Virus & Anti-virus

Sébastien LERICHE

ENAC

Version janvier 2017

Objectifs

- Apprécier les enjeux de la protection virale
- Décrire les différents types d'infection informatique
- Comprendre les techniques virales et antivirales
- Réagir en cas d'infection

Plan

- Partie 1 : Virus
  - Historique
  - Dégâts
  - Taxonomie
  - Cycle de vie
  - Modes de reproduction
  - Techniques anti-détection

Plan

- Partie 2 : Anti-virus
  - Le théorème de Cohen
  - Techniques statiques
  - Techniques dynamiques
  - Efficacité
  - Conduite à tenir

- Partie 3 : TP
  - Manipulations en bash
  - Étude de virus polymorphes, furtifs
  - Ver Windows
  - Anti-virus
  - Analyse de virus réels

Préliminaires

- Les concepts pratiques montrés dans ce cours ne doivent pas être expérimentés en dehors de l'établissement
- Loi Godfrain (art. 323), modifiée par la LCEN en 2004
  - Manipulations limitées à des systèmes virtualisés
  - Motif légitime : compréhension des mécanismes intrinsèques de fonctionnement des virus et anti-virus

– http://www.legifrance.gouv.fr/affichCode.do;jsessionid=543CB324820DC19FC267087CCF082019.tpdjo14v_3?idSectionTA=LEGISCTA000006149839&cidTexte=LEGITEXT000006070719&dateTexte=20100321

Partie 1 : Virus

Historique (1).

- 1940 – 1950 : théorie
  - Automate Von Neumann
  - Capacité d'un programme de se reproduire
  - Automates cellulaires autoreproducteurs
    - http://fr.wikipedia.org/wiki/Automate_cellulaire
- 1960-1980 : premiers virus « innocents »
  - Jeu « Core Wars »
  - Programmes d'administration bugués
  - Hypothétiques travaux militaires

Historique (2)

- 1980 – 1988 : Premières menaces virales
  - Premiers virus / vers (Xerox, virus pour Apple II, Elk Cloner pour AppleDOS, Brain...
  - Travaux et thèse de Fred Cohen en 1984-1986
    - Terminologie « Virus » et formalisation des infections
  - RTM en 1985, « vers » malveillant, 5% internet
- 1989 – 1995 : Complexité
  - Virus évolutifs « polymorphiques »
  - Apparition des anti-virus (~50 virus connus en 1992)
  - Générateurs de virus, intérêt de la communauté « hacker » => virus nombreux et complexes

Historique (3)

- 1995 – 1999 : Virus de documents
  - Macros, Suite Office (Concept, diffusé par Microsoft)
  - 40k virus connus en 1998
  - 80% des alertes sont des virus scriptés (vs programmes)
- 1999-2000 : Bulle Internet
  - Virus « mass-mailer » (propagation par mail)
  - Melissa fait le tour du monde en 2 jours en 1999
  - IloveYou en seulement 2 heures en 2000
  - 56k virus en l'an 2000

Historique (4)

- 2001-2004 : Vers Internet
  - Nimda, multipropagation (faille IIS)
  - CodeRed, 350k machines infectées en 24h
  - SQLSlammer, autant en 10 minutes
  - Propagation des virus par échange de fichiers P2P
- 2004- 2009 : nouveaux matériels & social
  - Cabir, infection de smartphones Symbian (BT)
  - Psyb0t, infection des routeurs + modems adsl
  - StormWorm, infection multiple à grande échelle
  - StalkDaily, Twitter / Koobface, Réseaux sociaux

Historique (5)

- 2010-2015 : bancaire et espionnage
  - Flame, « boîte à outil » d'espionnage
  - Reveton, « ransomware »
  - ZeuS, cheval de troie, Tor (70M$)
  - Obad, réseau de zombies d'appareil mobiles (Android)
  - Stuxnet, attaque des centrifugeuses en Iran (SCADA)
  - Regin, cible d'autres ORG (EU, centres de recherche…)

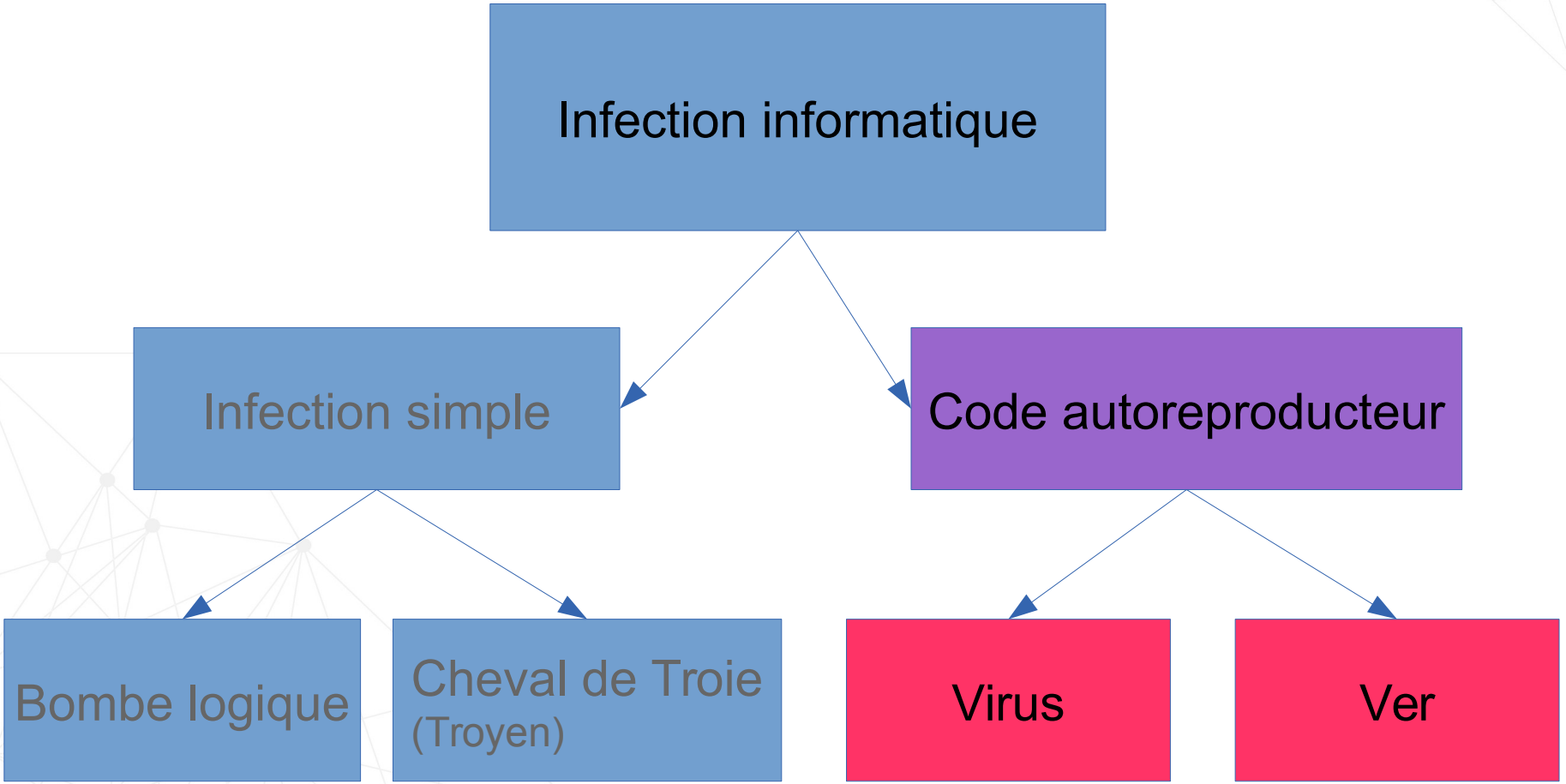
Sources d'information

- Web
  - http://secuser.com/alertes/index.htm
  - http://wildlist.org
  - http://www.clusif.asso.fr/fr/production/infovir/
  - http://spyware32.com
- Sites des éditeurs d'anti-virus
  - Ex : Sophos, F-Secure, Symantec, AVP…
- Presse Spécialisée
  - Linux magazine hors série 32
  - Revue MISC numéro 20

Dégats

- Sur les données, les logiciels et même le matériel
  - Sobig.F, 2003, 100M utilisateurs
  - Lovsan/Blaster, tous les abonnés d'un grand fournisseur d'accès Internet
  - CIH / Thernobyl, 1998 destruction du BIOS => changement de carte mère
- Selon les sources la moyenne des coûts pour un macro-ver comme Melissa est de 1,1 milliard d'euros, 8,75 milliards pour un ver d'e-mails type ILoveYou
  - http://desktoplinux.com/articles/AT3307459975.html

Taxonomie


Infection informatique
Infection simple
Code autoreproducteur
Bombe logique
Cheval de Troie
(Troyen)
Virus
Ver

Définitions (1)

- Infection informatique
  - Programme à caractère offensif, s'installant dans le système d'information, en vue de porter atteinte à la confidentialité, à l'intégrité ou la disponibilité de ce système
- Bombe logique
  - Programme offensif
- Cheval de Troie (ou Troyen)
  - Programme d'apparence annodine, mais contenant une activité malveillante

Définitions (2)

- Virus
  - Séquence de symboles interprétés modifiant d'autres séquences de symboles de manière à y include une copie de lui-même, éventuellement évoluée (F. Cohen)
  - Infection de programmes « hôtes »
- Ver
  - Similaire aux virus, mais pas d'infection de programme « hôtes », la propagation et la duplication se font par d'autres moyens (réseau...)

Bestiaire de virus et vers (1)

- Nomenclature plus très utile car les virus récents combinent plusieurs aspects.
- Illustration de la diversité des hôtes ou des modes de fonctionnement
  - Virus de programmes exécutables (exécutables, drivers, bibliothèques...)
    - Exécution de code binaire
  - Virus de documents
    - Exécution par interprétation de script contenu dans le document
    - Danger maximum si aucune action de contrôle de l'utilisateur (scripts WSH, Office, XML...)

Bestiaire de virus et vers (2)

- Virus de démarrage
  - Difficulté de détection par un anti-virus (code chargé postérieurement)
  - Aucune protection du système (accès disques)
  - BIOS (Basic Input Output System)
  - MBR (Master Boot Record)
- Virus comportementaux
  - Résidents (pas de trace sur le système de fichiers)
    - Inactivés à l'arrêt de la machine
    - Détournements de fonctions systèmes (hooks)
  - Binaires ou n-aires ou combinés
    - Code divisé sur plusieurs programmes a priori indépendants
  - Blindés (fonctions anti-antivirus)

Bestiaire de virus et vers (3)

- Virus psychologiques
  - Désinformation de l'utilisateur consistant à le pousser à exécuter une action malveillante pour son propre système
  - Utilisation de techniques d'ingénierie sociale
  - Ex. : effacement de fichiers systèmes soi-disant contaminés

- 3 classes de vers
  - Simples (worms)
    - Activité exploitant une faille d'un logicielle ou d'un OS
    - Duplication par le réseau internet

Bestiaire de virus et vers (4)

- Macro-vers
  - Programmes hybrides disséminés dans des documents bureautiques via des pièces jointes de courriers électroniques (type Melissa)
  - Infection du logiciel de bureautique après ouverture du document
  - Propagation via exploitation du carnet d'adresses
- Vers d'email
  - Propagation également par pièce jointe activé directement par le client de messagerie (type IloveYou)
  - Infection du client de messagerie
  - Propagation similaire

Infections Informatiques

Qu'est ce donc ?

1. Programme (dispositif principalement logiciel)
2. S'installe à l'insu du ou des utilisateurs,
3. Objectif : porter atteinte au système

Infections Informatiques

En mode furtif:

L'utilisateur ne se rend pas compte qu'un tel programme est présent

En mode résident:

Processus actif en mémoire afin de pouvoir agir en permanence (démon ou service)

Invisible lors de l'affichage des processus

En mode persistent:

Capable par différentes techniques, de se réinstaller dans la machine.

Localisation multiple (base de registre + programme résident + système de fichier...)

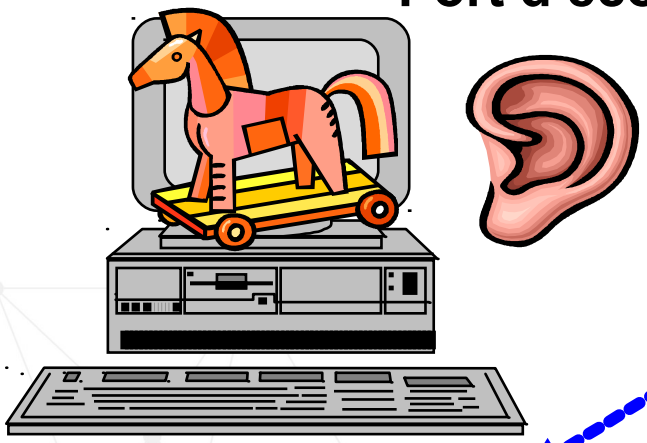
Éradication difficile (Toutes les actions de désinfection doivent être simultanées)

Cheval de Troie ou Troyen

Il est constitué de 2 parties

poste victime

Port d'écoute



module serveur

Réseau

Internet ou local

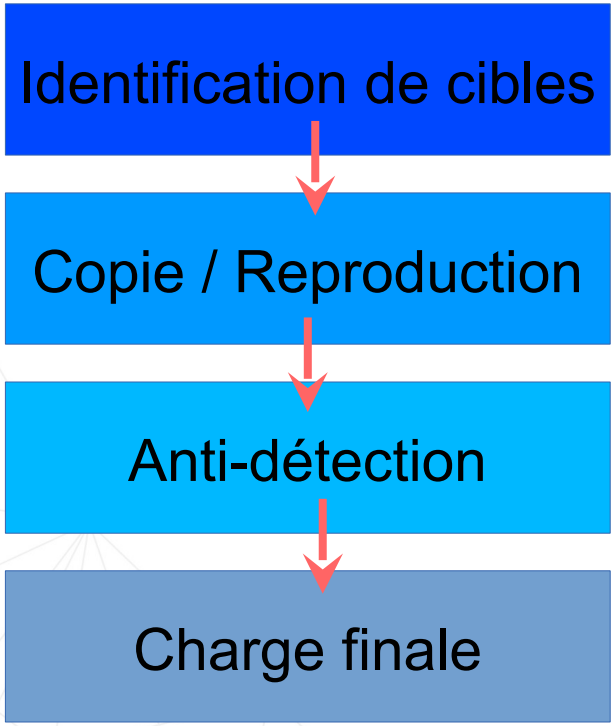
poste agresseur

module client

Il autorise un accès distant

à la victime

Diagramme fonctionnel



Fichier exécutable non déjà infecté, connexion réseau, exploitation de faille de sécurité...

Écrasement, ajout….de code

Polymorphisme, furtivité, chiffrement...

Activité malveillante

Cycle de vie (1)

- Diffusion initiale
  - Programme d'apparence inoffensive = « Dropper »
  - Adapté aux victimes
  - Ingéniérie sociale (logiciel de jeux, exécutables / animation type flash, pornographie...)
- Infection
  - Passive : l'utilisateur copie le dropper (P2P, forum, mail...)
  - Active : l'utilisateur exécute le dropper (« primo-infection ») ou un fichier déjà contaminé
    - A cette étape, un virus duplique son code, mais pas un ver

Cycle de vie (2)

- Incubation
  - Objectif = Survie du virus
  - Copies multiples, supports multiples
  - Limitation de la détection, contournement des tests des anti-virus
- Maladie
  - Activation de la charge finale (bombe logique)
  - Systématique dès la primo-infection
  - Différée à une « gachette » (imagination du hacker)
  - Létale (CIH) ou non (ex : Coffee Shop, légalisation de la marijuana...)

Routines

- Routine de Recherche
  - Détermine l'étendue de l'infection
  - Et sa rapidité
  - Trouver le format de cible correct
  - Éviter la surinfection (à l'aide de signatures)
- Routine de copie
- Routine d'anti-détection
- Routine offensive (charge finale)

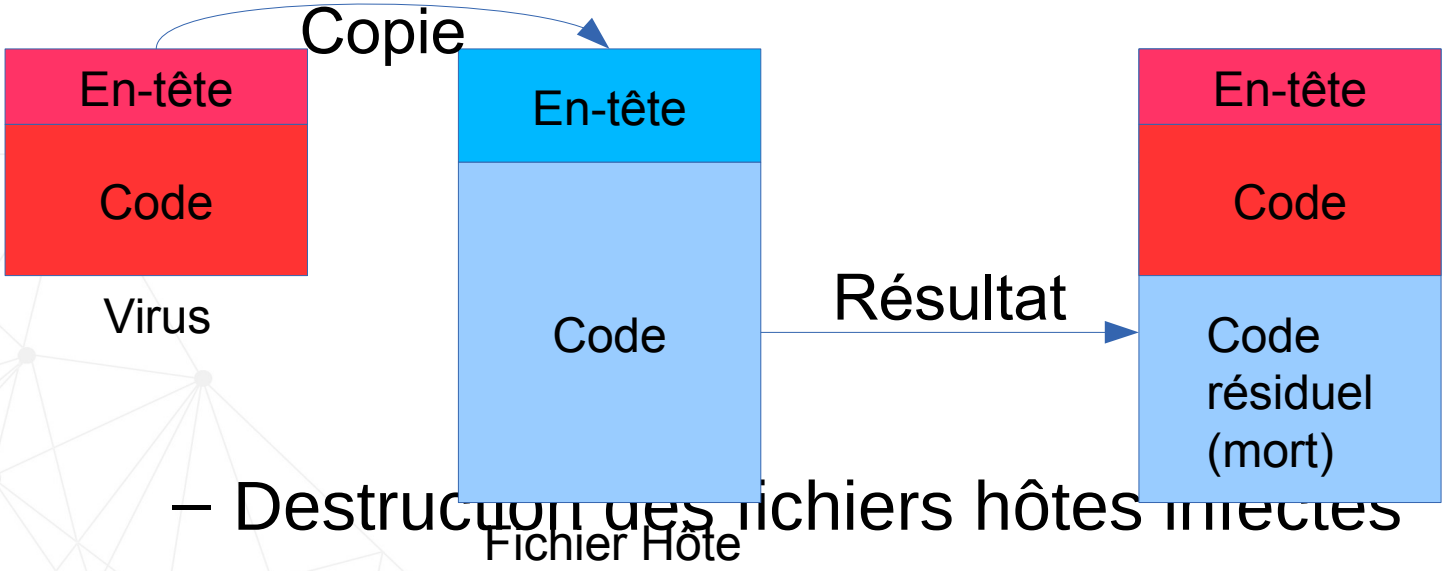
Routines

Programmes Autoreproducteurs

- Routine de Recherche
- Routine de copie
  - Écrasement de code
  - Ajout de code
  - Entrelacement de code
  - Accompagnement de code
- Routine d'anti-détection
- Routine offensive (charge finale)

Reproduction d'un virus (1)

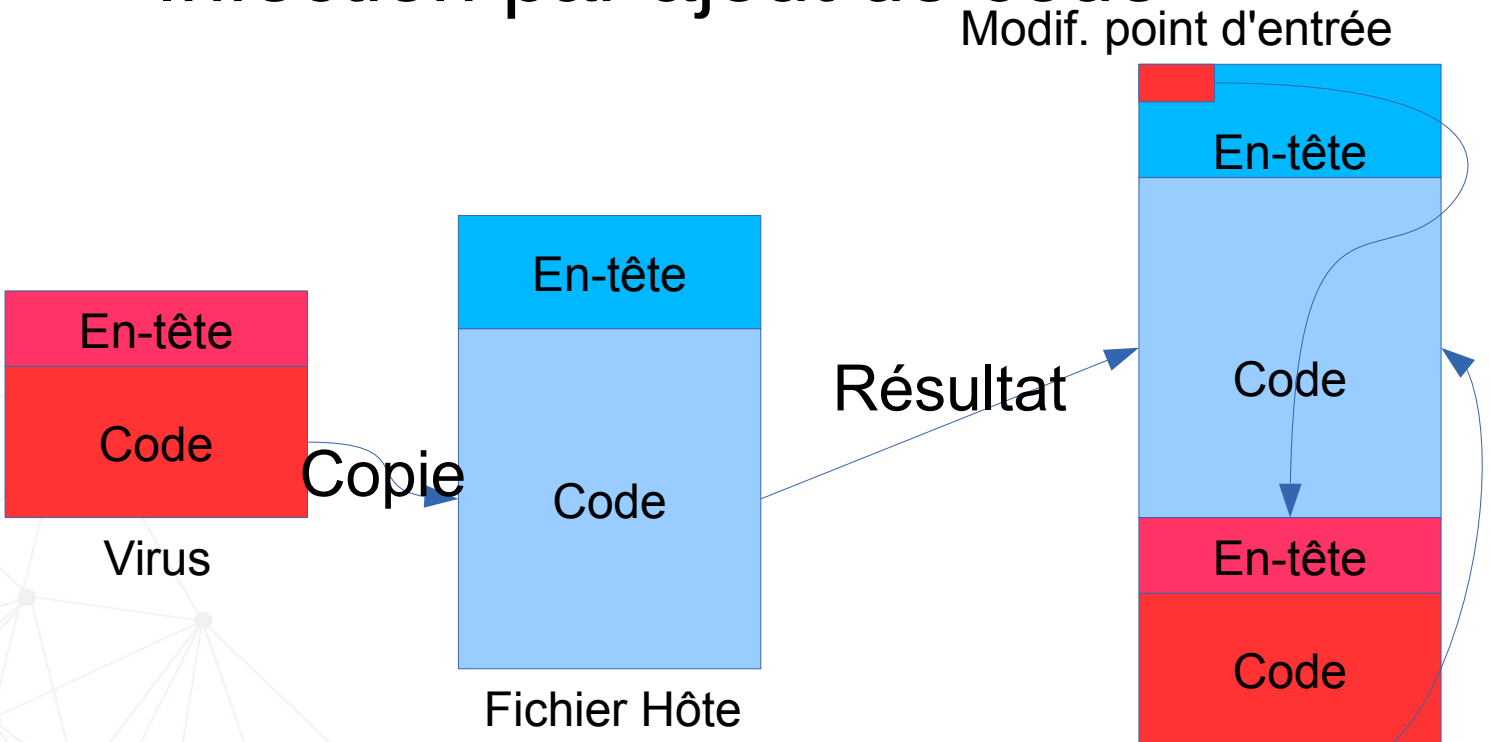
- Infection par écrasement ou recouvrement de code



  - Destruction des fichiers hôtes infectés
  - Taille du fichier d'origine conservée

Reproduction d'un virus (2)

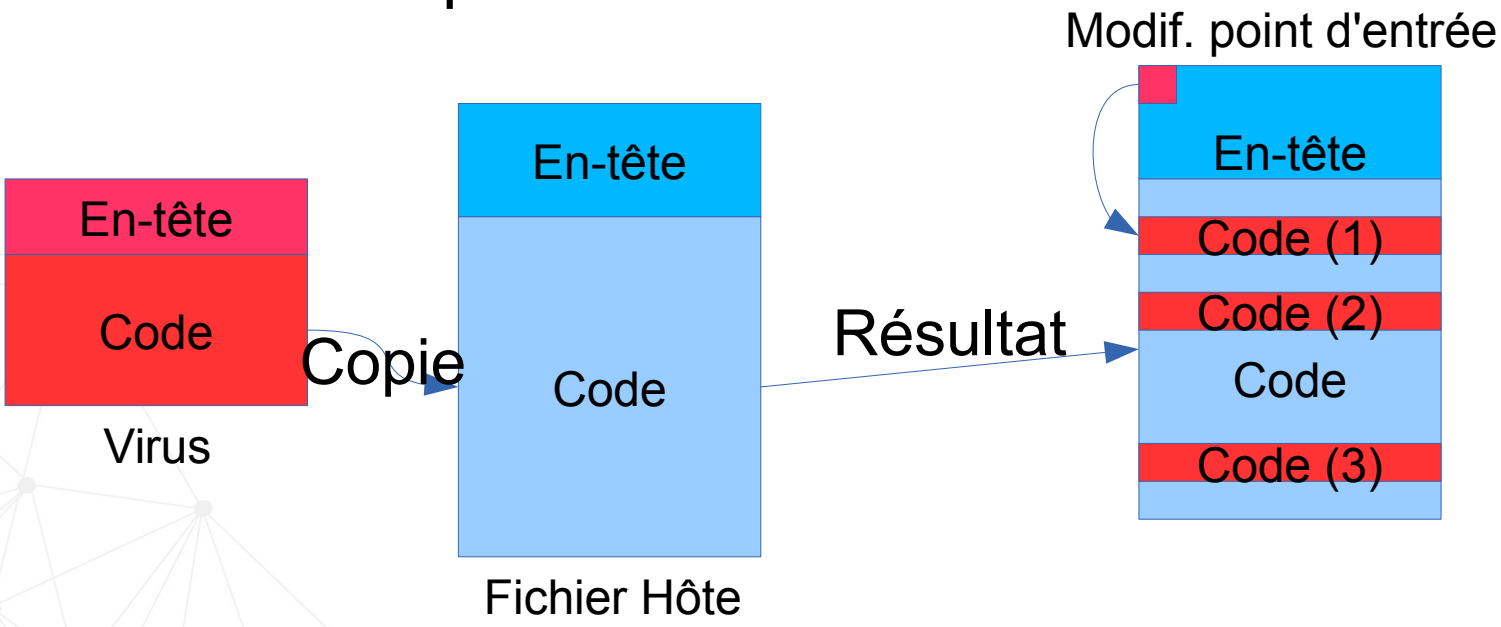
- Infection par ajout de code



  - En début ou en fin de code, plus simple en fin
  - Le code modifié en début de programme est « corrigé » temporairement par le virus

Reproduction d'un virus (3)

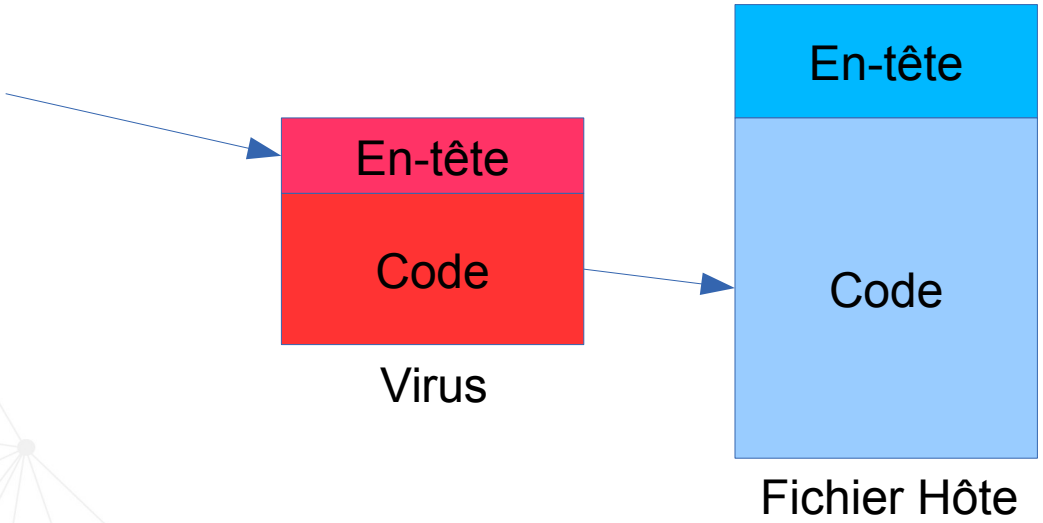
- Infection par entrelacement



  - Exploitation de zones de code inutilisées
  - Taille d'origine conservée

Reproduction d'un virus (4)

- Infection par accompagnement (compagnon)

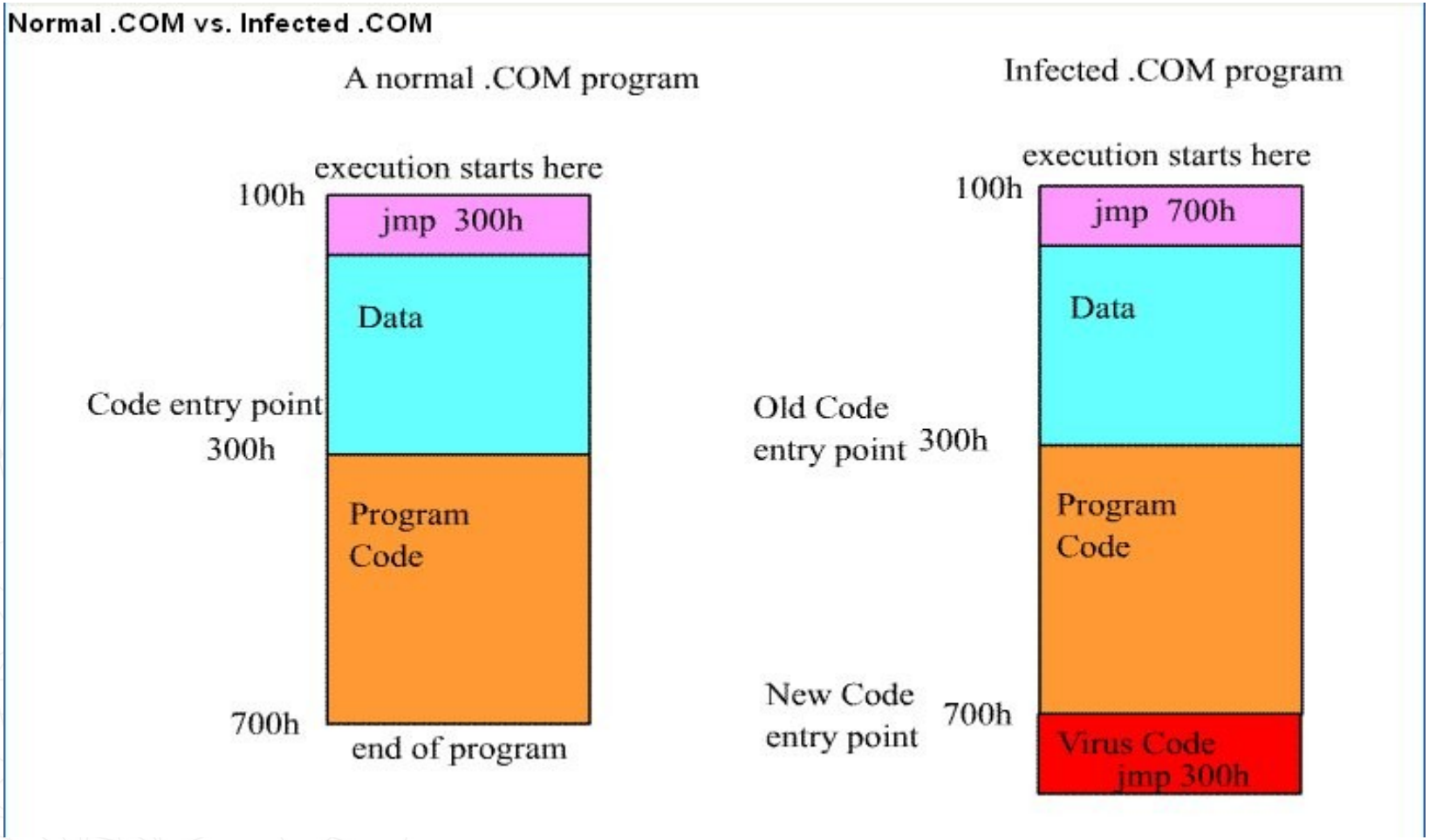


Fichier hôte intact

- Préemption d'exécution (path, chemin d'exécution...)
- Renommage et copie sous le même nom que le fichier hôte

Progammes
Autoreproducteurs

Exemple : infection d'un .com



Programmes Autoreproducteurs

Routines

- Routine de Recherche
- Routine de copie

- Routine d'anti-détection

contrer l'action des antivirus pour assurer la survie du virus

  - Méthode passive:
    - Furtivité
    - Polymorphisme
  - Méthode active
    - mise en veille,
    - saturation
    - désinstallation de l'antivirus.
  - Routine offensive (charge finale)

Infection simple

- Programme résident
  - Processus actif, démon, service
  - Peut être furtif
    - Invisible lors de l'affichage des processus
- Mode persistant
  - Localisation multiple (base de registre + programme résident + système de fichier...)
  - Éradication difficile, car toutes les actions de désinfection doivent être simultanées

A vous de jouer (1)

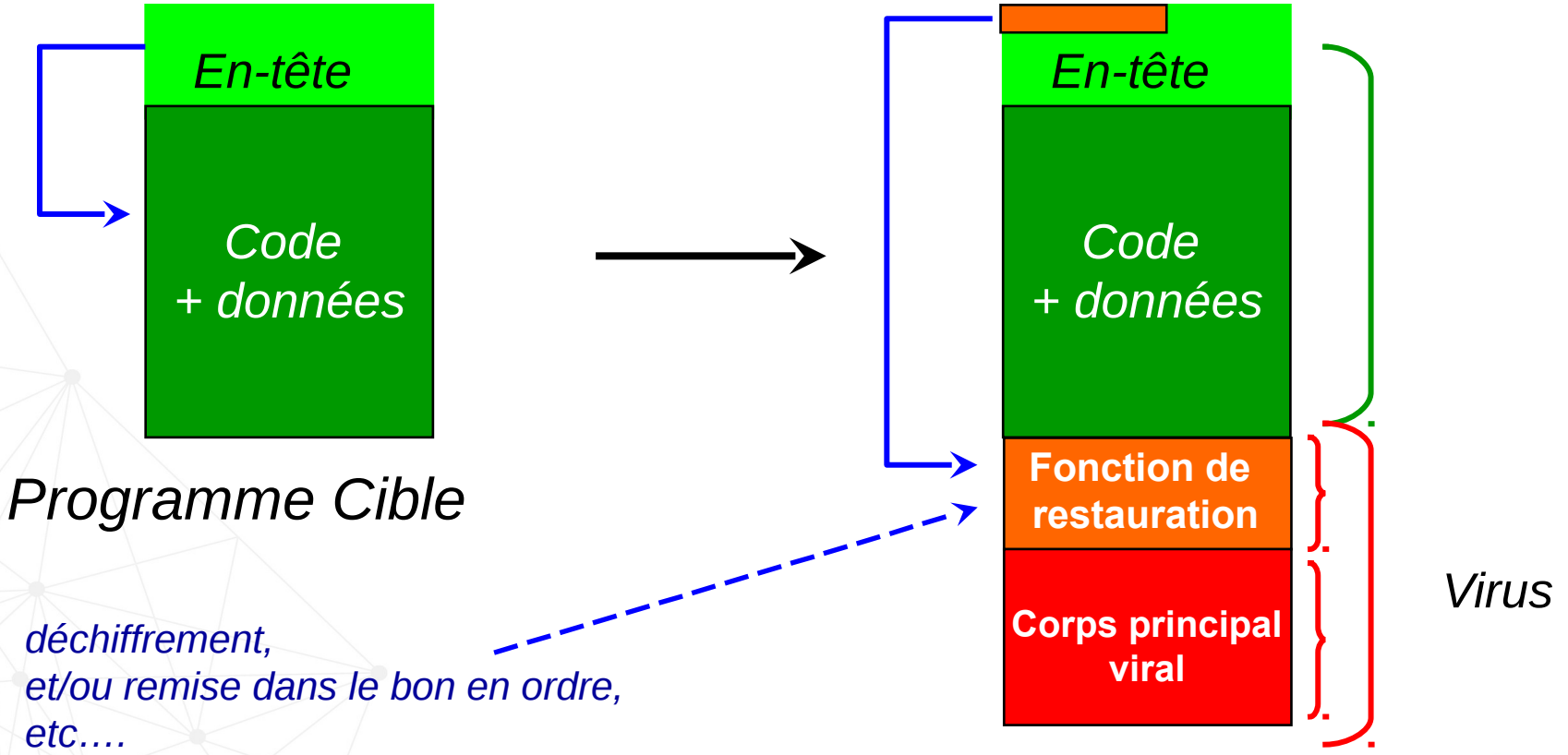
- Utilisation du langage BASH / Linux
  - http://www.gnu.org/software/bash/manual/bashref.html
  - Que fait le programme ci-dessous (vbash) ?
  - Quel est le type d'infection ?
  - Modifiez le programme pour éviter d'infecter plusieurs fois le même hôte (signature)

```
for i in  *.sh; do
    if test "./$i" != "$0"; then
        tail -n 5 $0 | cat >> $i ;
    fi
done
```

Techniques anti-détection (1)

- Polymorphisme
  - Exécution d'un code de forme différent mais de sémantique inchangée (réécriture du code)
  - DEC EAX → SUB EAX,1
  - Ordre des instructions
  - « Garbage code » (NOP, XOR REG,0...)
  - Metamorphisme (méta-langage pour encoder le module de polymorphisme)

Infection par du code polymorphe



Les virus et vers Polymorphes

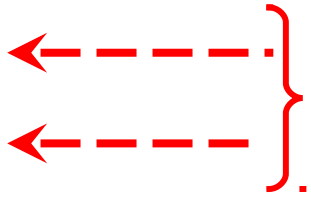
Réécriture de code (1)

le code de déchiffrement suivant en langage assembleur:

```
loc_401010:
    cmp ecx, 0
    jz short loc 40101C
    sub byteptr [eax], 30h
    inc eax
    dec ecx
    jmp short loc 401010
```

Peut être réécrit de la manière équivalente suivante :

```
loc_401010:
    cmp ecx, 0
    jz short loc 40101C
    add byte ptr [eax], <valeur aléatoire>
    sub byte ptr [eax], 30h
    sub byte ptr [eax], <même valeur aléatoire>
    inc eax
    dec ecx
    jmp short loc 401010
```



Ajout puis retrait de la même valeur aléatoire

Si la première version de code constitue la signature, la seconde ne sera pas détectée.

Réécriture de Code (2)

```
if (flag) infection();
        else charge_finale();
```

Est équivalent à

```
(flag)?infection():charge_finale();
```

obfuscation

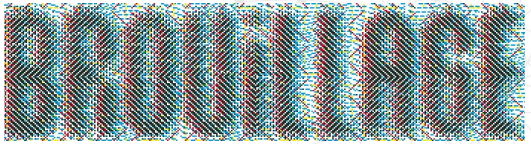
- L'obfuscation consiste à partir d'un programme P de produire un nouveau programme T(P) qui possède deux propriétés:
  - T(P) a les mêmes fonctionnalités que P. i.e T(P) calcule les mêmes fonctions que P
  - T(P) est inintelligible dans le sens où T(P) constitue une boîte noire virtuelle
- Cela revient à chiffrer du code clair avec du code clair
- Exemples en Langage C sur http://www.ioccc.org/ (*The International Obfuscated C Code Contest*)

Obfuscation

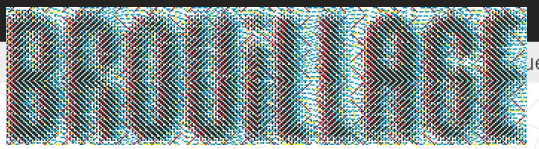
Objectifs

- Outre au changement de forme du code, l'obfuscation contribue à
  - Rendre plus difficile l'analyse du code
    - Contourner les moteurs d'analyse spectrale,
    - rendre plus difficile la tâche des désassembleurs et débogueurs
    - Détecter et contrer l'exécution pas à pas, l'exécution sous émulateur ou virtualisation
  - Camoufler certaines structures de données et certaines zones de code (par exemple les clés et algorithmes de chiffrement)

Obfuscation
Brouillage



- Un programme difficile à analyser doit posséder au moins les propriétés suivantes:
  - Le début et la fin du programme doivent être difficile à localiser
  - Le code ne possède pas de motif facilement identifiable
  - La structure des données n'est pas parlante. Aucune cohérence ne ressort de la structure des données.
- ≠Techniques
  - Optimisation de code
    - Supprimer les redondances
  - Mélange des instructions
  - Insertion de code inutile (code mort)
    - Mais attention, un code factice doit être suffisamment complexe pour ne pas être supprimé lors d'une phase d'optimisation du code compilé



Obfuscation - Brouillage

Insertion de code **(1)**

1

```
.486
.model flat,stdcall
option casemap:none
include r1.inc
.DATA
tit             db "a=",0
res             db 0
.CODE
main:
        mov     eax, 1
        push    eax
        mov     ebx, eax
        shl     ebx, 1
        add     eax, ebx
        pop     ebx
        add     eax, '0'
        mov     dword ptr [res], eax
        call    MessageBoxA, 0, offset res, offset tit, MB_OK
        call    ExitProcess, 0
end main
```

Code initial

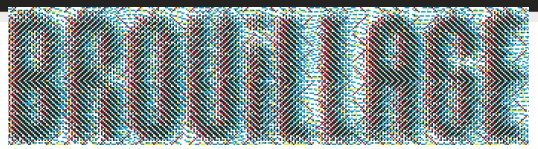
2

```
.486
.model flat,stdcall
option casemap:none
include r1.inc
.DATA
tit             db "a=",0
res             db 0
.CODE
main:
        mov     eax, 1
        push    eax
        mov     ebx, eax
        inc     ecx
        shl     ebx, 1
        sub     eax, ecx
        add     eax, ebx
        add     ecx, -1
        add     eax, ecx
        add     eax, 1
        pop     ebx
        add     eax, '0'
        mov     dword ptr [res], eax
        call    MessageBoxA, 0, offset res, offset tit, MB_OK
        call    ExitProcess, 0
end main
```

Code après ajout de code factice

Obfuscation - Brouillage

Insertion de code **(2)**



1

```
.486
.model flat,stdcall
option casemap:none
include r1.inc
.DATA
tit             db "a=",0
res             db 0
.CODE
main:
      mov   eax, 1
      push  eax
      mov   ebx, eax
      shl   ebx, 1
      add   eax, ebx
      pop   ebx
      add   eax, '0'
      mov   dword ptr [res], eax
      call  MessageBoxA, 0, offset res, offset tit, MB_OK
      call  ExitProcess, 0
end main
```

Code initial

2

```
.486
.model flat,stdcall
option casemap:none
include r1.inc
.DATA
tit             db "a=",0
res             db 0
.CODE
main:
      mov   eax, 1
      push  eax
      mov   ebx, eax
      shl   ebx, 1
      jc    L
      add   eax, ebx
      pop   ebx
      add   eax, '0'
      mov   dword ptr [res], eax
      call  MessageBoxA, 0, offset res, offset tit, MB_OK
      call  ExitProcess, 0
L:
      add   eax, ebx
      pop   ebx
      add   eax, '0'
      mov   dword ptr [res], eax
      call  MessageBoxA, 0, offset res, offset tit, MB_OK
      call  ExitProcess, 0
end main
```

Code après insertion d'1 bloc recopié depuis le code initial

Obfuscation - Transformation Intra-procédurale

Dégénérescence

1

```
#include <stdio.h>
int main() {
int a=1, b=1;
while (a<10) {
   b+=a;
   if (b>10)
        b--;
   a++;
}
printf("b=%d\n", b);
}
```

Programme initial

2

```
#include <stdio.h>
int main() {
int a=1, b=1;
L1:
  if (a>=10)
       goto L4;
  b+=a;
  if (b<=10)
       goto L2;
  b--;
L2:
  a++;
  goto L1;
L4:
printf("b=%d\n", b);
}
```

Conversion des flux de contrôle de haut niveau en If-then-goto

3

```
#include <stdio.h>
int main() {
int a, b , sw;
sw=1;
s:
switch(sw) {
case 1:
   a=1, b=1;
   sw=2;
   goto s;
case 2:
   if (a>=10)
        sw=6;
   else
        sw=3;
   goto s;
case 3:
   b+=a;
   if (b<=10)
        sw=5;
   else
        sw=4;
   goto s;
case 4:
   b--;
   sw=5;
   goto s;
case 5:
   a++;
   sw=2;
   goto s;
case 6:
   break;
}
printf("b=%d\n", b);
}
```

Dégénérescence des flux de contrôle

Obfuscation - Transformation Inter-procédurale

Modification des appels

```
#include <stdio.h>
int x;
void fonct2(){printf("fonct2\n");}
void fonct3(){printf("fonct3\n");}
void fonct1() {
      if (x>4)
            fonct2();
      else
            fonct3();
}
int main () {
      x=4; fonct1();
      x=5; fonct1();
}
```

1

Programme initial

```
#include <stdio.h>
int x;
void fonct2(){printf("fonct2\n");}
void fonct3(){printf("fonct3\n");}
void fonct1(void (*fptr1)(),
            void (*fptr2)()) {
      void (*ptr)();
      if (x>4)
            ptr=fptr1;
      else
            ptr=fptr2;
      (*ptr)();
}
int main () {
      x=4; fonct1(fonct2, fonct3);
      x=5; fonct1(fonct2, fonct3);
}
```

2

Appels de fonctions transformés

Techniques anti-détection (2)

- Furtivité
  - Activité pré-OS, stockage en RAM < 640ko
  - Activité pré-OS, hook noyau / appels systèmes
- Chiffrement
  - Chiffrement ad-hoc
  - Mieux : génération environnementale de clés
- Compression
  - Méthode de chiffrement / furtivité pour limiter l'efficacité de l'AV

ver Kelaino

La section *données* du code est chiffrée ….

```
DATA:00402799   aVvqajprXSsuqrp db 'v6-C~jPR{oc~Offi-CRPl ¢oc~Offi-Cp~066-Cu-Cufi~6-C~n=:aNmtio6fijPac~loP}ouu'
DATA:00402799   db '~uo=:}y}u]ao6uO-Cffij Pa~'=:s Cffifioffifi]aoaojP~NcfiOa~6fi_~O£ook=:PPPP'
DATA:00402799   db 'PPPPm-CNffio~6omR]]]] mA-o£fig~6fiA'"A'"eA'artubus~hrbhfs"R=:e],
DATA:00402799   db 'g60-C60fiojPc=:e]}a} ~Oc]g60-C60fiojP--C6~~c=:e]affiuoffifijPa=:e]}O~o‘
```

….par une simple addition avec la valeur constante 30H

Procédure de déchiffrement située au début du code viral

```
00401000  start             proc near
00401000  mov               ecx, addr_fin_data          ; ECX = Adresse de la fin des data
00401005  sub               ecx, addr_deb_data          ; ECX = 402D5D - 402000 = taille des data.
0040100B  mov               eax, 402000h                ; EAX = adresse du début des data
00401010
00401010  boucle_decrypte:                              ; CODE XREF: start+1A^Yj
00401010  cmp               ecx, 0          ; ECX sert de  compteur.
00401013  jz short decryptage_termine  ; tant que ecx != 0 on boucle.
00401015  sub               byte ptr [eax], 30h    ; on soustrait 30h a l'octet pointé par EAX
00401018  inc               eax             ; on passe au prochain octet à décrypter
00401019  dec               ecx             ; on décremente le compteur
0040101A  jmp               short boucle_decrypte       ; on boucle tant que ECX est différent de 0
```

Auto-détection des codes polymorphes

- Fixer la date du fichier à une valeur constante.
- Fixer la taille d'un fichier infecté à une certaine taille significative, telle qu'un multiple de 5678.
- Masquage de données.
- Créer une clé dans le base de registre de windows
- Attributs peu connus de systèmes de fichiers

Auto-détection de code polymorphe

Attributs NTFS

C:\temp> **dir cible.com**
Le volume dans le lecteur C s'appelle Mon-Doudou.
Le numéro de série du volume s'appelle DEDE-BAFE

Répertoire de C:\temp
11/07/2007 11:29 0 **cible.com**
1 Fichier(s) 0 octets
0 Rep(s) 13,797,240,832 octets libres

C:\temp> **echo oui > cible.com:infected**

C:\temp> **dir cible.com**
Le volume dans le lecteur C s'appelle Mon-Doudou.
Le numéro de série du volume s'appelle DEDE-BAFE

Répertoire de C:\temp
11/07/2007 11:30 0 **cible.com**
1 Fichier(s) 0 octets
0 Rep(s) 13,797,240,832 octets libres

C:\temp> **more < cible.com:infected**
oui

On liste
Le fichier

Ajout de L'attribut infected

Pas facile à voir

Mais bien Présent !

Partie 2 : Anti-Virus

Le théorème de Cohen

- Littérature peu abondante, codes sources quasi inexistants, protection d'un marché (juteux)
- Cause : un théorème démontré par F. Cohen
  - La détection virale est un problème indécidable
  - Conséquence 1 : il est impossible de prévenir une attaque virale « nouvelle »
  - Conséquence 2 : la complexité de la détection d'une attaque « nouvelle » implique l'utilisation de techniques probabilistes / statistiques => protection du savoir-faire
  - Conséquence 3 : la confiance accordée à un anti-virus doit rester relative...

Contrôle d'intégrité

- la modification des fichiers sensibles (exécutables, documents...) est surveillée.
- calcul une empreinte numérique de chaque fichier surveillé
- En cas de modification, la vérification de l'empreinte est négative et une infection suspectée.

Contrôle d'intégrité : les limites

- Difficile à mettre en pratique
  - Systèmes complexes où le taux de variabilité des fichiers toujours croissant rend cette prise en compte impossible en pratique.
- peut être contournée
  - Virus compagnons (pas de modification de la cible)
  - Virus furtifs (simulation d'une modification légitime par le système)
- Fausses alarmes
  - Si on surveille des fichiers dynamiques
  - environnements ou les fichiers de configuration sont susceptibles d'être modifies fréquemment.
- l'infection est détectée mais trop tard puisque c'est le résultat d'une infection qui est constate.

Techniques statiques (1)

- Recherche de signature(s)
  - Série d'octets caractéristiques d'un virus
    - Discriminante : signature spécifique à un unique virus
    - Non incriminante : ne correspond à aucun autre virus ou programme sain
  - En général séquence d'instructions ou message spécifique du virus
  - Emplacement (début, fin, offset)
  - Facile à contourner (polymorphisme, chiffrement)

Virus de test EICAR

- signature du virus de test EICAR:

X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*

- Le fichier est programme DOS, qui lorsqu'il est lancé affiche le message:

"EICAR-STANDARD-ANTIVIRUS-TEST-FILE!"

- www.eicar.org

Techniques statiques (2)

- Analyse spectrale
  - Liste des instructions « peu courantes » ou « caractéristiques » de programmes malveillants
  - Estimation de la « normalité » d'un programme par un test statistique
  - Détection possible de virus inconnus
  - Faux positifs
  - Inefficace sur du code chiffré ou compressé

Techniques statiques (3)

- Analyse heuristique
  - Étude du comportement du programme, détection de certains appels de fonctions ou séquences d'instructions utilisées par les programmes malveillants
  - Définition de règles et stratégies de décision de viralité du programme
  - Détection possible de virus inconnus
  - Faux positifs
  - Inefficace sur du code chiffré ou compressé

Techniques statiques (4)

- Contrôle d'intégrité
  - Calcul d'empreintes numériques sur chaque fichier sensible (MD5, SHA-1, CRC...)
  - Vérification de la modification par re-calcul de l'empreinte
  - Problème : efficace sur des programmes, mais pas sur des documents dont le contenu et donc la signature numérique évoluent
  - Contournable par des virus compagnons

Techniques dynamiques (1)

- Surveillance comportementale
  - Anti-virus résident en mémoire
  - Détection de comportement « suspect » (cf. techniques précédentes)
    - Accès en écriture sur programme exécutable
    - Tentative de « hook » sur des fonctions systèmes
    - …
  - Détection de virus-inconnus
  - Faux positifs

Techniques dynamiques (2)

- Emulation de code
  - Analyse statique ou type « sandbox » du code
  - Adapté aux programmes polymorphes, chiffrés ou compressés
  - Détection de virus inconnus
  - Faux positifs

Efficacité des anti-virus (1)

- Sur les virus « connus »
  - Base de données de références
  - Taux proche de 100%
  - Peu de fausses alarmes
- Sur les virus « inconnus »
  - 80% si utilisation de techniques de protection connues
  - Sinon, pas de statistiques...
  - La réalité montre que les virus novateurs sont conçus avec des mécanismes d'anti-détection efficaces...

Efficacité des anti-virus (2)

- Détection de vers
  - Très difficile (Nimda, Klez, BugBear...)
  - Anticipation quasi impossible
  - Désinfection locale avec des outils spécifiques
  - Dissémination distribuée => risque de ré-infection rapide
- L'utilisation d'anti-virus est nécessaire (pour filtrer les programmes malveillants connus) mais non suffisante (pour contrer l'apparition de programmes innovants)
- Mises à jour quotidiennes au minimum

Conduite à tenir en cas d'infection (1)

- Absence de détection par un anti-virus
  - Phénomènes visibles anormaux symptomatiques
  - Isolation du système
  - Sauvegarde (en espérant qu'une MAJ de l'AV permettra la récupération des données)
  - Analyse en détail
  - Récupération des données à partir d'une source sûre et antérieure à l'infection
  - Mesures post-infection identiques

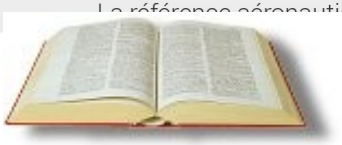
Conduite à tenir en cas d'infection (2)

- Détection par un anti-virus
  - Attention aux faux positifs
  - Isoler du réseau la machine incriminée
  - Sauvegarde des données (risque de sauvegarder des données corrompues/infectées)
  - Passer l'antivirus en mode éradication
  - Éteindre la machine, puis rallumer et relancer l'AV
    - Si infection persistante : formatage bas niveau des disques
  - Appliquer les mesures de compromission
    - Changement des mots de passe...
    - Correctifs de sécurité
  - Information des acteurs de la sécurité (RSSI...)

A vous de jouer (1)

- Proposez une méthode de détection de votre virus bash
- Implémentez en bash un détecteur de votre virus par signature
  - Lorsqu'il détecte un hôte infecté, faites-en la désinfection

Bibliographie



- Linux magazine
  Hors série numéro 32: Virus
- MISC (Multisystem & Internet Security Cookbook) magazine bimestriel
  Numéro 20 – polymorphisme/obfuscation/blindage de code
- Les virus informatiques : théorie, pratique et applications de Eric Filiol chez les éditions Springer
- Les virus informatiques : techniques virales et antivirales avancées de Eric Filiol aux Editions Springer, collection IRIS, 2007.

Conclusion

- Utilisation de techniques (cryptographie, obfuscation, programmation..) avancées

- Mais les virus innovants sont somme toute assez rare
  - Dans la majorité des cas, réutilisation de techniques connues

- La mise à jour des antivirus ne concerne pas seulement la base de signature, une m.a.j du moteur antiviral est quelquefois nécessaire
  - Car si les éditeurs de solutions antivirales analysent les malwares, les auteurs de codes malveillants analysent de leur côté les codes antiviraux ce qui leur permet de connaitre par exemple la base de comportements
  - tout le jeu du programmeur de virus consistera à utiliser cette connaissance pour mieux contourner la protection.