SLO

# Sécurité logicielle

## Chapitre 4.
## Reverse engineering logiciel

Sylvain Pasini
sylvain.pasini@heig-vd.ch
Mars - Juillet 2014

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

## Aperçu du cours

1. Introduction
2. Top 25 des erreurs de sécurité logicielle
3. Rappel d'architecture des systèmes à processeurs
4. **Reverse engineering de logiciels**
5. Attaques logicielles
6. Protections logicielles
7. Développement sécurisé

# Aperçu

4. **Reverse engineering logiciel**
   - 4.1. Définition et introduction
   - 4.2. Objectifs (légaux / illégaux)
   - 4.3. Aspect légal
   - 4.4. Rétro-ingénierie logicielle
   - 4.5. Outils courants (Linux)



# Aperçu

4. **Reverse engineering logiciel**
   - 4.1. **Définition et introduction**
   - 4.2. Objectifs (légaux / illégaux)
   - 4.3. Aspect légal
   - 4.4. Rétro-ingénierie logicielle
   - 4.5. Outils courants (Linux)

# Reverse engineering

- Aussi appelé :
  - rétro-ingénierie, rétro-conception, ingénierie inversée
- Définition :
  - étudier un objet pour en **déterminer le fonctionnement interne** ou la **méthode de fabrication**

- Vieux film (BIOS d'IBM) :
  - http://www.dailymotion.com/video/x3i8at_retroingenierie_tech



# Aperçu

4. **Reverse engineering logiciel**
   - 4.1. Définition et introduction
   - **4.2. Objectifs (légaux / illégaux)**
   - 4.3. Aspect légal
   - 4.4. Rétro-ingénierie logicielle
   - 4.5. Outils courants (Linux)

# Objectifs (généralement légaux)

- comprendre le fonctionnement (utiliser, modifier, ...)
- valider la sortie d'un compilateur
- information de debug
- analyse d'interopérabilité
- apprentissage
- curiosité
- veille concurrentielle
  - détection d'éventuelles violations de brevets
- analyse de maliciels



# Analyse de produits / brevets

- un brevet est forcément public,
  donc accessible à tous les concurrents
- très difficile de savoir si un concurrent utilise un de "nos" brevets
- le **reverse engineering** est parfois la seule solution pour savoir si un produit viole un brevet !
- processus très couteux
  - temps
  - argent
  - tout cela, sans être certain de parvenir à notre fin...

# Interopérabilité

- Plusieurs logiciels pouvant communiquer
  - compatibilité : peut fonctionner dans un environnement donné
- Logiciel livré sous forme de code binaire
  - Les fabricants peuvent ne pas fournir de documentation
- Difficile d'écrire un logiciel interopérable ou un plugin
- Nécessité de réaliser du **reverse engineering**...
- Exemples historiques :
  - BIOS (sur les PC, ROM)
  - Samba (étude du protocol pour le porter sur d'autres plateformes)
  - OpenOffice (étude du format de fichier pour le porter sur d'autres plateformes)
  - Wine



# Valider la sortie d'un compilateur

- Il est parfois nécessaire de vérifier ce qu'a produit le compilateur.
  - optimisation
  - exactitude
  - sécurité
- Lorsque l'on écrit du code sensible, il y a quelques feintes sécuritaires...
- Exemples,
  - des boucles "qui ne servent fonctionnellement à rien" (masquage, obfuscation, détection de fautes, ...)
    - vérifier que les boucles n'aient pas été supprimées / optimisées
  - en crypto, on souhaite du code "temps constant" / sans condition
    - vérifier qu'il n'y ait pas d'ajout de sauts conditionnels ou autres

# Analyse de maliciels

- Les auteurs de maliciels fournissent rarement le code source...
- Les **reverse engineering** est forcément nécessaire pour comprendre le fonctionnement du maliciel.

- Analyse statique
  - Comprendre son fonctionnement en étudiant son code exécutable.
- Analyse dynamique
  - Comprendre son fonctionnement en l'exécutant dans un environnement contrôlé (émulé / virtualisé) et en étudiant le tout.



# Objectifs (généralement illégaux)

- fabriquer une contre-façon (généralement illégale)
- fabriquer un nouveau produit ayant des fonctionnalités identiques
  - éviter les couvertures de brevets
  - piratages, emulateurs, ...
- espionnage industriel
  - vol de produits, d'algorithmes, de logiciels, ...
- espionnage militaire
- recherche de vulnérabilités, de backdoors, conception d'exploits
- piratage de logiciel
  - retirer les protections, patch, keygen, ...

- et d'autres

# Conception d'exploits

- 3 phases :
  1. découverte de vulnérabilités
     - fuzzing, statistique, ...
  2. analyse de vulnérabilités
     - **reverse engineering...**
  3. conception de l'exploit



# Piratage de logiciels (cracking)

- Pour "casser" un logiciel, il est souvent nécessaire de :
  - trouver un numéro de série
    - copie facile
    - le reverse permet de faire sauter la vérification
  - trouver un numéro de série pas encore activé (clé d'activation)
    - le reverse permet de connaître l'algo interne et de construire un keygen
    - le reverse permet de faire sauter la vérification
- En résumé, le reverse est utile pour
  - modifier des variables de l'application
    - activated = true (au lieu de false)
  - sauter / retirer le code de vérification
  - retrouver l'algorithme de vérification
  - et autres

# Aperçu

4. **Reverse engineering logiciel**
   - 4.1. Définition et introduction
   - 4.2. Objectifs (légaux / illégaux)
   - **4.3. Aspect légal**
   - 4.4. Rétro-ingénierie logicielle
   - 4.5. Outils courants (Linux)



# Aspect légal

- Domaine complexe...
  - Loi Droit d'Auteur, Loi des Brevets, Loi Concurrence Déloyale, droit communautaire, ...
- Le droit communautaire permet la traduction, l'adaptation, l'arrangement et toute autre *transformation* d'un programme d'ordinateur lorsque ces actes sont nécessaires pour **permettre à l'acquéreur légitime de l'utiliser d'une manière conforme à sa destination**, y compris pour corriger des erreurs.
- La question de savoir s'il est permis de procéder à l'ingénierie inverse pour assurer l'utilisation du logiciel reste cependant **controversée**.
- Si par exemple l'analyse d'un élément du programme est indispensable pour la **réparation de défauts** ou pour la d**écouverte de lacunes de sécurité** et les informations nécessaires ne sont pas accessibles d'une autre façon, **l'ingénierie inverse peut être admise *ultima ratio***.
- Une analyse motivée par des **soucis de piratage** ou par pure **curiosité** n'est en revanche pas couverte par la condition de l'utilisation conforme.
- Source :
  - http://www.advobern.ch/files/aufsaetze/I%20ingenierie_inverse_et_la_propriete_intellectuelle.pdf

# Aperçu

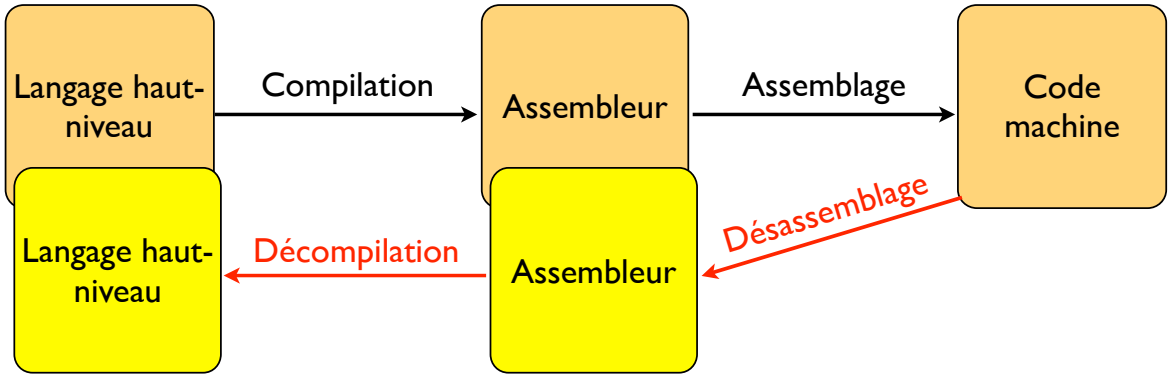
4. **Reverse engineering logiciel**
   - 4.1. Définition et introduction
   - 4.2. Objectifs (légaux / illégaux)
   - 4.3. Aspect légal
   - **4.4. Rétro-ingénierie logicielle**
   - 4.5. Outils courants (Linux)



# Reverse d'un logiciel

- Analyse statique
  - Processus permettant d'évaluer un programme ou une partie d'un programme en se basant sur sa **forme**, sa **structure**, son **contenu** ou sa **documentation**.
  - Analyse par "lecture", pas d'exécution.
  - Outils : désassembleur, décompilateur
- Analyse dynamique
  - Processus permettant d'évaluer un programme ou une partie d'un programme en se basant sur son **comportement à l'exécution**.
  - Analyse du comportement
  - Outils : débogueurs

# Reverse par analyse statique






# Reverse par analyse statique

- Désassembleur :
  - A partir d'un code binaire, il permet de retrouver le code assembleur.
  - En général, il n'y a rien de plus :-(
  - Certains désassembleurs performants permettent
    - de retrouver les types des variables
    - de renommer les variables
  - Exemples : disas de GDB, disas d'objdump, IDA
- Décompilateur :
  - A partir d'un code assembleur, il permet de retrouver "un" code source.
  - Exemples : Hex-Rays Decompiler (IDA), Jad (Java), Reflector (.NET)

# Reverse par analyse dynamique

- Débogueur
  - permet de déboguer un logiciel, soit
    - contrôler l'exécution, mettre des arrêts,
    - suivre le déroulement pas à pas,
    - lire/modifier les registres, la mémoire
  - meilleure analyse grâce à l'observation en temps réel
  - possibilité de modifications en cours d'exécution
    - registres, mémoires (stack, heap, instructions, ...)
    - exemple sauter une routine de vérification de licence
  - plus intrusifs que l'analyse passive

  - Exemples : GDB, OllyDbg, WinDbg, IDA Pro



# Aperçu

4. **Reverse engineering logiciel**
   - 4.1. Définition et introduction
   - 4.2. Objectifs (légaux / illégaux)
   - 4.3. Aspect légal
   - 4.4. Rétro-ingénierie logicielle
   - **4.5. Outils courants (Linux)**

# Outils utiles

- file
  - indique le type de fichier
- strings
  - affiche tous les caractères imprimables d'un executable
- objdump
  - désassembleur (objdump -d)
- gdb (Gnu Debugger)
  - débogueur avec désassembleur intégré (disas)
- nm
  - affiche tous les symbols
- ldd
  - affiche les librairies partagées
- strace / ltrace
  - trace les appels systèmes / les appels aux librairies



# GDB : GNU Debugger

- Un débogueur pour plusieurs langages (C, C++, …)
- Permet d'inspecter précisément un programme en temps réel
  - Registres
  - Mémoire
  - Exécution / code
- Très pratique pour le développement, recherche d'erreurs
- Très pratique pour le reverse, comprendre le fonctionnement
- Pas très « user friendly ».

# IDA

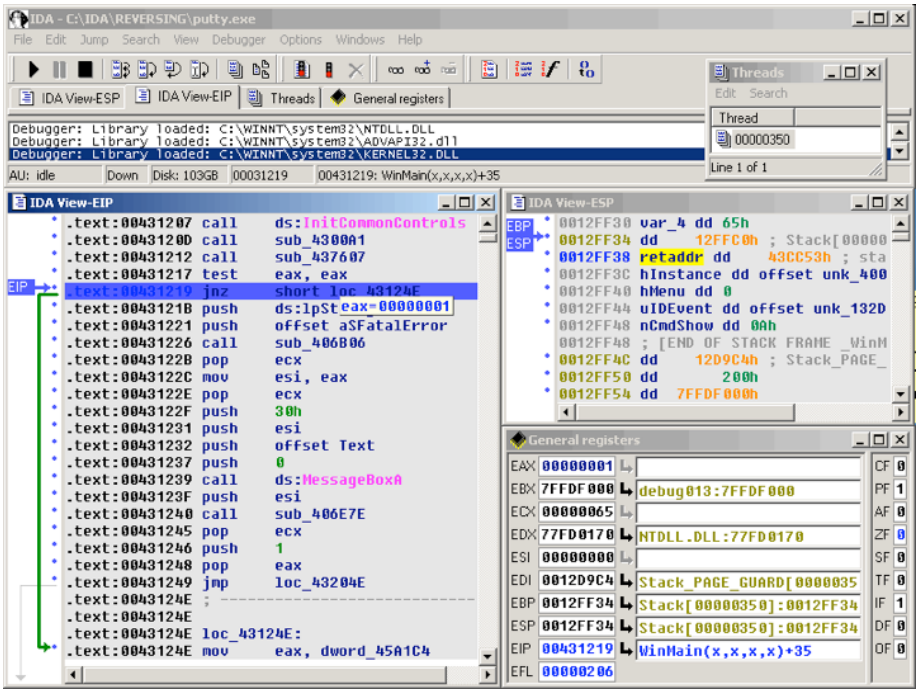
- IDA : Interactive DisAssembler
  - Désassembleur
  - Débogueur (interface)
- Interactivité
  - possibilité de naviguer dans le code ASM et de l'annoter, modifier, ....
- Le meilleur au monde !
- Disponible pour Linux, Mac OS et Windows
- Versions actuelles :
  - IDA **Free** : gratuit, IDA version 5.0 (23 mars 2006)
    (http://www.hex-rays.com/products/ida/support/download.shtml)
  - IDA **Pro** : payante, IDA version 6.4 (6 mars 2013)



# IDA

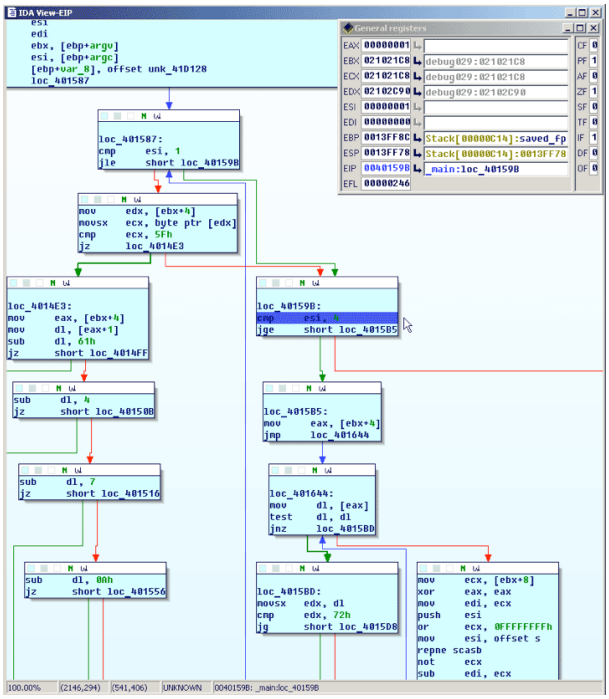
- Il permet une analyse très fine du code, de l'utilisation des registres et de la pile (stack).
- Cela permet d'isoler les fonctions, relations, variables globales et locales, etc.
- Il existe des plugins coûteux...

# IDA


IDA - C:\IDA\REVERSING\putty.exe
File Edit Jump Search View Debugger Options Windows Help
IDA View-ESP
IDA View-EIP
Threads
General registers
Debugger: Library loaded: C:\WINNT\system32\NTDLL.DLL
Debugger: Library loaded: C:\WINNT\system32\ADVAPI32.dll
Debugger: Library loaded: C:\WINNT\system32\KERNEL32.DLL
AU: idle
Down
Disk: 103GB
00031219
00431219: WinMain(x,x,x,x)+35
IDA View-EIP
.text:00431207 call ds:InitCommonControls
.text:0043120D call sub_4300A1
.text:00431212 call sub_437607
.text:00431217 test eax, eax
.text:00431219 jnz short loc_43124E
.text:0043121B push ds:lpSt eax=00000001
.text:00431221 push offset aSFatalError
.text:00431226 call sub_406B06
.text:0043122B pop ecx
.text:0043122C mov esi, eax
.text:0043122E pop ecx
.text:0043122F push 30h
.text:00431231 push esi
.text:00431232 push offset Text
.text:00431237 push 0
.text:00431239 call ds:MessageBoxA
.text:0043123F push esi
.text:00431240 call sub_406E7E
.text:00431245 pop ecx
.text:00431246 push 1
.text:00431248 pop eax
.text:00431249 jmp loc_43204E
.text:0043124E loc_43124E:
.text:0043124E mov eax, dword_45A1C4
IDA View-ESP
0012FF30 var_4 dd 65h
0012FF34 dd 12FFC0h ; Stack[00000
0012FF38 retaddr dd 43CC53h ; sta
0012FF3C hInstance dd offset unk_400
0012FF40 hMenu dd 0
0012FF44 uIDEvent dd offset unk_132D
0012FF48 nCmdShow dd 0Ah
0012FF48 ; [END OF STACK FRAME _WinM
0012FF4C dd 12D9C4h ; Stack_PAGE_
0012FF50 dd 200h
0012FF54 dd 7FFDF000h
General registers
EAX 00000001
EBX 7FFDF000 debug013:7FFDF000
ECX 00000065
EDX 77FD0170 NTDLL.DLL:77FD0170
ESI 00000000
EDI 0012D9C4 Stack_PAGE_GUARD[0000035
EBP 0012FF34 Stack[00000350]:0012FF34
ESP 0012FF34 Stack[00000350]:0012FF34
EIP 00431219 WinMain(x,x,x,x)+35
EFL 00000206



# IDA


IDA View-EIP
General registers
loc_401587:
cmp esi, 1
jle short loc_40159B
mov edx, [ebx+4]
movsx ecx, byte ptr [edx]
cmp ecx, 5Fh
jz loc_4014E3
loc_4014E3:
mov eax, [ebx+4]
mov dl, [eax+1]
sub dl, 61h
jz short loc_4014FF
loc_40159B:
cmp esi, 4
jge short loc_4015B5
sub dl, 4
jz short loc_40150B
loc_4015B5:
mov eax, [ebx+4]
jmp loc_401644
sub dl, 7
jz short loc_401516
loc_401644:
mov dl, [eax]
test dl, dl
jnz loc_4015BD
sub dl, 0Ah
jz short loc_401556
loc_4015BD:
movsx edx, dl
cmp edx, 72h
jg short loc_4015D8
mov ecx, [ebx+8]
xor eax, eax
mov edi, ecx
push esi
or ecx, 0FFFFFFFFh
mov esi, offset s
repne scasb
not ecx
sub edi, ecx