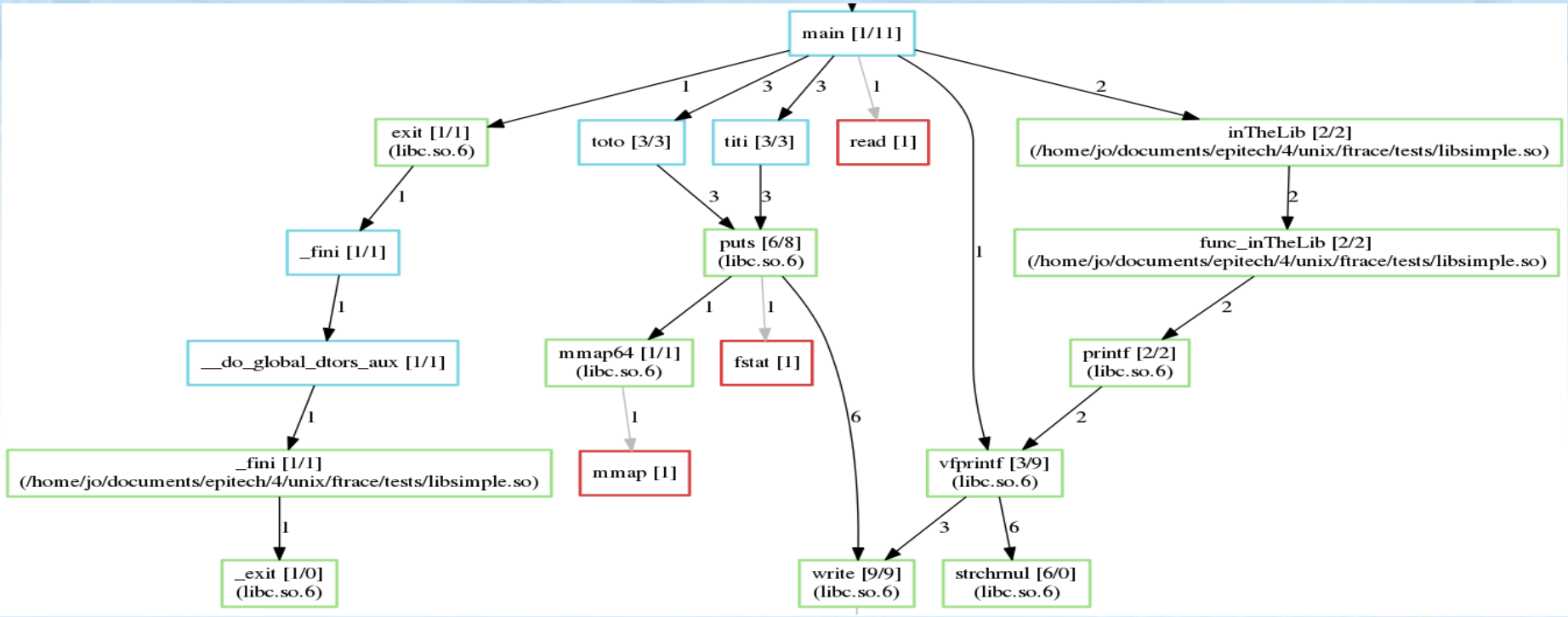

EPITECH.
L'ECOLE DE L'EXPERTISE INFORMATIQUE
Quand un passionné devient un expert reconnu
European Institute of Information Technology
Titre homologué par l'Etat - Niveau I (CNCP)
medega_j

FTRACE : Qu'est ce que c'est ?

- Générateur de CallGraph
- Permet d'avoir une vue d'ensemble du flux d'exécution d'un processus (appels de fonctions -statiques & dynamiques-, syscalls, signaux, ..)
- Bien plus complexe que le Strace, demande une parfaite compréhension des mécanismes d'édition de lien dynamique & des différents appels.



Subdivisions

La réalisation d'un Ftrace peut se diviser en 4 grandes parties :

- Core de tracing basé sur PTRACE (Cf. Strace)
- Détection des opcodes d'appel de fonction
- Résolution des adresses de tous les symboles (fonctions) indexés par le binaire (statiques ou dynamiques)
- Génération du graphique logique

Core de tracing : Rappel

fork (si le processus à tracer n'existe pas)

PTRACE_TRACEME dans le fils (ou PTRACE_ATTACH si le processus existe déjà)

wait4 dans le père (pour attendre les changements de statut du fils)

execve dans le fils (pour lancer le binaire à tracer)

Tracer instruction par instruction à l'aide de PTRACE_SINGLESTEP

A chaque retour de wait4, faire le travail de Ftrace

Passer à l'instruction suivante

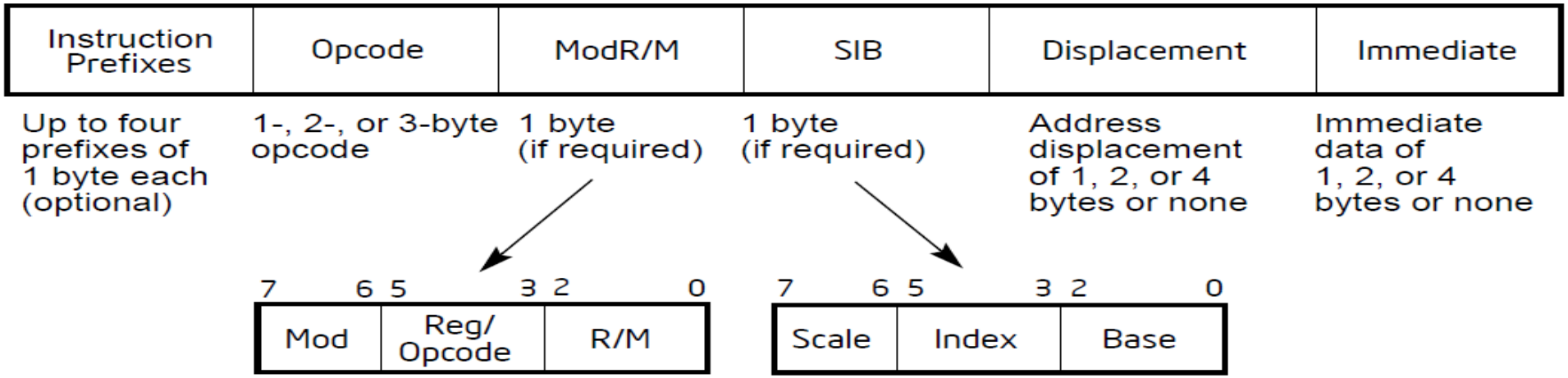
Appel de fonction : CALL

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
E8 *cw*	CALL *rel16*	M	N.S.	Valid	Call near, relative, displacement relative to next instruction.
E8 *cd*	CALL *rel32*	M	Valid	Valid	Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.
FF /2	CALL *r/m16*	M	N.E.	Valid	Call near, absolute indirect, address given in *r/m16*.
FF /2	CALL *r/m32*	M	N.E.	Valid	Call near, absolute indirect, address given in *r/m32*.
FF /2	CALL *r/m64*	M	Valid	N.E.	Call near, absolute indirect, address given in *r/m64*.
9A *cd*	CALL *ptr16:16*	D	Invalid	Valid	Call far, absolute, address given in operand.
9A *cp*	CALL *ptr16:32*	D	Invalid	Valid	Call far, absolute, address given in operand.
FF /3	CALL *m16:16*	M	Valid	Valid	Call far, absolute indirect address given in *m16:16*. In 32-bit mode: if selector points to a gate, then RIP = 32-bit zero extended displacement taken from gate; else RIP = zero extended 16-bit offset from far pointer referenced in the instruction.
FF /3	CALL *m16:32*	M	Valid	Valid	In 64-bit mode: If selector points to a gate, then RIP = 64-bit displacement taken from gate; else RIP = zero extended 32-bit offset from far pointer referenced in the instruction.

Appel de fonction : Le Mod R/M & SIB

- Le Mod R/M & le SIB sont des octets supplémentaires qui sont rajoutés aux opcodes afin d'encoder de manière efficiente, les paramètres d'adressages.

- Man intel : sections #2.1, #3.1.1.1, …
(*http://download.intel.com/products/processor/manual/325383.pdf*)



Les symboles : Linkage

- Linkage statique : Tous les symboles référencés par le binaire sont inclus dans celui ci, aussi bien le code des fonctions écrites par l'utilisateur que celui provenant de librairies externes.

- Linkage dynamique : Les symboles externes référencés par le binaire sont chargés dans l'espace mémoire du processus, au lancement de celui-ci, & sont rendus accessibles depuis le code utilisateur par un processus de résolution (l'éditeur de lien dynamique).

Les symboles : GOT & PLT

- *Global Offset Table* : Sorte de tableau/cache/index contenant des adresses, dont les adresses résolues des symboles importés.

- *Procedure Linkage Table* : Section contenant du code (un set de 3 instructions par fonction importée) qui est appelé depuis le code utilisateur & qui sert de pont vers le code d'une fonction disponible dans une librairie dynamique, quelque part dans l'espace mémoire du processus. Ce code dans un premier temps, consulte la GOT, au cas où l'adresse du symbole en question aurait déjà été résolue, le cas échéant, redirige l'exécution à cette adresse là, sinon appelle la fonction (l'éditeur de lien dynamique) qui va s'occuper de la résolution à proprement dit (cette dernière s'occupe de mettre à jour la GOT une fois l'adresse du symbole résolue afin de « cacher » le résultat de la résolution pour les prochains appels à la fonction (comportement modifiable avec la variable d'environnement *LD_BIND_NOT* (« man ld »)).

- Pour retrouver les noms des symboles importés, on va exploiter la section (de type *SHT_REL(A*) (man elf)) du binaire qui contient les informations de relocations des symboles de type : *R_386_JUMP_SLOT*. Chacun de ses objets (Elf64_Rel) nous donnera les informations (adresse dans la GOT, offset dans une string table du nom de la fonction) via des macros utilisées avec son champ « *r_info* ».

Graphique : Dotty

- Framework simplifié de génération de graphes (utilisé par doxygen par exemple..)
- Syntaxe intuitive (langage DOT) :

```
digraph G {
  a -> b [label="hello", style=dashed];
  a -> c [label="world"];
  c -> d; b -> c; d -> a;
  b [shape=Mdiamond, label="this is b"];
  c [shape=polygon, sides=5, peripheries=3];
  d [style=bold];
}
```

Précisions

- Avec un CALL va un RET
- /proc/[PID]/ (maps, …)
- ldd
- Link_map
- Breakpoints (INT 0x03)
- Droit à la libelf
- Pensez aux symboles importés dynamiquement (dlopen, dlsym, ..)
- Résolution des symboles statiques & même dynamiques des librairies dynamiques
- Bonus : Symboles de DEBUG (Stabs/Dwarf) => Droit à la libdwarf

Questions ?