

Forum:[Hack Forums](#)

Thread: [./Dark Legion/Digital Shadows: Unmasking the Art of Process Hollowing in the Hacker's Underworld](#)

Dread.Reaper - April 8, 2023, 10:17 am

@Genuine@

Digital Shadows: Unmasking the Art of Process Hollowing in the Hacker's Underworld

Written By: Dread.Reaper - Dark Legion

Introduction:

In the shadowy recesses of cyberspace, a realm governed by bits and bytes, an elite group of digital warriors continuously seeks new methods for penetrating the defenses of their unsuspecting targets. These hackers, skilled in the arcane arts of computer manipulation, employ a diverse arsenal of techniques to infiltrate systems, exfiltrate sensitive data, and wreak havoc upon the digital landscape. Among their most potent weapons lies a cunning and elusive method known as process hollowing, which masterfully evades detection while sowing chaos in its wake.

Process hollowing, a sophisticated form of code injection, is the digital equivalent of a chameleon. Its raison d'être is to obfuscate malicious code, surreptitiously embedding it within the confines of seemingly innocuous processes. This cunning subterfuge allows the nefarious payload to slip past the watchful eyes of even the most advanced security systems, silently biding its time until it springs forth to execute its malevolent intentions.

To accomplish this devious feat, hackers leverage a series of intricate API function calls, manipulating the inner workings of the operating system to their advantage. The process begins with the creation of a legitimate process in a suspended state, such as the seemingly harmless notepad.exe. The unsuspecting victim remains oblivious, unaware that the seeds of destruction have already been sown. Next, the hacker unmaps the memory content of the target process, hollowing it out to create a clandestine lair for their malicious payload. With the hollowed space prepared, the attacker allocates new memory and writes the malicious code into the vacated recesses.

As the final act of subterfuge, the hacker manipulates the thread context, directing the hijacked process to execute the malicious payload upon resumption. With the stage set and the machinations complete, the suspended process springs back to life, carrying out its sinister payload unbeknownst to its hapless host.

Process hollowing is but one of many treacherous tools in the hacker's toolkit, an insidious reminder of the ever-evolving nature of cyber threats. As defenders scramble to adapt and respond, hackers continue to push the boundaries of their craft, constantly seeking new vulnerabilities to exploit and innovative methods to remain undetected. In this digital game of cat and mouse, the stakes are high, and the battle rages on.

I. Introduction

Process hollowing, an advanced code injection technique, is utilized by malicious actors to execute harmful payloads within legitimate processes.

By leveraging process hollowing, attackers can evade detection from security tools and knowledgeable users.

II. Overview of Process Hollowing

Process hollowing involves the following main steps:

- Creating a legitimate process in a suspended state
- Unmapping the memory content of the legitimate process
- Allocating new memory space for the malicious payload
- Writing the malicious payload into the newly allocated memory space
- Resuming the suspended process to execute the malicious payload

III. Key API Function Calls in Process Hollowing

The process-hollowing technique employs a series of API function calls, which include:

- `CreateProcess`: Initiates a legitimate process in a suspended state
- `ZwUnmapViewOfSection`/`NtUnmapViewOfSection`: Unmaps the memory space of a specific section of a process, creating a hollowed section
- `VirtualAllocEx`: Allocates new memory space within the hollowed section
- `WriteProcessMemory`: Writes the malicious content into the allocated memory space
- `SetThreadContext` and `ResumeThread`: Restores the context to the thread and resumes the process, allowing the execution of the malicious payload

IV. Step-by-Step Implementation of Process Hollowing

Creating a Legitimate Process in a Suspended State

- Choose a target process: Select a legitimate process to serve as the host for the malicious payload. A commonly targeted process is `notepad.exe`, as it is less likely to arouse suspicion.
- Use `CreateProcess` API function: Utilize the `CreateProcess` function with the `dwCreationFlags` parameter set to `CREATE_SUSPENDED`, which ensures that the process remains in a suspended state until further action is taken.

Unmapping the Memory Content of the Legitimate Process

- Locate the process's entry point: Retrieve the base address of the target process's entry point by accessing the PEB (Process Environment Block) structure.
- Use `ZwUnmapViewOfSection`/`NtUnmapViewOfSection` API functions: Invoke these functions to unmap the entire memory space of a specific section of the process, creating a hollowed section.

V. In-Depth Technical Breakdown

Creating a Legitimate Process in a Suspended State

1. Choose a target process
 - i. Identify a suitable process: Select a process that is less likely to raise suspicion, such as `notepad.exe`, `calc.exe`, or `explorer.exe`.
2. Use `CreateProcess` API function
 - i. Set up `PROCESS_INFORMATION` and `STARTUPINFO` structures: These structures will store information about the new process and its main thread.
 - ii. Call `CreateProcess`: Invoke `CreateProcess` with the chosen target process, passing the appropriate flags (`CREATE_SUSPENDED`) and structures (`PROCESS_INFORMATION` and `STARTUPINFO`) as parameters.
 - iii. Verify successful process creation: Ensure that the process is created in a suspended state and that the necessary handles (process and thread handles) are obtained.

Unmapping the Memory Content of the Legitimate Process

1. Locate the process's entry point
 - i. Access the PEB structure: Read the target process's PEB by using the `ReadProcessMemory` API

function with the process handle obtained in step 1.

- ii. Retrieve the base address: Extract the base address of the target process's entry point from the PEB's ImageBaseAddress field.
2. Use ZwUnmapViewOfSection/NtUnmapViewOfSection API functions
 - i. Obtain a handle to ntdll.dll: Load ntdll.dll dynamically and acquire a handle using the GetModuleHandle API function.
 - ii. Locate ZwUnmapViewOfSection/NtUnmapViewOfSection: Use the GetProcAddress API function to find the addresses of the ZwUnmapViewOfSection or NtUnmapViewOfSection functions.
 - iii. Call ZwUnmapViewOfSection/NtUnmapViewOfSection: Invoke the chosen function, passing the process handle and the base address of the target process's entry point as parameters.
 - iv. Verify successful unmapping: Confirm that the memory space of the specific section of the process has been unmapped, creating a hollowed section.

Allocating New Memory Space for the Malicious Payload

1. Determine the size of the malicious payload: Calculate the memory size required to accommodate the malicious payload.
2. Use VirtualAllocEx API function
 - i. Prepare parameters for VirtualAllocEx: Set up the appropriate parameters, including the target process handle, the base address of the target process's entry point, the calculated memory size, and allocation type (MEM_COMMIT | MEM_RESERVE) and protection flags (PAGE_EXECUTE_READWRITE).
 - ii. Call VirtualAllocEx: Invoke the VirtualAllocEx function with the prepared parameters to allocate new memory space within the hollowed section of the target process.
 - iii. Verify successful allocation: Ensure that the new memory space has been allocated and that the base address of the allocated memory matches the base address of the target process's entry point.

Writing the Malicious Payload into the Newly Allocated Memory Space

1. Prepare the malicious payload: Obtain or generate the malicious payload to be injected into the target process.
2. Use WriteProcessMemory API function
 - i. Set up parameters for WriteProcessMemory: Prepare the necessary parameters, including the target process handle, the base address of the newly allocated memory, the malicious payload, and the size of the payload.
 - ii. Call WriteProcessMemory: Invoke the WriteProcessMemory function with the prepared parameters to write the malicious payload into the allocated memory space within the hollowed section of the target process.
 - iii. Verify successful writing: Ensure that the malicious payload has been written to the allocated memory space without any errors.

Resuming the Suspended Process to Execute the Malicious Payload

1. Use SetThreadContext API function
 - i. Obtain the original thread context: Retrieve the original thread context of the target process's main thread using the GetThreadContext API function with the thread handle obtained in step 1.
 - ii. Modify the thread context: Update the thread context's EIP/RIP (Instruction Pointer) register value to point to the entry point of the malicious payload within the newly allocated memory space.
 - iii. Set the new thread context: Invoke the SetThreadContext function with the modified thread context and the thread handle as parameters, applying the changes to the target process's main thread.
2. Use ResumeThread API function
 - i. Call ResumeThread: Invoke the ResumeThread function with the thread handle as a parameter to resume the suspended process and initiate the execution of the malicious payload.
 - ii. Verify successful resumption: Ensure that the process has been resumed and that the malicious

payload is executing as intended.

VI. Detection and Prevention Techniques

Detecting Process Hollowing

1. Monitoring process behavior: Observe processes for unusual behavior, such as a legitimate process spawning another process or establishing network connections that are not typical for the process in question.
2. Memory scanning: Perform memory scans to identify discrepancies between the process memory and the corresponding file on disk.
3. Heuristic analysis: Use heuristic algorithms to detect potential process hollowing indicators, such as a high number of API calls associated with process manipulation.

Prevention Techniques

1. Employ strong security software: Implement comprehensive security solutions, such as antivirus software, intrusion detection systems (IDS), and host-based intrusion prevention systems (HIPS) to monitor and block suspicious activities.
2. Regularly update software: Keep operating systems, applications, and security software up-to-date to minimize vulnerabilities that attackers might exploit.
3. Limit user privileges: Restrict user privileges to minimize the potential impact of an attack by ensuring that users have the least amount of access necessary to perform their tasks.
4. Use application whitelisting: Implement application whitelisting to prevent the execution of unauthorized programs or scripts.
5. Employee training and awareness: Educate users about the risks associated with process hollowing and other advanced threats, encouraging them to follow best security practices.

VII. Code Examples of Process Hollowing

Please note that the following code examples are for educational purposes only and should not be used for malicious activities. The examples are presented using C++.