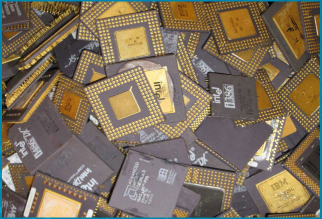


cpu_rec.py, a tool for statistical recognition of exotic architectures in a binary

Louis Granboulan

SSTIC, 9 juin 2017, Rennes

This work was partially done within project EIC of IRT SystemX

Abstract

1. Problem to solve: recognise the architecture in a binary
2. Scientific content: learning and statistics
3. User manual: with binwalk, or standalone
4. Examples
5. Conclusion, perspectives

What means "recognise the architecture in a binary"?

1. First, what is a binary?

Of course, everything in a is binary... we only consider

- files
- containing instructions to be executed by a microprocessor
- directly readable (no compression and no encryption)
- with unknown content
- that we want to analyse (by reverse-engineering)

Examples

- Unknown file format (not COFF, ELF, Mach-O, ...)
- Bare metal firmware
- Memory dump (RAM, flash, ROM...)

What means "recognise the architecture in a binary"?

2. And what is an architecture?

ISA (Instruction Set Architecture) could be a definition, but

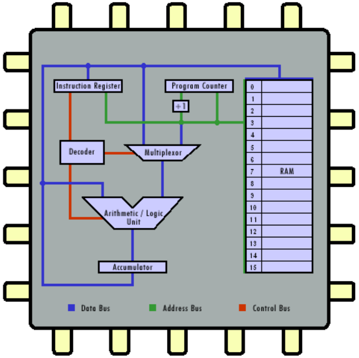
- Two different microprocessors ana have the same ISA or an ISA subset of the other (e.g. Core i5 vs. Core i3 vs. Pentium III)
- One microprocesseur can handle multiple ISA (e.g. MIPS big or little endian, ARM Thumb or not)
- What is in the binary depend on the compiler, which can select a subset of the ISA
- My definition : an architecture is what is recognised by `cpu_rec.py`!

The 72 architectures recognised by `cpu_rec.py` (default corpus)

- 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, and 6502 compiled with `cc65`

What means "recognise the architecture in a binary"?

3. Why does `cpu_rec.py` use statistical properties?



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

- The composent in the CPU that read the instructions is named the decoder
- The decoder always rends the instructions byte per byte, both for fixed-length and variable length instruction sets
- Therefore, the probability distribution of one byte depends on the value of the previous one

Core principle of `cpu_rec.py`

This remark is enough to be able to recognise an architecture

Learning and corpus

Learning technique

- Start from a corpus with labels (examples of each architecture)
- Learn from this corpus
- Classify binaries of unknown architecture

Available corpus

- For usual architectures (x86, ARM, SPARC, MIPS, Alpha, IA64, PPC, PA-RISC, ...) examples are easy to find
- For rare or exotic architectures (PIC16, Mico32, RISC-V, Cray, RL78, M-CORE, CLIPPER, ...) there are few available examples
- For the moment, 72 architectures are included in the corpus

Machine learning and binaries

State of the art

- To experiment with machine learning, the `scikit-learn` library is often recommended
- A binary is a sequence of bytes, therefore it can be seen as a text where bytes are the alphabet, and we can apply standard techniques designed for text documents

It does not work well

- Because the available corpus is too small (only 100 Ko for some architectures) and/or too heterogenous
- Because I am not a specialist of machine learning
- But *Multinomial Naive Bayes* on (2,3)-grams gives acceptable results

Markov chains and Kullback-Leibler distance

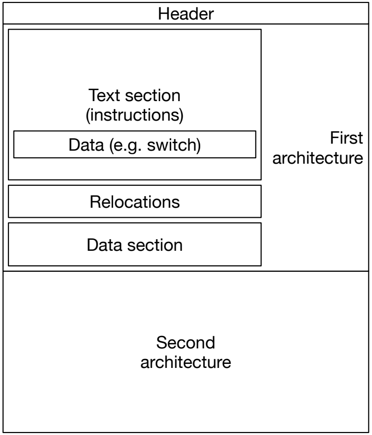
Markov chains

- Conditional probabilité of the value of a byte, knowing the previous one
- It is équivalent to the probabilité distribution of bigrams
- We can model each architecture by its distribution of bigrams and it distribution of trigrams

Distance between Markov chains / between distributions of n-grams

- It is named Cross entropy or Kullback-Leibler divergence
- It results in the same classification as *Multinomial Naive Bayes* but we have a measure of confidence
- `scikit-learn` does not know Cross entropy, there `cpu_rec.py` does not use `scikit-learn`

Where to look?



We are not interested by everything:

- The binary does not contain only instructions (there can be many sections, with data and instructions interleaved...)
- It can contain multiple architectures

Solution: sliding window

Not too big, not too small: 0x1000 by default, but smaller for small binaries

We may also have a window of variable size, depending on how close we are from known distributions... but the tool would be much slower

Command-line software

Software in python with no dependencies

- Available at `https://github.com/airbus-seclab/cpu_rec/`
- Together with a corpus including 72 architectures
- Compatible with python 3 and python 2 (at least 2.4)

Arguments and result

- The arguments are the files to analyse, and one or more `-v` for verbosity
- The files can be compressed, or firmware in HEX format
- Output: what architecture is recognised (whole file and text section) plus result of the sliding window analysis (adress and size of the zone containing instructions)

```
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 for binwalk

Integration in binwalk

- Installation in `~/.config/binwalk/modules/`
- Only for recent binwalk, works with python 2 or python 3
- Sliding windows analysis, with heuristics to eliminate outliers

Performances

- Creation of corpus signatures for 72 architectures : 25s et 1Go de RAM
- 60s per Mo of file analysed

```
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)
```

Sometimes there are small errors (e.g. IA-64 above, in zone which is the address table of DirExport)

A few examples of results obtained with `cpu_rec.py`

These examples have been chosen to highlight corner cases and limitations of the tool

- File with multiple architectures and imprecise analysis
- File with multiple architectures and too few instructions
- Known exotic architecture
- Unknown exotic architecture

File with multiple architectures and imprecise analysis

Mach-O FAT file

- `OSXII`, which is a MacOSX executable with two architectures: ppc and i386

Actual file content:

```
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)
```

There is a small zone in the middle of the text section for i386 which is not recognised. In practice, a reverse-engineer won't care.

File with multiple architectures and too few instructions

Mach-O FAT file

- SweetHome3D, with three architectures: ppc, i386, x86_64

Actual file content:

```
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)
```

Only PowerPC is detected, because the text sections for other architectures are too small. If we extract one Mach-O thin from this file, then its architecture is detected, because the size of the sliding windows is adapted.

```
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)
```

Known exotic architecture

Clipper architecture

- `https://en.wikipedia.org/wiki/Clipper_architecture`
- RISC 32-bit, used by Intergraph between 1986 and 1990

- The corpus is using C-Kermit: `cku196.clix-3.1` copied from `ftp://kermit.columbia.edu/kermit/bin/`
- `cpu_rec.py` can recognise binaries from `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)
```

Some small zones are not well recognised. In practice, it is not a problem: we see that there are three main zones containing CLIPPER instructions.

`cku196.clix-3.1` has been chosen for the corpus because it is a variant of COFF. I have no information on the file format of `boot.1`.

Unknown exotic architecture

Industrial automaton

- Bus Controller Module X20BC0083 from B&R Automation
- Firmware available on `https://www.br-automation.com/fr-ch/telecharger/software/automation-studio/hw-upgrades/v26-hw-upgrade-x20bc0083/`
- Contains 4 files `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)
```

Nothing is recognised. But if one of those files is added to the corpus, then we recognise that the other have the same architecture.

Discussions with B&R Automation showed that it is a homemade FPGA.

Conclusion

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

It works!

- Very simple approach, no understanding of how works a CPU is needed to add it to the corpus
- Not precise, but that's expected for a statistical approach
- A reverser facing an unknown binary will know where to look

What could be added?

- Missing architectures (please help me!)
- Other tools, dedicated to some architectures, to get more precise and complete information; for example:
  - More precise detection of the start of the executable code
  - If the code is not PIE, find the loading address
  - For ARM, better find where is each variant (especially Thumb)