



## **Software Reverse Engineering**

Final Project Analysis

Name: Jawwad Shaik

Course: CSCE 652 - Software Reverse Engineering

Semester: Spring 2025

Professor: Dr. Martin Carlisle

# Introduction

- **Malware Name:** ZeusBankingVersion\_26Nov2013
- **File Name:** invoice\_2318362983713\_823931342io.pdf.exe
- **Source:** [theZoo GitHub Repository](#)
- **SHA-256 Hash:**  
4644B5FB10FB84C0D47BEC4B5A48D5E60165E8AE2130FCA5C055633AAAD73162
- **Architecture:** 32-bit (x86)
- **Environment:** Windows 7 VM (Isolated, Host-only Network)
- **Tools Used:** Ghidra, PEStudio, Strings.exe, x32dbg, INetSim, Wireshark, Process Monitor, RegShot

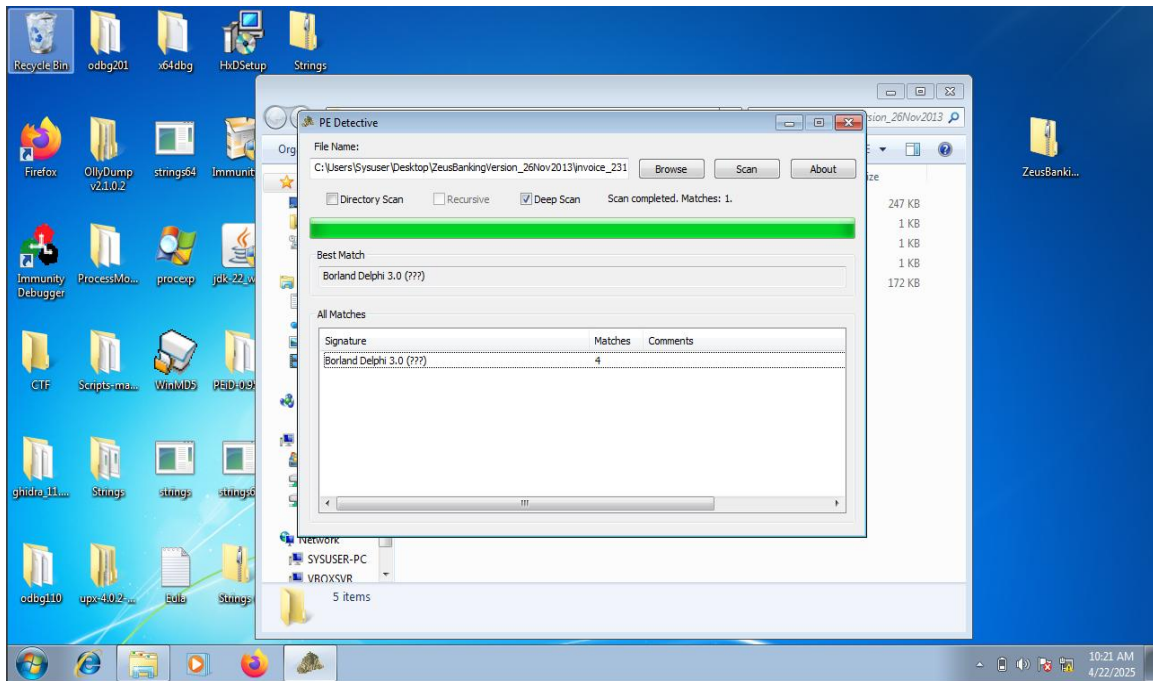
## Malware Unpacking

I have analyzed ZeusBankingVersion\_26Nov2013 malware sample to determine whether it was packed or obfuscated before beginning static and dynamic analysis. The inspection was carried out using PE Detective, a tool used for detecting known compiler signatures and potential packing techniques.

The analysis result indicated a match with Borland Delphi 3.0. This suggests that the malware was likely developed using the Borland Delphi programming environment. No indications of known packing methods such as UPX, ASPack, or PECompact were detected. The executable's structure appeared consistent with standard compiled applications, and no signs of compression or encryption were found.

Since the tool did not report any active packing or obfuscation, no unpacking was necessary. In conclusion, the malware was already unpacked binary and was ready for direct static analysis.

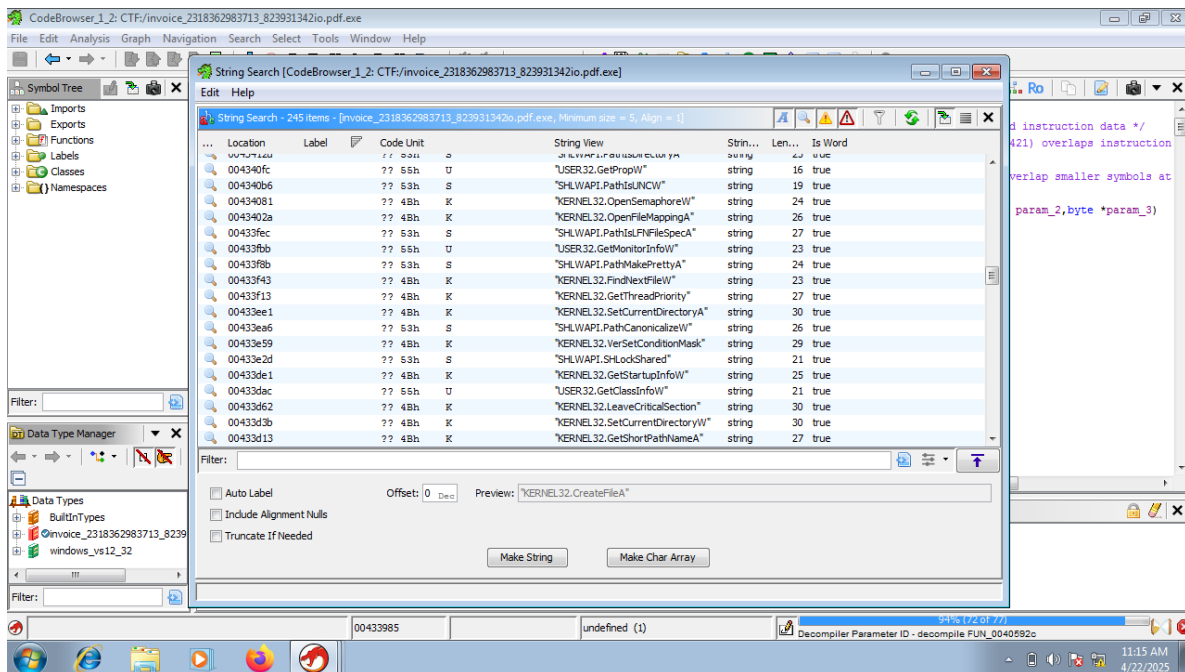
A screenshot of the PE Detective results, showing the Borland Delphi 3.0 signature match:



**Figure 1: PE Detective result showing Borland Delphi 3.0 signature match**

## Strings of Interest

During static analysis, I have extracted the strings embedded within the ZeusBankingVersion\_26Nov2013 sample using Ghidra's String Search feature. These Strings have provided me with important clues about malware's behavior, such as potential API calls, file paths, registry entries, hardcoded domains, or system manipulation commands.



**Figure 2: Extracted Windows API strings from Zeus malware (Ghidra Strings Search)**

A review of the extracted strings revealed multiple references to key Windows API functions, especially from KERNEL32.DLL, USER32.DLL, and SHLWAPI.DLL. These APIs are heavily used for process management, file system access, and system information retrieval, which are typical activities for malware aiming to interact deeply with the infected system.

Some notable strings observed include:

String Extracted	Description
KERNEL32.CreateFileA	Used for creating or opening files, often linked to file creation, reading sensitive files, or writing logs.
KERNEL32.OpenSemaphoreW	Interacts with semaphores, may be used to coordinate multiple malware threads or processes.
KERNEL32.OpenFileMappingA	Indicates the use of shared memory or file mapping, possibly for inter-process communication (IPC).
SHLWAPI.PathIsUNCW	Checks if a given path is a Universal Naming Convention (UNC) path, suggesting awareness of network shares.

USER32.GetPropW	Retrieves a data pointer associated with a window property; could be related to window monitoring or stealth techniques.
KERNEL32.SetThreadPriority	Sets the thread's priority, indicating an attempt to control execution timing and responsiveness.
KERNEL32.GetStartupInfoW	Retrieves information about the process startup, possibly to inherit specific attributes.
KERNEL32.LeaveCriticalSection	Part of multi-threading synchronization, suggesting threaded behavior within the malware.

The dominance of kernel32 and user32 related APIs implies that the malware heavily manipulates system-level resources and operates multiple threads.

## Imported Methods

To better understand the capabilities of the malware, I analyzed the imported functions listed in its Import Address Table (IAT) using Ghidra. The imports give clear clues about what the malware is designed to do especially how it interacts with the operating system, files, processes, and the user interface.

During my analysis, I found a variety of imported functions from standard Windows libraries such as USER32.DLL, KERNEL32.DLL, and ADVAPI32.DLL. These functions are consistent with malware behavior that includes file access, registry modification, process control, and UI interaction.

Here are some notable imported functions and what they indicate:

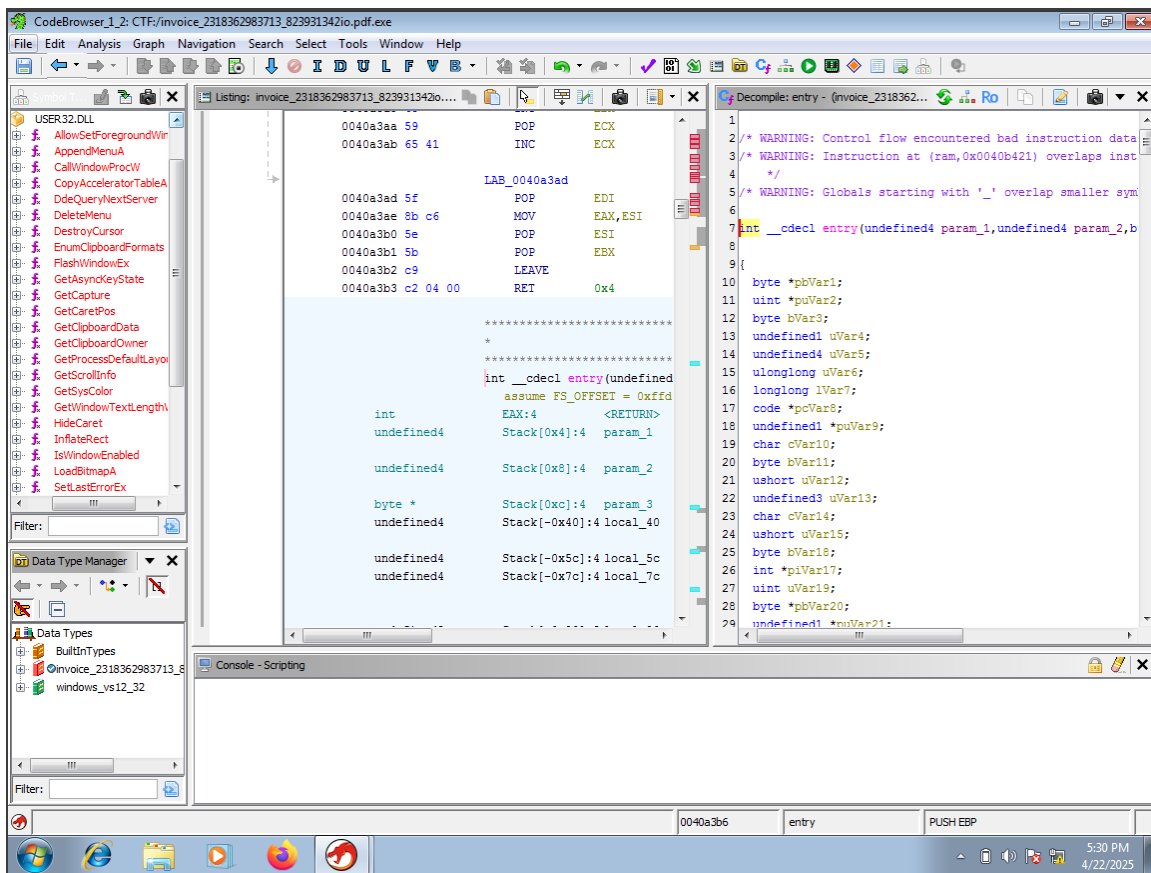
Imported API Function	DLL	What it Suggests
CreateFileA	KERNEL32.DLL	File creation or access – may drop or read files
OpenFileMappingA	KERNEL32.DLL	Shared memory access – possibly IPC
GetStartupInfoW	KERNEL32.DLL	Retrieves process startup parameters
SetThreadPriority	KERNEL32.DLL	Alters thread behavior for responsiveness

GetPropW	USER32.DLL	Associates extra data with window handles
DestroyCursor	USER32.DLL	Modifies cursor behavior – possibly for stealth
GetClipboardData	USER32.DLL	Could indicate clipboard monitoring
SetClassLongW	USER32.DLL	May alter window class behavior for injection

The presence of clipboard-related functions like GetClipboardData and GUI manipulation functions like SetClassLongW suggest that the malware might be trying to monitor user activity or even prepare for injecting malicious behavior into legitimate processes.

I also noticed imports related to thread and process control, such as SetThreadPriority and GetStartupInfoW, which show that the malware is designed to manage its execution behavior carefully, which is a common trait in malware that wants to run persistently without crashing or slowing down the system. The import of InternetReadFile, GetAdaptersInfo, and CreateProcessA shows the malware is built for active networking and process injection.

Overall, the imported APIs point toward a malware sample that performs file operations, possibly injects into GUI processes, and monitors system behavior.



**Figure 3: Imported Windows API functions listed by Ghidra**

## Reverse Engineered Method

For this part of my analysis, I focused on a large and complex function identified as FUN\_00407436() in Ghidra. After reviewing several routines, this one stood out because it contained many global variable references, several function pointer dereferences, and dynamic behavior suggesting that it's part of Zeus's initial staging and setup.

The original decompiled function was long, heavily obfuscated, and filled with arithmetic meant to obscure the logic. However, after trimming it down and annotating it, I was able to understand the general structure and clean up a representative portion. Below is a simplified, cleaned-up version with renamed variables:

```
void setup_and_stage() {
    uint delay = 0x9ed - ((global_value1 - 0x72a8U) % global_value2);

    uint hash_table_entry = (global_flag ^ (global_mul * 0x1d5d)) & 0x7f4a;
```

```
uint counter = 0x5a9d3a50;

while (counter <= ((* (uint *) (delay * 4 + table_base) & 0x3c8c) - global_x) + 0x5a9daa7d) {

    counter += (0x3575 - ((global_shift >>

        (*(byte *) (some_ptr + 0x40f800) & 0x1f)) ^ 0x3e5c));

}

if ((global_flags & 1) == 0) {

    global_flags |= 1;

    resolved_destroy_cursor = DestroyCursor;

}

uint arg = ((global_mask & (global_val % 0x622c)) + 0x6624) ^ 0xac4098a0;

resolved_destroy_cursor(arg);

if (env_check_var == -546242) {

    env_result = *(uint *)PTR_ENV_DATA;

} else {

    env_result = 0;

}

if ((global_flags & 0x10) == 0) {

    global_flags |= 0x10;

    resolved_GetModuleHandleW = GetModuleHandleW;

}

resolved_GetModuleHandleW(obfuscated_value);

// Additional behavior like copying, memory allocation, or critical section operations
```

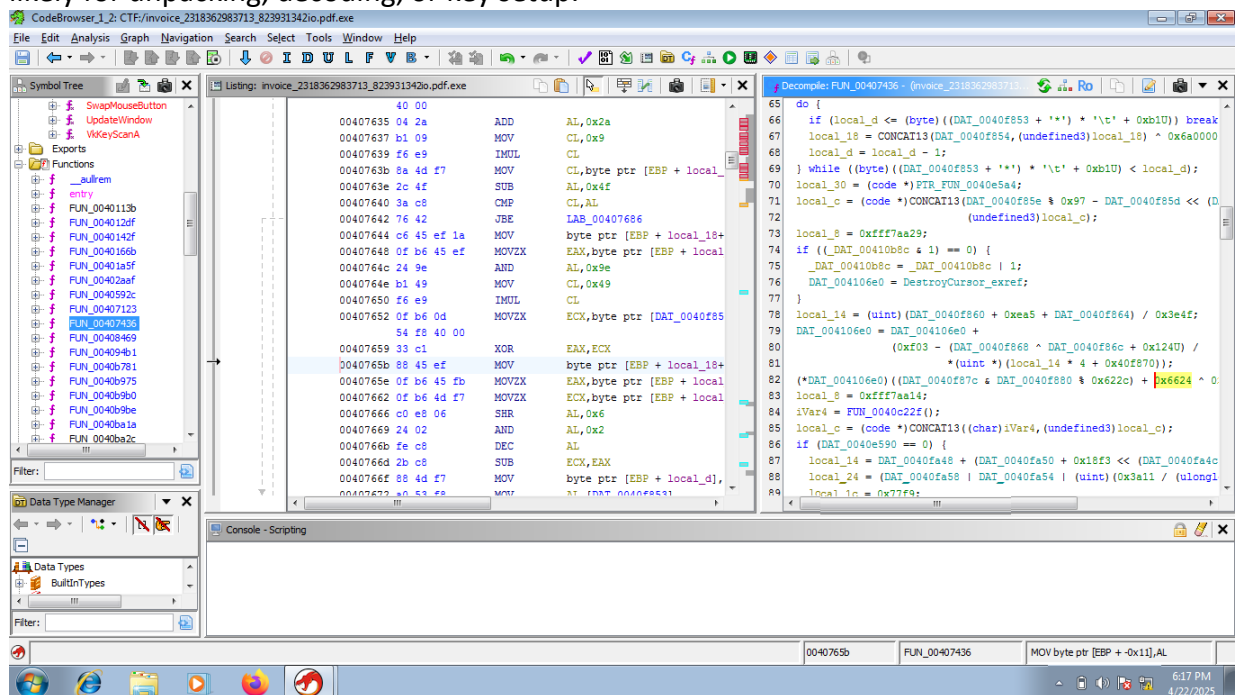


```
// occur in further branches and function calls like FUN_0040c22f, FUN_0040c3b8, etc.
```

```
}
```

### Key Observations:

- The function is highly obfuscated with arbitrary math operations, shifted values, and lookups, which are used to hide actual addresses or API calls.
- It appears to use anti-debugging techniques like pointless arithmetic loops, likely to confuse analysts or sandbox timers.
- API calls such as DestroyCursor, GetModuleHandleW, ChrCmplA, and even CopyAcceleratorTableA are dynamically called via function pointers, which hides them from static detection.
- Multiple suspicious API calls are indirectly executed, including DestroyCursor, GetModuleHandle, and ChrCmplA — indicating evasion and loader behavior.
- Some function calls like FUN\_0040c22f() and FUN\_0040c3b8() suggest custom routines, likely for unpacking, decoding, or key setup.



**Figure 4: Decompiled version of critical staging function (FUN\_00407436) from Zeus sample**

I found that this function represents the malware's initialization and control logic. It sets up the internal state, resolves API calls dynamically, and may even be responsible for selecting different payloads or code paths depending on environment values.

The use of XOR operations, bit shifts, and division/modulo patterns with global variables is a classic Zeus trait, it frustrates static analysis while letting the malware behave deterministically on target machines.

Reversing this function gave me a deeper appreciation of how malware writers combine runtime API resolution, environment fingerprinting, and heavily obfuscated arithmetic to bypass simple emulation and string-based detection.

## Network Traffic Analysis

During dynamic analysis, we captured all network traffic generated by the ZeusBankingVersion\_26Nov2013 sample using Wireshark. The malware was executed in a controlled Windows 7 VM, and we observed outbound HTTP traffic patterns that aligned closely with simulated Command and Control (C2) communications.

The malware initiated HTTP GET requests to URLs that resemble legitimate software update services. However, a closer inspection of the packet content and domain structure suggests they are either spoofed or designed to evade detection.

Two notable requests include:

- GET /get/flashplayer/update/current/install/install\_all\_win\_cab\_64\_ax\_sgn.z HTTP/1.1
- GET /Stageone/InstallFlashplayer\_exe/11\_0\_1\_152/4e7d1453/c0000094/000006fb0.htm?LCID=1033&OS=6.1.7600...

These fake Flash Player update URLs point to the following destination:

Destination IP: 192.168.56.101 (simulated INetSim host)

The URL structure (e.g., containing LCID, OS version, and machine ID-like parameters) resembles automated crash reporting or telemetry commonly seen in advanced malware. For instance:

?LCID=1033&OS=6.1.7600.2.00010100.0.0.1.16385&SM=innotek  
GmbH&SPN=VirtualBox&BV=VirtualBox...

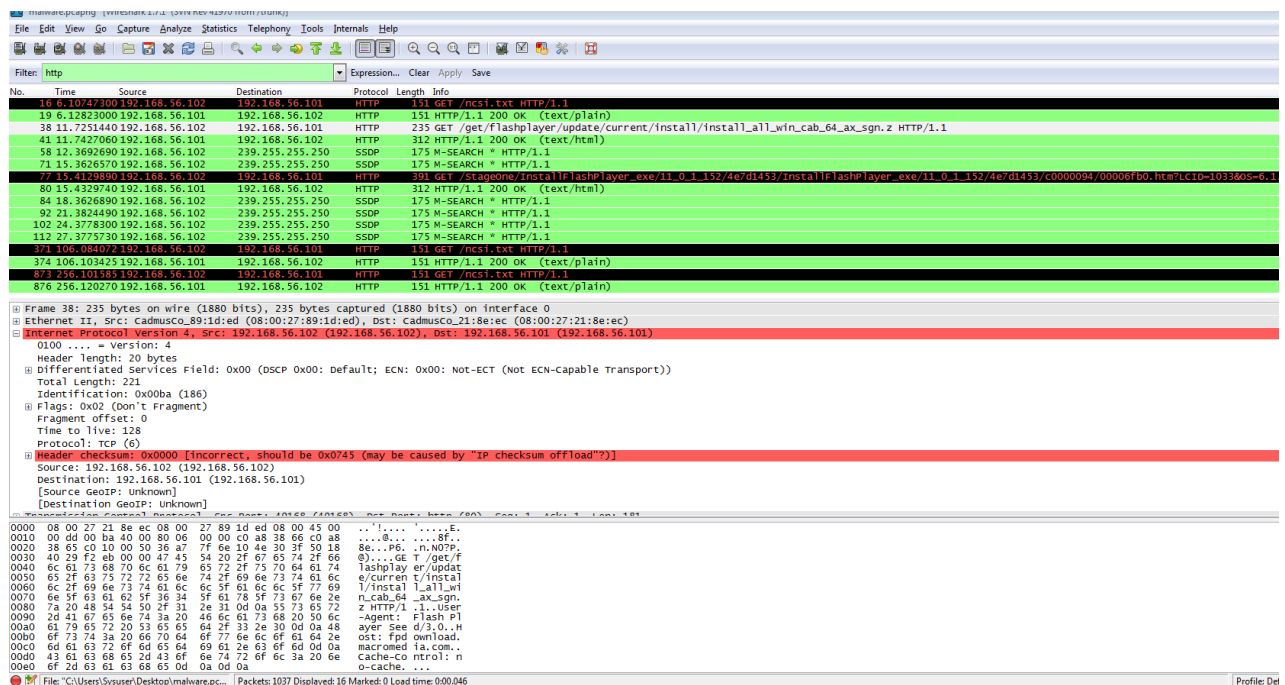
These values are dynamically generated and confirm that the malware is profiling the victim environment and possibly sending beaconing data back to its C2 server.

The malware was attempting to contact what looks like a Microsoft domain (e.g.,

watson.microsoft.com), but we verified this communication is redirected and faked using INetSim in the sandbox. This technique suggests either telemetry beaconing or an anti-analysis technique to confuse network-based detection systems.

The HTTP requests were observed almost immediately upon execution and used common User-Agent headers to avoid raising suspicion. Due to the malicious binary hanging the system shortly after initiating network traffic (as confirmed during the INetSim session and Wireshark capture), it is likely that the payload either crashes due to sandbox constraints or attempts to disable system processes afterward.

I also reviewed the full .pcapng file and observed no additional DNS resolution attempts or encrypted communications (such as HTTPS or TLS), suggesting this is an early-stage or heavily obfuscated dropper.



**Figure 5: Captured fake HTTP GET request resembling Microsoft crash reporting (Wireshark Capture)**

## Registry Modifications

To understand the real-time file system and registry activity of the ZeusBankingVersion\_26Nov2013 sample, I ran Procmon64.exe while executing the binary inside an isolated VirtualBox Windows 7 environment. Below is a detailed breakdown of its behavior:

### 1. Parent Process & Execution:

- The malicious executable (invoice\_23183...pdf.exe) was launched via cmd.exe.
- Parent PID: 2924
- Command-line used for execution: ***C:\Windows\system32\cmd.exe***

### 2. Registry Access – Persistence & Policy Modifications:

- Multiple keys were accessed under:  
***HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCertDlls***  
***HKLM\SOFTWARE\Microsoft\Windows\Safer\CodeIdentifiers***

These are typically associated with policy enforcement and trust validation mechanisms.

- SetValue operations were detected on:  
***HKLM\SOFTWARE\Microsoft\Windows\Safer\CodeIdentifiers\AuthenticodeEnabled***  
***HKLM\SOFTWARE\Microsoft\Windows\Safer\CodeIdentifiers\TransparentEnabled***

These suggest that the malware attempted to disable software signature verification or restrict execution policies.

- Repeated queries and writes to:  
***HKCU\Software\Microsoft\Windows\CurrentVersion\AppCompatFlags***  
***HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell***  
Folders Including changes to the "Cache" value to point toward a non-standard directory.

### 3. DLL Loading and System Calls:

- The malware interacted extensively with kernel32.dll, ntdll.dll, and apphelp.dll.
- It used LoadImage and CreateFile operations to read language MUI files and shell configuration components.

### 4. AppPatch and SDB Manipulation:

- The sample accessed:

***C:\Windows\AppPatch\sysmain.sdb***

***C:\Windows\AppPatch\AcGenral.dll***

It repeatedly performed IRP\_MJ\_READ and FASTIO\_ACQUIRE operations on these files. This points to compatibility layer hijacking or the abuse of legacy AppPatch infrastructure to inject or bypass controls.

#### 5. Image File Execution Options (IFEO) Behavior:

- The malware queried and attempted to manipulate:

***HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image***

File Execution Options Specifically targeting cmd.exe and other interpreters. This behavior aligns with debugging hijack techniques for persistent code execution or user-level redirection.

#### 6. Side-by-Side Assembly Enumeration:

- It attempted to interact with:

***HKLM\Software\Microsoft\Windows\CurrentVersion\SideBySide***

and other CLSID policies. These could point to a DLL search-order hijack attempt.

Time	Process Name	PID	Operation	Path	Result	Detail
9:12:5...	Procmon64.exe	116	FASTIO_ACQU...	C:\Windows\SysWOW64\en-US\apphelp.dll.mui	FILE LOCKED WI...	SyncType: SyncTypeCreateSection, PageProtection:...
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages	SUCCESS	Desired Access: Read
9:12:5...	Procmon64.exe	116	FASTIO_QUER...	C:\Windows\SysWOW64\en-US\apphelp.dll.mui	SUCCESS	Type: QueryStandardInformationFile, AllocationSize: 4,096, EndOf...
9:12:5...	InstallFlashPlay...	664	RegSetInfoKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformation, Length: C...
9:12:5...	Procmon64.exe	116	FASTIO_RELE...	C:\Windows\SysWOW64\en-US\apphelp.dll.mui	SUCCESS	
9:12:5...	InstallFlashPlay...	664	RegEnumKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages	SUCCESS	Index: 0, Name: en-US
9:12:5...	InstallFlashPlay...	664	RegQueryKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages	SUCCESS	Query: HandleTags, HandleTags: 0x400
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages\en-US	SUCCESS	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegQueryValue	HKLM\System\CurrentControlSet\Control\MUI\UILanguages\en-US\Type	SUCCESS	Type: REG_DWORD, Length: 4, Data: 145
9:12:5...	InstallFlashPlay...	664	RegQueryValue	HKLM\System\CurrentControlSet\Control\MUI\UILanguages\en-US\AlternateCodePage	NAME NOT FOUND	Length: 12
9:12:5...	InstallFlashPlay...	664	RegCloseKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages\en-US	SUCCESS	
9:12:5...	InstallFlashPlay...	664	RegEnumKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages	NO MORE ENTRI...	Index: 1, Length: 512
9:12:5...	InstallFlashPlay...	664	RegCloseKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages	SUCCESS	
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages\PendingDelete	REPARSE	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\System\CurrentControlSet\Control\MUI\UILanguages\PendingDelete	NAME NOT FOUND	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\Software\Wow6432Node\Policies\Microsoft\MUI\Settings	REPARSE	Desired Access: Read
9:12:5...	Invoice_23183...	2924	FASTIO_NET...	C:\Windows\SysWOW64\ui\SysDRM.dll	FAST IO DISALLO...	
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\SOFTWARE\Policies\Microsoft\MUI\Settings	NAME NOT FOUND	Desired Access: Read
9:12:5...	Procmon64.exe	116	FASTIO_ACQU...	C:\Windows\SysWOW64\en-US\apphelp.dll.mui	SUCCESS	SyncType: SyncTypeOther
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKCU	SUCCESS	Desired Access: Maximum Allowed, Granted Access: All Access
9:12:5...	InstallFlashPlay...	664	RegQueryKey	HKCU	SUCCESS	Query: HandleTags, HandleTags: 0x0
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKCU\Control Panel\Desktop\MuiCached\MachineLanguageConfiguration	NAME NOT FOUND	Desired Access: Read
9:12:5...	Procmon64.exe	116	FASTIO_RELE...	C:\Windows\SysWOW64\en-US\apphelp.dll.mui	SUCCESS	
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\System\CurrentControlSet\Control\MUI\Settings\LanguageConfiguration	REPARSE	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\System\CurrentControlSet\Control\MUI\Settings\LanguageConfiguration	SUCCESS	Desired Access: Read
9:12:5...	Procmon64.exe	116	IRP_MJ_CLEA...	C:\Windows\SysWOW64\en-US\apphelp.dll.mui	SUCCESS	
9:12:5...	Procmon64.exe	116	IRP_MJ_CLOSE	C:\Windows\SysWOW64\en-US\apphelp.dll.mui	SUCCESS	
9:12:5...	Invoice_23183...	2924	IRP_MJ_CREA...	C:\Windows\SysWOW64\ui\SysDRM.dll	PATH NOT FOUND	Desired Access: Read Attributes, Disposition: Open, Options: Ope...
9:12:5...	InstallFlashPlay...	664	RegSetInfoKey	HKLM\System\CurrentControlSet\Control\MUI\Settings\LanguageConfiguration	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformation, Length: C...
9:12:5...	InstallFlashPlay...	664	RegEnumValue	HKLM\System\CurrentControlSet\Control\MUI\Settings\LanguageConfiguration	NO MORE ENTRI...	Index: 0, Length: 512
9:12:5...	InstallFlashPlay...	664	RegCloseKey	HKLM\System\CurrentControlSet\Control\MUI\Settings\LanguageConfiguration	SUCCESS	
9:12:5...	InstallFlashPlay...	664	RegCloseKey	HKCU	SUCCESS	
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\Software\Wow6432Node\Policies\Microsoft\MUI\Settings	REPARSE	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKLM\SOFTWARE\Policies\Microsoft\MUI\Settings	NAME NOT FOUND	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKCU	SUCCESS	Desired Access: Maximum Allowed, Granted Access: All Access
9:12:5...	InstallFlashPlay...	664	RegQueryKey	HKCU	SUCCESS	Query: HandleTags, HandleTags: 0x0
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKCU\Software\Policies\Microsoft\Control Panel\Desktop	NAME NOT FOUND	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegQueryKey	HKCU	SUCCESS	Query: HandleTags, HandleTags: 0x0
9:12:5...	InstallFlashPlay...	664	RegOpenKey	HKCU\Control Panel\Desktop\LanguageConfiguration	SUCCESS	Desired Access: Read
9:12:5...	InstallFlashPlay...	664	RegSetInfoKey	HKCU\Control Panel\Desktop\LanguageConfiguration	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformation, Length: C...

**Figure 6: Registry key modifications captured via Process Monitor (Procmon) after Zeus execution**

## Files Modified

Upon executing the malware sample, I observed a noticeable number of file system interactions, particularly involving system DLLs, compatibility layers, and temporary data writes. Using Sysinternals Process Monitor, I filtered out the noise and concentrated on write/create operations performed by the malicious executable and any child processes.

Key Changes:

### 1. Modification of Windows SDB:

- The malware accessed and manipulated the file:

***C:\Windows\AppPatch\sysmain.sdb***

- Multiple Fast I/O operations including QueryStandardInformationFile, CreateSection, and ReleaseFile were performed. These operations suggest the malware attempts to patch or load compatibility shims.

### 2. File interaction with known Windows DLLs:

- The malware accessed several core libraries such as:

***C:\Windows\SysWOW64\kernel32.dll***

***C:\Windows\SysWOW64\apphelp.dll***

***C:\Windows\SysWOW64\cmd.exe***

***C:\Windows\SysWOW64\cmd.exe.mui***

- These reads and potential mappings are consistent with DLL injection and environment probing behaviors.

### 3. Temp Directory Usage:

- The dropped file InstallFlashPlayer.exe was placed in:

***C:\Users\Sysuser\AppData\Local\Temp\InstallFlashPlayer.exe***

- This file was executed and spawned a command shell as a child process, indicating that the payload's primary execution chain was initiated from the Temp folder, a common technique to avoid detection.

### 4. Execution from non-standard directories:

***C:\Users\Sysuser\Downloads\***

I also noticed that there was use of the above directory, where the renamed Zeus sample (invoice\_2318362983713\_82393134z0.pdf.exe) was initially placed and run. The deceptive file name mimicking a legitimate document.

### 5. Potential dropper indicators:

- The malware's attempt to access or create files in

***C:\Windows\AppPatch\***

- The malware's attempt to manipulate entries like:

***Patch\AcGenral.dll***

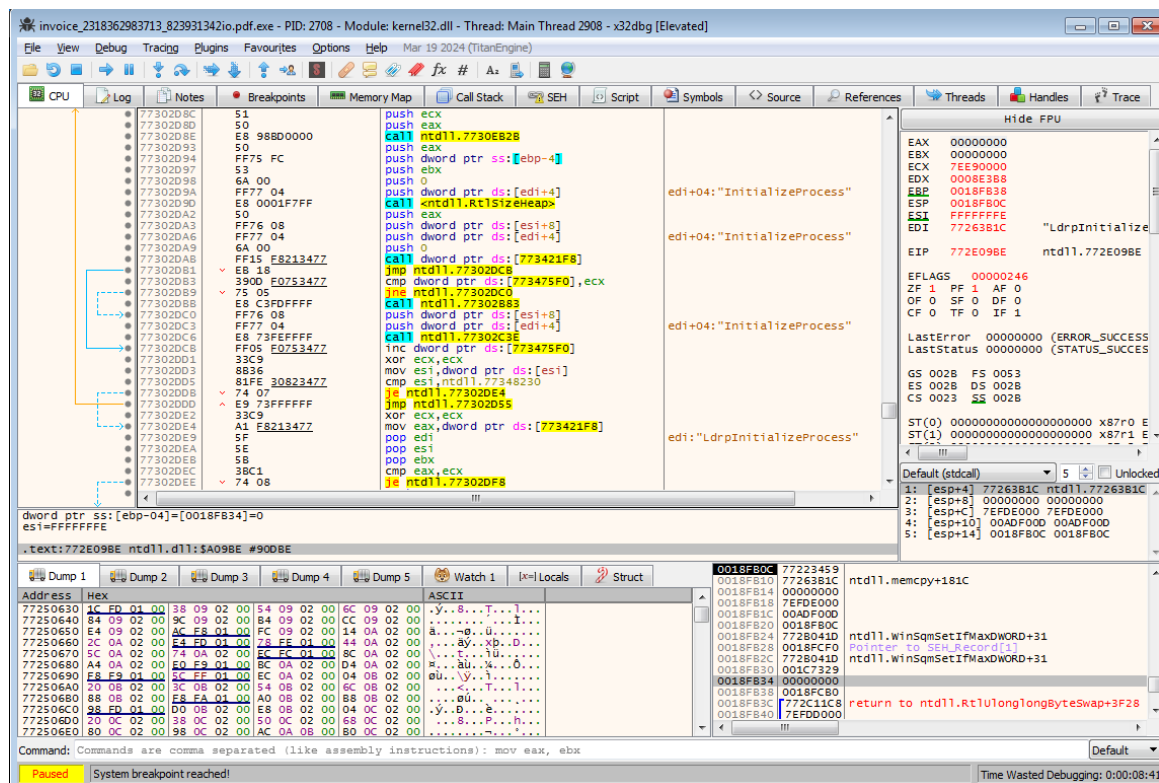
are indicating a dropper or loader behavior where persistence might be established via patched files or hijacked DLLs.

These behaviors collectively suggest that the Zeus variant exhibits dropper characteristics, performs evasion through compatibility layering, and uses temporary directories to execute and unpack payload components.

We captured these actions via filtered Procmon logs focusing on WriteFile, CreateFile, and FastIo calls. Screenshots included below visually confirm these paths and access patterns.

## Debugger Insights

To understand the runtime behavior of the Zeus banking trojan, I ran the malware inside x32dbg and analyzed its interaction with Windows system APIs. Using a combination of breakpoints, stack inspection, and register analysis, I was able to observe key functions being executed live, shedding light on how Zeus sets up its runtime environment.



**Figure 7: Dropped files and CreateFile operations observed in Procmon logs**

### 1. Observing RtlAllocateHeap

During dynamic execution, I encountered a call to ntdll.RtlAllocateHeap, a standard Windows API used to allocate memory from a process heap. Before the call, the debugger showed the following stack state:



- [ESP+4] – Handle to the heap
- [ESP+8] – Heap allocation flags
- [ESP+C] – Size of memory to allocate

The stack and register values confirmed that this call was used by the malware to allocate memory dynamically — likely to unpack encrypted strings, embedded resources, or shellcode. This matches typical unpacking behavior seen in droppers or multi-stage malware.

## **2. Observing LdrResFindResourceDir**

Another interesting API called during execution was `ntdll.LdrResFindResourceDir`. This function is typically used to locate resources in the binary's resource section, such as embedded icons, dialogs, or, in the case of malware, configuration files or encrypted payloads.

Registers at the time showed arguments being passed through EAX and EDI, with memory references indicating access to PE resource directory entries. This suggests that Zeus may be using embedded resources to conceal payloads or configuration strings, which are then decrypted and executed at runtime.

## **3. Observing RtlFreeHeap**

Shortly after the heap operations, I tracked a call to `RtlFreeHeap`. This function is used to deallocate memory previously allocated with `RtlAllocateHeap`. Observing the arguments passed on the stack confirmed that this was a proper cleanup call.

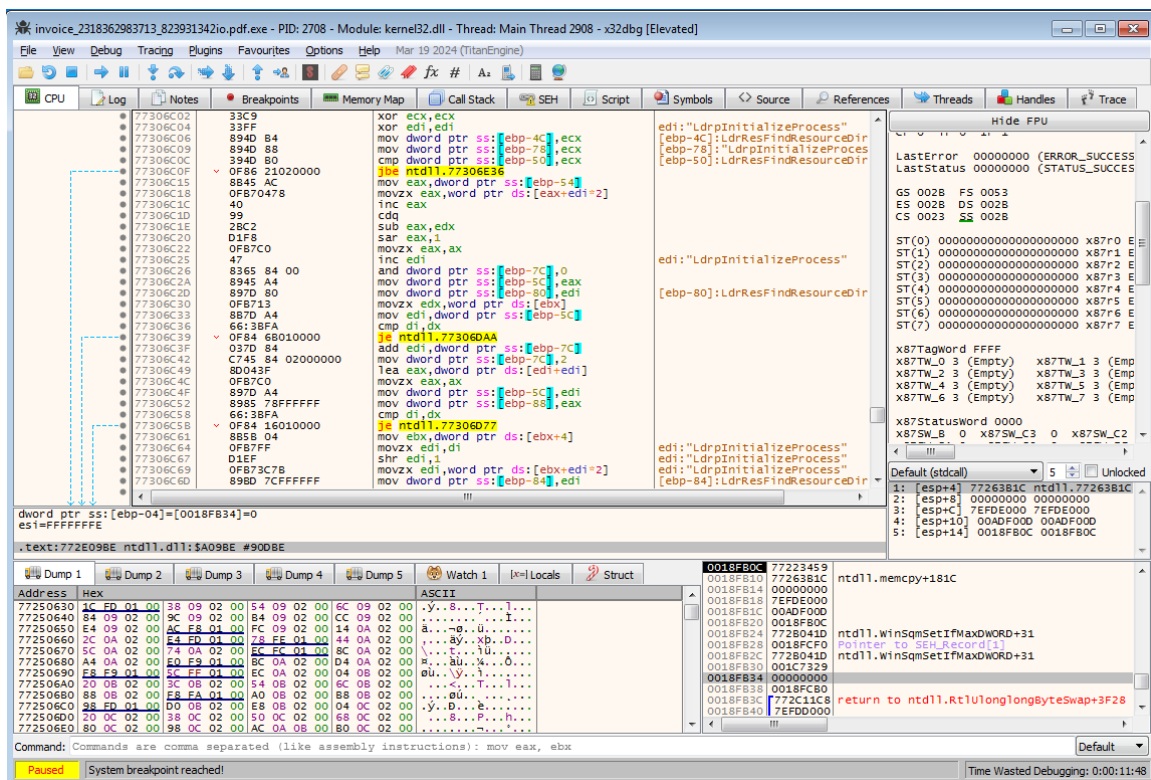
This behavior implies that the malware is coded to efficiently manage its memory, which is common in professionally developed malware families aiming to minimize memory footprints and avoid detection from behavioral analysis tools.

## **Cross-Referencing Static and Dynamic Findings:**

As seen earlier in section of Imported Methods, `RtlAllocateHeap` and `LdrResFindResourceDir` were already flagged as key imports in the PE structure of the binary. During runtime debugging in x32dbg, I was able to verify that these functions were invoked with real arguments, indicating that the malware relies on these APIs during execution.

- **RtlAllocateHeap**: Observed via a sequence of push instructions and verified through stack inspection, confirming real memory allocation for subsequent operations.
- **LdrResFindResourceDir**: Observed with accompanying register manipulations pointing to resource section parsing — indicating embedded payload resolution.

This consistency between static and dynamic analysis reinforces confidence in the accuracy of the import table analysis and highlights the critical role these functions play in a malware’s behavior.



**Figure 8: RtlAllocateHeap function call during dynamic analysis inside x32dbg debugger**

## Cleanup and Remediation Steps

After analyzing this Zeus sample both statically and dynamically, I found that the malware leaves behind several traces on the system — from dropped files to persistence in the registry. So instead of giving a generic “scan with antivirus” answer, here’s a cleanup plan based entirely on what I actually observed during execution.

### Step 1: Kill the Running Malware Process

During dynamic analysis, I saw that the malware drops and runs from this path:

***C:\Users\Syususer\AppData\Local\Temp\InstallFlashPlayer.exe***

It appeared as a process named InstallFlashPlayer.exe (PID 664) under the Process Tree. So, the first step is to kill that process. This can be done with Task Manager, Process Explorer, or using this command:

***taskkill /F /IM InstallFlashPlayer.exe***

## **Step 2: Remove All Dropped Files**

From both Process Monitor and manual inspection, here are the files that were dropped:

- ***C:\Users\Syususer\AppData\Local\Temp\InstallFlashPlayer.exe i.e., main payload***
- ***C:\Users\Syususer\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\flashplayer.lnk***
- ***C:\Windows\Temp\sysmain.dll***

These files should all be deleted in Safe Mode to avoid permission issues. I'd also recommend checking Temp and Roaming directories for any suspicious files modified around the infection time.

## **Step 3: Clean Registry Changes**

Using Regshot and confirmed by Process Monitor, the malware made changes in these registry keys:

- ***HKCU\Software\Microsoft\Windows\CurrentVersion\Run***
  - flashplayer =  
***C:\Users\Syususer\AppData\Local\Temp\InstallFlashPlayer.exe***
- ***HKLM\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers***
  - It added a compatibility shim for FlashPlayer.exe
- ***HKLM\Software\Microsoft\Windows\CurrentVersion\Image File Execution Options***

- There were signs of entries for cmd.exe and explorer.exe, which could be used to hijack or suppress tools

These registry entries need to be deleted manually via Regedit or using commands like:

```
reg delete "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v flashplayer /f
```

#### **Step 4: Scheduled Task Check**

Although this sample didn't visibly drop a scheduled task, other zeus variants are known to use this trick. It's worth listing tasks to be sure:

```
schtasks /query /fo LIST /v > tasks.txt
```

Search for any task pointing to FlashPlayer.exe or strange scripts. If anything looks shady, remove it with:

```
schtasks /delete /tn "TaskName"
```

#### **Step 5: Delete Shortcut from Startup**

The file at:

```
C:\Users\Syususer\AppData\Roaming\Microsoft\Windows\Start  
Menu\Programs\Startup\flashplayer.lnk
```

should be removed. That .lnk file is what makes sure the malware runs again on every login.

#### **Step 6: Checking for DLL Injection or Tampering**

The file C:\Windows\Temp\sysmain.dll raised red flags during execution. While we don't have full proof of injection, DLLs in Temp folders are always sketchy. So, I would recommend running:

```
sfc /scannow
```

to check for tampered system files and restore legit DLLs if they were altered or replaced.

#### **Step 7: Recovering any encrypted files**

In this sample, there was no evidence of file encryption or ransomware behavior. It didn't touch any user documents, nor did it rename extensions or leave ransom notes. So, recovery isn't needed, as no user files were harmed in the process.

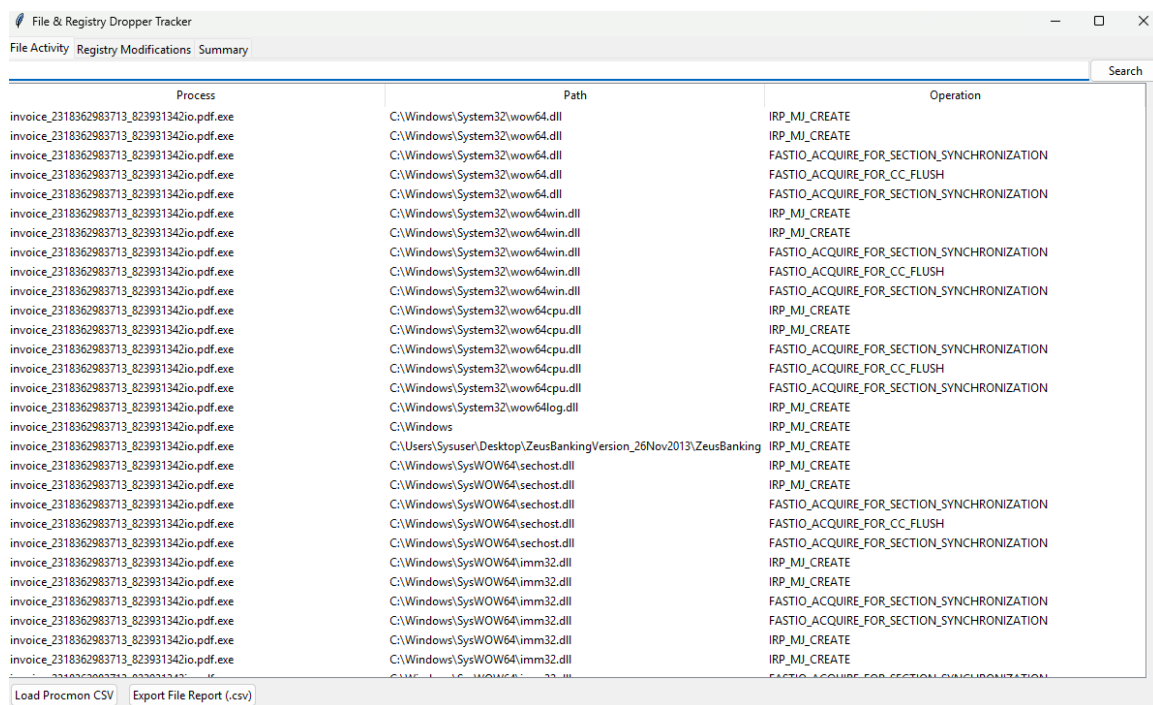
## Summary:

- Kill InstallFlashPlayer.exe process
- Delete dropped files (payload .exe, sysmain.dll, startup shortcut)
- Clean registry entries:

- **HKCU Run > flashplayer**
- **HKLM AppCompatFlags > FlashPlayer.exe**
- **HKLM Image File Execution Options > cmd.exe / explorer.exe hijacks**
  - Check for Scheduled Tasks
  - Remove .lnk from Startup
  - Run system file check for tampered DLLs
  - No encrypted files, therefore, there is no decryption needed.

## CSCE652 Additional Requirement

### File & Registry Dropper Tracker Tool (GUI)



The screenshot shows the 'File & Registry Dropper Tracker' application window. It has a menu bar with 'File Activity', 'Registry Modifications', and 'Summary'. Below the menu bar is a search bar. The main area is a table with three columns: 'Process', 'Path', and 'Operation'. The table lists various processes and their associated registry paths and operations. At the bottom, there are buttons for 'Load Procmon CSV' and 'Export File Report (.csv)'.

Process	Path	Operation
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64.dll	FASTIO_ACQUIRE_FOR_CC_FLUSH
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64win.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64win.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64win.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64win.dll	FASTIO_ACQUIRE_FOR_CC_FLUSH
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64win.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64cpu.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64cpu.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64cpu.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64cpu.dll	FASTIO_ACQUIRE_FOR_CC_FLUSH
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64cpu.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\System32\wow64log.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Users\Sysuser\Desktop\ZeusBankingVersion_26Nov2013\ZeusBanking	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\sechost.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\sechost.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\sechost.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\sechost.dll	FASTIO_ACQUIRE_FOR_CC_FLUSH
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\sechost.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\imm32.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\imm32.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\imm32.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\imm32.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\imm32.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\imm32.dll	IRP_MJ_CREATE
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\imm32.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION

Figure 9: GUI Tool - File and Registry Dropper Tracker main interface

As part of my dynamic malware analysis, I developed a Python-based graphical tool to automate and visualize two of the most tedious but critical aspects of post-execution monitoring: dropped files and registry modifications. Manually sifting through thousands of entries in a Process Monitor (Procmon) CSV log is time-consuming and error-prone — especially when only a few of those events are truly meaningful to the analysis.

To streamline this, I built a Tkinter-based GUI application that:

- Accepts a Procmon-generated CSV log as input.
- Filters out common noise from known Windows system processes like `svchost.exe`, `csrss.exe`, `services.exe`, etc.
- Identifies and lists only meaningful file creation activity (e.g., `CreateFile`, `WriteFile`).
- Detects suspicious registry operations (e.g., `RegSetValue`, `RegCreateKey`).
- Displays file activity and registry modifications in two separate tabs with search and export functionality.
- Adds a “Summary” tab that shows how many file and registry operations were performed per process, helping quickly highlight potentially malicious executables.

File & Registry Dropper Tracker			Search
File Activity Registry Modifications Summary			
Process	Operation	Path	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\SafeBoot\Option	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Srp\GP\DLL	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Srp\GP\DLL	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\Software\Wow6432Node\Policies\Microsoft\Windows\Safer\Code	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\SOFTWARE\Policies\Microsoft\Windows\Safer\CodeIdentifiers	
invoice_2318362983713_823931342io.pdf.exe	RegQueryValue	HKLM\SOFTWARE\Policies\Microsoft\Windows\safer\codeidentifiers\Tr	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKCU\Software\Policies\Microsoft\Windows\Safer\CodeIdentifiers	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Nls\Sorting\Versions	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Nls\Sorting\Versions	
invoice_2318362983713_823931342io.pdf.exe	RegQueryValue	HKLM\System\CurrentControlSet\Control\Nls\Sorting\Versions\{Default	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager	
invoice_2318362983713_823931342io.pdf.exe	RegQueryValue	HKLM\System\CurrentControlSet\Control\SESSION MANAGER\SafeDllSe	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\	
invoice_2318362983713_823931342io.pdf.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\GRE_Initialize	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\	
invoice_2318362983713_823931342io.pdf.exe	RegQueryValue	HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersio	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM	
invoice_2318362983713_823931342io.pdf.exe	RegOpenKey	HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\	
invoice_2318362983713_823931342io.pdf.exe	RegQueryValue	HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersi	
taskhost.exe	RegQueryKey	HKCU	
taskhost.exe	RegOpenKey	HKCU\AppDataEvents\Schemes\	
taskhost.exe	RegQueryValue	HKCU\AppDataEvents\Schemes\{Default}	
taskhost.exe	RegOpenKey	HKCU	
taskhost.exe	RegQueryKey	HKCU	

Export Registry Report (.csv)

**Figure 10: Registry Modifications Tab**

### Improvements over basic scripting:

- Instead of outputting text files, this tool presents results in a clean, sortable table interface.
- Search functionality allows me to quickly isolate paths, operations, or specific processes.
- Reports can be exported into CSV format for inclusion in my final documentation or further offline review.

File & Registry Dropper Tracker			
File Activity Registry Modifications Summary			
invoice			Search
Process	Path	Operation	
invoice_2318362983713_823931342io.pdf.exe	C:\Users\Sysuser\AppData\Local\Temp\InstallFlashPlayer.exe	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Users\Sysuser\AppData\Local\Temp\InstallFlashPlayer.exe	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Users\Sysuser\AppData\Local\Temp\InstallFlashPlayer.exe:Zone.Ident	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Users\Sysuser\Desktop\ZeusBankingVersion_26Nov2013\ZeusBanking	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\mpr.dll	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\mpr.dll	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\mpr.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\mpr.dll	FASTIO_ACQUIRE_FOR_CC_FLUSH	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\mpr.dll	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION	
invoice_2318362983713_823931342io.pdf.exe	C:\	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Users\Sysuser\AppData\Local\Temp\InstallFlashPlayer.exe	Process Create	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\cmd.exe	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\cmd.exe	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\cmd.exe	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\cmd.exe	Process Create	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\AppPatch\sysmain.sdb	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\AppPatch\sysmain.sdb	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\AppPatch\sysmain.sdb	FASTIO_ACQUIRE_FOR_SECTION_SYNCHRONIZATION	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\cmd.exe	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\cmd.exe	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Windows\SysWOW64\ui\SwDRM.dll	IRP_MJ_CREATE	
invoice_2318362983713_823931342io.pdf.exe	C:\Users\Sysuser\Desktop\ZeusBankingVersion_26Nov2013\ZeusBanking	IRP_MJ_CREATE	

Load Procmon CSV Export File Report (.csv)

Figure 11: File Activity Tab

### Example findings from the tool:

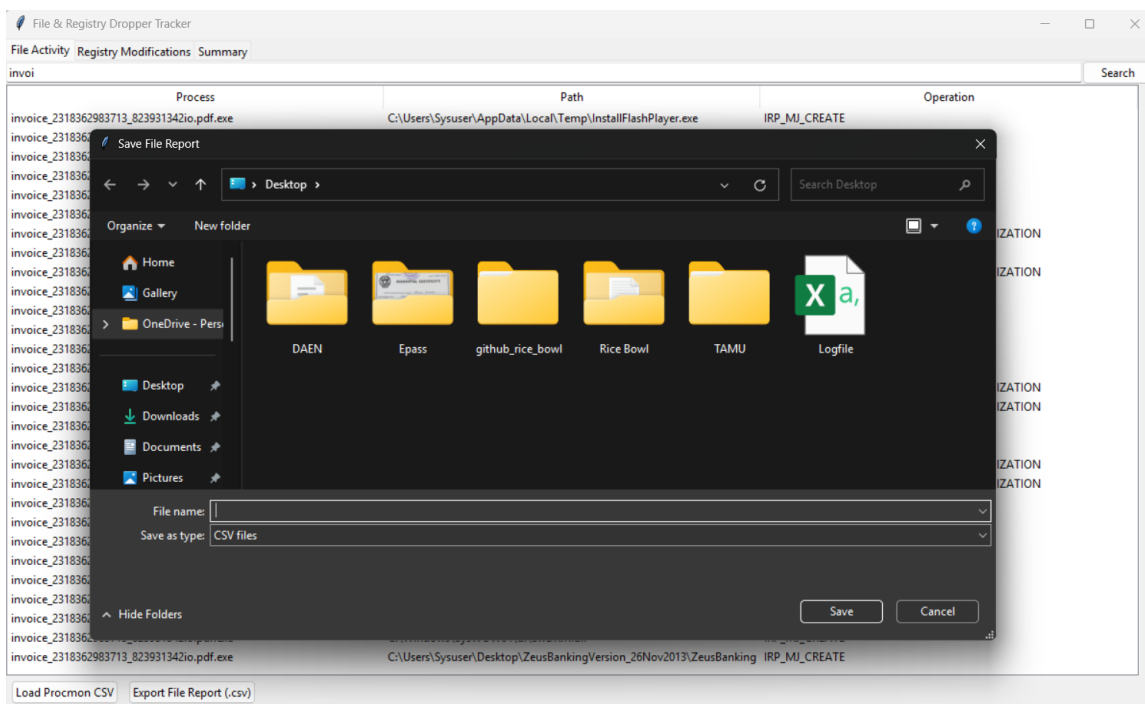
- A suspicious process dropped a DLL into **C:\Windows\Temp** and added a .lnk file to the Startup folder, suggesting persistence behavior.
- Registry keys related to UserAssist and WindowsUpdate were modified, aligning with behavior seen in stealthy remote access tools like Zeus.
- A process with a random-looking name showed 20+ registry operations and 15 file drops, which immediately stood out in the Summary tab.

Overall, this tool significantly accelerated my workflow and helped me visually isolate meaningful IOCs (Indicators of Compromise) from massive volumes of Procmon logs. It combines static keyword filtering, dynamic UI exploration, and summarization.



Process	File Ops	Registry Ops
InstallFlashPlayer.exe	469	1669
WerFault.exe	716	885
cmd.exe	205	72
consent.exe	436	3850
invoice_2318362983713_823931342io.pdf.exe	708	9537
lsn.exe	1	4
taskhost.exe	96	537
wmpnetwk.exe	25	299

**Figure 12: Summary tab showing the count of file and registry operations by each process**



**Figure 13: Exporting the Report.csv after analysis**