

Introduction au Reverse Engineering

Thomas BYGODT

Novembre 2022

- ENSSAT Promo Info 2017
- Auditeur/Pentester Senior chez Orange Cyberdefense depuis 5 ans
- Ancien membre de l'équipe française de l'ECSC 2019

Mail

`thomas.bygodt@univ-rennes1.fr`

- Plateforme TP : `https://ctfd.xrsecware.fr`
- Registration code : `ReverseExploit2022Enssat!!!!`
- Mail : `enssat.fr` ou `univ-rennes1.fr`

- 1 Contexte
- 2 Binaires et assembleur
- 3 Intel IA-32
- 4 Les outils
- 5 Format de fichier
- 6 Méthodologies d'analyses
- 7 Exemple de techniques d'anti-debug, d'obfuscation

1 Contexte

- Qu'est-ce que le Reverse Engineering ?
- Objectifs

Qu'est-ce que le Reverse Engineering

Définition (source : Larousse)

Étude d'un produit ou d'un système existant dans le but de déterminer son fonctionnement et la manière dont il a été conçu.

Qu'est-ce que le Reverse Engineering

Exemples

- Analyser un malware afin de connaître son fonctionnement
- Trouver des vulnérabilités dans un produit
- Proposer un logiciel libre et open source (ex : driver)

Qu'est-ce que le Reverse Engineering

Exemples

- Analyser un malware afin de connaître son fonctionnement
- Trouver des vulnérabilités dans un produit
- Proposer un logiciel libre et open source (ex : driver)

Mais aussi

- Décompiler une application Android
- Déobfusquer un script Python / Javascript / .NET

Difficultés

- Nécessite une bonne connaissance dans de nombreux domaines (système, API, développement)
- Peu de documentation (ex : firmwares)
- Défaire les techniques d'obfuscation (VM, code mort, chiffrement...)
- Nécessite du temps et de l'entraînement pour lire facilement de l'assembleur

Objectifs

- Apprendre les méthodologies d'analyse courantes
- Être capable d'analyser un programme x86-64
- Découvrir les outils d'analyse statique et dynamique

Programme

- Découverte de l'architecture Intel
- Présentation des formats des exécutables (ELF, PE)
- Présentation de quelques techniques d'obfuscation et d'anti-débug

Astuces

- Commencer avec des programmes simples (quitte à les faire soit même !)
- Trouver le compromis entre l'analyse bas niveau et haut niveau
- Commencer par une analyse Top/Down ou Down/Top

Astuces

- Commencer avec des programmes simples (quitte à les faire soit même !)
- Trouver le compromis entre l'analyse bas niveau et haut niveau
- Commencer par une analyse Top/Down ou Down/Top

Challenges

- Root-me : <https://root-me.org/>
- Crackme : <https://crackmes.one/>

2 Binaires et assembleur

- Généralités
- Assembleur

Les outils

- Compilateurs : gcc, llvm,...
- Débogueur : gdb, LLDB, x64dbg, WinDbg,...
- Outils CLI : objdump, strings, binwalk,...
- Outils GUI : Ghidra, Hex Rays IDA, Binary Ninja
- Scripting : Bash, Python3

De la source à l'exécutable et inversement

Code Source

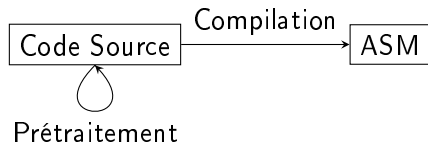
De la source à l'exécutable et inversement

Code Source

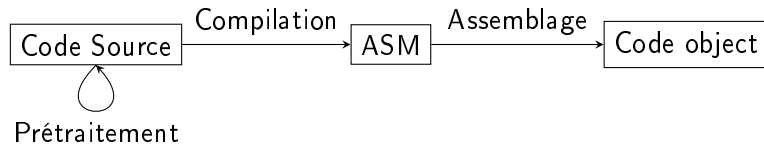


Prétraitement

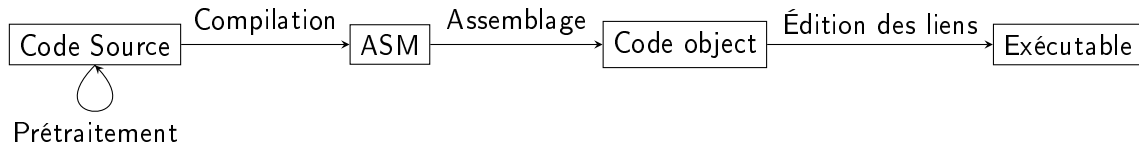
De la source à l'exécutable et inversement



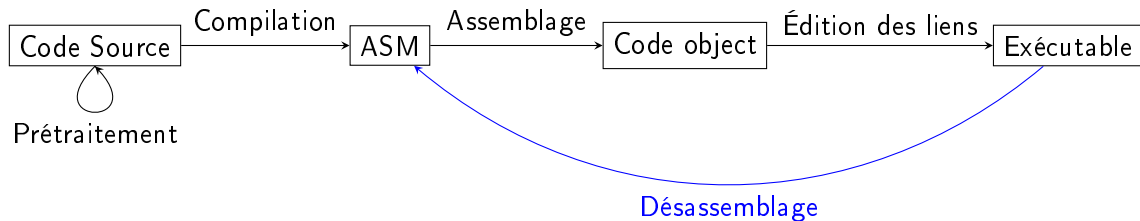
De la source à l'exécutable et inversement



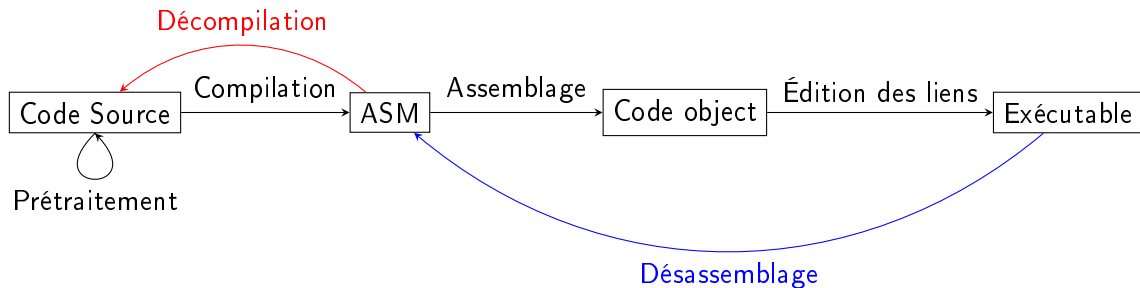
De la source à l'exécutable et inversement



De la source à l'exécutable et inversement



De la source à l'exécutable et inversement



Exemple

- Assembleur online : <https://godbolt.org/>
- En local : `gcc -S -masm=intel src.c`

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
int main(int argc, char** argv)
{
    int sum;
    sum = argc * 2;
    printf("%d\n", sum);
    return 0;
}
```

Exemple

ASM associé :

```
push    rbp
mov     rbp, rsp
sub     rsp, 32
mov     DWORD PTR [rbp-20], edi
mov     QWORD PTR [rbp-32], rsi
mov     eax, DWORD PTR [rbp-20]
add     eax, eax
mov     DWORD PTR [rbp-4], eax
mov     eax, DWORD PTR [rbp-4]
mov     esi, eax
mov     edi, OFFSET FLAT:..LCO
mov     eax, 0
call    printf
mov     eax, 0
leave
ret
```


Exemple

```
> gcc -c example_asm.s && objdump -M intel -d example_asm.o
```

```
example_asm.o:      format de fichier elf64-x86-64  
Déassemblage de la section .text :
```

```
0000000000000000 <main>:
```

0:	55	push	rbp
1:	48 89 e5	mov	rbp, rsp
4:	48 83 ec 20	sub	rsp, 0x20
8:	89 7d ec	mov	DWORD PTR [rbp-0x14], edi
b:	48 89 75 e0	mov	QWORD PTR [rbp-0x20], rsi
f:	8b 45 ec	mov	eax, DWORD PTR [rbp-0x14]
12:	01 c0	add	eax, eax
14:	89 45 fc	mov	DWORD PTR [rbp-0x4], eax
17:	8b 45 fc	mov	eax, DWORD PTR [rbp-0x4]
1a:	89 c6	mov	esi, eax
1c:	bf 00 00 00 00	mov	edi, 0x0
21:	b8 00 00 00 00	mov	eax, 0x0
26:	e8 00 00 00 00	call	2b <main+0x2b>
2b:	b8 00 00 00 00	mov	eax, 0x0
30:	c9	leave	
31:	c3	ret	

Exemple

```
> gcc -o example_asm example_asm.o && objdump -M intel -d example_asm.o
example_asm:      format de fichier elf64-x86-64
Déassemblage de la section .init :
```

```
00000000004003c8 <_init>: [...]
0000000000400410 <__libc_start_main@plt>: [...]
0000000000400430 <_start>: [...]
0000000000400526 <main>:
 400526:      55                push    rbp
 400527:      48 89 e5          mov     rbp, rsp
 40052a:      48 83 ec 20        sub     rsp, 0x20
 40052e:      89 7d ec          mov     DWORD PTR [rbp-0x14], edi
 400531:      48 89 75 e0        mov     QWORD PTR [rbp-0x20], rsi
 400535:      8b 45 ec          mov     eax, DWORD PTR [rbp-0x14]
 400538:      01 c0             add     eax, eax
 40053a:      89 45 fc          mov     DWORD PTR [rbp-0x4], eax
 40053d:      8b 45 fc          mov     eax, DWORD PTR [rbp-0x4]
 400540:      89 c6             mov     esi, eax
 400542:      bf e4 05 40 00    mov     edi, 0x4005e4
 400547:      b8 00 00 00 00    mov     eax, 0x0
 40054c:      e8 af fe ff ff    call    400400 <printf@plt>
 400551:      b8 00 00 00 00    mov     eax, 0x0
 400556:      c9                leave
 400557:      c3                ret
 400558:      0f 1f 84 00 00 00 nop     DWORD PTR [rax+rax*1+0x0]
```

Exemple

undefined8		Stack[-0x28]:8local_28		
		main		XREF[5]
00400526	55	PUSH	EBP	
00400527	48 89 e5	MOV	EBP,ESP	
0040052a	48 83 ec 20	SUB	ESP,0x20	
0040052e	89 7d ec	MOV	dword ptr [EBP + local_1c],EDI	
00400531	48 89 75 e0	MOV	qword ptr [EBP + local_28],RSI	
00400535	8b 45 ec	MOV	EAX,dword ptr [EBP + local_1c]	
00400538	01 c0	ADD	EAX,EAX	
0040053a	89 45 fc	MOV	dword ptr [EBP + local_c],EAX	
0040053d	8b 45 fc	MOV	EAX,dword ptr [EBP + local_c]	
00400540	89 c6	MOV	ESI,EAX	
00400542	bf e4 05 40 00	MOV	EDI=>DAT_004005e4,DAT_004005e4	
00400547	b8 00 00 00 00	MOV	EAX,0x0	
0040054c	e8 af fe ff ff	CALL	printf	
00400551	b8 00 00 00 00	MOV	EAX,0x0	
00400556	c9	LEAVE		
00400557	c3	RET		

```
1 |
2 | undefined8 main(int iParam1)
3 |
4 | {
5 |     printf("%d\n", (ulong)(uint) (iParam1 * 2));
6 |     return 0;
7 | }
8 |
```

CPU

- Le jeu d'instruction est dépendant du CPU et de son architecture
- Intel et AT&T sont les syntaxes les plus répandues
- CISC architecture : Complex Instruction Set Computer (ex : Intel IA-32)
 - Jeu d'instructions très varié
 - Instruction couteuse
 - Instructions de longueur variable
- RISC architecture : Reduced Instruction Set Computer (ex : ARM, MIPS, SPARC)
 - Jeu d'instructions limité
 - Necessite plus d'instruction pour une fonction complexe
 - Taille des instructions fixe

Les tailles mémoires

Nom	Taille	Exemple(s) de type C
Byte	8	char
Word	16	short
Double word	32	int, long
Quad word	64	long long

Boutisme (Endianness)

Little-Endian

- Les octets de poids faible sont stockés dans les adresses plus petites.
- Ex : La valeur 0x12345678 est représentée en RAM par 78 56 34 12

Big-Endian

- Les octets de poids fort sont stockés dans les adresses plus petites.
- Ex : La valeur 0x12345678 est représentée en RAM par 12 34 56 78

Boutisme (Endianness)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

00	00	32	34	00	42	45	4e	53	53	41	54	00	00	00	00	00
10	00	00	01	72	65	76	65	72	73	65	00	00	00	00	00	00

Questions

- word à l'adresse 0x1 en mode little-endian ?
- Double Word à l'adresse 0x4 en mode little-endian ?
- Quad Word à l'adresse 0x13 en mode big-endian ?

Boutisme (Endianness)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

00	00	32	34	00	42	45	4e	53	53	41	54	00	00	00	00	00
10	00	00	01	72	65	76	65	72	73	65	00	00	00	00	00	00

Questions

- word à l'adresse 0x1 en mode little-endian ? **0x3432**
- Double Word à l'adresse 0x4 en mode little-endian ?
- Quad Word à l'adresse 0x13 en mode big-endian ?

Boutisme (Endianness)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

00	00	32	34	00	42	45	4e	53	53	41	54	00	00	00	00	00
10	00	00	01	72	65	76	65	72	73	65	00	00	00	00	00	00

Questions

- word à l'adresse 0x1 en mode little-endian ? **0x3432**
- Double Word à l'adresse 0x4 en mode little-endian ? **0x534e4542**
- Quad Word à l'adresse 0x13 en mode big-endian ?

Boutisme (Endianness)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

00		00	32	34	00	42	45	4e	53	53	41	54	00	00	00	00
10		00	00	01	72	65	76	65	72	73	65	00	00	00	00	00

Questions

- word à l'adresse 0x1 en mode little-endian ? 0x3432
- Double Word à l'adresse 0x4 en mode little-endian ? 0x534e4542
- Quad Word à l'adresse 0x13 en mode big-endian ? 0x7265766572736500

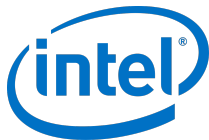
- 3 Intel IA-32
 - Généralités
 - Registres
 - Instructions

Présentation

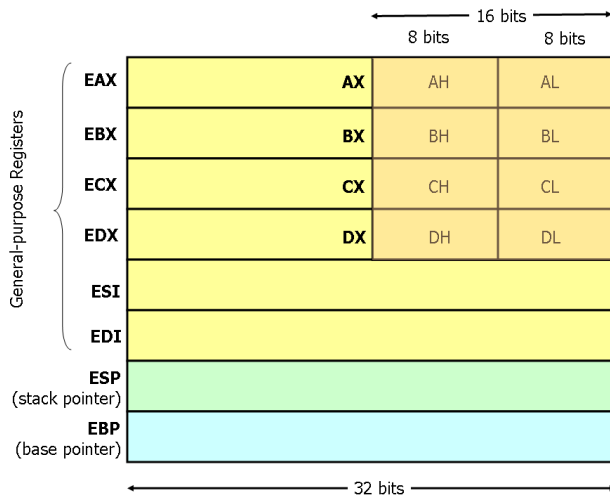
L'architecture Intel est l'une des plus répandues, aussi bien dans sa version 32 bits ou 64 bits.

Elle date de 1985.

Elle peut porter différents noms : x86, x86_32, i386.



Registres x86 génériques



Registres x86

Voici les utilisations les plus courantes des registres :

- EAX : sert d'accumulateur et pour renvoyer les résultats
- EBX : pointeur sur les données
- ECX : compteur
- EDX : utilisé pour les entrées/sorties, et les opérations arithmétiques.
- EIP : Pointeur d'instruction
- EDI/ESI : pointeur de destination ou de sorties
- ESP : Pointeur sur le sommet de la pile (stack)
- EBP : Pointeur sur une adresse particulière de la pile (stack)

Registres x86

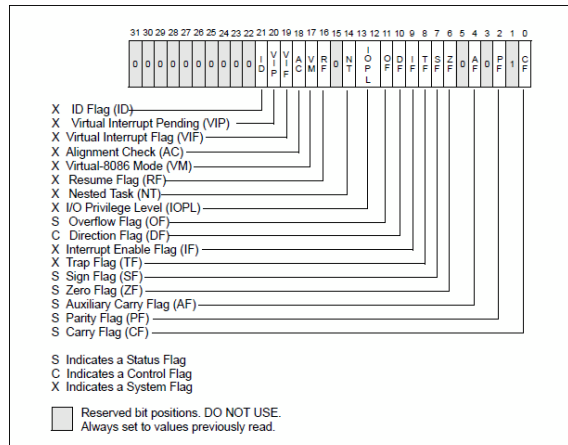
Voici les utilisations les plus courantes des registres :

- EAX : sert d'accumulateur et pour renvoyer les résultats
- EBX : pointeur sur les données
- ECX : compteur
- EDX : utilisé pour les entrées/sorties, et les opérations arithmétiques.
- EIP : Pointeur d'instruction
- EDI/ESI : pointeur de destination ou de sorties
- ESP : Pointeur sur le sommet de la pile (stack)
- EBP : Pointeur sur une adresse particulière de la pile (stack)

Nom des registres (amd64)	Taille
rax, rbx, rcx, rdx, rdi, rsi, rbp, rsp, r8, r9, ..., r15	64
eax, ebx, ecx, edx, edi, esi, ebp, esp, r8d, r9d, ..., r15d	32 (31 :0)
ax, bx, cx, dx, di, si, bp, sp, r8w, r9w, ..., r15w	16 (15 :0)
ah, bh, ch, dh, r9h, ..., r15h	8 haut (15 :8)
al, bl, cl, dl, dil, sil, bpl, spl, r8b, r9b, ..., r15b	8 bas (7 :0)

Le registre EFLAG

- Généralement utilisé par les sauts conditionnels
- Modifiés par de nombreuses instructions (**CMP**, **TEST**, ...); implicitement pour certaines (**ADD**, **SHR**, ...)



EFLAGS Register

source : sop.upv.es

Instruction x86

Définition

Opcode = Mnemonic + destination + paramètre(s)

Instruction x86

Définition

Opcode = Mnemonic + destination + paramètre(s)

Exemple

```
C7 03 0E 00 00 00    mov    DWORD PTR [ebx], 14
8B 03                mov    eax, DWORD PTR [ebx]
6B C0 0E            imul    eax, eax, 14
```

Exemple : MOV

MOV

Instruction permettant de déplacer une donnée

- Registre vers registre : `mov eax, ebx`
- Registre vers mémoire : `mov [0x42424242], ebx`
- Mémoire vers registre : `mov eax, [0x42424242]`
- Valeur vers registre : `mov eax, 0x42`
- Valeur vers mémoire : `mov [eax], 0x42`

r/m format.

$\text{base} + (\text{index} \times \text{scale}) + \text{displacement}; \text{scale} \in \{1, 2, 3, 4\}$

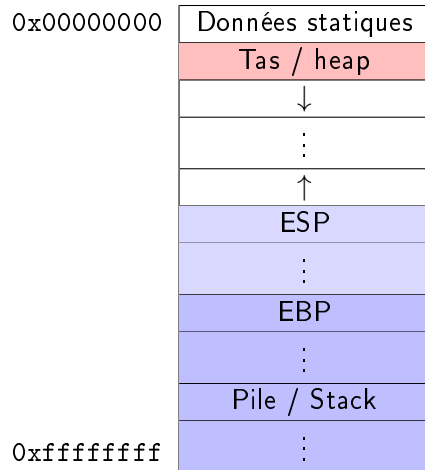
Instruction x86

Voici les instructions les plus courantes :

- Transfert : MOV, MOVSX (sign extend), MOVZX (zero extend), 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
- Opérations sur les chaînes de caractères : STOS{B,W,D}, MOVS, CMPS
- Opérations répétées : REP MOV{B,W,D}, REP {B,W,D}
- Instruction spéciale : NOP

La pile / The Stack

- Pile de type **L**ast **I**n **F**irst **O**ut
- Contient les variables locales
- Séparée en frame contenant le contexte de chaque fonction
- **ESP** : Pointe vers le début de la stack
- **EBP** : Pointe vers le contexte précédent



Exemple : PUSH/POP

PUSH

Instruction permettant d'ajouter une valeur sur la stack

POP

Instruction permettant de récupérer une valeur de la stack

Exemple : PUSH/POP

PUSH

Instruction permettant d'ajouter une valeur sur la stack

POP

Instruction permettant de récupérer une valeur de la stack

```
push 0x123  
mov eax, 0xdeadbeef  
push eax  
push 0x123  
pop eax
```

0x100	
0x104	
0x108	
0x10C	0x0

ESP : 0x10C

Exemple : PUSH/POP

PUSH

Instruction permettant d'ajouter une valeur sur la stack

POP

Instruction permettant de récupérer une valeur de la stack

```
push 0x123  
mov eax, 0xdeadbeef  
push eax  
push 0x123  
pop eax
```

0x100	
0x104	
0x108	0x123
0x10C	0x0

ESP : 0x108

Exemple : PUSH/POP

PUSH

Instruction permettant d'ajouter une valeur sur la stack

POP

Instruction permettant de récupérer une valeur de la stack

```
push 0x123  
mov eax, 0xdeadbeef  
push eax  
push 0x123  
pop eax
```

0x100	
0x104	0xdeadbeef
0x108	0x123
0x10C	0x0

ESP :0x104

Exemple : PUSH/POP

PUSH

Instruction permettant d'ajouter une valeur sur la stack

POP

Instruction permettant de récupérer une valeur de la stack

```
push 0x123  
mov eax, 0xdeadbeef  
push eax  
push 0x123  
pop eax
```

0x100	0x123
0x104	0xdeadbeef
0x108	0x123
0x10C	0x0

ESP : 0x100

Exemple : PUSH/POP

PUSH

Instruction permettant d'ajouter une valeur sur la stack

POP

Instruction permettant de récupérer une valeur de la stack

```
push 0x123  
mov eax, 0xdeadbeef  
push eax  
push 0x123  
pop eax
```

0x100	
0x104	0xdeadbeef
0x108	0x123
0x10C	0x0

ESP :0x104

Contrôle de flux

Il y a deux types de contrôle de flux :

Conditionnel

Sauter à une adresse spécifique si une condition est remplie.

Exemple : if, loop

Inconditionnel

Sauter à une adresse spécifique dans tous les cas.

Exemple : goto, exceptions, interruptions, appels de procédure

Exemple JMP/JXX

JMP

- Saut relatif : Saut à une adresse relative à l'adresse présente dans l'EIP.
- Saut absolu : Saut à une adresse voulue (opérande selon le format r/m).

JXX : saut conditionnel

- JZ/JE \leftrightarrow if (ZF == 1)
- JNZ/JNE \leftrightarrow if (ZF == 0)
- JNG/JLE \leftrightarrow if (ZF == 1 || SF != OF)
- JNL/JGE \leftrightarrow if (ZF == OF)
- JBE \leftrightarrow if (CF == 1 || ZF == 1)
- JB \leftrightarrow if (CF == 1)
- ...

Exemple CMP/TEST

CMP

- Basé sur la **soustraction**
- Modifie les drapeaux : CF, OF, SF, ZF, AF, PF.

TEST

- Basé sur l'opération logique **AND**
- Modifie les drapeaux : SF, ZF, PF

Exemple 1

```
    mov eax, 7
A:   cmp eax, 0
     jl  C
     sub eax, 1
     jmp A
C:   ; [...]
```

Exemple 1

```
    mov eax, 7
A:   cmp eax, 0
     jl  C
     sub eax, 1
     jmp A
C:   ; [...]
```

```
int a = 7;
while(a){
    a -= 1;
}
```


Exemple 2

```
    cmp eax, 7
    jle B
A:
    imul eax, eax
    jmp next
B:
    mov eax, 1
next:
    ; [...]
```

Exemple 2

```
    cmp eax, 7
    jle B
A:
    imul eax, eax
    jmp next
B:
    mov eax, 1
next:
    ; [...]
```

```
if(a > 7){
    a *= a; // A
} else {
    a = 1; // B
}
// C
```

Les fonctions

- `call addr` permet d'appeler une fonction
- `ret` permet de sortir d'une fonction
- Définie par une pile d'exécution ("stack frame")
- Dépend de la convention utilisée

Prologue

```
push ebp
mov  ebp, esp
sub  esp, SIZE_LOCAL_VARS
```

Epilogue

```
mov esp, ebp ; leave
pop ebp      ; leave
ret          ; ret SIZE_PARAMS
```

Conventions d'appels

Éléments communs

- Les paramètres sont passés de la droite vers la gauche sur la stack
- EAX ou EDX : EAX contient la valeur de retour

Voici la différence entre les deux conventions les plus courantes :

cdecl

La stack est nettoyée par la fonction appelante

stdcall

La stack est nettoyée par la fonction appelée

4 Les outils

- Les outils d'observation
- Les outils d'analyse

Observation statiques

- Basé sur l'observation du binaire
- Indétectable, car il n'y a pas d'exécution
- Utile pour déterminer :
 - les imports et/ou exports
 - l'entropie du binaire (détection de malware)
- Ex :
 - objdump, binwalk, strings, ldd, LIEF, CFF Explorer, PEiD, etc...
 - Moteurs d'analyses statiques des antivirus

Observation dynamique

- Analyse en mode boîte noire
- Observer les interactions avec le système (fichiers, api système, connexion)
- Bien que faiblement intrusif, la détection est possible
- Ex :
 - ps, ltrace, strace,
 - Sysinternals

Les débogueurs

- **Contrôler l'exécution** du binaire
- **Consulter l'état** des variables, des registres, de la stack/heap
- **Désassembler** le binaire
- Très intrusif
- Ex :
 - Linux : gdb, IDA, Ghidra, LLDB
 - Windows : x64dbg (userland : ring 3), WinDbg (kernel land : ring 0)

Les déssassembleurs & décompilateurs

Désassembleurs

- **Désassemble** un binaire au travers d'une interface
- Fonctionnalités avancées (diagramme de flux, suivi des variables, etc...)
- Pas d'exécution donc indétectable
- Ex : **Ghidra**, **IDA**, Radare2 (R2)

Décompilateurs

- **Tente de retrouver le code source** d'un binaire
- Dépend du compilateur, des techniques d'obfuscation, etc...
- Ex : **Ghidra**, **IDA**

GDB - cheatsheet

Running

```
# gdb <program> [core dump]
    Start GDB (with optional core dump).

# gdb --args <program> <args...>
    Start GDB and pass arguments

# gdb --pid <pid>
    Start GDB and attach to process.

set args <args...>
    Set arguments to pass to program to
    be debugged.

run
    Run the program to be debugged.

kill
    Kill the running program.
```

Breakpoints

```
break <where>
    Set a new breakpoint.

delete <breakpoint#>
    Remove a breakpoint.

clear
    Delete all breakpoints.

enable <breakpoint#>
    Enable a disabled breakpoint.

disable <breakpoint#>
    Disable a breakpoint.
```

Watchpoints

```
watch <where>
    Set a new watchpoint.

delete/enable/disable <watchpoint#>
    Like breakpoints.
```

<where>

```
function_name
    Break/watch the named function.

line_number
    Break/watch the line number in the
    current source file.

file:line_number
    Break/watch the line number in the
    named source file.
```

Conditions

```
break/watch <where> if <condition>
    Break/watch at the given location if
    the condition is met.
    Conditions may be almost any C
    expression that evaluate to true or false.

condition <breakpoint#> <condition>
    Set/change the condition of an
    existing break- or watchpoint.
```

Examining the stack

```
backtrace
where
    Show call stack.

backtrace full
where full
    Show call stack, also print the
    local variables in each frame.

frame <frame#>
    Select the stack frame to operate on.
```

Stepping

```
step
    Go to next instruction (source line),
    diving into function.
```

```
next
    Go to next instruction (source line)
    but don't dive into functions.

finish
    Continue until the current function
    returns.

continue
    Continue normal execution.
```

Variables and memory

```
print/format <what>
    Print content of variable/memory
    location/register.

display/format <what>
    Like „print“, but print the
    information after each stepping
    instruction.

undisplay <display#>
    Remove the „display“ with the
    given number.

enable display <display#>
disable display <display#>
    En- or disable the „display“
    with the given number.

x/nfu <address>
    Print memory.
    n: How many units to print (default 1).
    f: Format character (like „print“).
    u: Unit.

    Unit is one of:
        b: Byte,
        h: Half-word (two bytes)
        w: Word (four bytes)
        g: Giant word (eight bytes)).
```

GDB - cheatsheet

Format

<i>a</i>	Pointer.
<i>c</i>	Read as integer, print as character.
<i>d</i>	Integer, signed decimal.
<i>f</i>	Floating point number.
<i>o</i>	Integer, print as octal.
<i>s</i>	Try to treat as C string.
<i>t</i>	Integer, print as binary (<i>t</i> = „two“).
<i>u</i>	Integer, unsigned decimal.
<i>x</i>	Integer, print as hexadecimal.

<what>

expression

Almost any C expression, including function calls (must be prefixed with a cast to tell GDB the return value type).

file_name::variable_name

Content of the variable defined in the named file (static variables).

function::variable_name

Content of the variable defined in the named function (if on the stack).

{type}address

Content at *address*, interpreted as being of the C type *type*.

\$register

Content of named register. Interesting registers are *\$esp* (stack pointer), *\$ebp* (frame pointer) and *\$eip* (instruction pointer).

Threads

thread <thread#>

Chose thread to operate on.

Manipulating the program

set var <variable_name>=<value>
Change the content of a variable to the given value.

return <expression>
Force the current function to return immediately, passing the given value.

Sources

directory <directory>

Add *directory* to the list of directories that is searched for sources.

list

list <filename>:<function>

list <filename>:<line_number>

list <first>,<last>

Shows the current or given source context. The *filename* may be omitted. If *last* is omitted the context starting at *start* is printed instead of centered around it.

set listsize <count>

Set how many lines to show in „list“.

Signals

handle <signal> <options>

Set how to handle signals. Options are:

(no)print: (Don't) print a message when signals occurs.

(no)stop: (Don't) stop the program when signals occurs.

(no)pass: (Don't) pass the signal to the program.

Informations

disassemble

disassemble <where>

Disassemble the current function or given location.

info args

Print the arguments to the function of the current stack frame.

info breakpoints

Print informations about the break- and watchpoints.

info display

Print informations about the „displays“.

info locals

Print the local variables in the currently selected stack frame.

info sharedlibrary

List loaded shared libraries.

info signals

List all signals and how they are currently handled.

info threads

List all threads.

show directories

Print all directories in which GDB searches for source files.

show listsize

Print how many are shown in the „list“ command.

whatis <variable_name>

Print type of named variable.

GEF (GDB Enhanced Features)

Layout et fonctionnalités supplémentaires
pour GDB :
<https://github.com/hugsy/gef>

```
[ Legend: Modified register | Code | Heap | Stack | String ]
registers
Rax : 0x0
Rbx : 0x0
Rcx : 0x0007fffffcca0 + 0x0004095d00000000
Rdx : 0x0
Rop : 0x0007fffffca30 + 0x0000000000000000
Rbp : 0x0007fffffca50 + 0x00000000000007f0 + <__libc_csu_init+0> push r15
Rsi : 0x0007fffffca170 + 0x0000000000002000 + 0x0000000000000000
Rdi : 0x20000
Rip : 0x0000000000000799 + <main+64> mov QWORD PTR [rbp-0x28], rax
R8 : 0x0007fffffca700 + 0x0007fffffca700 + [loop detected]
R9 : 0x1
R10 : 0x0
R11 : 0x246
R12 : 0x0000000000000100 + <_start+0> mov ebp, ebp
R13 : 0x0007fffffca40 + 0x0000000000000001
R14 : 0x0
R15 : 0x0
[flags: CARRY PARITY adjust ZERO sign trap INTERRUPT direction overflow resume virtualx86 identification]
Sni: 0x002b Sdi: 0x0033 Sdi: 0x0000 Sdi: 0x0000 Sdi: 0x0000 Sdi: 0x0000
stack
0x0007fffffca30 +0x0000: 0x0000000000000000 + $rsp
0x0007fffffca35 +0x0000: 0x0000000000000000
0x0007fffffca40 +0x0010: "myfile.txt"
0x0007fffffca45 +0x0010: 0x00000000000007f0 ("x")
0x0007fffffca50 +0x0010: 0x0007fffffca50 + 0x0000000000000001
0x0007fffffca55 +0x0010: 0x0007fffffca55 + 0x0000000000000000
0x0007fffffca60 +0x0010: 0x00000000000007f0 + <__libc_csu_init+0> push r15 + $rsp
0x0007fffffca65 +0x0010: 0x0007fffffca65 + <__libc_start_main+240> mov edi, eax
code:1308;x86-64
0x00079c <main+51> mov esi, 0x400874
0x00079d <main+52> mov rdi, rax
0x00079e <main+53> call 0x400530 <fopen@plt>
0x000799 <main+64> mov QWORD PTR [rbp-0x28], rax
0x00079d <main+65> cmp QWORD PTR [rbp-0x28], 0x0
0x00079e <main+71> jne 0x40079c <main+99>
0x00079a <main+75> lea rax, [rbp-0x20]
0x00079b <main+79> mov rsi, rax
0x00079b <main+82> mov edi, 0x400876
source:vsnsprintf.c:20
15 int main ()
16 {
17     FILE * pFile;
18     char szFileName[]="myfile.txt";
19
20     // pFile=0x0007fffffca30 + 0x0000000000000000, szFileName=0x0007fffffca50 + "myfile.txt"
21     if (pFile == NULL)
22         printfError ("Error opening '%s'",szFileName);
23     else
24     {
25         // file successfully open
threads
[00] Id 1, Name: "vsnsprintf", stopped, reason: SINGLE STEP
trace
[00] 0x400799 + Name: main()
gef>
```

Ghidra

- Développé par la NSA, révélé en 2017 par WikiLeaks, opensourcé en mars 2019 lors de la conférence RSA
- Collaboratif, retour en arrière, beaucoup d'architectures déjà présentes, multiplateforme, extensible







Ghidra - cheatsheet

Key		
Action Context	Mods + Key	Menu → Path
The action may only be available in the given context.		
❖ Indicates the context menu, i.e., right-click.		
The Ctrl key is replaced by the command  key on Macintosh.		

Load Project/Program		
New Project	Ctrl+N	File → New Project
Open Project	Ctrl+O	File → Open Project
Close Project ¹	Ctrl+W	File → Close Project
Save Project ¹	Ctrl+S	File → Save Project
Import File ¹	I	File → Import File
Export Program	O	File → Export Program
Open File System ¹	Ctrl+I	File → Open File System


¹ These actions are only available if there is an active project. Create or open a project first.

Help/Customize/Info		
Ghidra Help Hover on action	F1	Help → Contents
About Ghidra		Help → About Ghidra
About Program		Help → About <i>program name</i>
Preferences		Edit → Tool Options
Set Key Binding Hover on action	F4	
Key Bindings		Edit → Tool Options →  Key Bindings
Processor Manual	❖	→ Processor Manual

Markup		
 Undo	Ctrl+Z	Edit → Undo
 Redo	Ctrl+Shift+Z	Edit → Redo
 Save Program	Ctrl+S	File → Save <i>program name</i>
Disassemble	D	❖ → Disassemble
Clear Code/Data	C	❖ → Clear Code Bytes
Add Label Address field	L	❖ → Add Label
Edit Label Label field	L	❖ → Edit Label
Rename Function Function name field	L	❖ → Function → Rename Function
Remove Label Label field	Del	❖ → Remove Label
Remove Function Function name field	Del	❖ → Function → Delete Function
Define Data	T	❖ → Data → Choose Data Type ❖ → Data → <i>type</i>
Repeat Define Data	Y	❖ → Data → Last Used: <i>type</i>
Rename Variable Variable in decompiler	L	❖ → Rename Variable
Retype Variable Variable in decompiler	Ctrl+L	❖ → Retype Variable

Cycle Integer Types	B	❖ → Data → Cycle → byte, word, dword, qword
Cycle String Types	'	❖ → Data → Cycle → char, string, unicode
Cycle Float Types	F	❖ → Data → Cycle → float, double
Create Array ²	[❖ → Data → Create Array
Create Pointer ²	P	❖ → Data → pointer
Create Structure Selection of data	Shift+[❖ → Data → Create Structure
New Structure Data type container		❖ → New → Structure
Import C Header		File → Parse C Source
Cross References		❖ → References → Show References to <i>context</i>

² When possible, arrays and pointers are created of the data type currently applied.

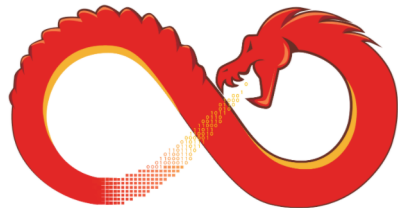
Miscellaneous		
Select		Select → <i>what</i>
Program Differences	2	Tools → Program Differences
 Rerun Script	Ctrl+Shift+R	
Assemble	Ctrl+Shift+G	❖ → Patch Instruction

Ghidra - cheatsheet

Navigation		
Go To	G	Navigation → Go To
Back	Alt+←	
Forward	Alt+→	
Toggle Direction	Ctrl+Alt+T	Navigation → Toggle Code Unit Search Direction
Next Instruction	Ctrl+Alt+I	Navigation → Next Instruction
Next Data	Ctrl+Alt+D	Navigation → Next Data
Next Undefined	Ctrl+Alt+U	Navigation → Next Undefined
Next Label	Ctrl+Alt+L	Navigation → Next Label
Next Function	Ctrl+Alt+F	Navigation → Next Function
	Ctrl+J	Navigation → Go To Next Function
Previous Function	Ctrl+↑	Navigation → Go To Previous Function
Next Non-function Instruction	Ctrl+Alt+N	Navigation → Next Instruction Not in a Function
Next Different Byte Value	Ctrl+Alt+V	Navigation → Next Different Byte Value
Next Bookmark	Ctrl+Alt+B	Navigation → Next Bookmark

Windows		
Bookmarks	Ctrl+B	Window → Bookmarks
Byte Viewer		Window → Bytes: <i>program name</i>
Function Call Trees		
Data Types		Window → Data Type Manager
Decompiler	Ctrl+E	Window → Decompile: <i>function name</i>
Function Graph		Window → Function Graph
Script Manager		Window → Script Manager
Memory Map		Window → Memory Map
Register Values	V	Window → Register Manager
Symbol Table		Window → Symbol Table
Symbol References		Window → Symbol References
Symbol Tree		Window → Symbol Tree

Search		
Search Memory	S	Search → Memory
Search Program Text	Ctrl+Shift+E	Search → Program Text
Search For ...		
Matching Instructions		
Address Tables		
Direct References		
Instruction Patterns		
Scalars		
Strings		
		Search → For what



GHIDRA

Ghidra Cheat Sheet

Ghidra is licensed under the Apache License, Version 2.0 (the "License"); Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Frida

- Framework d'instrumentation dynamique
- Souvent utilisé dans le monde mobile
(<https://github.com/sensepost/objection>)
- Permet de suivre les appels de fonctions, les registres, les adresses mémoires, etc...

FRIDA

Frida

```
import Frida, sys, os

pid = frida.spawn(["bin", "c"*0x12])
session = frida.attach(pid)
script = '''
    //javascript goes here
'''

def on_message(message, data):
    print(message)

script = session.create_script(script)
script.on('message', on_message)
script.load()

frida.resume(pid)
sys.stdin.read()
```

```
function patch(addr, code, log){
    Memory.protect(addr, code.length, 'rwx');
    Memory.writeByteArray(addr, code);
    Memory.protect(addr, code.length, 'r-x');
    console.log("[+] Patched ", log);
}

var baseAddr = Module.findBaseAddress('bin');
console.log('baseAddr: ' + baseAddr);

Interceptor.attach(
    baseAddr.add(0x123456), {
        onEnter: function(args) {
            send('todo');
        }
    }
);
console.log("[+] Interceptor 0x123456")
```

Frida : astuces

EBFE

Pour "arrêter" un programme, on va le faire boucler à l'infini avec un `JMP -2`
`patch(baseAddr.add(0x1821), [0xeb, 0xfe, 0x00, 0x00], "EBFE")`

- Mettre BEAUCOUP de `console.log()` pour debug un script Frida.

5 Format de fichier

- Introduction
- PE : Portable Executable
- ELF : Executable and Linkable Format

Généralités

ELF : Executable and Linkable Format

- Linux
- Android
- Console de jeux (PS2-PS5)

PE : Portable Executable

- Windows
- UEFI

Mach-O (Apple)

- macOS
- iOS

Présentation

- Introduit depuis Windows NT 3.1
- 2 en-têtes : MZ (historique) et ensuite PE
- l'en-tête PE est un ensemble de structure décrivant comment charger le programme

En-tête MZ-DOS
Section DOS
En-tête PE
En-têtes optionnels
Tables des sections
Section 1
⋮
Section n

Exemple

PORTABLE EXECUTABLE

ANGE ALBERTINI 
<http://www.corkami.com>

D:\>mini.exe

D:\>echo %errorlevel%
42

```

0 1 2 3 4 5 6 7 8 9 A B C D E F
000: .M .Z
030:                                     40 00 00 00
040: .P .E 00 00 4C 01
050:           02 00 0B 01
060:               40 01 00 00
070:           00 00 40 00 01 00 00 00 01 00 00 00
080:               04 00
090: 60 01 00 00 40 01 00 00           03 00
140: B8 2A 00 00 00 C3
  
```

MINI.EXE

DOS HEADER
IT'S A BINARY

FIELDS

e_magic
e_lfanew

VALUES

MZ
0x40 → PE Header

PE HEADER
IT'S A 'MODERN' BINARY

Signature
Machine
Characteristics

PE\0\0
0x14C [intel 386]
2 [executable]

OPTIONAL HEADER
EXECUTABLE INFORMATION

Magic
AddressOfEntryPoint
ImageBase
SectionAlignment
FileAlignment
MajorSubsystemVersion
SizeOfImage
SizeOfHeaders
Subsystem

0x10B [32b]
0x140
0x400000
1
1
4 [NT 4 or later]
0x160
0x140
3 [CLI]

CODE

X86 ASSEMBLY

mov eax, 42
retn

EQUIVALENT C CODE

return 42;

Présentation

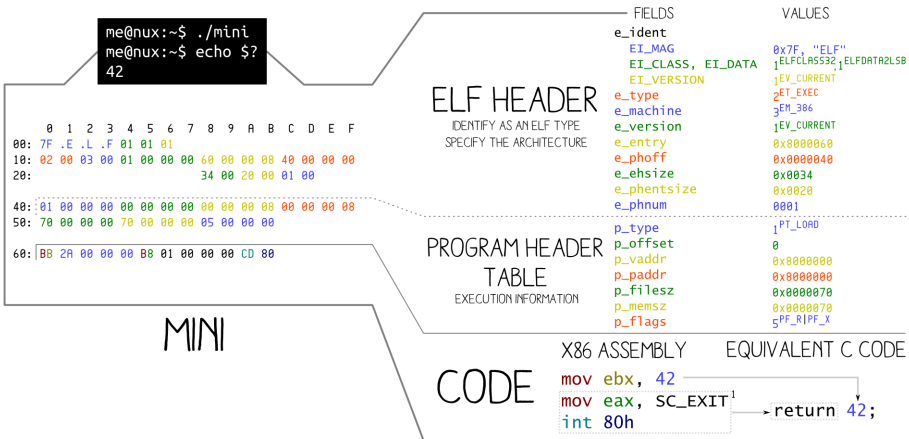
- Collections d'en-têtes décrivant l'architecture mémoire du binaire
- Utilisé pour les exécutables, objets, bibliothèques partagées
- Contient les informations :
 - Les symboles importés/exportés
 - Informations nécessaires à l'exécution (adressage, point d'entrée...)
 - Dépendances et bibliothèques dynamiques
- l'en-tête PE est un ensemble de structure décrivant comment charger le programme

Pour plus d'information : `man elf`

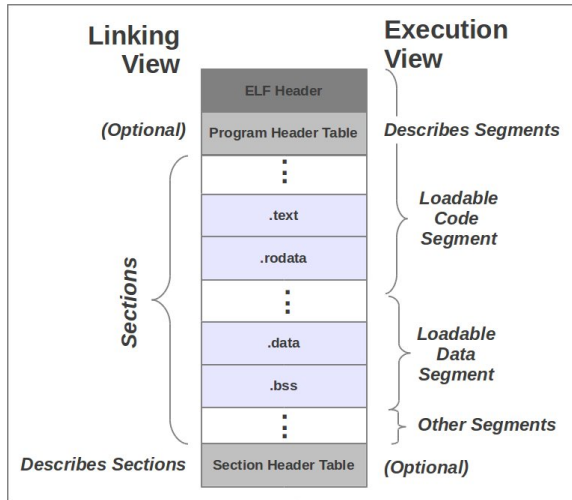
En-tête ELF EHdr
Table d'en-tête de programme (Phdr)
Section 1
⋮
Section n
Table des en-têtes des sections (Shdr)

Exemple

EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
<http://www.corkami.com>

Section VS Segment



Source : Mugabi Siro

6 Méthodologies d'analyses

- Introduction
- Graphe de flot de contrôle
- Graphe d'appels de fonctions
- Trucs et astuces

Généralités

Méthode descendante (top-down)

- Commencer par le point d'entrée de du programme
- Analyse macroscopique des fonctions
- Raffiner l'analyse sur les fonctionnalités pertinentes

Outil

- Graphe de flot de contrôle (CFG)
- Graphe des appels de fonctions

Généralités

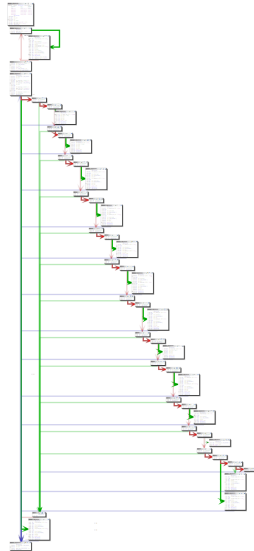
Méthode ascendante (bottom-up)

- Commencer par le code assembleur
- Suivre une chaîne de caractères, une opération sur une variable
- Remonter la chaîne d'appels

Outil

- Analyse dynamique (GDB)
- strings

Exemple de CFG



Structure conditionnel

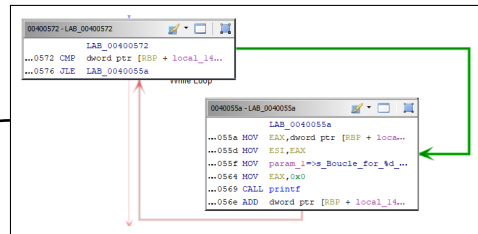
```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char** argv)
{
    // gcc control_flow.c -o control_flow
    int sum, i;
    printf("%d\n", argc * 2);
    for (i = 0; i < 10; i++){
        printf("Boucle for %d", i);
    }
    if (argc > 2){
        printf("argc > 2");
    } else{
        printf("argc <= 2");
    }
    while(i < 20){
        printf("boucle while %d", i); i++;
    }
    return 0;
}
```

Structure conditionnel

```
#include <stdlib.h>
#include <stdio.h>

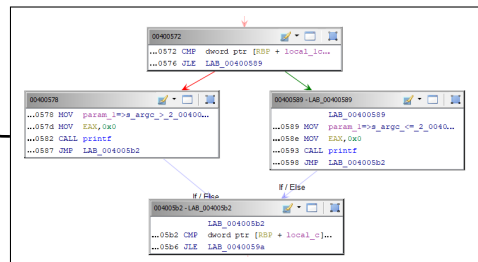
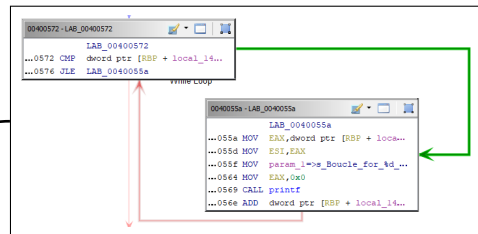
int main(int argc, char** argv)
{
    // gcc control_flow.c -o control_flow
    int sum, i;
    printf("%d\n", argc * 2);
    for (i = 0; i < 10; i++){
        printf("Boucle for %d", i);
    }
    if (argc > 2){
        printf("argc > 2");
    } else{
        printf("argc <= 2");
    }
    while(i < 20){
        printf("boucle while %d", i); i++;
    }
    return 0;
}
```



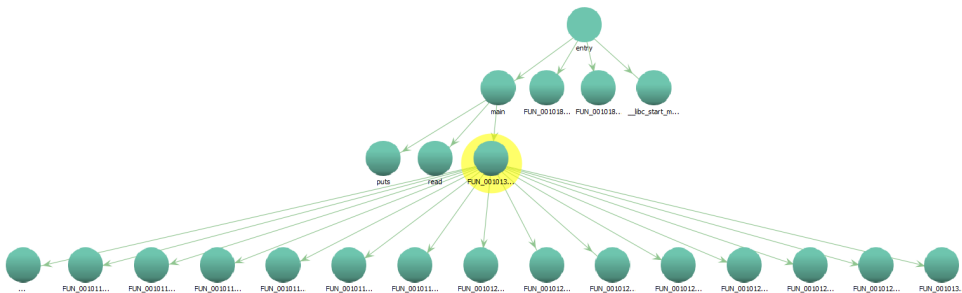
Structure conditionnel

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char** argv)
{
    // gcc control_flow.c -o control_flow
    int sum, i;
    printf("%d\n", argc * 2);
    for (i = 0; i < 10; i++){
        printf("Boucle for %d", i);
    }
    if (argc > 2){
        printf("argc > 2");
    } else{
        printf("argc <= 2");
    }
    while(i < 20){
        printf("boucle while %d", i); i++;
    }
    return 0;
}
```



Graphe d'appels de fonctions



L'observation

- Beaucoup d'informations simples à récupérer facilement
- Se faire une idée sur quelle partie du programme il faut se focaliser
- Ce qui est observable :
 - Les entrées et sorties du programme
 - Les informations des en-têtes, potentiellement son langage d'origine
 - Les interactions avec l'environnement (API système)

Analyse statique ou dynamique ?

Analyse statique

Permet

- Désassembler l'application
- lire le contenu statique non chiffré

Mais

- Obfuscation
- Prend beaucoup de temps
- Entièrement en mode boîte noire

Analyse dynamique

Permet

- Récupérer les entrées/sortie
- Debugging, Hooking, Emulation

Mais

- Anti-debug/tampering/dump
- Obfuscation
- Aucune certitude ce qui est constaté

Cibler l'analyse

À éviter

- Tout débbugger
 - compliqué de mise en oeuvre
 - génère des erreurs la plupart du temps
- Se lancer tête baissée dans l'assembleur
 - Le point d'entrée choisi sera très probablement le mauvais
 - Prend BEAUCOUPS trop de temps

Préférer une approche hybride

- Toujours commencer par des choses simples
- Récupérer un maximum d'informations sur l'architecture, le langage, les libs utilisés
- Internet (stackoverflow, github, etc...) est votre ami !

Scripter

- Connaitre un langage de scripting (Python3 pour GDB, Ghidra via Jython)
- Ne pas hésiter à recoder dans le langage une portion du code décompilé
- Automatiser vos approches récurrentes (si c'est pertinent)
 - Recherche de structure identique dans un bloc de donnée.
- Utiliser les outils de la communauté

Connaitre différents langages

- Le langage de programmation utilisé va complètement modifier le code généré
- Le concept d'objet peut particulièrement être déroutant comme en C++ ou en Go
- Pour apprendre, développer ou trouver un code source équivalent, compilez-le et analysez-le

7 Exemple de techniques d'anti-debug, d'obfuscation

- Généralités
- Anti-debug
- Détection des environnements de tests
- Obfuscation
- Les machines virtuelles

Généralités

Objectifs

- Rendre le programme suffisamment compliqué pour que la majorité des gens ne puisse pas le cracker
- Ralentir le plus possible les attaquants pour protéger les premières semaines d'exploitations
- Protéger un secret algorithmique

Généralités

Objectifs

- Rendre le programme suffisamment compliqué pour que la majorité des gens ne puisse pas le cracker
- Ralentir le plus possible les attaquants pour protéger les premières semaines d'exploitations
- Protéger un secret algorithmique

Mais...

- C'est le jeu du chat de la souris
- Possibilité de modifier un binaire pour outrepasser une vérification
- Le coût financier d'une protection, l'avis du public sur les DRM
- Toute protection possède un coût en performance (ex : Denuvo)

Vérifier la présence d'un débogueur

Modifier le comportement du programme lors de son utilisation avec un débogueur

Vérifier la présence d'un débogueur

Modifier le comportement du programme lors de son utilisation avec un débogueur

Linux

- Vérifier PTRACE_TRACEME

```
if (ptrace(PTRACE_TRACEME, 0, NULL, 0) == -1)
    printf("traced!\n");
```

- Vérifier le contenu de /proc/\$PID/cmdline avec \$pid = getpid()
- Lever un SIGTRAP

Vérifier la présence d'un débogueur

Modifier le comportement du programme lors de son utilisation avec un débogueur

Linux

- Vérifier PTRACE_TRACEME

```
if (ptrace(PTRACE_TRACEME, 0, NULL, 0) == -1)
    printf("traced!\n");
```

- Vérifier le contenu de /proc/\$PID/cmdline avec \$pid = getppid()
- Lever un SIGTRAP

Windows

- Vérifier kernel32!IsDebuggerPresent ou kernel32!CheckRemoteDebuggerPresent
- Vérifier ntdll!NtQueryInformationProcess
- Créer un thread avec ThreadHideFromDebugger

Détection des environnements de tests

Pourquoi une VM/sandbox ? (vmware, virtualbox)

- Analyser un malware dans un environnement jetable
- L'automatisation
- Il est facile de suivre ce qui est fait dans une VM

Détecter une VM

- La plupart des astuces sont dépendantes des environnements ciblés
- Les malwares incorporent nombre de protection pour éviter qu'ils soient exécutés dans le cloud des moteurs antivirus

Exemple

Techniques classiques

- Vérifier les identifiants du matériel du système d'exploitation (MAC, fichiers, registres)
- Utiliser des instructions CPU spécifiques comme CPUID, SIDT
- Vérifier le temps d'exécution
- Utiliser la backdoor entre l'hôte et l'invité

Exemple

Techniques classiques

- Vérifier les identifiants du matériel du système d'exploitation (MAC, fichiers, registres)
- Utiliser des instructions CPU spécifiques comme CPUID, SIDT
- Vérifier le temps d'exécution
- Utiliser la backdoor entre l'hôte et l'invité

Exemples

- Vérification du préfix de l'adresse MAC ("\\x00\\x05\\x69", "\\x00\\x0C\\x29", ...)
- Vérification du process vboxservices
- Vérifier les informations du processeur

```
unsigned int cpuInfo[4];  
__cpuid((int*)cpuInfo, 1);  
return ((cpuInfo[2] >> 31) & 1) == 1;
```

Obfuscation

Généralités

- Réduire le nombre d'informations accessibles facilement
- Compliquer au maximum l'analyse

Applications

- DRM : Digital Right Management (ex : Widevine)
- Protection des données (ex : clé de chiffrement)
- Malware

Exemple

Les classiques

- Insertion de code mort
- Modifier/supprimer les symboles lors de la compilation
- Chiffrer les chaînes de caractères

Les techniques

- Modifier le flot de contrôle (tout mettre à plat)
- Packager le code et le chiffrer

Les avancées

- L'utilisation des prédicats opaques
- Utilisation d'une VM (au sens ASM)

Modifier les symboles

Exemple d'un malware en Go :

main.main	00517fc0	int main.main...
main.Run	005180a0	undefined mai...
main.rzSF37vf8khc8Vye	005181a0	undefined mai...
main.g7Srk53hUCFt86bQ	00518300	undefined mai...
main.QJ4rg98Fn233nn4s	00518500	undefined mai...
main.DrERWCNyzs9az2eW	00518620	undefined mai...
main.JJ23FehAEuwgD2Qv	005187a0	undefined mai...
main.KyRf6LTuBVzUfacr	00518920	undefined mai...
main.ZCxFSYJQ7tsSP3Zn	00518b60	undefined mai...
main.TsPDE4w9RXwcc4rm	00518d60	undefined mai...
main.Tf5QX6MLXHpCESbu	00518e80	undefined mai...
main.s3jaCsNTu4J8cqth	00518fc0	undefined mai...
main.mUBSP5wQrvrhp5FC	005191a0	undefined mai...
main.mjDE8mmD57D2pk6L	005193c0	undefined mai...
main.Run.func1	005195e0	undefined mai...

Modifier le flot de contrôle

Exemple d'un binaire ne possédant qu'une seule fonction :



Les prédicats opaques

Objectifs

Complexifier la lecture des conditions avec l'utilisation de prédicats toujours vrais ou faux complexes.

Les prédicats opaques

Objectifs

Complexifier la lecture des conditions avec l'utilisation de prédicats toujours vrais ou faux complexes.

Exemple

```
int i = 0, sum = 0;
while(i < 42 && !((3*(i+1)*(i+1)*(i+1)+2*(i+1)*(i+1)+(i+1))%6 == 0)) {
    sum += numbers[i++];
}
k = 0
while(i < 100) {
    sum += numbers[i++];
}
```

Les prédicats opaques

Objectifs

Complexifier la lecture des conditions avec l'utilisation de prédicats toujours vrais ou faux complexes.

Exemple

```
int i = 0, sum = 0;
while(i < 42 && !((3*(i+1)*(i+1)*(i+1)+2*(i+1)*(i+1)+(i+1))%6 == 0)) {
    sum += numbers[i++];
}
k = 0
while(i < 100) {
    sum += numbers[i++];
}
```

L'équation $(3 * x^3 + 2 * x^2 + x) \% 6 == 0$ est toujours vraie

Outrepasser les prédicats opaques

- Executer/Émuler le code pour étudier son comportement
- Utiliser des solvers (ex : <https://github.com/Z3Prover/z3>)
- L'exécution symbolique dynamique peut aider (ex : <https://github.com/JonathanSalwan/Triton>)

Généralités

Convertir le jeu d'instruction dans un nouveau jeu d'instructions.

Généralités

Convertir le jeu d'instruction dans un nouveau jeu d'instructions.

Avantages

- Obligation d'écrire un outil de décodage
- Il est très difficile de comprendre le nouveau set d'instruction
- Le flot de contrôle est complètement différent depuis l'extérieur de la VM
- L'empreinte mémoire est faible

Généralités

Convertir le jeu d'instruction dans un nouveau jeu d'instructions.

Avantages

- Obligation d'écrire un outil de décodage
- Il est très difficile de comprendre le nouveau set d'instruction
- Le flot de contrôle est complètement différent depuis l'extérieur de la VM
- L'empreinte mémoire est faible

Mais...

- Le coût en ressource CPU est élevé
- Il est difficile de mettre à jour une fois la logique de la protection comprise

Architecture d'une VM

