

Malicious Code Analysis

Fangtian Zhong
CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu





Overview

01

Malware Binary Analysis

02

Static Malware Analysis I



Part One

01

2025-10-19

Malware Binary Analysis

An isometric illustration of a digital workspace. It features several stylized figures: a person in a suit standing near a large screen displaying a grid, a person in a white shirt walking, and a man and woman in business attire standing together. The background is filled with floating rectangular panels containing various icons like stars, a calendar, a padlock, and a lightbulb, all in a light blue and teal color palette.



Malware Binary Analysis

.exe

```
01001010100101010
10101010011010101
01001010100101010
10101010011010101
01001010100101010
10101010011010101
01001010100101010
10101010011010101
10101010011010101
01001010100101010
10101010011010101
```



- What does the malware do
- How does it do it
- identify triggers
- What is the purpose of the malware
- Is this an instance of a known threat or a new malware
- who is the author
- ...

- 🛡 Typically a stripped binary with no debugging information.
- 🛡 In the case of malicious code, it is often obfuscated and packed.
- 🛡 Often has embedded suicide logic and anti-analysis logic.

Challenges:

- lack of automation
- time-critical analysis
- labor intensive
- requires a human in the loop



Malware Analysis Systems

- **Malware analysis** involves dissecting malware to understand how it works, and determine its functionality, origin, and potential impact
 - Malware analysis is essential for any business and infrastructure that responds to cybersecurity incidents
- Malware analysis systems can be classified into two broad categories
 - **Static analysis systems** (pre-execution analysis)
 - Process malware without running it, and extract features to be used for malware detection and classification
 - **Dynamic analysis systems** (post-execution analysis)
 - It involves running the malware either in a physical or virtual environment, and searching for indicators of malicious activities
- Some references also add a class of **hybrid analysis systems**, that combine static and dynamic analysis

Static Analysis VS Dynamic Analysis

Static Analysis

- ❑ The process of analyzing a malware sample without actually executing it.
- ❑ **Tools:** VirusTotal, strings, a disassembler like angr, IDA Pro, etc.

Dynamic Analysis

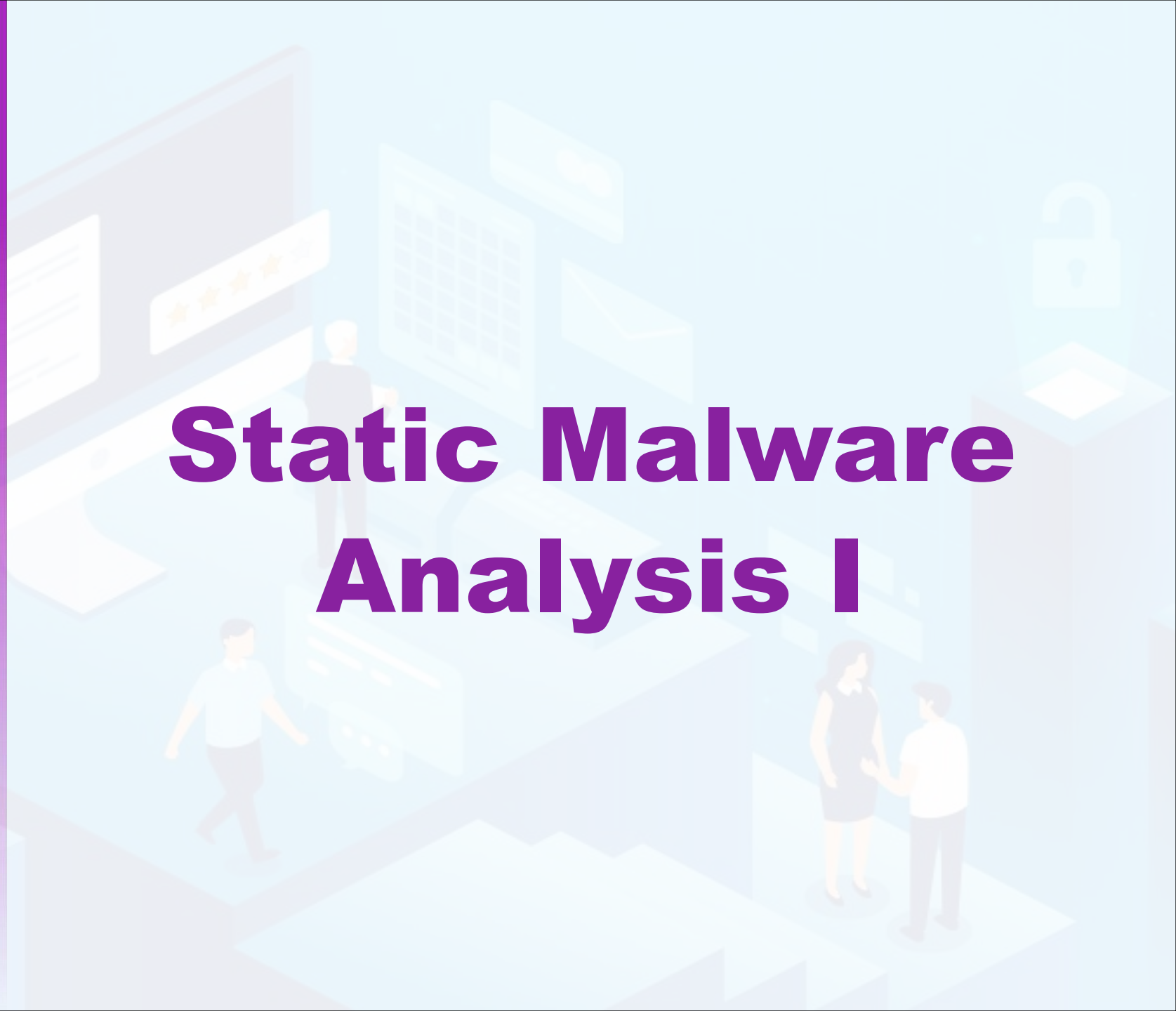
- ❑ The process of analyzing malware by executing it in a controlled environment to observe its behavior and identify its functionality. Use a virtual machine and take snapshots.
- ❑ **Tools:** angr, RegShot, Process Monitor, Process Hacker, CaptureBAT.
- ❑ **RAM Analysis:** Mandiant's Redline, Volatility.



Part Two

02

Static Malware Analysis I





Basic Static Analysis



Basic static analysis

- ❑ View malware without looking at instructions.
- ❑ Tools: VirusTotal, strings.
- ❑ Quick and easy but fails for advanced malware and can miss important behavior.





Advanced Static Analysis



Advanced static analysis

- ❑ Reverse-engineering with a disassembler.
- ❑ Complex, requires understanding of assembly code.





Static Analysis Systems

- **Static analysis** provides information about the functionality of the file, and it produces a set of signature features (without executing the file)
 - The extracted information is used to predict whether the file is malicious software
 - The disadvantage of static analysis is that the “true features” of the code may be missed
- **Static analysis can include:**
 - Analyzing PE header and sections
 - PE header provides information about linked libraries and imported/exported functions, as well as contains metadata about the executable
 - Strings of characters can contain references to modified files or accessed file paths by the executable (e.g., URLs, domain names, IP addresses, names of loaded DLLs, registry keys, etc.)
 - Search for packed/encrypted code that is used by malware developers to make their manipulated files more difficult to analyze
 - Disassembling the program – translating machine code into assembly language code
 - Load the executable into a disassembler to translate it into assembly language, and obtain a better understanding of what the program does



Static Features for Malware Classification

- **Static features**

- In *Windows* systems, static features are extracted from either the PE file header and sections, or assembly language source file (obtained after disassembling the file)

- **Strings** – sequence of characters, related to URLs, IP addresses, accessed file paths, registry keys, or names of modified files by the executable

- [Ye et al. \(2008\)](#) used extracted strings from PE files as input features to an SVM ensemble with bagging model for malware detection

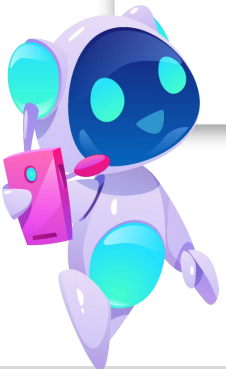
- **Byte n -grams** – sequence of n bytes in PE header or the assembly language code

- An *n -gram* is a sequence of n adjacent items in sequential data
- A large number of sequences of n bytes (n ranging from 1 to 8) are used as input features for ML model training
- Different ML models (Decision Trees, Random Forests, Deep Belief Nets) have been implemented using byte n -grams , e.g., by [Jain and Meena \(2011\)](#), [Yuxin et al. \(2019\)](#)
- Challenges include the large number of n -grams for each file (which often requires reducing the dimensionality of the feature vectors)



String

- ★ Any sequence of printable characters is a **string**
- ★ Strings are terminated by a **null** (0x00)
- ★ ASCII characters are 8 bits long
Now called ANSI
- ★ Unicode characters are 16 bits long
Microsoft calls them "wide characters"





String

ASCII

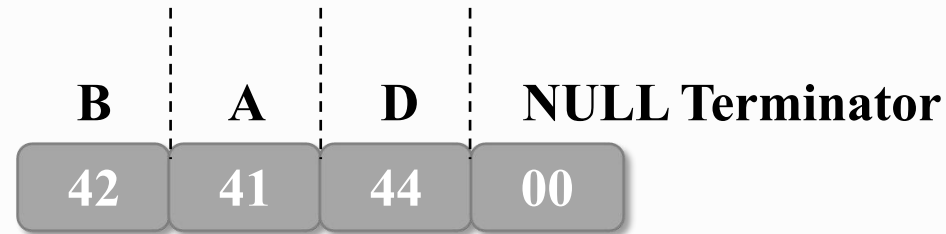


Figure 1. ASCII representation of the string BAD

Unicode

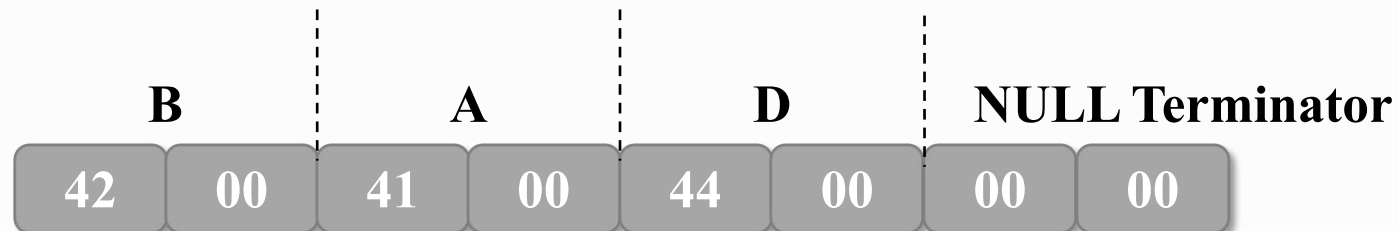



Figure 2. Unicode representation of the string BAD

String

 Native in Linux, also available for Windows.

 Finds all strings in a file 3 or more characters long.





Strings Command

💡 GetLayout and SetLayout are Windows functions.

💡 GDI32.DLL is a Dynamic Link Library

```
C:>strings bp6.ex_  
VP3  
VW3  
t$@  
D$4  
99.124.22.1 4  
e-@  
GetLayout 1  
GDI32.DLL 3  
SetLayout 2  
M}C  
Mail system DLL is invalid.!Send Mail failed to  
send message. 5
```



Example

```
char *sneaky = "SOSNEAKY";
int authenticate(char *username, char *password)
{
    char stored_pw[9];
    stored_pw[8] = 0;
    int pwfile;

    // evil back d00r
    if (strcmp(password, sneaky) == 0) return 1;

    pwfile = open(username, O_RDONLY);
    read(pwfile, stored_pw, 8);

    if (strcmp(password, stored_pw) == 0) return 1;
    return 0;
}

int accepted()
{
    printf("Welcome to the admin console, trusted user!\n");
}

int rejected()
{
    printf("Go away!");
    exit(1);
}
```

```
int main(int argc, char **argv)
{
    char username[9];
    char password[9];
    int authed;

    username[8] = 0;
    password[8] = 0;

    printf("Username: \n");
    read(0, username, 8);
    read(0, &authed, 1);
    printf("Password: \n");
    read(0, password, 8);
    read(0, &authed, 1);

    authed = authenticate(username, password);
    if (authed) accepted();
    else rejected();
}
```




Static Features for Malware Classification

- **Opcode (mnemonic) n -grams** – n consecutive opcodes (i.e., operational code instructions) in the assembly language source code
 - Assembly instructions are composed of an operational code and operand
 - E.g., for the instruction sequence: “call sub_401BCD”, “add eax 1”, “mov ebx ebx”, the 3-gram opcode is: CALL-ADD-MOV
 - Malware samples from the same family often use the same opcodes
 - [Santos et al. \(2013\)](#) selected the top 1,000 features using 1 or 2-gram opcodes and trained an SVM malware classifier
- **API function call** – request to the OS for accessing system resources, such as networking, security, file management, etc.
 - Application Programming Interfaces (API) function calls are very discriminative features, as they can provide key information to reveal the behavior of malware
 - E.g., certain sequences of API function calls are often found in malware, but rarely in benign files
 - [Ahmadi et al. \(2016\)](#) used the frequency of 794 API function calls to develop an ML system for classifying malware into families

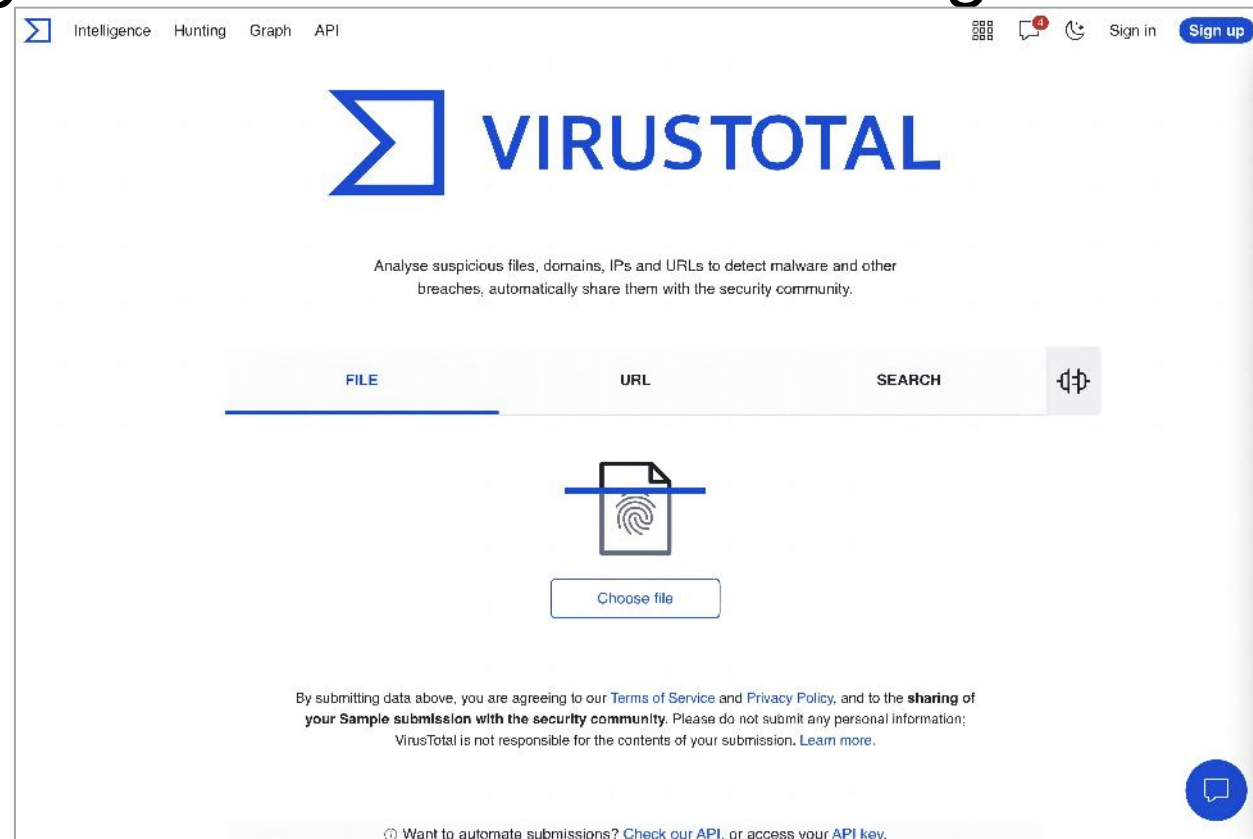


Generic Detection



VirusTotal is a free online service that allows users to scan files, URLs, and IP addresses for viruses, malware, and other security threats using more than 70 antivirus engines and other security tools.

<https://www.virustotal.com/gui/home/upload>



675af60cbe5382f35842d4f32b383d1eadbac55d5be1d735379ad69ab6373f77



Sign in

Sign up



Community Score



8 security vendors and no sandboxes flagged this file as malicious



675af60cbe5382f35842d4f32b383d1eadbac55d5be1d735379ad69ab6373f77

18.00 KB
Size

2023-03-04 23:45:57 UTC
7 days ago



Stardust 2.EXE

peexe assembly runtime-modules idle direct-cpu-clock-access 64bits

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.reverseshell

Threat categories trojan

Family labels reverseshell

Security vendors' analysis

Do you want to automate checks?

Antiy-AVL	Trojan[Backdoor]/Win64.ReverseShell	CrowdStrike Falcon	Win/malicious_confidence_70% (D)
Cynet	Malicious (score: 100)	Elastic	Malicious (high Confidence)
Fortinet	W64/ReverseShell.O!tr	McAfee	Artemis!979A6C706551
McAfee-GW-Edition	Artemis	Symantec	ML.Attribute.HighConfidence
Acronis (Static ML)	Undetected	AhnLab-V3	Undetected





Hashing

- ★ Static malware analysis by hashing involves calculating a cryptographic hash of a suspicious file to determine if it matches any known malware.
- ★ To perform static malware analysis by hashing, the following steps can be taken:
 - ❑ Obtain a suspicious file that needs to be analyzed.
 - ❑ Calculate the cryptographic hash of the file using a hashing algorithm such as MD5, SHA-1, or SHA-256.
 - ❑ Compare the calculated hash value against a database of known malicious hash values. This database could be a local repository of known malware hashes or an online database such as VirusTotal.
 - ❑ If the hash value matches a known malicious hash, the file is likely to be malware and should be treated accordingly. If the hash value does not match any known malicious hash, the file may still be suspicious and further analysis is needed.



Hashes

- 🏆 MD5, SHA-1 or SHA-256.
- 🏆 Condenses a file of any size down to a fixed-length fingerprint.
- 🏆 Uniquely identifies a file well in practice.
 - ❑ There are MD5 collisions but they are not common.
 - ❑ Collision: two different files with the same hash.



HashCalc

H HashCalc

Data Format: File Data: C:\Users\student\Desktop\p3.pcap ...

☐ HMAC Key Format: Text string Key:

☒ MD5 52583b5e2c99d19c046915181fd7b29b

☐ MD4

☒ SHA1 991d4e880832dd6aaebadb8040798a6b9f163194

☐ SHA256



Hash Uses



Label a malware file



Share the hash with other analysts to identify malware



Search the hash online to see if someone else has already identified the file





Static Features for Malware Classification

- **Function–call graph** – is a directed graph whose vertices represent the functions of a program, and the edges symbolize function calls
 - [Kinable et al. \(2011\)](#) developed an approach for clustering malware based on the structural similarities between function-call graphs
- **Control–flow graph** – is a directed graph in which the nodes represent basic blocks, and the edges represent control-flow paths
 - A basic block is a linear sequence of program instructions having an entry point (the first instruction executed) and an exit point (the last instruction executed)
 - The control-flow graph is a representation of all the paths that can be traversed during a program's execution
 - [Faruki et al. \(2012\)](#) used a Random Forest classifier for detecting malware using control-flow graphs of various API calls

THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2023.11.28