ClubHACK 2010

The art of making bad jokes : from the easiest to the most effective

DAVID Baptiste & FILIOL Eric
bdavid@et.esiea-ouest.fr
filiol@esiea.fr

Laboratoire de virologie et de cryptologie opérationnelles - Corsaire pas pirate

esiea recherche (C + V)°

Agenda :

Introduction :

→ Malware and bad jokes ?

I) The art of paralyzing users :

→ The screen
→ The mouse
→ The keyboard
→ The system

II) Malware and the Internet :

→ Attacking the reputation of one person by downloading prohibited pictures.
→ Transfering the user's documents by mail.

III) Some more complex malware :

→ Create a companion virus with a polymorphic payload.
→ Create K-ary viruses (Filiol 2007).

IV) Conclusions :

→ General Conclusions

V) Appendix :

Introduction :

Why create bad jokes or malware ?

→ For fun ?

→ Revenge (against a boss, a colleague, a competitor, an enemy, etc…)

→ Get a reputation

→ Paralyze an opponent

→ Economic interest (blackmail, paralysis of competitive economic activities, spam activities...)

→ Theft of confidential data (credit card, other documents ...)

→ Industrial espionage

→ Mafia activities or unofficial military operations

Types of malicious actions and famous examples :

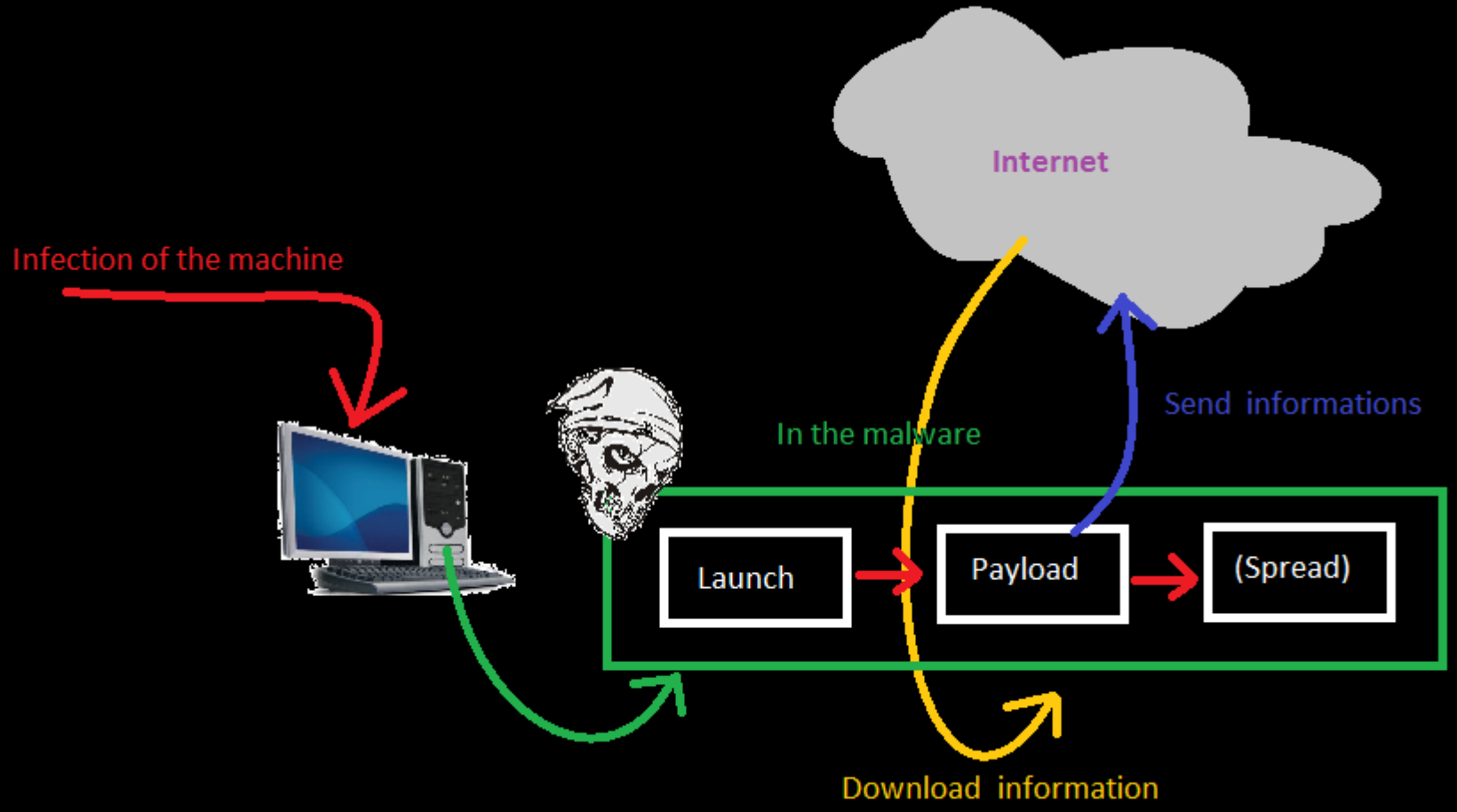
→ Paralyze the actions of the user (Sasser).

→ Create panic (I Love You, Michelangelo).

→ Spy users (Trojan : Back Orifice).

→ Ransom users (PGPCoder or GPCode).

The simplified steps of malware :



I) The art of paralyzing users :

The art of paralyzing users (with demos and explanation of code) :

→ Screen :

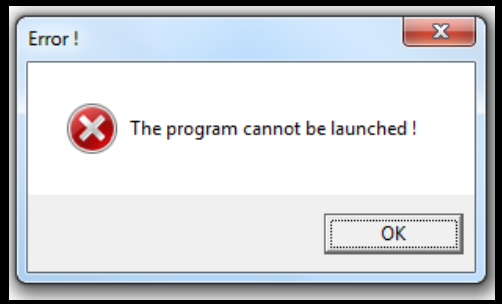
→ Use the screensaver.
→ Draw on the desktop !

→ Mouse :

→ Jail the mouse in a window.
→ Invert its movements.
→ Prevent the use of clicks.

→ Keyboard :

→ Neutralize most of the keys.



→ System :

→ Prevent the launch of programs.

Attack against the screen :

1) Use the screensaver to do it :

→ Use the SendMessage function explained in the MSDN.

```
SendMessage(
        __in  HWND hWnd,
        __in  UINT Msg,
        __in  WPARAM wParam,
        __in  LPARAM lParam
);
```

Where :

→ hWnd is a handle to the windows which will receive the message. If this argument is HWND_BROADCAST ((HWND)0xffff), the message is sent to all top-level windows in the system.

→ Msg is the message to be sent. Here, we will use WM_SYSCOMMAND (0x0112).

→ wParam is additional message-specific information. Here, according to WM_SYSCOMMAND we will control the monitor with SC_MONITORPOWER (0xF170).

→ lParam is another additional message detal. With the use of SC_MONITORPOWER, we have the choice to use :

- → -1 (the display is powering on)
- → 1 (the display is going to low power)
- → 2 (the display is being shut off)

Attacks against the screen :

2) Draw on the screen :

→ Get a handle on the desktop. (function : GetDesktopWindow();)

→ Get a special handle in order to draw on the desktop. (function : GetWindowDC();)

→ Create tools to draw on the desktop. (CreateSolidBrush(), CreatePen())

→ In an infinite loop : (while(1){ /* … */})

→ Draw a big black rectangle. (function : Rectangle();)

Advantage → Fast.

Disadvantage → Just a single color.

→ Draw pixel by pixel (or area by area).

Advantage : → Possibility of making real drawings (skulls, hearts, etc...)

Disadvantage : → Slow.

Attacks against the mouse :

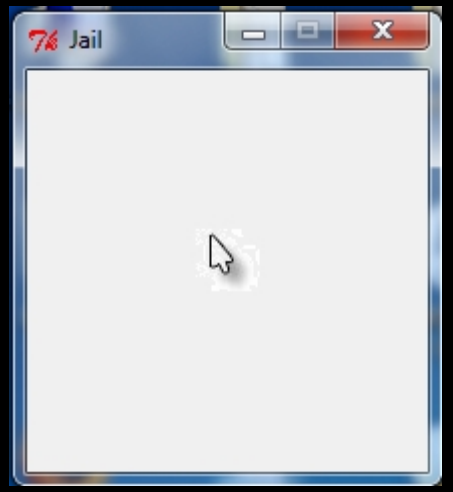
1) Jail the mouse in a window :

→ Create a Window (with Perl/Tk for example).

→ With the functions of Perl Tk (rootX (), rooty (), pointerx (), pointery ()) takes the cursor position.

→ In an infinite loop :

→ Check the position of the cursor.
→ If the cursor tries to go out of the window :
→ Set the cursor at the center of the window with MouseMoveAbsPix() function extracted from the Win32::GuiTest module.



→ Ideas to go further :

→ Prevent windows from closing :

```
$windows->protocol( "WM_DELETE_WINDOW", sub { return; } );
```

→ Prevent the use of interrupt signals for the program :

```
$SIG{INT} = sub { return; };
```

Attacks against the mouse :

2) Invert the mouse mouvements :

→ Detection of the position of the mouse at all times.

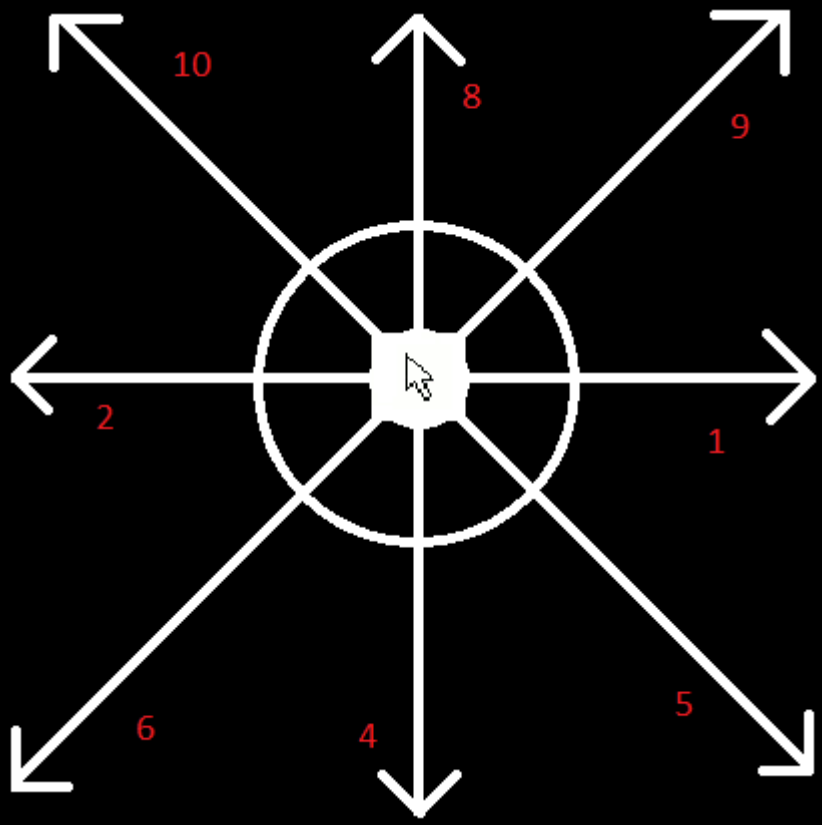
→ If there is a movement :

→ Calculate the difference between the initial position and the new one.

→ Calculate the speed of the cursor.

→ Determine the opposite position.

→ Move the cursor to the opposite position from where it should have been.

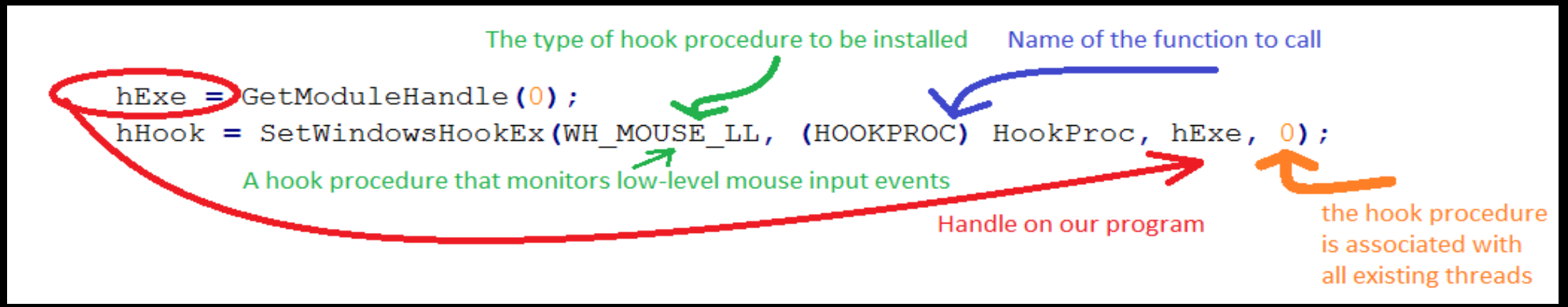


Attacks against the mouse :

3) Prevent the use of clicks.

→ Use the low level hook system from Windows.

→ Install a low level hook on the mouse.



→ Wait for messages from Windows (GetMessage(&MSG, NULL, 0, 0))

→ Broadcast windows messages on the hooks. (TranslateMessage() DispatchMessage())

→ In the HookProc, do not dispatch messages which say that the left button has been pressed. if (wParam == WM_LBUTTONDOWN) { return 1; }

Attacks against the keyboard :

Goal : Put an hook on the keyboard and do not spread the intercepted messages.

→ Possibility to use SetWindowsHookEx (); TranslateMessage (); DispatchMessage (); as seen previously for the mouse (but the hook is set with WH_KEYBOARD_LL).

→ Using RegisterHotKey( ) and GetMessage ( ) is also an intersting possibility.

→ Do not dispatch messages once they are intercepted.

The problem of taskmanager (ctrl+alt+sup) :

To Disable the Taskmanager, in the registry of windows :

In HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System

→ Set the key DisableTaskMgr to 1.

→ To really hook the ctrl + alt + sup, it would be necessary to go into kernel mode and make a driver that filters the keyboard.

Attacks against the system :

The goal is to prevent the launch of new programs :

→ In Perl :

→ Use the Win32::Process::List module.

→ Create the list of current processes with Win32::Process::List->new();

→ View the processes using GetProcesses();

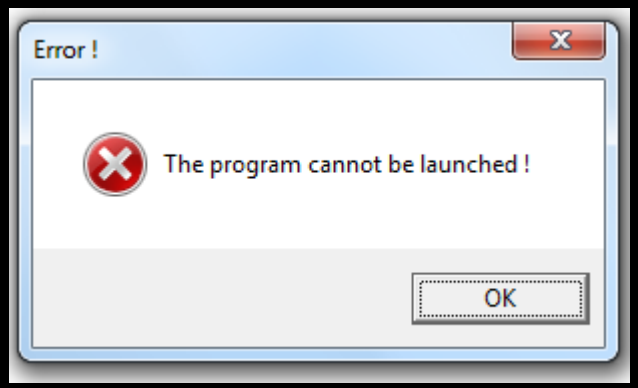
→ In an infinite loop :

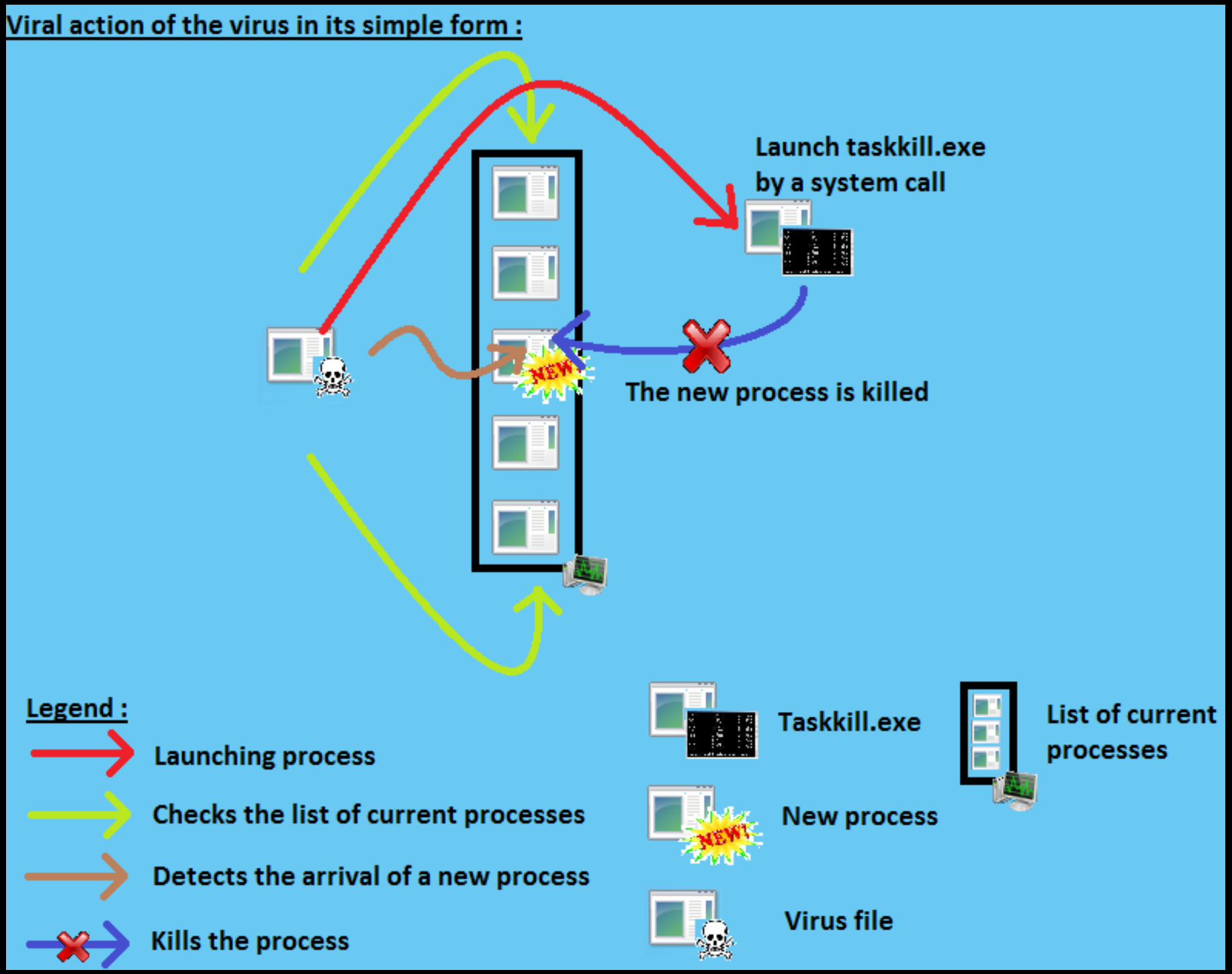
→ If there is a new process :

→ Kill it with the API or a system call.

→ If it's not possible

→ Throw an exception.





From iAwacs 2010 : DAVID Baptiste

II) Malware and the Internet :

Attacking the reputation of one person by downloading prohibited pictures.

→ In Perl :

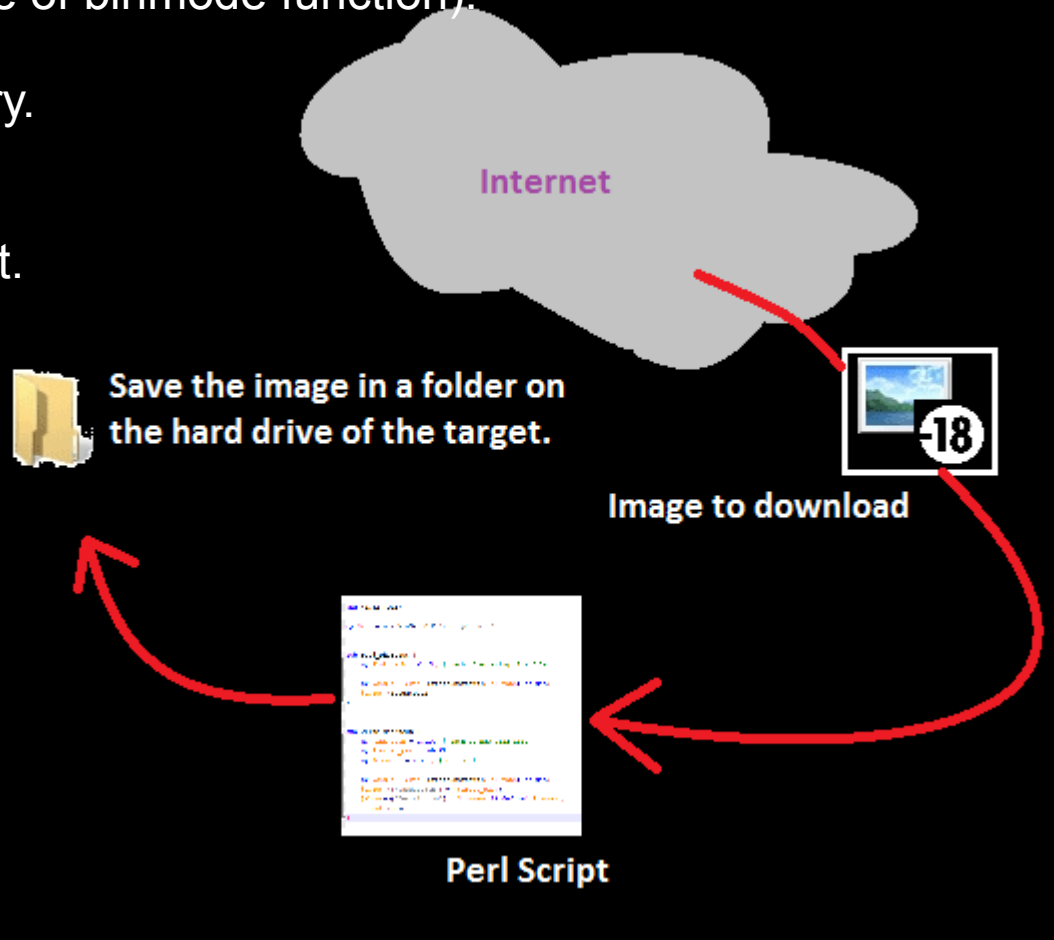
→ Downloading an image from the web with LWP::Simple.

→ Use get() method to do that.

→ Write the image in binary mode (use of binmode function).

→ Save the image in a hidden directory.

→ The FireWall of Windows 7 doesn't react.



Transfer the user's documents by mail.

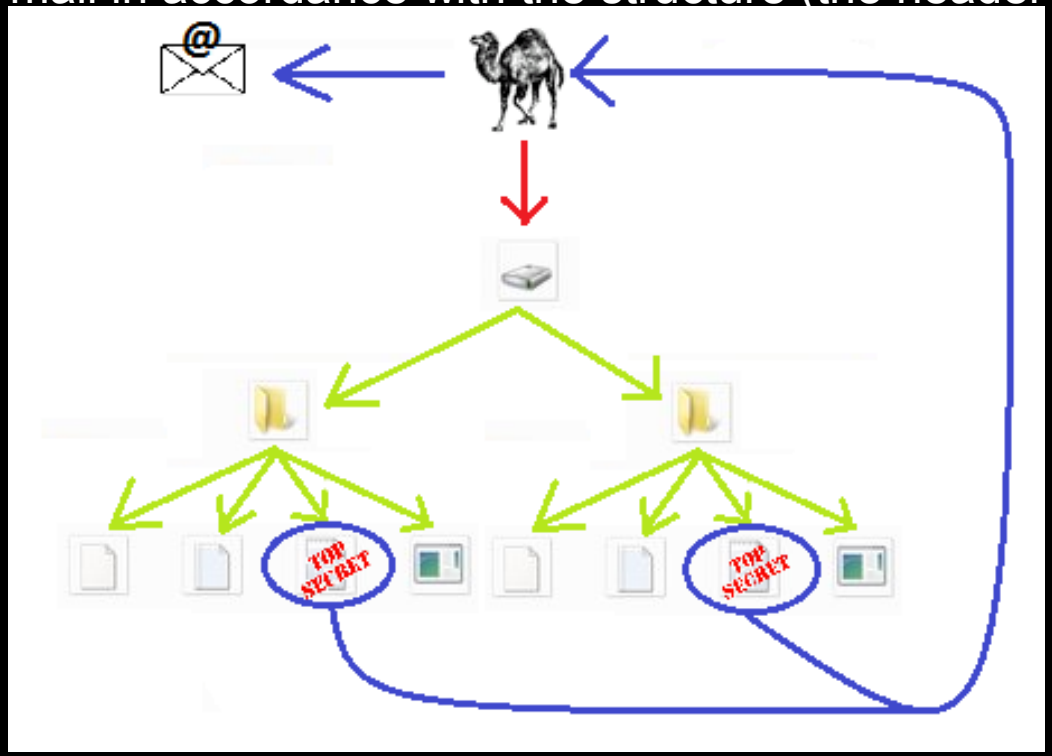
→ Browse the user's files to search an index in order to determine if files are interesting or not.

→ When an interesting file is selected :

→ Read the file.

→ Use the module Net:: SMTP in order to send the file by mail.

→ Write the mail in accordance with the structure (the header and the body) of a mail.



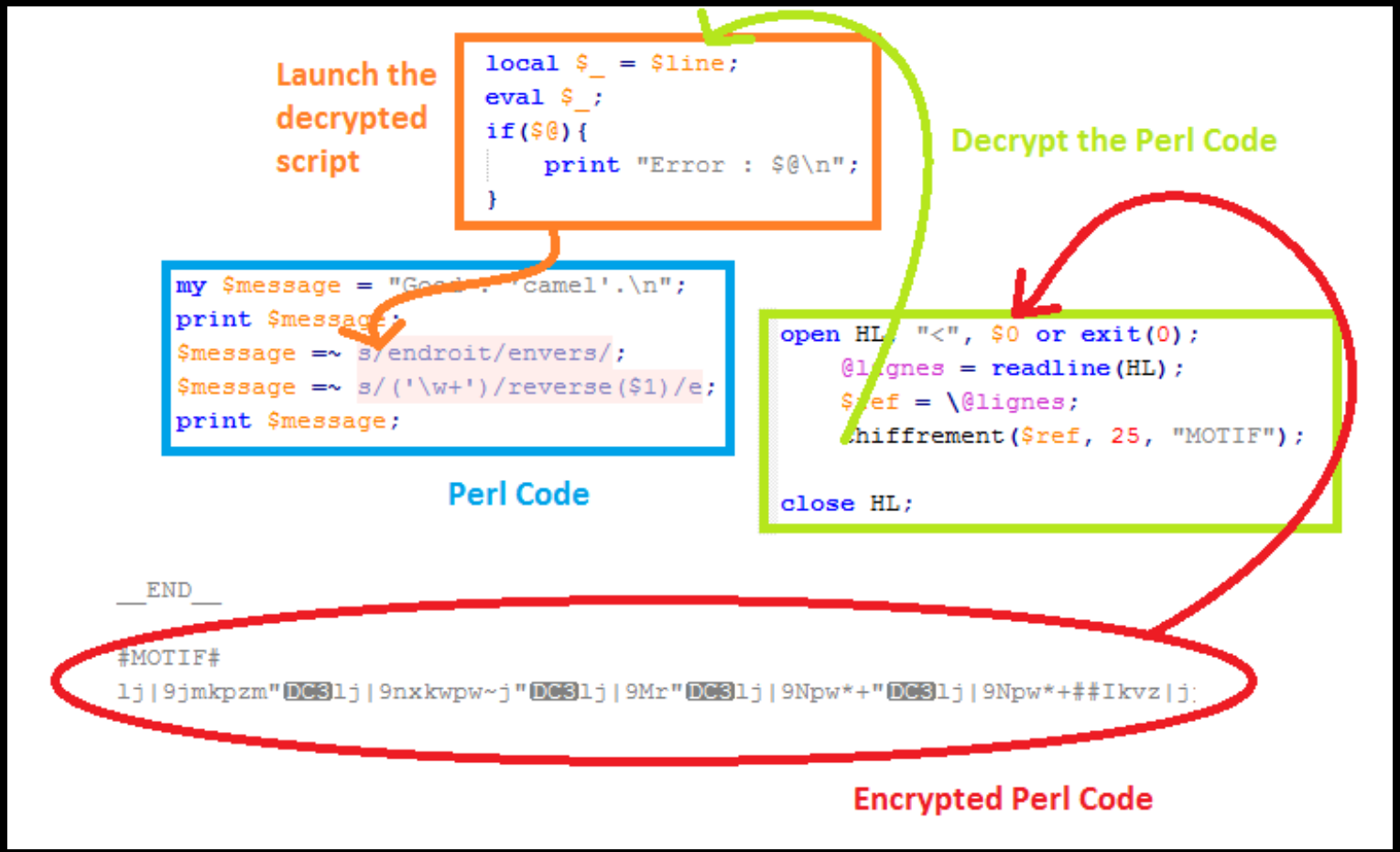
III) Some more complex malware :

How to create a simple polymorphic virus :

Goal : Create a companion virus with a polymorphic payload.

→ In Perl :

→ Use the payload from the jailed mouse.
→ To decrypt them, put the encrypted payload and the code, into another Perl script.
→ Load the Perl code decrypted from the script attacked.
→ Run the code decrypted by the attacked script.

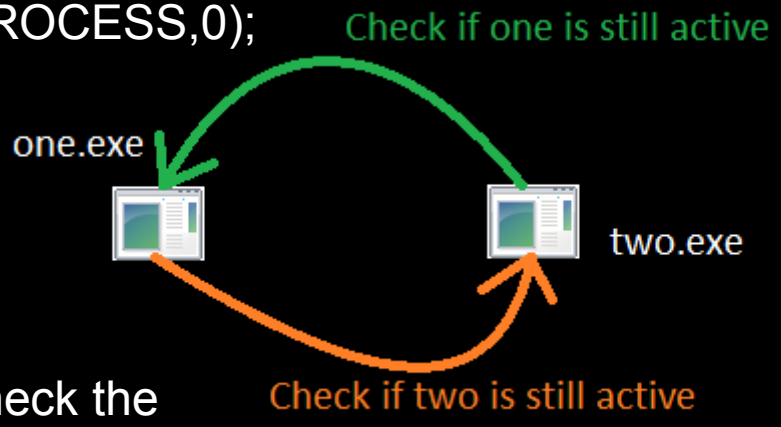


Creat K-ary viruses (Filiol 2007) :

→ The two parts of the virus create a self-checking "circle" in order to revive one of them if necessary.

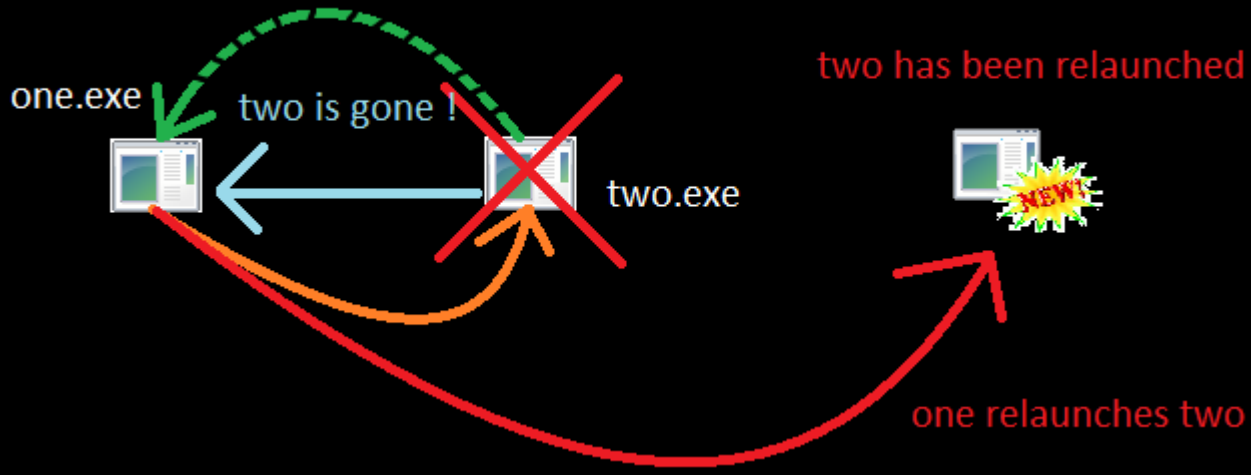
→ Use CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0); in order to get a handle on the process list.

→ Use the structure PROCESSENTRY32 from the Windows API to access the different process informations.



→ First take the list of running processes, and then check the presence of the viruses.

→ If one part of the virus disappears from the list of running processes, relaunch it.



IV) General Conclusion :

From a human point of view :

→ The reasons for creating malware are numerous. That means that many people may find reasons to creat malware.

→ Anyone can creat malware.

→ The hardest part may be about to find an idea for the viral payload.

From a technical point of view :

→ A large number of malware concern only small automation tasks.

→ Using the Windows API makes a lot of actions possible on the system.

→ The use of internet opens very interesting possibilities for remote control and spying.

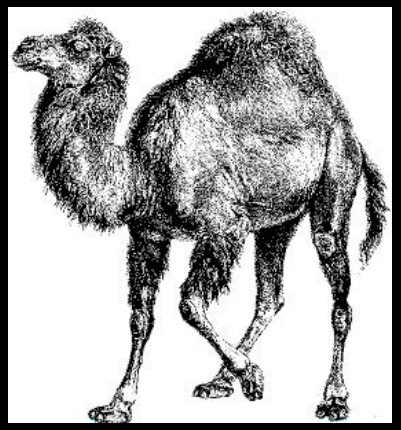
→ Ease of use and action concerning scripting languages (Python, Perl, JavaScript, Ruby)...

→ Usability of the Windows API with C language.

Thank you very much for your attention.

If you have any questions, I would be happy to answer them...

Appendix A : Perl Reminders :



The scalar values :

Represented by : $

Ex : → $a = 42;

→ $str = « I'm a string !\n »;

The Arrays :

Represented by : @

Ex : → @t = (1, 2, 3, 4);

→ @b = (« toto », 42, « foo »);

→ The access to the first element is given by : $b[0] (ie : toto).

→ The access to the last element is given by : $b[$#b] ↔ $b[-1] (ie : foo)

The hash tables :

Represented by : %

Ex : → %hash = (key => value);

→ $hash{$key} = $value;

The Scalar references :

Represented by : \$

Ex : → $A = 42; $refA = \$A; %hash = (foo => bar); $refHash = \%hash;

→ print $$refA; # (ie : 42).

→ print $refHash→{foo}; # (ie : bar).

Perl Reminders :

The conditions :

→ if (cond) { }
→ swtich { case : }
→ else { } or elsif (cond) { }

The main loops :

→ while(cond){ }
→ do { } while(cond);
→ for(initialization ; condition; increment) { }
→ foreach (@tab) { }
→ goto.

The operator conditions :

→ ==	: means equal between two scalars.
→ >= or <=	: are the inequality operators.
→ eq ne	: means equal or not equal between two strings.
→ cond ? True : False	: is the ternary operator.
→ || && or and not xor	: are the condition operators used in the conditions.
→ =~	: the operator which is used with the regular expressions.

Perl Reminders :

Some array functions :

Perl code :

```
#!  /usr/bin/perl -w          # Called the Shebang

my @t = (1,2,3,4);            # Is an array, the '#' is used for comments.
unshift(@t,-1,0);             # @t = (-1, 0, 1, 2, 3, 4)
my $f = shift(@t);            # @t = (0, 1, 2, 3, 4) and $f = -1
push(@t, 5, 6);               # @t = (0, 1, 2, 3, 4, 5, 6)
my $l = pop(@t);              # @t = (0, 1, 2, 3, 4, 5) and $l = 6
```

The subroutines :

→ Create a function :

```
sub foo {
        my $arg1 = shift;
        my $arg2 = shift;
        # … do something … #
        return $toBeReturned;
}
```

Concerning the modules :

Load a module	:	use myModule;
Access to a subroutine in the module	:	function() or myModule::function();

Appendix B : Use the shortcut as trigger :

Goal : Creating a trigger modulated by the user.

→ Using shortcuts in order to create the trigger.

→ Launch arbitrary code before the execution of the application referenced by the Shortcut.

→ Try to change the behavior of programs launched from shortcuts..

The tools to do that :

→ In Perl :

→ use Win32::Shortcut; # Module to handle shortcuts.

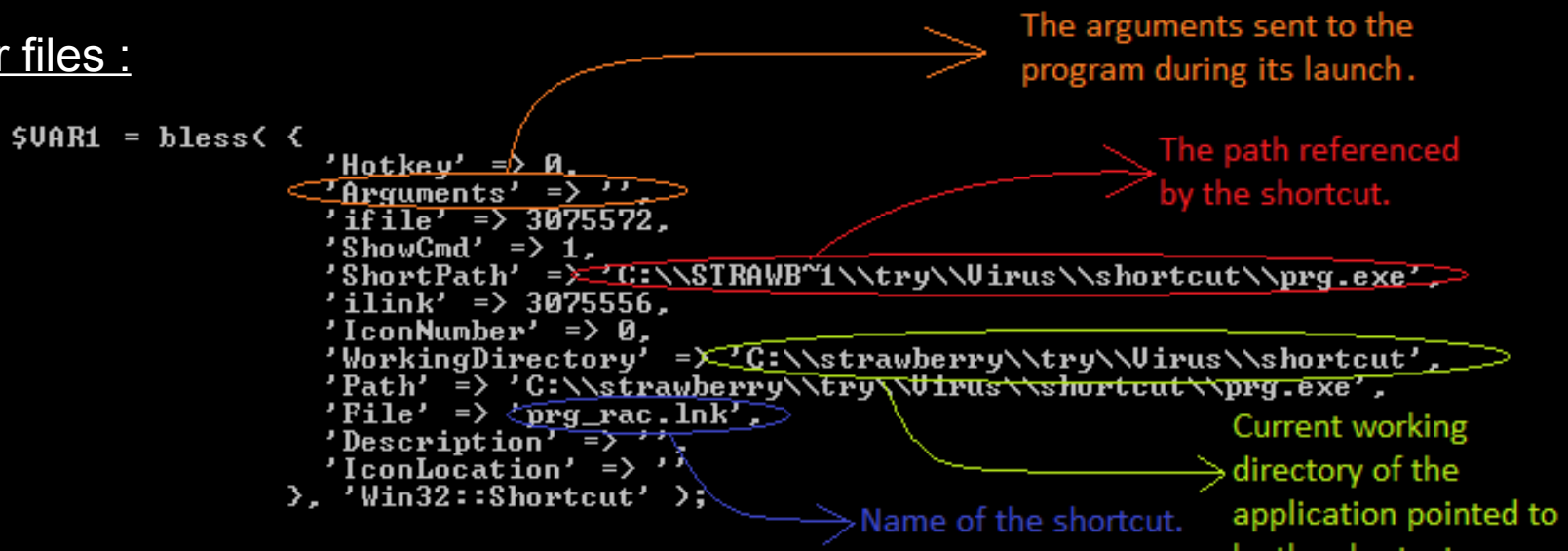
→ The elements of the shortcuts in the Shortcut module :

→	Path, ShortPath	↔ The target of the shortcut.
→	WorkingDirectory	↔ The working directory.
→	Description	↔ An optional description given to the shortcut.
→	ShowCmd	↔ The condition of the window in which the program will be executed
→	Hotkey	↔ The hotkey associated with the shortcut.
→	IconLocation	↔ The file that contains the icon for the shortcut.
→	IconNumber	↔ The number of the icon for the shortcut in the file pointed by IconLocation
→	Argument	↔ The arguments associated with the shell link object.
→	Filename	↔ The filename of the shortcut file opened.

Use the shortcut as trigger :

For directories :

For files :

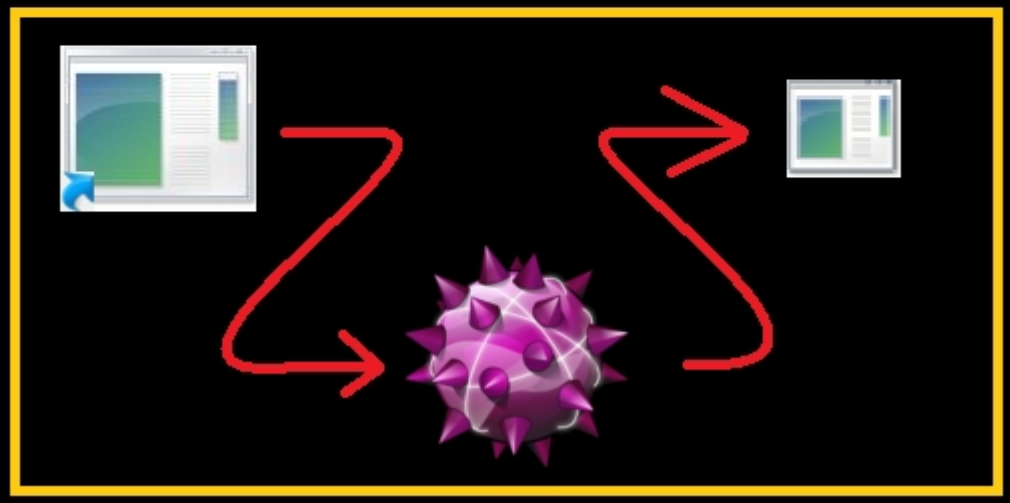


Use the shortcut as trigger :

For the files :

→ Normal Use :

→ Attack of the shortcut :



Use the shortcut as trigger :

For the directories :

→ Normal Use :

→ Attack of the shortcut :

Appendix C : Use the url to dispatch information for a computer under attack :

→ The goal is to send information taken by a keylogger on an attacked computer to the computer of the attacker.