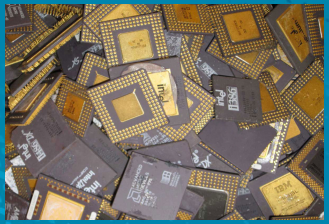


cpu_rec.py, un outil statistique pour la reconnaissance d'architectures binaires exotiques

Louis Granboulan

SSTIC, 9 juin 2017, Rennes

Ces travaux ont été en partie réalisés au sein du projet EIC de l'IRT SystemX

AIRBUS

Résumé

1. Le problème à résoudre : reconnaissance d'architecture dans un binaire
2. Contenu scientifique : apprentissage et statistiques
3. Mode d'emploi : binwalk, ou bien standalone
4. Exemples
5. Conclusion, perspectives

Qu'est-ce que la « reconnaissance d'architectures dans un binaire » ?

1. D'abord, c'est quoi un binaire ?

Évidemment, tout le contenu d'un ordinateur est binaire... on se limite

- à des fichiers
- contenant des instructions à faire exécuter par un microprocesseur
- directement lisibles (pas de compression ni de chiffrement)
- dont on ne connaît pas bien le contenu
- et qu'on veut analyser (rétro-ingénierie)

Exemples

- Format inconnu (pas COFF, ELF, Mach-O, ...)
- Firmware *bare metal* (directement exécuté sur le microprocesseur)
- Dump mémoire (RAM, flash, ROM...)

Qu'est-ce que la « reconnaissance d'architectures dans un binaire » ?

2. Et c'est quoi une architecture ?

ISA (Instruction Set Architecture) décrit un jeu d'instruction de microprocesseurs

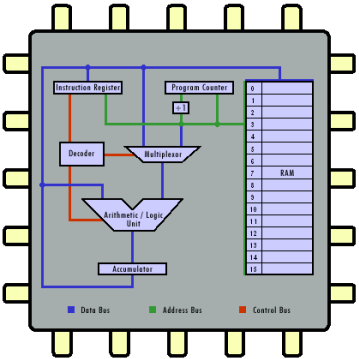
- Deux microprocesseurs différents peuvent avoir le même ISA ou des ISA sous-ensembles l'un de l'autre (e.g. Core i5 vs. Core i3 vs. Pentium III)
- Un microprocesseur peut gérer plusieurs ISA (e.g. MIPS big ou little endian, ARM Thumb ou non)
- Le compilateur peut sélectionner un sous-ensemble des instructions
- Ma définition : une architecture, c'est ce que reconnaît `cpu_rec.py` !

Les 72 architectures reconnues par `cpu_rec.py` (corpus par défaut)

- 68HC08, 68HC11, 8051, ARM64, ARMeb, ARMel, ARMhf, ARcompact, AVR, Alpha, AxisCris, Blackfin, CLIPPER, Cell-SPU, CompactRISC, Cray, Epiphany, FR-V, FR30, FT32, H8-300, HP-Focus, HP-PA, IA-64, IQ2000, M32C, M32R, M68k, M88k, MCore, MIPS16, MIPSeb, MIPSel, MMIX, MN10300, MSP430, Mico32, MicroBlaze, Moxie, NDS32, NIOS-II, OCaml, PDP-11, PIC10, PIC16, PIC18, PIC24, PPCeb, PPCel, RISC-V, RL78, ROMP, RX, S-390, SPARC, STM8, Stormy16, SuperH, TILEPro, TLCS-90, TMS320C2x, TMS320C6x, V850, VAX, Visium, WE32000, X86, X86-64, Xtensa, Z80, i860, et 6502 compilé avec `cc65`

Qu'est-ce que la « reconnaissance d'architectures dans un binaire » ?

3. Pourquoi `cpu_rec.py` utilise des propriétés statistiques ?



Source : `http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/index.html`

- Le module du CPU qui interprète les instructions est le décodeur
- Que les instructions aient une taille fixe ou une taille variable, le décodeur fonctionne octet par octet
- Donc la distribution de probabilité d'un octet dépend en particulier de la valeur de l'octet précédent

Idée fondatrice de `cpu_rec.py`

Cela suffit à reconnaître une architecture

Apprentissage et corpus

Principes de l'apprentissage

- On part d'un corpus annoté (des exemples de chaque architecture)
- Apprentissage de ce corpus
- Utilisation du résultat pour classifier des binaires d'architecture inconnue

Corpus disponible

- Pour les architectures courantes (x86, ARM, SPARC, MIPS, Alpha, IA64, PPC, PA-RISC, ...) : plein d'exemples
- Pour des architectures plus rares ou exotiques (PIC16, Mico32, RISC-V, Cray, RL78, M-CORE, CLIPPER, ...) : peu d'exemples disponibles
- Actuellement, 72 architectures sont présentes dans le corpus

Machine learning et binaires

État de l'art

- Pour expérimenter avec le *machine learning* ou l'*apprentissage* il y a la bibliothèque `scikit-learn`
- Un binaire est une suite d'octets, donc un texte sur un alphabet, donc les méthodes d'analyse de texte et de classification de documents s'appliquent

Ça marche plutôt mal

- Parce que le corpus disponible pour l'apprentissage est trop petit (100 Ko pour certaines architectures) et/ou trop hétérogène
- Parce que je ne suis pas un spécialiste de machine learning
- Mais résultats acceptables avec *Multinomial Naive Bayes* sur des (2,3)-grammes

Chaînes de Markov et distance de Kullback-Leibler

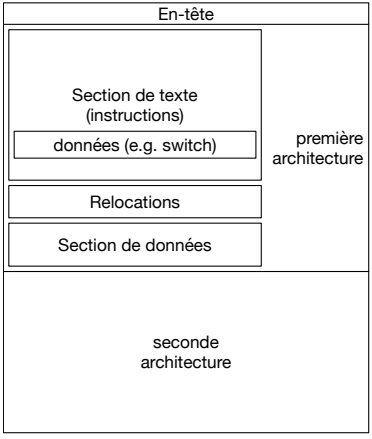
Chaînes de Markov

- Probabilité conditionnelle d'un octet connaissant le précédent
- Équivalence avec la distribution de probabilité des bigrammes
- Modélisons chaque architecture par la distribution des bigrammes/trigrammes

Distance entre chaînes de Markov / distribution de n-grammes

- Ça s'appelle *entropie croisée* ou *divergence de Kullback-Leibler*
- Ça donne la même classification que *Multinomial Naive Bayes* mais avec une mesure de confiance
- `scikit-learn` ne connaît pas, donc `cpu_rec.py` ne l'utilise pas

Où regarder ?



Ce qui nous intéresse n'est pas partout :

- Un binaire ne contient pas que des instructions (plusieurs sections, données et instructions entrelacées...)
- Il peut contenir plusieurs architectures

Solution : fenêtre glissante

Ni trop longue, ni trop courte : 0x1000 par défaut, mais moins pour les petits binaires

On pourrait aussi avoir une fenêtre de taille variable, en utilisant comme indicateur la distance aux distributions apprises... mais ce serait plus lent

Mode d'emploi

Logiciel autonome en python

- Disponible sur `https://github.com/airbus-seclab/cpu_rec/`
- Fourni avec un corpus reconnaissant 72 architectures
- Compatible avec python 3 et python 2 (si au moins 2.4)

Arguments et résultat

- En arguments, les fichiers à analyser, et un ou plusieurs `-v` pour la verbosité
- Les fichiers peuvent avoir été comprimés, ou être au format HEX (firmware)
- En sortie, l'architecture suggérée (pour tout le fichier et la section de texte) et le résultat de l'analyse par fenêtre glissante (adresse et taille de la zone détectée)

```
shell_prompt> cpu_rec.py corpus/PE/PPC/NTDLL.DLL corpus/MSP430/goodfet32.hex
corpus/PE/PPC/NTDLL.DLL       full(0x75b10)None    text(0x58800)PPCel chunk(0x4c800;153)PPCel
corpus/MSP430/goodfet32.hex   full(0x61ac) None                       chunk(0x5200;41)  MSP430
```

Module pour binwalk

Intégration dans binwalk

- Installation dans `~/.config/binwalk/modules/`
- Versions récentes de binwalk uniquement, python 2 ou python 3
- Analyse par fenêtre glissante, heuristiques pour éliminer les outliers

Performances

- Création des signatures pour 72 architectures : 25s et 1Go de RAM
- 60s par Mo de fichier analysé

```
shell_prompt> binwalk -% corpus/PE/PPC/NTDLL.DLL

DECIMAL     HEXADECIMAL   DESCRIPTION
-----------------------------------------------
0           0x0           None (size=0x5800)
22528       0x5800        PPCel (size=0x4c800)
335872      0x52000       None (size=0x1000)
339968      0x53000       IA-64 (size=0x800)
342016      0x53800       None (size=0x21800)
```

Parfois de petites erreurs (e.g. IA-64 ci-dessus, zone qui en réalité est dans la table d'adresses du répertoire Export)

Quelques exemples de résultats obtenus avec `cpu_rec.py`

Ces exemples sont choisis pour montrer des cas limites d'utilisation de l'outil

- Fichier avec plusieurs architectures et imprécision d'analyse
- Fichier avec plusieurs architectures et trop peu d'instructions
- Architecture exotique connue
- Architecture exotique inconnue

Fichier avec plusieurs architectures et imprécision d'analyse

Fichier Mach-O FAT

- `OSXII`, exécutable MacOSX avec deux architectures : ppc et i386

Schéma du contenu du fichier.

```
0x00000 Mach-O Header
0x01000 ppc     Macho-O Header
0x02180 ppc     __TEXT,__text (size=0x1AB5C)
0x281E0 ppc     end
0x29000 i386    Mach-O Header
0x2A7D0 i386    __TEXT,__text (size=0x1B87D)
0x51F5C i386    end
```

```
shell_prompt> binwalk -% OSXII
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------
0             0x0             None (size=0x1800)
6144          0x1800          PPCeb (size=0x1b800)
118784        0x1D000         None (size=0xd000)
172032        0x2A000         X86 (size=0x2000)
180224        0x2C000         None (size=0x800)
182272        0x2C800         X86-64 (size=0x800)
184320        0x2D000         X86 (size=0x18800)
284672        0x45800         None (size=0xc000)
```

Une petite zone au milieu de la section exécutable de l'architecture i386 n'est pas bien reconnue. Ça n'est pas gênant en pratique.

Fichier avec plusieurs architectures et trop peu d'instructions

Fichier Mach-O FAT

- `SweetHome3D`, avec trois architectures : ppc, i386, x86_64

Schéma du contenu du fichier.

```
0x00000 Mach-O Header
0x01000 ppc     Macho-O Header
0x01CC8 ppc     __TEXT,__text (size=0xD38)
0x06F00 ppc     end
0x07000 i386    Mach-O Header
0x07BF0 i386    __TEXT,__text (size=0x2EC)
0x0C9F0 i386    end
0x0D000 x86_64 Mach-O header
0x0DB44 x86_64 __TEXT,__text (size=0x235)
0x11750 x86_64 end
```

```
shell_prompt> binwalk -% SweetHome3D
DECIMAL     HEXADECIMAL   DESCRIPTION
--------------------------------------------------
0           0x0           None (size=0x1c00)
7168        0x1C00        PPCeb (size=0x1000)
11264       0x2C00        None (size=0xe800)
```

Seule l'architecture PowerPC a été détectée, les sections exécutables pour les autres architectures sont trop petites. Mais si on extrait un Mach-O thin, alors son architecture est détectée, car la taille des fenêtres est ajustée.

```
shell_prompt> binwalk -% SweetHome3D.x86_64
DECIMAL     HEXADECIMAL   DESCRIPTION
--------------------------------------------------
0           0x0           None (size=0xa00)
2560        0xA00         X86-64 (size=0x400)
3584        0xE00         None (size=0x200)
4096        0x1000        OCaml (size=0x200)
4608        0x1200        None (size=0x3400)
```

Architecture exotique connue

Architecture Clipper

- `https://en.wikipedia.org/wiki/Clipper_architecture`
- RISC 32-bit, utilisé par Intergraph entre 1986 et 1990

- Le corpus se base sur C-Kermit : `cku196.clix-3.1` copié de `ftp://kermit.columbia.edu/kermit/bin/`
- `cpu_rec.py` sait reconnaître l'architecture des binaires issus de `https://web-docs.gsi.de/~kraemer/COLLECTION/INTERGRAPH/starfish.osfn.org/Intergraph/index.html`

```
shell_prompt> binwalk -% boot.1
DECIMAL      HEXADECIMAL   DESCRIPTION
--------------------------------------------------
0            0x0           CLIPPER (size=0x22000)
139264       0x22000       None (size=0x1000)
143360       0x23000       IA-64 (size=0x800)
145408       0x23800       PIC24 (size=0x800)
147456       0x24000       None (size=0x10800)
215040       0x34800       CLIPPER (size=0x2e800)
405504       0x63000       None (size=0x18000)
503808       0x7B000       CLIPPER (size=0x19000)
606208       0x94000       None (size=0x4a800)
```

Quelques petites zones ne sont pas bien reconnues. Ça n'est pas gênant en pratique. On voit apparaître trois grandes zones contenant des instructions.

Vu que je ne connais pas le format de ce fichier (contrairement à `cku196.clix-3.1` qui est une variante de COFF) je n'en sais pas plus.

Architecture exotique inconnue

Automate industriel

- Bus Controller Module X20BC0083 de B&R Automation
- Firmware accessible sur `https://www.br-automation.com/fr-ch/telecharger/software/automation-studio/hw-upgrades/v26-hw-upgrade-x20bc0083/`
- Contient 4 fichiers `7966_0.fw`, `7966_1.fw`, `7966_3.fw` et `7966_4.fw`

```
shell_prompt> binwalk -% 7966_3.fw
DECIMAL      HEXADECIMAL   DESCRIPTION
--------------------------------------------------
0            0x0           None (size=0x800)
2048         0x800         IA-64 (size=0x400)
3072         0xC00         None (size=0x17800)
99328        0x18400       IA-64 (size=0x200)
99840        0x18600       None (size=0x2a800)
273920       0x42E00       IA-64 (size=0x200)
274432       0x43000       None (size=0x4600)
```

Rien n'est reconnu. Mais si l'un des quatre fichiers est rajouté au corpus, les autres sont reconnus comme ayant la même architecture.

Après quelques échanges avec B&R Automation, c'est un FPGA programmé maison.

Conclusion

https://github.com/airbus-seclab/cpu_rec/

Ça marche !

- Approche simple, ne nécessitant aucune compréhension du fonctionnement d'un CPU pour le rajouter au corpus
- Des imprécisions, mais c'est normal pour une approche statistique
- Réduit grandement l'espace des possibilités pour un reverser

Que faire d'autre ?

- Rajouter des architectures manquantes (aidez-moi !)
- D'autres outils, dédiés à certaines architectures, pour avoir des informations plus complètes et plus précises ; par exemple :
  - Repérage plus précis du début du code exécutable
  - Si pas PIE, trouver l'adresse de chargement en mémoire
  - Pour ARM, mieux localiser les diverses variantes (en particulier Thumb)