Malware Classification

Using Static Disassembly & Machine Learning

Zhenshuo Chen

School of Computing, Dublin City University

BACKGROUND

Traditional manual malware analysis is difficult to cope with the rapidly increasing number of malicious programs and cyber attacks.

Symantec's Global Intelligence Network blocked **142 million** threats daily in 2019.

China's National Internet Emergency Center recorded **18 million** malicious samples in the first half of 2020.

Objective

The primary goal is to develop an automatic classifier that can classify malware into different families with high accuracy and efficiency.

Main Work

1. Designed macroscopic, small-scale features suitable for static disassembly.
2. Trained models using automatic machine learning.
3. Analyzed the limitations of static disassembly.
4. Designed a practical workflow and developed a malware classifier used with IDA Pro.

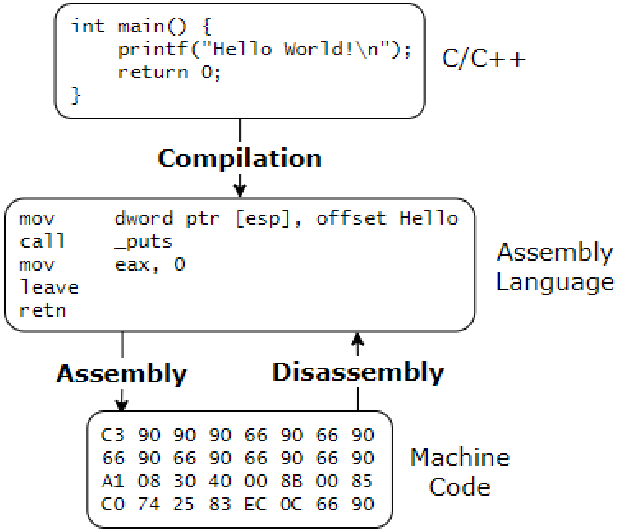
1

Basic Concepts

Machine Code & Assembly Languages

For malware analysts, there is no high-level source code for review but only executable files.

Because of the correspondence between assembly languages and machine code, executable files can be translated into assembly instructions.



Code Obfuscation & Encryption

Anti-reverse-engineering techniques can be used to obstruct disassembly.

Not only malware authors, but also software companies use them to protect commercial software from being cracked or pirated.

```
00401439  ^ EB E4           jmp main.40141F
0040143B    8BF7            mov esi,edi
0040143D  ^ EB CA           jmp main.401409
0040143F    33C0            xor eax,eax
00401441  ^ EB EC           jmp main.40142F
00401443    66:2B03         sub ax,word ptr ds:[ebx]
00401446  ^ EB D5           jmp main.40141D
00401448    8D1D 00304000   lea ebx,dword ptr ds:[403000]
0040144E  v EB 00           jmp main.401450
00401450  ^ EB F1           jmp main.401443
```

Obfuscation: disrupted execution flow

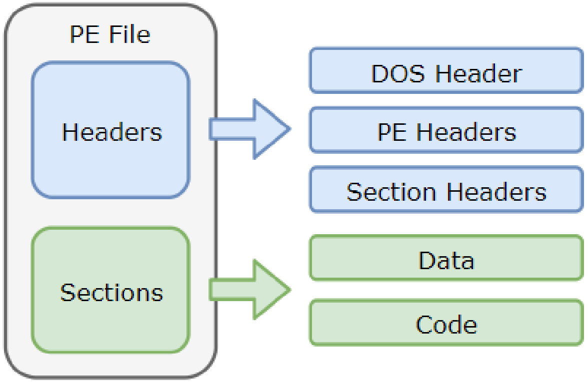
```
dd 0AE2C543Fh, 2F10C7A2h, 8059DFA5h, 1108E115h, 0CE3D9038h
dd 2007DFBBh, 77DA9179h, 904F0AD2h, 0DBD3E36Dh, 0D0BCF948h
dd 0DB7695C0h, 0B9B72C77h, 849CAA3Ah, 5AC7847Dh, 0BAA2BFBFh
dd 9D6EE0FAh, 0E275F58Ch, 0E172C2F0h, 0D3BCB558h, 27A93062h
dd 319BB966h, 320907F5h, 0C42A8F80h, 6A86F551h, 37EC06DAh
dd 4EB946Bh, 162FBB98h, 73A6A2DEh, 0CFE5D957h, 6DC5B790h
```

Encryption: failed disassembly

Windows PE Format

The Portable Executable (PE) format is a file format for executable files on Windows system, consisting of several headers and sections.

PE Headers can be regarded as metadata encapsulating system-related information, such as export and import APIs, resources (icons, images and audio, etc.), and the distribution of data and code.



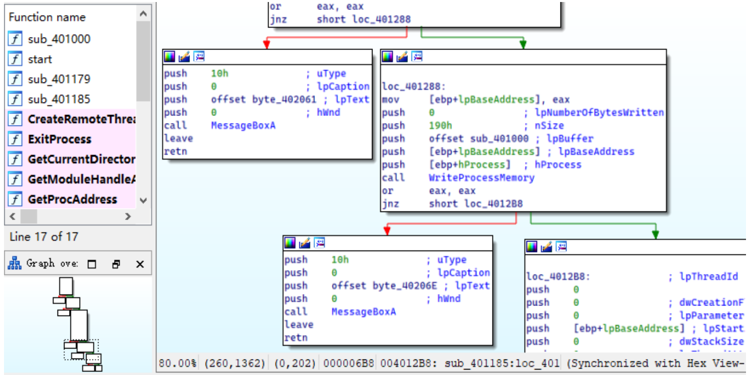
2

Traditional Analysis

Static Disassembly Analysis

Static analysis must disassemble machine code firstly via professional tool. Analysts explore control flows and assembly instructions to understand malicious behaviors.

In theory, it provides complete code coverage, but is time-consuming and vulnerable to code obfuscation or encryption.

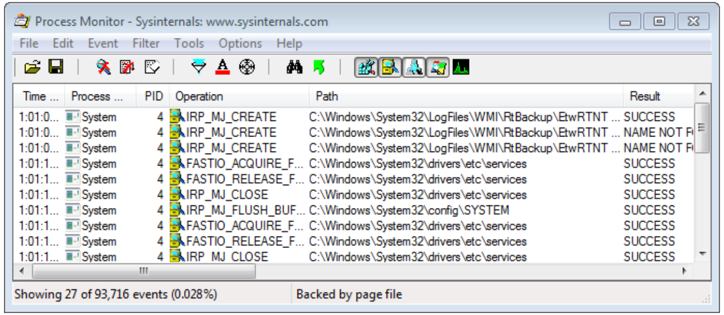


IDA Pro: a powerful disassembler

Dynamic Behavior Analysis

Dynamic analysis focuses on system events. It is not susceptible to encryption since encrypted files must decrypt themselves before execution.

However, it is costly and requires virtual environments to run malware. In addition, malicious behaviors may not be recorded because the environment does not meet the execution conditions. Malware can also detect virtual environments and hide itself.



3

Dataset

2015 Microsoft Malware Classification Challenge

ID	Name	# Samples	Type
1	Ramnit	1541	Worm
2	Lollipop	2478	Adware
3	Kelihos_ver3	2942	Backdoor
4	Vundo	475	Trojan
5	Simda	42	Backdoor
6	Tracur	751	Trojan Downloader
7	Kelihos_ver1	398	Backdoor
8	Obfuscator.ACY	1228	Obfuscated malware
9	Gatak	1013	Backdoor

2015 Microsoft Malware Classification Challenge

Each sample has two files of different forms:

- machine code
- disassembly script generated by IDA Pro

```
100010A0 D6 89 35 04 90 00 10 5E 5F 5B C9 C3 56 68 80 00
100010B0 00 00 FF 15 9C 80 00 10 8B F0 56 FF 15 98 80 00
100010C0 10 85 F6 59 59 A3 5C E4 00 10 A3 58 E4 00 10 75
100010D0 05 33 C0 40 5E C3 83 26 00 E8 D8 03 00 00 68 DA
100010E0 14 00 10 E8 BC 03 00 00 C7 04 24 F3 13 00 10 E8
100010F0 B0 03 00 00 59 33 C0 5E C3 8B 44 24 08 55 33 ED
```

```
.text:100010AC 56                    push    esi
.text:100010AD 68 80 00 00 00        push    80h
.text:100010B2 FF 15 9C 80 00 10     call    ds:_malloc_crt
.text:100010B8 8B F0                 mov     esi, eax
.text:100010BA 56                    push    esi
.text:100010BB FF 15 98 80 00 10     call    ds:_encode_pointer
.text:100010C1 85 F6                 test    esi, esi
```

4

Feature Extraction

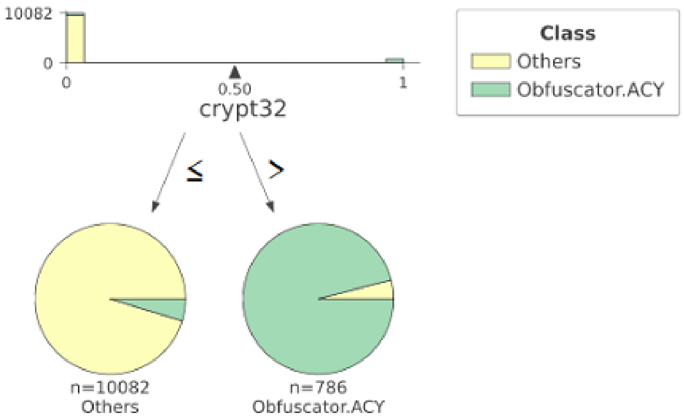
New Proposed Features

- Import Library
- PE Section Size
- PE Section Permission
- Content Complexity

Import Library

Each malware family has distinct functions. They must import system or third-party libraries to achieve.

We used the One-Hot Encoding to indicate whether a library is imported by malware.



PE Section Size

PE files consist of several sections. Each section stores different types of bytes.

The number of sections, their uses and attributes are defined by software development tools and programmers based on functionality.

Section	Description
`text`	Executable code
`data`	Normal data
`idata`	Import libraries
`rdata`	Read-only data

PE Section Permission

PE sections have access permissions, which are combinations of readable, writable and executable. We calculated the total size of readable data, writable data and executable code separately for each malware sample.

This can be regarded as a summary of PE section sizes and provides a more macroscopic view.

Content Complexity

Content complexity has six fixed dimensions: the original sizes, compressed sizes and compression ratios of disassembly and machine code files.

We compressed samples and recorded size changes. They can roughly reflect function complexity, code encryption and obfuscation.

```
mov     eax, esi
ror     eax, 8
mov     ecx, esi
ror     ecx, 5
```

Disassembly snippet with the largest compression ratio

```
dd   66E5C7CBh, 0BABBBB02h, 8525F0EEh
dd   8ED64AF8h, 0F93780DFh, 80B1924Ah
dd   3C99EBD7h, 0AF12FC33h, 837520D0h
```

Disassembly snippet with the smallest compression ratio

Existing Old Features

- File Size
- API Sequence
- Opcode Sequence

File Size


Distribution of File Sizes
1e7
Machine Code Size
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
1
2
3
4
5
6
7
8
1e7
Disassembly Size
Class
1
2
3
4
5
6
7
8
9

API Sequence

The API sequence is almost the most commonly used feature. It directly uses malicious or suspicious API sequences to classify malware.

API	Function
`VirtualAllocEx`	Allocate memory in the target
`WriteProcessMemory`	Write a library path into the target
`CreateRemoteThread`	Make the target load the library

The API sequence of dynamic-link library injection, used to inject malicious code into a running program

Opcode Sequence

The opcode sequence is also commonly used. It focuses on disassembly instructions.

Opcodes are defined by CPU architectures, not by systems as in the case of APIs. They are compatible with different systems built on the same architecture.

```
     jmp     next
     db      10
next:
     mov     eax, 0
```

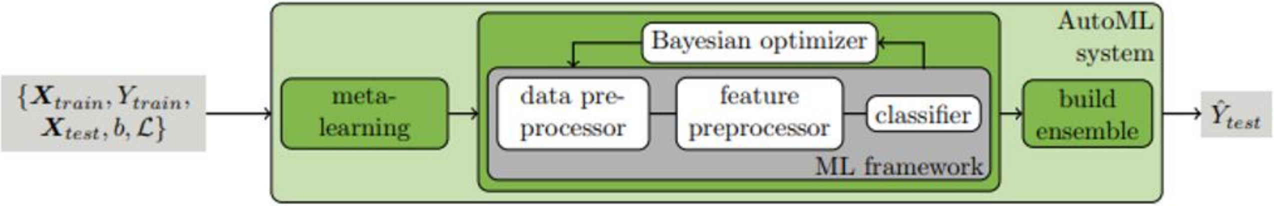
Its opcode sequence is `jmp`, `db`, `mov`

5

Training Models

Automatic Machine Learning

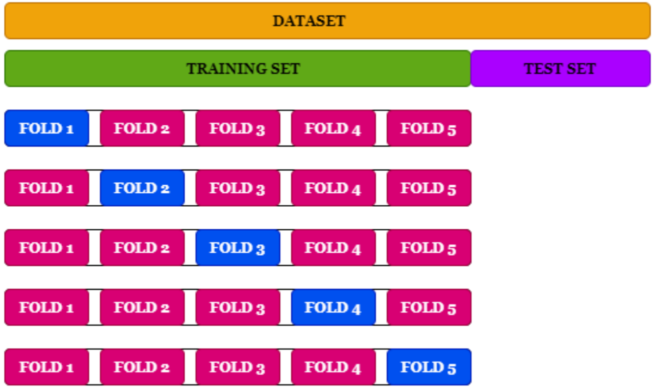
We used automatic machine learning library `auto-sklearn` to search for the best parameters, relying on Bayesian optimization, meta-learning and ensemble construction.



Cross-Validation

The 80% of the dataset was used as a training set and `auto-sklearn` evaluated models on it using 5-fold cross-validation.

After optimal parameters were determined, we used the remaining 20% as a test set to calculate classification accuracy.



Accuracy Results

Feature	Dimension	Best Accuracy	Best Model
All Features	1812921 → 10343	0.9948	Random Forest
Section Size + Section Permission + Content Complexity	861 → 40	0.9940	Random Forest
Section Size + Section Permission + Content Complexity + Import Library	1431 → 340	0.9922	Random Forest
Opcode 4-gram	1408515 → 5000	0.9908	Random Forest
File Size + API 4-gram + Opcode 4-gram	1811490 → 10003	0.9899	Random Forest
Content Complexity	6	0.9811	Random Forest
Section Size	846 → 25	0.9775	Random Forest
Section Permission	9	0.9701	Random Forest
Import Library	570 → 300	0.9393	Random Forest
File Size	3	0.9352	Random Forest
API 4-gram	402972 → 5000	0.5796	Random Forest

Accuracy Results

Feature(s)	Dimension	Accuracy
Section Size, Section Permission, Content Complexity	40	0.9940
Opcode 4-gram	5000	0.9908
Content Complexity	6	0.9811
Section Size	25	0.9775
Section Permission	9	0.9701
Import Library	300	0.9393
File Size	3	0.9352
API 4-gram	5000	0.5796

6

Practical Applications

IDA Pro Classifier Plug-in

With the fitted machine learning model, we developed an IDA Pro classifier plug-in.

When an analyst opens a malware sample with IDA Pro, the plug-in can produce required files in the similar format as the files in the dataset and calculate the probability that malware belongs to each family.

```
---------------------------------------------------
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28
IDAPython v7.4.0 final (serial 0) (c) The
---------------------------------------------------
0.53 -> Ramnit
0.24 -> Lollipop
0.17 -> Obfuscator.ACY
0.05 -> Gatak
0.01 -> Simda
0.01 -> Vundo
0.00 -> Tracur
0.00 -> Kelihos_ver1
0.00 -> Kelihos_ver3
```

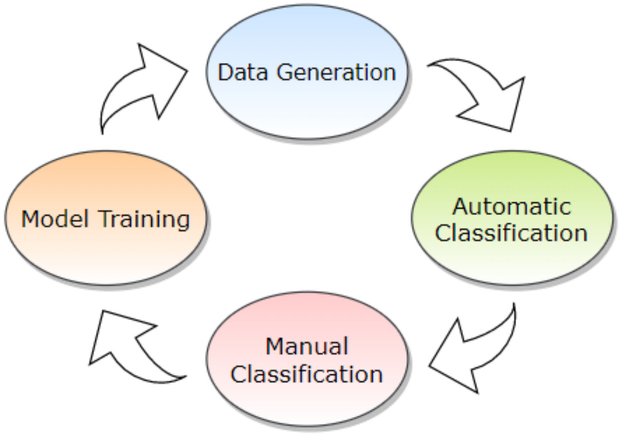
Concept Drift

In many other machine learning applications like digit classification, the mapping learned from historical data will be valid for new data in the future.

However, for malware, due to function updates, code obfuscation and bug fixes, the similarity between previous and future versions will degrade slowly over time, decaying the detection accuracy.

Automatic Workflow

To solve the concept drift challenge, we designed an automatic malware classification workflow to apply and enhance our classifier in practice with IDA Pro's Python development kit.



Automatic Workflow

1. **Data Generation**

Producing training data from executable malware.

2. **Automatic Classification**

Getting classification results using the model.

3. **Manual Classification**

Conducting in-depth analysis manually and determining accurate results.

4. **Model Training**

Retraining the model with new samples and their class label.

7

Limitations of Static Disassembly

Lazy Loading

When extracting import libraries, only the libraries in the Import Table can be extracted. These libraries will be automatically loaded when malware starts.

To make malicious behavior more hidden, developers can use lazy loading to load a library just before it is about to be used. Here are the top libraries based on Information Gain. They are ubiquitous and have no special significance for malware classification.

Library	Function
`MSASN1`	Abstract Syntax Notation One Runtime
`MSVCRT`	Microsoft Visual C++ Runtime
`UXTheme`	Microsoft Windows Controls

Name Mangling

Name mangling allows different programming entities to be named with the same identifier, like C++ overloading. Internally, compilers need different identifiers to distinguish them.

It adds noise to the API sequence extraction. For the same or similar functions, we may extract more than one name.

Compiler	`void fun(int)`	`void fun(double)`
GCC 8	`_Z3funi`	`_Z3fund`
Visual C++ 2022	`?fun@@YAXH@Z`	`?fun@@YAXN@Z`

Jump Thunk

Many compilers generate a jump thunk, a small code snippet, for each external API, then convert all calls to an API into calls to its jump thunk. It can provide an interface proxy.

Jump thunks make API sequences inaccurate. If we use linear scanning to extract external API calls from this code, we will get the sequence `WriteFile`, `ReadFile`. But the true sequence is `ReadFile`, `WriteFile`, `ReadFile`.

```
j_write_file    proc
    jmp       WriteFile
j_write_file    endp

j_read_file     proc
    jmp       ReadFile
j_read_file     endp

call      j_read_file
call      j_write_file
call      j_read_file
```

THANK YOU

https://www.linkedin.com/in/zhenshuo-chen