

Introduction au Reverse Engineering

Airbus

Formation **TLS-SEC**, Toulouse, 2025/2026

The Airbus logo, consisting of the word "AIRBUS" in a bold, blue, sans-serif font.

Auteurs originaux

Jean-Baptiste Bédrune (Apple), Alexandre Gazet (Apple)

Améliorations et corrections

Raphaël Rigo (Apple), Julien Lenoir (Apple), Damien Aumaitre (Qb), Sébastien Renaud (SAFA), Sylvain Peyrefitte (Airbus), Simon Garrelou (Airbus) (et d'autres très probablement)

- Analyse en boîte noire d'un système en vue d'en comprendre le fonctionnement interne
- Idée de reconstruire une représentation dans un niveau d'abstraction supérieur
- IT sec: traditionnellement appliqué sur du code binaire

Préambule

- Ce cours est (un peu) barbant (rappels architecture) :)
- Seules les architectures x86/x86-64 seront étudiées
- Orienté vers les environnements Windows
- Posez des questions!

Aperçu

- Pourquoi un cours sur le reverse-engineering ?
- Architecture Intel IA32
- Assembleur x86/x86-64
- Le format PE
- Les outils

Part I

Motivations

Histoire

Défense

Attaque

Entre les deux

Dans les années 80/90s

- Cracking et démo¹ scene Amiga, Apple II², Atari, puis DOS, *etc.*
- Virus informatique³

¹<http://www.pouet.net/>

²<https://www.filfre.net/2015/12/a-pirates-life-for-me-part-1-dont-copy-that-floppy/>

³<https://archive.org/details/malwaremuseum&tab=collection>



⁴Source : <https://www.deviantart.com/demoscene-amiga/art/Amigascene-LogoMedley-202900140>



PARADOX ★

--> PARADOX <--

CAUSED ANOTHER VICTIM TODAY!

BANSHEE AGA FROM CORE DESIGN

CRACKED BY e.F.A.

SUPPLIED BY . HOT TUNA+MIMICOM

FADE

Cracking scene jeux/shareware

- Écosystème des shareware Windows
- Développement des outils (debuggers, patcher)
- Développement des compressors/packers/unpackers (AsProtect, Armadillo, etc.)
- Exemples de protections : SecuROM, SafeDisc, StarForce

Mais aussi hack/exploit (monde Unix)

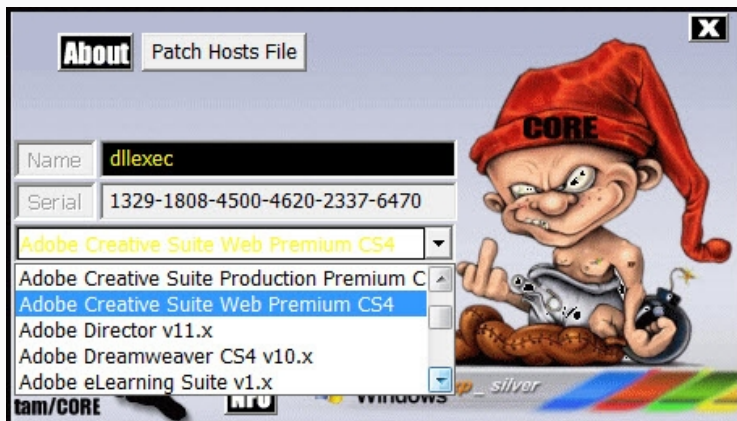
- Phrack⁵
- Smashing The Stack For Fun And Profit, Aleph One
- The Last Stage of Delirium⁶ (LSD)

⁵<http://www.phrack.org/>

⁶<http://lsd-pl.net/>







Changements importants, mi-2000

- Internet généralisé (say hi to Blaster)
- Numérisation++
- Puissance de calcul++
- Prise de conscience autour de la sécurité informatique
- Se développent également:
 - La distribution dématérialisée (musique, films, etc.)
 - Et son corollaire, les DRM
 - Plus les mêmes acteurs, ni les mêmes enjeux

Et c'est là que vous arrivez

- De plus en plus d'opportunités professionnelles

Histoire

Défense

Attaque

Entre les deux

Entités

- Vendeurs de solutions anti-virus
- Security Operation Center (SOC)/ Computer Emergency Response Team (CERT)⁷
- Product Security Incident Response Team (PSIRT)
- Éditeurs jeux-vidéos/solution anti-cheats ⁸
- “Étatiques” : ANSSI

Missions

- Analyse de malware, exploits, incidents, etc.
- Analyse bot, wall/aim/* hack, cheat-engine, développement de contre-mesures
- Forensic/investigation numérique

⁷<https://www.ssi.gouv.fr/agence/cybersecurite/ssi-en-france/les-cert-francais/>

⁸[https://recon.cx/2018/brussels/resources/slides/RECON-BRX-2018-Starcraft-Emulating-a-buffer-overflow-for-fun-](https://recon.cx/2018/brussels/resources/slides/RECON-BRX-2018-Starcraft-Emulating-a-buffer-overflow-for-fun-and-profit.pdf)

Entités

- Security Operation Center (SOC)/ Computer Emergency Response Team (CERT)⁹

Missions

- Analyse de malware, exploits, incidents, etc.
- Forensic/investigation numérique

C'est nous!

- Équipe interne de réponse à incident pour le groupe Airbus
- Airbus, suppliers, affiliates
- Publis : <https://skyblue.team> et outils : <https://github.com/airbus-cert>

⁹ <https://www.ssi.gouv.fr/agence/cybersecurite/ssi-en-france/les-cert-francais/>

Histoire

Défense

Attaque

Entre les deux

Entités

- Très nombreuses boîtes de services/pentests/red team
- Boîtes d'intelligence économique
- "Étatiques" : DGSE¹⁰, DGSI, DGA, *etc.*

Missions

- Pentest/outillage
- Recherche de vulnérabilité
- Développement d'exploit
- Lutte Informatique Offensive (LIO^{11 12})

¹⁰ https://www.sstic.org/2018/presentation/2018_cloture/

¹¹ <http://www.livreblancdefenseetsecurite.gouv.fr/>

¹² <https://www.senat.fr/rap/r07-449/r07-4496.html>

Histoire

Défense

Attaque

Entre les deux

Modders/jailbreak

- Téléphones (ex : evad3rs)
- Consoles^{13 14} (jailbreak/émulation)
- Véhicules (interfaces propriétaires)
- IoT : aka *Junk Hacking*
- *etc.*

Enjeux

- Problématique générale du contrôle/verrouillage du hardware par le software
- Savoir ce que font vraiment les produits que vous utilisez

¹³ [https://recon.cx/2018/brussels/resources/slides/RECON-BRX-2018-Mess-with-the-best-die-like-the-rest_\(mode\).pdf](https://recon.cx/2018/brussels/resources/slides/RECON-BRX-2018-Mess-with-the-best-die-like-the-rest_(mode).pdf)

¹⁴ <https://fail0verflow.com/blog/>

Test subject #2: Miele G 651 I PLUS-3

- **Step 1:** finding diagnostic keys

- Results: 4660 and 22136 ✓

↓ ↓
(0x1234) (0x5678)



¹⁵ <https://media.ccc.de/v/39c3-hacking-washing-machines>

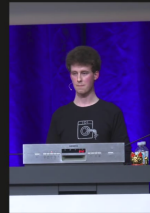
FreeMDU Home

- Smart home integration via MQTT
- Supports HA's MQTT autodiscovery
- **Almost no configuration required!**

Washing Machine

	Start Program	Press
	Display Contents	1.49
	Program Options	PrefWash WaterPlus
	Program Phase	Idle
	Program Spin Setting	SpinMax
	Program Temperature	75 °C
	Program Type	Cottons

 **POWER CYCLES**



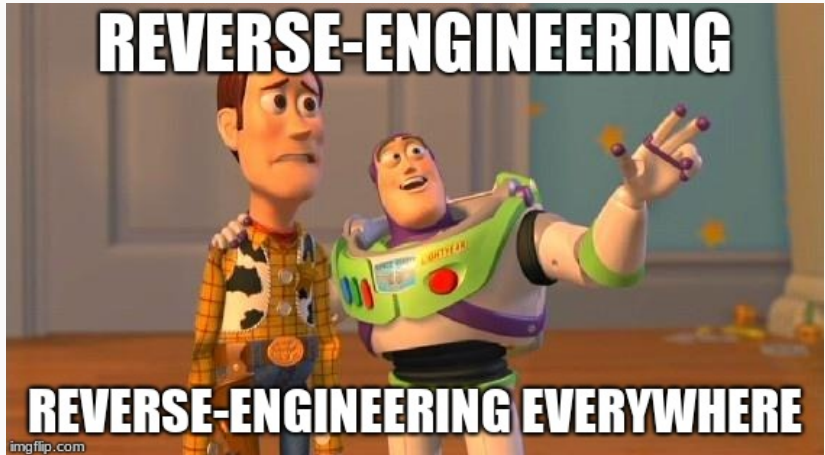
I then started exploring the Freebox.

Eventually found the library to control the front panel display.



30

¹⁶ <https://media.ccc.de/v/39c3-set-top-box-hacking-freeing-the-freebox>



Part II

Le format PE (Portable Executable)

Matière première

- Programmes informatiques distribués sous forme d'exécutables
- Code source $\Rightarrow$ chaîne de compilation $\Rightarrow$ exécutable

Objectifs

- Quelques bases sur le format PE (Windows)
- Concepts similaires dans le format ELF
- Transition fichier sur le système de fichiers vers un programme s'exécutant en mémoire

PE header, COFF header et Optional header

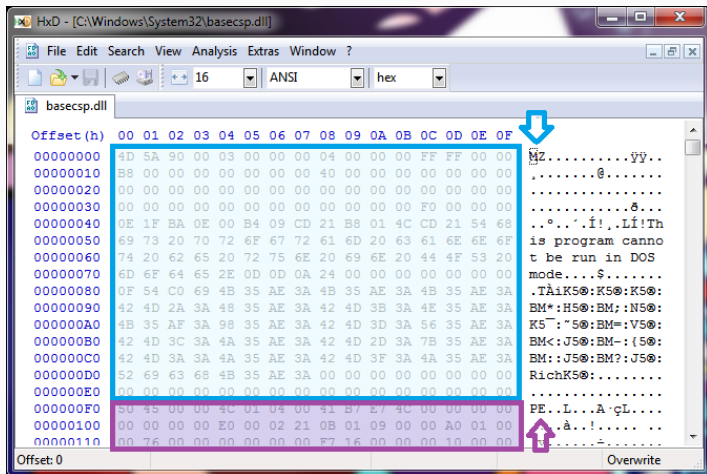
En-tête PE

Mini TP PE

Les protections autour du binaire Windows

Le loader Windows

- Format des fichiers exécutables : le format PE
- Introduit par Microsoft en même temps que Windows NT 3.1
- Initialement conçu à partir du format UNIX COFF
- L'extension 64-bit s'appelle PE+
- Deux signatures incontournables : **MZ** et **PE**:
 - Le format PE débute par un en-tête DOS, conservé uniquement pour des raisons de compatibilité.
 - Il est suivi par l'en-tête *utile* : le PE header.
- L'en-tête PE contient toutes les informations destinées au **chargeur dynamique d'exécutable** (*loader*).



PE header, COFF header et Optional header

En-tête PE

Mini TP PE

Les protections autour du binaire Windows

Le loader Windows

L'en-tête PE de type `IMAGE_NT_HEADER` référence un champ `IMAGE_OPTIONAL_HEADER` qui n'a rien d'optionnel car la plupart des informations essentielles y sont stockées, telles que :

- Le point d'entrée : adresse de la première instruction.
- L'image base : adresse favorite à laquelle le binaire sera chargé en mémoire.
- La taille du code.
- Une structure `IMAGE_DATA_DIRECTORY`, elle aussi très importante pour les imports, exports, relocations ...

Les informations à propos des sections sont stockées de l'en-tête PE dans la structure `IMAGE_FILE_HEADER`.

Tutoriel, Lord Julius, 1998

If the adress is relative to something, adding that something to it will remove the relativity.

Point sémantique

- Adresse virtuelle (**VA**) : désigne l'adresse en mémoire.
- *Image base* : **VA** à laquelle est chargé (mappé) le binaire.
- Adresse virtuelle relative (**RVA**) : adresse virtuelle relative à l'*image base*.

Tutoriel, Lord Julius, 1998

If the adress is relative to something, adding that something to it will remove the relativity.

Point sémantique

- Adresse virtuelle (**VA**) : désigne l'adresse en mémoire.
- *Image base* : **VA** à laquelle est chargé (mappé) le binaire.
- Adresse virtuelle relative (**RVA**) : adresse virtuelle relative à l'*image base*.

Traité de mathématiques appliquées

- La plupart des adresses présentes dans l'en-tête PE sont des **RVA**.
- $VA = RVA + ImageBase$

- Une section est un conteneur logique.
- Elle est décrite par un `IMAGE_SECTION_HEADER`.
- Un jeu de flags spécifie ses propriétés : R/W/X ...
- Elle est identifiée par un nom, le plus souvent lié au type de données qu'elle contient :
 - `.CODE` ou `.text` : segments de code, le point d'entrée se situe dans cette section (*a priori*).
 - `.bss` : données non initialisées, sa taille sur le disque dur est nulle.
 - `.data` : variables globales.
 - `.rsrc` : contient les ressources (icônes, images, ...).
 - etc.

La convention de nommage n'est donnée qu'à titre indicatif, néanmoins la majorité des compilateurs respectent ces conventions.

Tout binaire interagit d'une manière ou d'une autre avec le système d'exploitation. Pour cela il utilise l'API fournie par ce dernier.

- Sur un noyau Windows cette interface a été découpée en bibliothèques (*dynamic link library* ou DLL) comme `kernel32.dll`.
- La table d'imports permet au binaire de spécifier les fonctions dont il a besoin. Pour cela il spécifie le nom de la bibliothèque et de chacune des fonctions requises.
- Dans l'en-tête PE, les imports sont décrits à l'aide de la structure `IMAGE_DIRECTORY_ENTRY_IMPORT`.

Un binaire peut souhaiter rendre disponible une partie de son code à d'autres binaires. Ce cas de figure concerne avant tout les bibliothèques qui sont alors compilées sous forme de DLL.

Une interface dédiée est utilisée : la table d'exports, décrite à l'aide de la structure `IMAGE_DIRECTORY_ENTRY_EXPORT`.

- Deux possibilités pour exporter une fonction :
 - par nom, comme `kernel32.dll` qui exporte `ExitProcess`.
 - par ordinal : un entier (16-bits) identifie la fonction exportée au sein de la DLL de manière unique.
- La structure `IMAGE_EXPORT_DIRECTORY` permet de faire le lien entre une fonction exportée et son adresse dans le binaire.

Un binaire ne va pas connaître à l'avance l'adresse de base à laquelle il va être chargé. Certaines instructions sont dépendantes de cette position et doivent être recalculées lors du chargement. C'est ce que l'on appelle "relocation".

Il faut le voir comme un ensemble d'instructions de relocation qui seront appliquées lors du chargement en mémoire. Cette ensemble d'instructions est stocké dans la *table de relocation*, décrite à l'aide de la structure `IMAGE_DIRECTORY_ENTRY_BASERELOC`.

Le malware Relock utilisait cette table en tant que packer. En effet, sa section de code était vide, sa table de relocation contenait un ensemble d'instructions permettant de reconstruire le code du malware. Il utilisait aussi un bug dans Windows qui, quand l'adresse de base demandée par le programme était "0", chargeait le programme automatiquement à 0x40000, ce qui constituait un détournement de l'ASLR.

PE¹⁰¹ a windows executable walkthrough

Ange Albertini
corkami.com

Dissected PE

The diagram illustrates the internal structure of a Windows Portable Executable (PE) file, specifically 'simple.exe'. It is organized into several main sections, each with a corresponding hex dump and assembly code view.

- Header:** Contains the DOS header, PE header, optional header, data directories, sections table, code, imports, and data. The PE header includes fields like Signature, Machine, NumberOfSections, and Characteristics.
- Sections:** Lists sections such as .text, .data, .bss, .rsrc, and .reloc. Each section has a name, virtual size, virtual address, pointer to raw data, and characteristics.
- Imports:** Shows the import table, which lists the modules and functions imported by the executable.
- Exports:** Shows the export table, which lists the functions exported by the executable.
- Assembly:** Provides a detailed view of the assembly code for the .text section, showing instructions like push, mov, call, and ret.
- Imports structures:** Details the import structures for the kernel32.dll module, including the IAT (Import Address Table) and the IAT_Thunk (Import Name Table).
- Consequences:** Explains the consequences of the imports, such as the need to load the kernel32.dll module and the location of the IAT_Thunk.

The diagram also includes a 'simple.exe' icon and a 'header' section with a 'technical details about the execution' link. The 'sections' section includes a 'contents of the executable' link. The 'imports' section includes a 'list before the execution and (optional) character' link. The 'data' section includes a 'list before the execution and (optional) character' link.

Récapitulatif détaillé

OpenRCE's Portable Executable Format Layout

PE header, COFF header et Optional header

En-tête PE

Mini TP PE

Les protections autour du binaire Windows

Le loader Windows

`https://tls-sec.skyblue.team/mini.html`

Pour chaque binaire, à l'aide de Detect-it-Easy:

- noter l'architecture cible
- retrouver la version du *linker* et la date de compilation
- trouver la section la plus grosse: 1) sur le disque, 2) en mémoire
- noter un nom de DLL importée et un nom de fonction importée

PE header, COFF header et Optional header

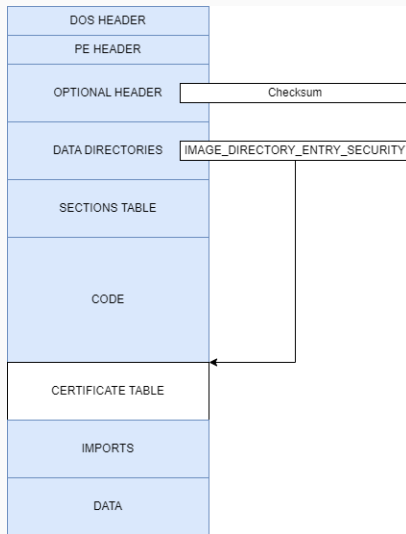
En-tête PE

Mini TP PE

Les protections autour du binaire Windows

Le loader Windows

- Sous Windows, on ne distingue pas les exécutables et les librairies, on parle de *module*
- Il est possible de signer chaque module afin d'être vérifié lors du chargement
- La signature se calcule en excluant
 - Le champ checksum présent dans IMAGE_OPTIONAL_HEADER
 - L'entrée IMAGE_DIRECTORY_ENTRY_SECURITY des DATA DIRECTORIES, qui contient l'adresse de la table des certificats
 - Et enfin la table des certificats définie dans la CERTIFICATE TABLE précédente



- Le fait d'exclure des parties du PE lors du calcul amène naturellement une zone d'ombre
- C'est l'objet de SigFlip.exe^{17 18}

¹⁷SigFlip : <https://github.com/med0x2e/SigFlip>

¹⁸Blackhat : <https://www.blackhat.com/docs/us-16/materials/us-16-Nipravsky-Certificate-Bypass-Hiding-And-Executing-Malware-From-A-Digitally-Signed-Executable-wp.pdf>

PE header, COFF header et Optional header

En-tête PE

Mini TP PE

Les protections autour du binaire Windows

Le loader Windows

- Composant essentiel et pourtant son utilisation est transparente : appel implicite lors du lancement d'un exécutable.
- Utilise les informations contenues dans l'en-tête PE pour assurer le chargement en mémoire du binaire.
- Responsable de la création des objets internes à l'OS.
 - Exemple : les objets *Process*, *Thread*.
- Notifie le système de la création et démarre le thread principal.

- Un processus $\Rightarrow$ un espace d'adressage virtuel indépendant (cf. CR3)
- Projection dans cet espace mémoire : ce n'est pas une simple copie.
 - Mapping des sections
 - Création de zones mémoire avec les attributs correspondants
 - Exemple : la section BSS (variables non initialisées).
- Le binaire est chargé à une adresse préférentielle : l'*image base*. Historiquement 0x400000 (mais PIE/ASLR peuvent le changer)
- Si le binaire n'a pu être chargé à son adresse préférentielle : utilisation des informations de la section `.reloc`.

Le loader charge toutes les bibliothèques nécessaires¹⁹ dans l'espace du processus :

- Il consulte la table des imports du processus en cours de création.
- Pour chaque DLL nécessaire, il la place dans l'espace d'adressage du processus et consulte sa table d'exports afin de déterminer les adresses des fonctions importées.
- Il assigne alors les champs de la table d'imports du processus avec ces adresses. Cette zone est appelée l'*import address table* (IAT).

¹⁹Pour aller plus loin:

<https://docs.microsoft.com/en-us/windows/desktop/dlls/dynamic-link-library-search-order>

Assez proche d'un PE sur le fond:

- le *file header* contient les informations critiques : architecture cible, *endianness*, point d'entrée, etc.
- les *program headers* définissent l'interpreteur (ld.so) et la cartographie mémoire,
- les *section headers* (optionnels) définissent les informations supplémentaires: bibliothèques à charger, symboles externes, relocations, etc.

```
$ ldd /bin/ls
linux-vdso.so.1 (0x00007ffdd65cb000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fa617bd1000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fa617832000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fa6175bf000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fa6173bb000)
/lib64/ld-linux-x86-64.so.2 (0x00007fa61801a000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fa61719e000)
```

Commande pour regarder un ELF: `readelf -a`

Quelques différences:

- Concept d'*interpréteur*:
 - Si `PT_INTERP` existe, le noyau charge l'exécutable pointé par cette entrée. Il est chargé de gérer lui-même la résolution des bibliothèques dynamiques.
 - Sinon, le noyau mappe les sections et jump au point d'entrée du ELF directement.
- Un exécutable statique ne dépend d'aucune bibliothèque externe: les appels système se font au travers de `syscall`.
 - Impossible sous Windows: les `syscall` ne sont pas stables d'une version à l'autre. On passe par des *stubs* dans `kernel32.dll` et autres

Part III

L'architecture Intel IA32

Description générale

Assembleur

Mini TP

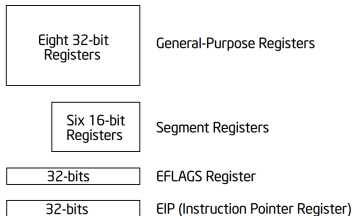
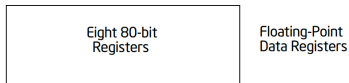
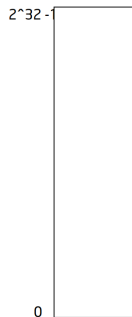
Exceptions et interruptions

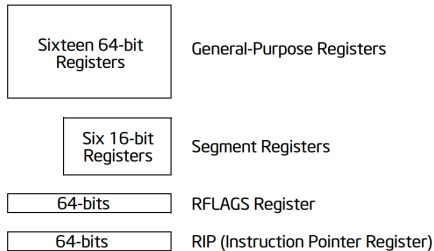
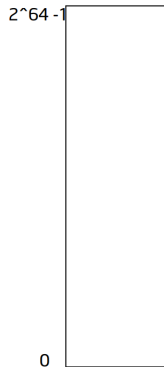
Éléments d'architecture

Principaux registres :

- 8 registres généraux 32 bits.
- 8 registres **FPU** 80 bits.
- 8 registres **MMX** 64 bits.
- 8 registres **XMM** 128 bits.
- le registre 32 bits **EFLAGS**.
- le registre 32 bits **EIP**.
- Un espace d'adressage jusqu'à $2^{32} - 1$ octets.

L'architecture x86 suit la convention *little-endian* pour manipuler les entiers.

Basic Program Execution Registers**FPU Registers****Address Space***

Basic Program Execution Registers**FPU Registers****Address Space**

General-Purpose Registers

31	16 15	8 7	0	16-bit	32-bit
	AH	AL		AX	EAX
	BH	BL		BX	EBX
	CH	CL		CX	ECX
	DH	DL		DX	EDX
	BP				EBP
	SI				ESI
	DI				EDI
	SP				ESP

- Zone contiguë de mémoire.
- Le sommet de la pile est pointé par **ESP** (32 bits) ou **SP** (16 bits).
- La pile est normalement alignée sur 32 ou 16 bits.
- **Last In First Out**.
- Procédure d'appel : adresse de retour, paramètres et *stackframe*.

PUSH

- ESP est décrémenté de la taille de valeur écrite.
- La valeur est écrite au sommet de la pile.

POP

- La valeur placée au sommet de la pile est lue.
- ESP est incrémenté de la taille de valeur lue.

Description générale

Assembleur

Mini TP

Exceptions et interruptions

Éléments d'architecture

- Langage de niveau directement supérieur au langage machine.
- À chaque instruction est associée un mnémonique (représentation textuelle).
- Assembleur assez simple, peu de registres et peu de comportements implicites
- Un peu moins d'un milliers d'instructions exprimables. En pratique une cinquantaine d'instructions constituent la base de tout programme.
- Fondamental pour le *reverse-engineering*, en particulier la compréhension des sauts conditionnels, boucles, et appels de fonctions.
- L'optimisation réalisée par les compilateurs peut rendre le code plus difficile à interpréter.

Instruction : format Intel

Instruction = Mnémonique + Opérande(s) optionnel(s)

2 opérandes : dest puis src

Exemple : ADD EAX, 8

- Transfert : MOV, PUSH, POP
- Opérations arithmétiques : ADD, SUB, DIV, MUL, INC, DEC, CMP
- Opérations logiques : NEG, NOT, AND, OR, XOR, SHL, SHR
- Branchement : CALL, RET, JMP, J (N)E/Z/B/A/C/G/L,

- Connaître la convention d'appel utilisée est nécessaire lors de l'analyse d'un binaire :
 - interprétation des arguments lors de de l'appel à une fonction.
 - interprétation de la valeur de retour.
- Influence de l'architecture matérielle : le nombre de registres est déterminant dans le choix.
- Le langage de programmation influence grandement le choix.

Architecture x86

- Peu de registres : préférences pour le passage de paramètres par la pile.
- La valeur de retour est passée par le registre EAX.

Le compilateur *Microsoft Visual C++* supporte les conventions d'appel suivantes :

Convention	Passage des arguments	Rétablissement de la pile
<i>cdecl</i>	poussés sur la pile de droite à gauche	par l'appelant
<i>stdcall</i>	poussés sur la pile de droite à gauche	par l'appelé
<i>fastcall</i>	par ECX puis EDX ensuite <i>stdcall</i>	par l'appelé
<i>thiscall</i>	this pointer par ECX ensuite <i>stdcall</i>	par l'appelé

Convention d'appel x64 Microsoft, utilisation des registres : ²⁰ ²¹

- 4-registres *fast-call*
- RCX, RDX, R8, R9
- arguments flottants et double-precision passés par XMM0-XMM3
- RAX, RCX, RDX, R8, R9, R10, R11 sont **volatiles**
- RBX, RBP, RDI, RSI, R12, R13, R14, R15 sont **non-volatiles**
- valeur de retour dans RAX

Sous Linux, OS X, BSD, etc. (System V):

- RDI, RSI, RDX, RCX, R8, R9, [XYZ]MM0-7²²

²⁰ [http://msdn.microsoft.com/en-us/library/7kcdt6fy\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/7kcdt6fy(v=vs.80).aspx)

²¹ http://www.codemachine.com/article_x64deepdive.html

²² https://en.wikipedia.org/wiki/X86_calling_conventions

```
1    func1(int a, int b, int c, int d, int e);
```

- a dans RCX
- b dans RDX
- c dans R8
- d dans R9
- e sur la pile

```
1    func3(int a, double b, int c, float d);  
2    // a in RCX, b in XMM1, c in R8, d in XMM3
```

- a dans RCX
- b dans XMM1
- c dans R8
- d dans XMM3

Description générale

Assembleur

Mini TP

Exceptions et interruptions

Éléments d'architecture

Compiler Explorer ²³ permet de regarder la sortie assembleur de différents compilateurs.

Exercice

Avec x86 MSVC v19.0 (WINE), en sélectionnant bien le langage C:

- retrouver le passage des paramètres dans `caller`
- leur lecture dans `sample`
- *idem* en changeant la convention d'appel en `int _fastcall sample`

```
1  int sample(int i, float f, unsigned char *ptr, unsigned long len) {
2      for(int j=0 ; j<len; j++)
3          ptr[j] = (unsigned char)((ptr[j]*i)-f);
4      return j;
5  }
6
7  void caller() {
8      unsigned char table[256];
9      for(int i=0; i<256; i++)
10         table[i] = i;
11      sample(12, 3.14159, table, 256);
12  }
```

²³Lien vers la source du TP : <https://godbolt.org/z/qhxbMMse7>

Description générale

Assembleur

Mini TP

Exceptions et interruptions

Éléments d'architecture

La terminologie Intel distingue les concepts d'interruption et d'exception :

Interruption

Événement asynchrone : typiquement généré par un dispositif d'entrée/sortie. Elles sont par exemple utilisées pour réaliser des communication non bloquantes.

Exemple : le clavier, en tant que périphérique, signale l'appui sur une touche au processeur à l'aide d'une interruption.

La terminologie Intel distingue les concepts d'interruption et d'exception :

Interruption

Événement asynchrone : typiquement généré par un dispositif d'entrée/sortie. Elles sont par exemple utilisées pour réaliser des communication non bloquantes.

Exemple : le clavier, en tant que périphérique, signale l'appui sur une touche au processeur à l'aide d'une interruption.

Exception

Événement synchrone généré par le processeur lui-même. La terminologie Intel spécifie trois classes d'exception : *faults*, *traps*, *abort*.

Exemple : une division par 0 lève une exception portant le numéro 0.

- Le processeur sauve son état interne sur la pile système.
- Traite l'événement : il accède à une routine spécifique grâce à une table nommée *interrupt descriptor table* (**IDT**). Chaque interruption est repérée par un numéro, on parle aussi de vecteur d'interruption.
- Restaure si possible l'état interne et reprend son exécution.
- Dans le cas contraire, le processeur peut faire appel à un gestionnaire de fautes ou tout simplement provoquer une erreur générale (**BSOD**)

Pour résumer sauvagement c'est un bloc *try/catch* très bas niveau.

L'int 1 : single-step

- Exception de classe *trap* ou *fault*.
- Levée après chaque instruction lorsque le processeur fonctionne en mode *pas à pas* ou *single-step*.
- Le *trap flag* (bit 8 du registre **EFLAGS**) permet d'activer ce mode.

L'int 1 : single-step

- Exception de classe *trap* ou *fault*.
- Levée après chaque instruction lorsque le processeur fonctionne en mode *pas à pas* ou *single-step*.
- Le *trap flag* (bit 8 du registre **EFLAGS**) permet d'activer ce mode.

L'int 3 : breakpoint (BP) logiciel

- Exception de classe *trap*.
- Se code sur un seul octet **0xCC**. Cette propriété lui permet d'être placé sur n'importe quelle instruction.
- Destinée à placer des points d'arrêts dans un programme.
- **BP** logiciel : traitement réalisé par l'OS.

Description générale

Assembleur

Mini TP

Exceptions et interruptions

Éléments d'architecture

DR0 à DR3 : Debug Address Register

- Chacun de ces 4 registres peut contenir une adresse qui sera monitorée : les *hardware breakpoints* (**HBP**).

DR6 : Debug Status Register

- Il indique quelles sont les conditions qui ont provoqué la levée de la dernière exception.
- Le bit BS indique que l'exception est due au mode *single-step*.

DR7 : Debug Control Register

- Un jeu de flags spécifie pour chaque **HBP**, dans quelle condition une exception doit être levée.
- En particulier le mode d'accès à l'adresse (r, w, x).
- Ils fixent aussi la nature du **HBP** qui peut être local ou global.

Les breakpoints matériels (HBP)

- Gérés par le processeur à l'aide des debug registers.
- Le processeur génère une **int1** lorsqu'il tente d'accéder à une adresse spécifiée par un des registres **DR0** à **DR3** dans le mode spécifié.
- Plus furtifs que les breakpoints logiciels : pas de modification de la mémoire.
- Moins pénalisant pour le temps d'exécution car tout est géré au niveau matériel.
- En revanche leur nombre est limité.

- L'architecture x86 définit 4 niveaux de privilèges (rings)
- Seuls 2 sont utilisés par les systèmes d'exploitation.

Ring0

- Mode noyau ou privilégié.
- Espace d'adresse *kernel land* : 0x80000000 – 0xFFFFFFFF.
- Accès complet à toutes les ressources matérielles : la mémoire, le processeur...

Ring3

- Mode utilisateur ou protégé.
- Espace d'adressage *user land* : 0x00000000 – 0x7FFFFFFF.
- Tributaire des services exposés par le noyau et donc du mode privilégié.

Intel Architecture Software Developer's Manual

- Les volumes les plus utiles :
 - Volume **1** : vue générale de l'architecture.
 - Volumes **2A** et **2B** : tout le jeu d'instructions.
 - Volumes **3A** et **3B** : présentation détaillée du fonctionnement du processeur.
- Se télécharge en pdf.
- Source d'information incontournable.

Le pendant de chacun de ces documents est aussi disponible chez **AMD**.

Part IV

Méthodologie d'analyse

Les niveaux d'abstractions

L'observation/instrumentation

Analyse de programmes exécutables

- Il existe une multitude d'approches possibles d'un exécutable : les niveaux d'abstractions.
- Depuis le plus bas : le code machine.
- Au plus haut : la *black-box*.
- Dans toute analyse le choix du niveau d'abstraction dépend à la fois de l'échantillon et de ce que l'on cherche à observer.
- Quelques abstractions *haut-niveau* extraites d'un exécutable :
 - Le graphe d'appel.
 - Le *Control Flow Graph* (CFG).
 - Le data flow graph.

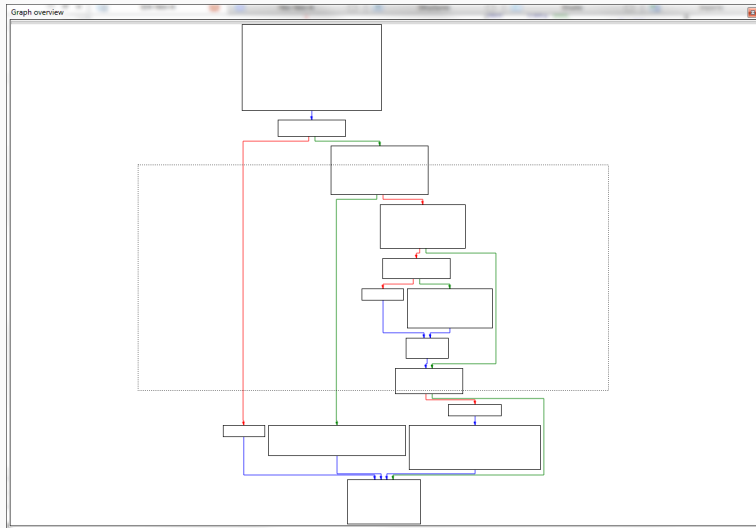
Control transfer instruction (CTI)

C'est une instruction qui peut briser la linéarité du code et transfère le contrôle : un saut, un RET, un CALL ou une INT.

Basic block

Un bloc basique est une séquence contiguë d'instructions ne comportant aucune **CTI** et n'étant la cible d'aucune **CTI**. Seule la première instruction de cette séquence est la cible d'une ou plusieurs **CTI**.

- Le control flow graph (**CFG**) est la combinaison de l'ensemble des basic blocks avec l'ensemble des **CTI**.
- Très intuitif pour comprendre la "logique" d'un programme.
- C'est le mode graphe d'**IDA**.



Data flow

- Chaque nœud représente l'accès en lecture ou en écriture d'une variable
- Backtracking sur les variables pour détecter les zones *sensibles*.
- Analyse combinée du data flow et control flow : *slicing*

Le graphe d'appel

- Permet de visualiser les relations de dépendances.
- Permet d'identifier les fonctions importantes
- Certaines figures sont aussi caractéristiques : machines virtuelles, fonction de chiffrement, etc.

- Comprendre le processus de la génération du code est important
- Faire le lien entre un CFG et les structures de plus hauts niveaux (décompilation mentale).
- L'exemple suivant permet de voir les structures conditionnelles basiques *IF* et *FOR* :

```
1      if (!crc_table_computed)  
2          make_crc_table();  
3  
4      for (n = 0; n < len; n++) {  
5          c = crc_table[(c ^ buf[n])  
6              & 0xff] ^ (c >> 8);  
      }
```

```
; Attributes: bp-based frame

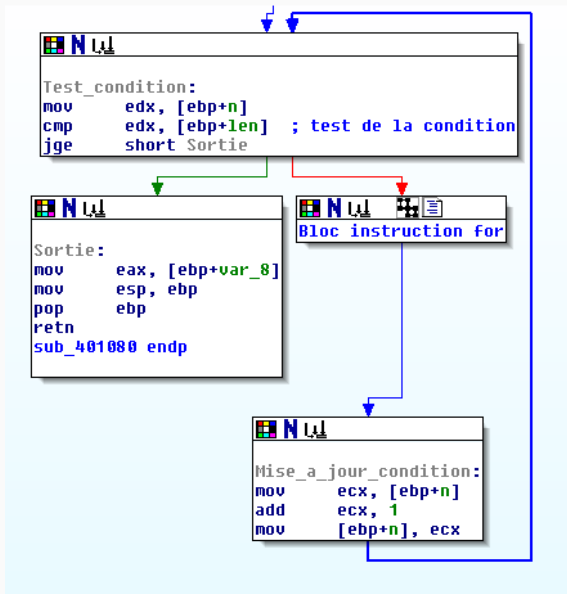
sub_401080 proc near

var_8= dword ptr -8
n= dword ptr -4
arg_0= dword ptr  8
arg_4= dword ptr  0Ch
len= dword ptr  10h

push    ebp
mov     ebp, esp
sub     esp, 8
mov     eax, [ebp+arg_0]
mov     [ebp+var_8], eax
cmp     crc_table_computed, 0
jnz     short Condition_fausse
```

```
Condition_vraie:
call    make_crc_table
```

```
Condition_fausse:
mov     [ebp+n], 0
jmp     short Test_condition
```



Structure conditionnelle IF THEN ELSE et DO WHILE

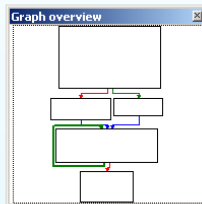
```
sub_401100 proc near  
; Attributes: bp-based frame  
  
sub_401100 proc near  
  
var_8= dword ptr -8  
param= dword ptr 8  
  
push    ebp  
mov     ebp, esp  
sub     esp, 8  
cmp     [ebp+param], 0 ; test de la condition  
jnz     short bloc_else
```

```
bloc_if:  
mov     [ebp+var_8], 1  
jmp     short do_while_loop
```

```
bloc_else:  
mov     [ebp+var_8], 2
```

```
do_while_loop:  
mov     eax, [ebp+param]  
sub     eax, 1  
mov     [ebp+param], eax  
cmp     [ebp+param], 0 ; test de la condition  
jg      short do_while_loop
```

```
mov     eax, [ebp+param]  
mov     esp, ebp  
pop     ebp  
retn  
sub_401100 endp
```



Les niveaux d'abstractions

L'observation/instrumentation

Analyse de programmes exécutables

L'analyse du code est un processus lent, trop coûteux sur de larges portions de code : il est nécessaire de cibler notre analyse.

Que pouvons-nous observer ?

- Le comportement du programme.
 - Exemple : attend-il des entrées de l'utilisateur ?
- Le programme lui-même : nombreuses informations contenues dans l'en-tête PE.
- Interaction du programme avec son environnement : accès au système de fichiers, utilisation d'API.

Un binaire peut comporter une multitude d'indices

- Nature du binaire à étudier : exécutable, DLL, driver ou autre
- **Toutes les chaînes de caractères**
- Reconnaissance du compilateur
- Bibliothèques liées statiquement
- Reconnaissance de packers

Cas pratique : message d'erreur/debug

"OpenSSL 1.0.2o-fips 27 Mar 2018"

- Indique que le binaire utilise la bibliothèque OpenSSL
- L'utilisation des signatures (motifs caractéristiques) correspondantes permettra de reconnaître les fonctions automatiquement : temps d'analyse économisé
- Utilisation possible de chiffrement/rng/hachage

- Analyse comportementale
- Caractérisation d'actions suspectes
- API spying, API hooking, ex : framework Detours de Microsoft²⁴
- OS instrumentés, sandbox, ex :
 - Cuckoo Sandbox
 - Joe Sandbox

Application à l'analyse de malware, ex. type de comportements suspects :

- Injection de thread.
- Création de fichiers/process.
- Écriture dans un process tiers.
- Accès à la base de registre :

`HKCU\Software\Microsoft\Windows\CurrentVersion\Run`

²⁴ <http://research.microsoft.com/en-us/projects/detours/>

Modification directe du code exécuté pour ajouter l'instrumentation

- Accès à l'état interne du programme durant son exécution.
- Beaucoup plus performant que le debug
- Granularité possible extrêmement fine : ex. log de tous les accès mémoire.
- Exemples :
 - Pin²⁵
 - Valgrind²⁶
 - DynamoRio²⁷
 - QBDI²⁸

²⁵ <https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>

²⁶ <http://valgrind.org/>

²⁷ <http://www.dynamorio.org/>

²⁸ <https://qbdi.quarkslab.com>

Les niveaux d'abstractions

L'observation/instrumentation

Analyse de programmes exécutables

- Démarche très coûteuse en temps, d'où la nécessité d'avoir ciblé les zones à analyser.
- Nous nous trouvons à un niveau d'abstraction très bas : le langage assembleur $\Rightarrow$ Il va falloir remonter, le plus souvent cela passe par la reconstruction du *control-flow-graph*.
- Deux approches possibles :
 - Dynamique à l'aide d'un débogueur
 - Statique grâce à un désassembleur
- Deux approches complémentaires : se servir du meilleur de chacune.

Avantages

- Vue globale du binaire : étude macroscopique puis microscopique.
- Reconstruction du *control-flow-graph*, *data-flow*.
- La référence : désassembleur **IDA Pro**.
- Technique de prédilection pour l'étude de code potentiellement malveillant car pas d'exécution.

Limitations

- Difficultés ou impossibilités à prédire la valeur de variables résolues dynamiquement.
- Le désassemblage peut être perturbé (au bas mot) par des techniques d'obfuscation.
- Difficultés avec les codes se déchiffrant en mémoire.

Avantages

- L'outil de prédilection est le débogueur : **x64dbg**, **WinDbG**, **GDB**, **textbfLLDB**, ...
- Accès direct à toutes les valeurs manipulées par le programme.
- Contrôle du flot d'exécution.
- Plus adaptée à une étude microscopique : mise en place de points d'arrêts et exécution pas-à-pas du code.

Limitations

- Obligation d'exécuter le code : code potentiellement dangereux pour le système d'exploitation hôte.
- Techniques relativement intrusives : contre-mesures éventuelles visant les outils d'analyse.

Part V

La sécurité logicielle

Enfin de la sécurité

- Sécurité hardware
- Sécurité software

Objectifs

- Comprendre les mécanismes de sécurité afin de les identifier lors de la phase de reverse
- Comprendre les différences entre l'assembleur et la vue décompilée

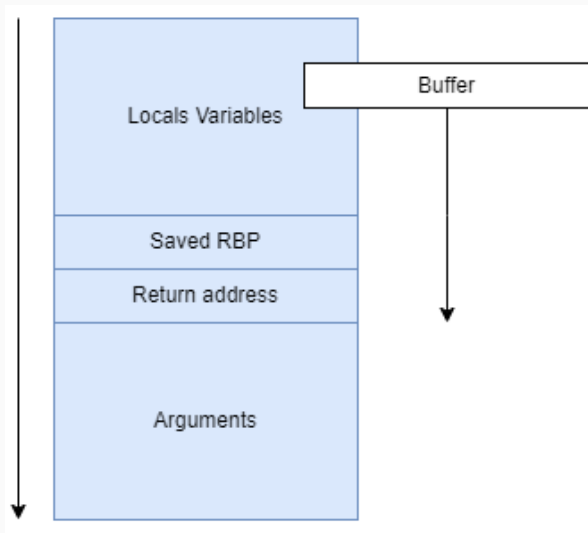
Les cookies

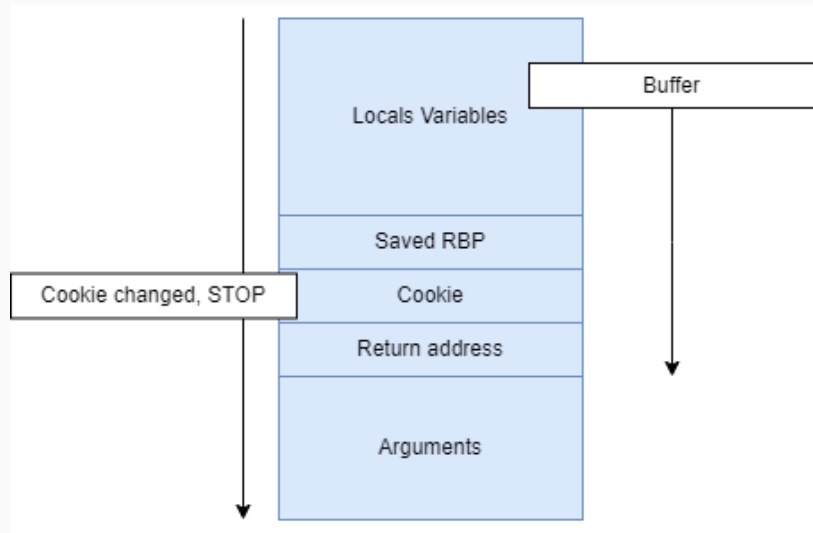
Les exceptions

Control Flow Guard

Le Canary cookie ou en français ... (pas trouvé dans le CyberDico ²⁹ de l'ANSSI) est un mécanisme de sécurité introduit à la compilation afin de se protéger des catégories d'attaques : stack overflow, return to stack

²⁹CyberDico : <https://cyber.gouv.fr/publications/cyberdico-quest-ce-que-cest>





```
sub_1400052C0 proc near
var_10038= qword ptr -10038h
ArgList= qword ptr -10030h
Buffer= word ptr -10028h
var_28= qword ptr -28h
arg_28= qword ptr 10h
arg_10= byte ptr 18h
arg_18= qword ptr 20h

; __unwind { // __GSHandlerCheck
mov     [rsp+arg_8], rdx
mov     qword ptr [rsp+arg_10], r8
mov     [rsp+arg_18], r9
push    rbx
push    rsi
push    rdi
mov     eax, 10040h
call    __alloca_probe
sub     rsp, rax
mov     rax, cs: __security_cookie
xor     rax, rsp
mov     [rsp+10058h+var_28], rax
mov     rbx, rdx
lea     rsi, [rsp+10058h+arg_10]
mov     rdi, rcx
call    sub_140004F98
mov     [rsp+10058h+ArgList], rsi ; ArgList
lea     rdx, [rsp+10058h+Buffer] ; Buffer
and     [rsp+10058h+var_10038], 0
mov     r9, rbx          ; Format
mov     r8d, 8000h       ; BufferCount
mov     rcx, [rax]        ; Options
call    __stdio_common_vsprintf_s
xor     r9d, r9d
lea     rdx, [rsp+10058h+Buffer]
mov     r8d, 7FFFFFFFh
mov     rcx, rdi
call    sub_14000827C
mov     rcx, [rsp+10058h+var_28]
xor     rcx, rsp          ; StackCookie
call    __security_check_cookie
add     rsp, 10040h
pop     rdi
ret
```

```
__int64 sub_1400052C0(__int64 a1, const wchar_t *a2, ...)
{
    unsigned __int64 *v4; // rax
    wchar_t Buffer[32768]; // [rsp+30h] [rbp-10028h] BYREF
    va_list va; // [rsp+10070h] [rbp+18h] BYREF

    va_start(va, a2);
    v4 = (unsigned __int64 *)sub_140004F98();
    _stdio_common_vswprintf_s(*v4, Buffer, 0x8000ui64, a2, 0i64, va);
    return sub_14000827C(a1, Buffer, 0x7FFFFFFFi64, 0i64);
}
```

Les cookies

Les exceptions

Control Flow Guard

Il existe un mécanisme d'exception présent sous Windows sous forme d'extension du langage C, précédant même le C++.

```
__try {  
    puts("in try");  
    __try {  
        puts("in try");  
        RAISE_AN_EXCEPTION();  
    } __finally {  
        puts("in finally");  
    }  
} __except( puts("in filter"), EXCEPTION_EXECUTE_HANDLER )  
    puts("in except");  
}
```

- Le mécanisme d'exception est une spécification du langage
- l'implémentation est libre

Sous Windows en x86, on parle de Structured Exception Handler (SEH)

- L'adresse des handlers d'exception est stockée dans une liste chaînée
- Les entrées de cette liste sont stockées directement sur la pile
- Elles ont donc été naturellement utilisées afin de détourner l'exécution, comme dans le cas des "return to stack"

```
1  typedef struct _EXCEPTION_REGISTRATION_RECORD
2  {
3      struct _EXCEPTION_REGISTRATION_RECORD *Next;
4      PEXCEPTION_ROUTINE                      Handler;
5  }
6  EXCEPTION_REGISTRATION_RECORD ,
7  *PEXCEPTION_REGISTRATION_RECORD;
```

Version sécurisée des précédentes SEH. Les handlers sont connus à la compilation, donc à l'exécution avant d'appeler une adresse on va vérifier si celle-ci est bien l'adresse d'un handler connu (peu importe lequel).

L'implémentation des exceptions en x64 est complètement différente et ne se base plus sur la pile.

En x64, une table avec une entrée pour chaque fonction du programme est générée à la compilation avec les informations nécessaires de `unwind`.

Les cookies

Les exceptions

Control Flow Guard

Un mécanisme similaire a été ajouté afin d'empêcher les attaques utilisant les vtables.

- Les vtables est un mécanisme permettant de gérer l'héritage en C++
- L'implémentation utilise une table d'adresse de fonction qui sera remplie lors de la création de l'objet
- Les attaquants peuvent utiliser cette table afin de rediriger le flux d'exécution lors d'un exploit
- Le CFG permet de restreindre les valeurs des adresses à une sous-partie connue de l'exécutable

Questions ?