosen OS and NOTHING ELSE. You're responsible for the propagation of your program.

- **WARNING!** There will only be one infection on said binary. Not more.
- Infections will start running on your OS binaries with a 64 bits architecture.

Chapter V

Example of use

Here is a potential use:

Lay the foundation:

```
# ls -al ~/Pestilence
total 736
drwxr-xr-x 3 root root 4096 May 24 08:03 .
drwxr-xr-x 5 root root 4096 May 24 07:32 ..
-rwxr-xr-x 1 root root 744284 May 24 08:03 Pestilence
```

We create a sample.c for our tests:

```
# nl sample.c
1 #include <stdio.h>
2 int
3 main(void) {
4     printf("Hello, World!\n");
5     return 0;
6 }
# gcc -m64 ~/Virus/sample/sample.c
#
```

We copy binaries (tests + ls) for our tests:

```
# cp ~/Virus/sample/sample /tmp/test2/.
# ls -al /tmp/test
total 16
drwxr-xr-x 2 root root 4096 May 24 08:07 .
drwxrwxrwt 13 root root 4096 May 24 08:08 ..
-rwxr-xr-x 1 root root 6712 May 24 08:11 sample
# /tmp/test/sample
Hello, World!
# file /tmp/test/sample
/tmp/test/sample: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=938[...]10b, not stripped
# strings /tmp/test/sample | grep "wandre"
# cp /bin/ls /tmp/test2/
# ls -al /tmp/test2
total 132
drwxr-xr-x 2 root root 4096 May 24 08:11 .
drwxrwxrwt 14 root root 4096 May 24 08:11 ..
-rwxr-xr-x 1 root root 126480 May 24 08:12 ls
# file /tmp/test2/ls
/tmp/test2/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=67e[...]281, stripped
#
```

Pestilence without the "test" process:

```
# pgrep "test"
# ./Pestilence
# strings /tmp/test/sample | grep "wandre"
Pestilence version 1.1 (c)oded may-2017 by wandre
# /tmp/test/sample
Hi!
# strings /tmp/test2/ls | grep "wandre"
Pestilence version 1.1 (c)oded may-2017 by wandre
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root 4096 May 2 10:11 .
drwxrwxrwt 14 root root 4096 May 2 10:17 ..
-rwxr-xr-x 1 root root xxxxxx May 2 10:12 ls
# gcc -m64 ~/Virus/sample/sample.c -o /tmp/test/sample
# ls -al /tmp/test
total 16
drwxr-xr-x 2 root root 4096 May 2 10:07 .
drwxrwxrwt 13 root root 4096 May 2 10:08 ..
-rwxr-xr-x 1 root root xxxx May 2 10:12 sample
# /tmp/test/sample
Hi!
# file /tmp/test/sample
/tmp/test/sample: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, not stripped
# strings /tmp/test/sample | grep "wandre"
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root 4096 May 2 10:11 .
drwxrwxrwt 14 root root 4096 May 2 10:17 ..
-rwxr-xr-x 1 root root xxxxxx May 2 10:12 ls
# strings /tmp/test/sample | grep "wandre"
Pestilence version 1.1 (c)oded may-2017 by wandre
#
```

Pestilence with the "test" process:

```
# pgrep "test"
22987
# ./Pestilence
# strings /tmp/test/sample | grep "wandre"
# /tmp/test/sample
Hi!
# strings /tmp/test2/ls | grep "wandre"
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root 4096 May 2 10:11 .
drwxrwxrwt 14 root root 4096 May 2 10:17 ..
-rwxr-xr-x 1 root root xxxxxx May 2 10:12 ls
# gcc -m64 ~/Virus/sample/sample.c -o /tmp/test/sample
# ls -al /tmp/test
total 16
drwxr-xr-x 2 root root 4096 May 2 10:07 .
drwxrwxrwt 13 root root 4096 May 2 10:08 ..
-rwxr-xr-x 1 root root xxxx May 2 10:12 sample
# /tmp/test/sample
Hi!
# file /tmp/test/sample
/tmp/test/sample: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, not stripped
# strings /tmp/test/sample | grep "wandre"
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root 4096 May 2 10:11 .
drwxrwxrwt 14 root root 4096 May 2 10:17 ..
-rwxr-xr-x 1 root root xxxxxx May 2 10:12 ls
# strings /tmp/test/sample | grep "wandre"
#
```


And now, last but not least, let's try to run **Pestilence** with **gdb**. It will be best understood with a little message:

```
# gdb -q ./Pestilence
(gdb) run
Starting program: /root/a.out
DEBUGGING..
[Inferior 1 (process 2683) exited with code 01]
# strings /tmp/test/sample | grep "wandre"
# /tmp/test/sample
Hi!
# strings /tmp/test2/ls | grep "wandre"
#
```

For the obfuscation part, since I don't want to demand anything or guide you, it will be mostly code review. If you can understand the infection routine execution too easily, it will mean your method is not enough! You can obfuscate your code with a key algorithm or using useless functions to complicate the reading and understanding of your virus.

You must understand that the harder the reading and understanding of your infection routine, the higher your grade. Be crafty!

Chapter VI

Bonus part

Bonus ideas:

- Being able to infect 32 bits binaries.
- Being able to infect all the files recursively from your OS root.



You must optimize this part executing infected binaries...

- Allowing infection on non-binary files.
- Using packing like methods directly on the virus. The aim being to make the binary as light as possible.
- You can play around adding a backdoor through your virus but make sure no error is visible... moreover if your backdoor provides a mean to open a port on your machine.



The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

During evaluation, you must be in a VM. For your information, the grading scale was built with a stable 64 bits Debian 7.0.