



# Project UNIX

Pestilence

42 Staff [pedago@staff.42.fr](mailto:pedago@staff.42.fr)

*Résumé: AND HIS NAME IS .....*



# Table des matières

<b>I</b>	<b>Préambule</b>	<b>2</b>
<b>II</b>	<b>Introduction</b>	<b>10</b>
<b>III</b>	<b>Objectifs</b>	<b>11</b>
<b>IV</b>	<b>Partie obligatoire</b>	<b>12</b>
<b>V</b>	<b>Exemple d'utilisation</b>	<b>14</b>
<b>VI</b>	<b>Partie bonus</b>	<b>17</b>
<b>VII</b>	<b>Rendu et peer-évaluation</b>	<b>18</b>

# Chapitre I

## Préambule

Parce qu'un peu de culture ne fait pas de mal, voici a quoi ressemble la gestion du CTRL+C sur 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
CMP DX,ES:[BP.dpb_first_sector] ; In directory?
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
```

# Chapitre II

## Introduction

L'obfuscation, assombrissement, ou obscurcissement est une stratégie de gestion de l'information qui vise à obscurcir le sens qui peut être tiré d'un message. Cette stratégie peut être intentionnelle ou involontaire.

Cette stratégie peut par exemple servir en matière de protection de la vie privée (par exemple, pour la protection des données personnelles ou la gestion de la réputation numérique), mais peut servir de base à un choix dans le contenu du message, à des tactiques de guerre ou à la sauvegarde de la confidentialité des informations.

On peut également parler de masquage ou d'opacification.

# Chapitre III

## Objectifs

Par le sujet Famine, vous avez maintenant une bonne vision sur la programmation de type auto-répliquant. Vous avez aussi pu voir la difficulté du sujet pour obtenir un simple petit "virus" pas utilisable dans un système à jour.

Nous allons maintenant rendre notre virus plus complexe dans sa méthode de lancement en introduisant des méthodes de programmation que vous n'avez pas forcément connu durant votre scolarité.

Votre petit programme Famine va prendre un niveau supplémentaire maintenant. Mais vous allez assez vite comprendre que si vous voulez valider ce projet, vous allez devoir opérer des changements drastiques dans votre code source. Car là où Famine ne vous demande que de patcher un binaire de manière discrète, Pestilence vous demandera de rendre votre binaire encore plus discret en vous obligeant d'appliquer une stratégie d'obfuscation "simple" dans votre code... enfin, simple, tout dépend du point de vue hein...

# Chapitre IV

## Partie obligatoire

**Pestilence** est un binaire de votre conception qui devra, comme **Famine**, appliquer une signature contenant au minimum vos logins respectifs sur des binaires placés dans un dossier spécifique, en reprenant les mêmes contraintes que **Famine**. La signature pourra par exemple avoir comme tête :

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

Toutefois, vous devrez coder **Pestilence** de telle sorte que :

- le code ne s'exécute pas si un processus spécifique est actif sur la machine cible.
- le code ne s'exécute pas si vous tentez d'utiliser un débogueur pour le lancer.
- Lorsqu'une modification manuelle sur le virus dans un but de contourner le lancement l'anti-debugging est réalisée, la routine d'infection doit être obfusquée partiellement.

A la vue de la difficulté de ce dernier point, le but sera ici de simplement rendre compliqué la lecture et la compréhension de votre routine d'infection pour une personne physique. Le problème étant toujours d'actualité dans le monde réel, le niveau d'obfuscation aura une influence sur votre note final.

Vos contraintes pour ce projet seront les suivantes :

- L'exécutable devra se nommer **Pestilence**.
- Cet exécutable est codé en assembleur, en C ou en C++ et rien d'autre.
- Votre programme ne va rien afficher sur la sortie standard ni d'erreur.
- Vous devrez **OBLIGATOIREMENT** travailler dans une VM.
- Le système d'exploitation cible est comme toujours libre de choix. Toutefois, vous devrez aménager lors de votre correction une VM appropriée.
- Votre programme va devoir agir sur les dossiers `/tmp/test` et `/tmp/test2` ou équivalent en fonction de votre système d'exploitation cible, et **UNIQUEMENT** dans

ces dossiers. Le contrôle de la propagation de votre programme est de votre responsabilité.

- **ATTENTION !** Une seule infection sur ledit binaire est possible, pas plus.
- Les infections se feront dans un premier temps sur des binaires du type de votre système d'exploitation ayant pour architecture 64 bits.

# Chapitre V

## Exemple d'utilisation

Voici un exemple d'utilisation possible :

On prepare le terrain :

```
# 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
```

On crée sample.c pour nos 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
#
```

On copie nos binaires ( tests + ls ) pour nos 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 sans le processus "test" :

```
# 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 avec le processus "test" :

```
# 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"
#
```



Et maintenant, le meilleur pour la fin, tentons de lancer **Pestilence** avec **gdb**. Avec un petit message, ce sera plus clair :

```
# 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"
#
```

Pour la partie obfuscation, puisque je ne veux pas donner d'obligation ou de guide, cette partie sera surtout du code review. Si vous arrivez à comprendre trop clairement l'exécution de la routine d'infection, et bien cela voudra simplement dire que votre méthode ne suffit pas ! Vous pouvez obfusquer votre code par un algorithme à clé ou encore par une utilisation de fonctions inutiles dans le but de rendre complexe la lecture et la compréhension de votre virus.

Comprenez bien que plus la lecture et la compréhension de la routine d'infection sera difficile à comprendre, plus votre note sera élevée à la soutenance. Soyez malins !

# Chapitre VI

## Partie bonus



Les bonus ne seront comptabilisés que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc ... Concrètement, cela signifie que si votre partie obligatoire n'est pas validée, vos bonus seront intégralement IGNORÉS.

Des idées de bonus :

- Pouvoir infecter les binaires d'architecture 32 bits.
- Pouvoir infecter tous les fichiers en partant de la racine de votre système d'exploitation de manière récursive.



Vous devez optimiser cette partie en exécutant les binaires infectés..

- Permettre une infection sur des fichiers non binaires.
- Utilisation de méthodes de type packing sur le virus directement dont le but sera de rendre le binaire le plus léger possible.
- Vous pouvez vous amuser à rajouter une backdoor par votre virus mais attention à faire en sorte qu'aucun erreur ne soit visible.. Surtout si votre backdoor permet d'ouvrir un port sur votre machine.

# Chapitre VII

## Rendu et peer-évaluation

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc).
- Rendez-votre travail sur votre dépôt **GiT** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance.
- Vous devez être sous une VM durant votre correction. Pour info, le barème a été fait avec une Debian 7.0 stable 64 bits.
- Vous être libre d'utiliser ce dont vous avez besoin, dans la limite des bibliothèques faisant le travail pour vous, ce qui correspondrait à un cas de triche.
- Vous pouvez poser vos questions sur le forum, sur jabber, IRC, slack...