# Automatic Classification of Object Code Using Machine Learning

## Architecture and Endianess

John Clemens


University of Maryland
Baltimore County (UMBC)
Baltimore, Maryland
clemej1 at umbc.edu


Johns Hopkins University
Applied Physics Laboratory (JHUAPL)
Laurel, Maryland
john.clemens at jhuapl.edu

*DFRWS 2015*

# Motivation

Reverse engineering and malware analysis efforts are extremely labor intensive and require expert domain knowledge

Enterprise computing environments and networks are diverse

- Types of systems: Laptops, Phones, Routers, IoT ...
- Within each individual system

Reverse engineering efforts are often "black box" tasks where nothing is known about the underlying system

Analysts are looking for tools and techniques to jumpstart this analysis

# Motivation

Machine Learning using byte histograms has already proven useful to classify general file types / file fragments (McDaniel and Heydari, 2003, among many others).

**Can we do better, and start to categorize within these general types?**

Start with compiled object code

- Very important for both malware analysis and reverse engineering
- Enterprise diversity means that the analyst will likely encounter multiple types of object code
- Accurate disassembly is *crucial*

Classification targets: *Architecture* and *Endianess*

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller
- SSD Controller

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller
- SSD Controller
- ....

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller
- SSD Controller
- ....

Example: My Motorola G (2015)

- Qualcomm ARM Cortex-A53

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller
- SSD Controller
- ....

Example: My Motorola G (2015)

- Qualcomm ARM Cortex-A53
- Adreno 405

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller
- SSD Controller
- ....

Example: My Motorola G (2015)

- Qualcomm ARM Cortex-A53
- Adreno 405
- Cell Baseband processor

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller
- SSD Controller
- ....

Example: My Motorola G (2015)

- Qualcomm ARM Cortex-A53
- Adreno 405
- Cell Baseband processor
- Battery controller

Image credit: Wikipedia

# Motivation

Computers are more than just the CPU

Example: My Thinkpad T400 (2008)

- Intel Core 2 Duo P8400
- Intel GMA4500
- ATI Radeon HD 3400
- Broadcom Wifi
- Intel NIC
- ACPI Embeded Controller
- SSD Controller
- ....

Example: My Motorola G (2015)

- Qualcomm ARM Cortex-A53
- Adreno 405
- Cell Baseband processor
- Battery controller
- ....

Image credit: Wikipedia

# Motivation

Malware authors are using this diversity to hide their software where existing countermeasures can't reach

> *"..but the most interesting finding is the malwares ability to reprogram the victims hard drives, making their implants invisible and almost indestructible..." - Kaspersky, Feb 17 2015*

> *"..The malware they created, called BadUSB, can be installed on a USB device to completely take over a PC, invisibly alter files installed from the memory stick, or even redirect the users internet traffic..." – Wired, July 2014*

> *"..Developers have published two pieces of malware [Jellyfish rootkit and Demon keylogger] that take the highly unusual step of completely running on an infected computer's graphics card, rather than its CPU..." – Ars Technica, May 7, 2015*

# Existing Methods

Existing methods either rely on file metadata to determine architecture, look for existing signatures within the sample, or just try to disassemble it and leave it up to the analyst to determine if the output is valid. Binwalk uses the last two methods.

- File metadata can be either missing or misleading
  - Unadorned firmware blobs
  - Obfuscated/packed malware
  - Incomplete file fragments or packet traces
  - Embedded code in native object files
- Signature detection requires prior knowledge of the architecture and can lead to false positives
- Disassembly methods can be misleading and require tools that support the architecture

# Proposed Method

## Hypothesis

Machine learning techniques can be applied to object code directly (without including metadata) to automatically classify the object code's target architecture and endianess

# Dataset

## Characteristics

- 16785 unique ELF files (exes/libs)
- 20 architectures
- Sources include Debian package repositories, Arduino development kits, and CUDA libraries

## Problems

- Only one compiler (GCC)
- CUDA sample size
- Need more embedded/micro-controller samples
- Heavy bias towards RISC

| Architecture | # Samples | Wordsize | Endianess |
|---|---|---|---|
| alpha | 1,383 | 64-bit | Big |
| hppa | 625 | 32-bit | Big |
| m68k | 1,296 | 32-bit | Big |
| arm64 | 1,134 | 64-bit | Little |
| ppc64 | 823 | 64-bit | Big |
| sh4 | 822 | 32-bit | Little |
| sparc64 | 752 | 64-bit | Big |
| amd64 | 965 | 64-bit | Little |
| armel | 960 | 32-bit | Little |
| armhf | 960 | 32-bit | Little |
| i386 | 967 | 32-bit | Little |
| ia64 | 650 | 64-bit | Little |
| mips | 960 | 32-bit | Big |
| mipsel | 960 | 32-bit | Little |
| powerpc | 992 | 32-bit | Big |
| s390 | 649 | 32-bit | Big |
| s390x | 653 | 64-bit | Big |
| sparc | 648 | 32-bit | Big |
| cuda | 17 | 32-bit | Little |
| avr | 596 | 8-bit | Little |
| Total | 16,785 | | |

# Intuition

```
64483304 25280000 007405e8 80f4ffff
4881c410 0100005b c3833d1c ca2d0000
7442833d cf4e2d00 297e3948 6315eecd
2d00488b 3dd7cd2d 00488b35 e8cd2d00
4839fa72 0931c080 3c162d75 17c38a04
16488d4a 013c0975 054889ca ebe23c20
74f7ebe1 8b0586cd 2d003d04 0100007f
```

```
0807025a 0c601200 d0881ff8 0ca01098
93182000 34180040 e84f1f20 08060259
87822050 0803025a e84f1b10 0c681200
34181ec6 379a0002 2330c000 e84f1e78
37390a00 08030259 e80000d0 081c025a
0c641200 e81f1f3d 34633fff 34840002
80838262 0c801005 0807025a 0c801200
```

Can you tell which architectures the above two samples target?

Instructions contain two parts:

- **opcode**: Unique to the architecture
- **operands**: Data to be operated on

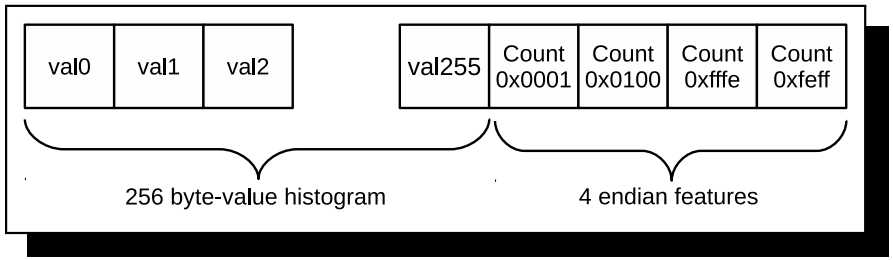
## Opcode Density

$$Opcode\_Density = \frac{length\_of\_opcode}{average\_instruction\_length}$$

# Feature Vector

To classify Architecture:

- Extract code sections of each ELF object
- Generate a normalized byte histogram to become a 256-entry attribute vector




To classify Endianess:

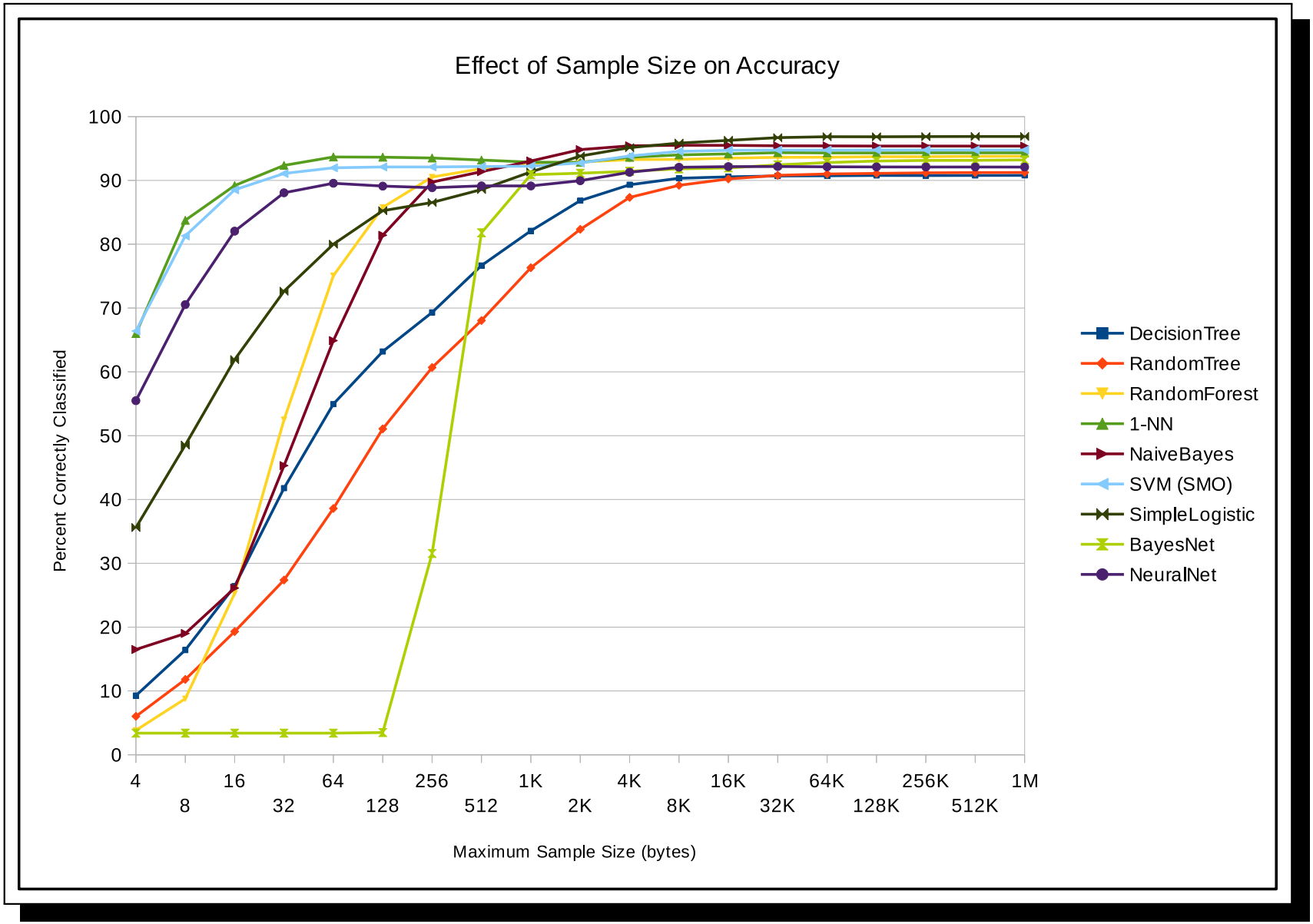
- Above is insufficient; endianess requires adjacency information lost in a byte histogram
- Look for distinctive patterns: encoding of '1' and '-1'
- Count the number of occurrences of '0x0001', '0x0100', '0xffe', '0xfeff'
- Use these four counts as additional endianess attributes
- Experiments with 2-byte bi-grams proved resource intensive and performed poorly compared to this method

# Results: Overall Performance

| Trained Model | Multi-class Strategy | WEKA Name | Histogram | Hist + Endian |
|---|---|---|---|---|
| 1-NN | Inherent | IBk | 89.3238% | 92.7256% |
| 3-NN | Inherent | IBk | 89.8660% | 94.9002% |
| Decision Tree | Inherent | J48 | 93.2976% | 98.0697% |
| Random Tree | Inherent | RandomTree | 87.8046% | 92.9461% |
| Random Forest | Inherent | RandomForest | 90.4617% | 96.4373% |
| Naive Bayes | Inherent | NaiveBayes | 92.5827% | 95.8951% |
| BayesNet | Inherent | BayesNet | 89.5144% | 92.2252% |
| SVM (SMO) | 1-vs-1 | SMO | 92.7256% | 98.3497% |
| Logistic Regression | Inherent | SimpleLogistic | 93.0831% | 97.9386% |
| Neural Net | Inherent | MultilayerPerceptron | 94.0244% | 97.9565% |

10-fold stratified cross validation accuracy for various models using the byte-value histogram alone, and the byte-value histogram augmented with heuristic-based endianess attributes.

# Results: Effect of Sample Size



# Summary

Contributions of this work include:

- A dataset of 16K samples of object code from 20 architectures
- Machine learning techniques using byte histograms can automatically classify object code's target architecture
- Endianess determined with high accuracy using with the addition of four extra heuristics in the feature vector
- High accuracy with a small sample size
- A method of automatic object code classification that does not require signatures, toolchain support, correct metadata, or any previous knowledge

# Future Directions / Questions?

Future Work

- More complete dataset (send me your esoteric samples!)
  - ★ Microcontroller / Embedded samples
  - ★ Virtual byte code (e.g. Python, Java, Dalvik/ART, CIL)
- Word-size detection
- Compiler attribution
- Detect embedded code (Thumb vs. ARM)
- Function / basic block boundary detection
- Plugin (IDA and friends)

I will post original dataset and updated dataset for download (location TBD)

Questions?