Introduction to EDR Evasion: API Hooking

redfoxsec.com/blog/introduction-to-edr-evasion-api-hooking

Shashi Kant Prasad January 24, 2024

- January 24, 2024
- Red Team
- Shashi Kant Prasad

Endpoint Detection and Response (EDR) solutions are essential for monitoring and responding to security incidents on endpoints. These solutions employ various techniques to identify malicious behavior, including the use of API hooking. API hooking is one of the commonly employed methods by EDRs. It helps to intercept and redirect the execution flow of specific functions for monitoring and analyzing their behavior.

When a function is hooked, the EDR loads a library into the target process. Afterward, it replaces the initial instruction of the function with a jump instruction leading to a routine within the loaded library.

This enables the EDR to track and analyze the actions of the hooked function. It then facilitates the detection of malicious activities such as process injection. In this blog, we will delve into the increasingly sophisticated realm of 'EDR Evasion' – a crucial concept in the cybersecurity world where attackers continually develop advanced techniques to circumvent Endpoint Detection and Response (EDR) systems.

Windows Syscalls and the Role of NTDLL

To understand EDR evasion techniques, it is crucial to grasp the concept of Windows syscalls. Syscalls, also known as the native API for Windows, provide a way for Windows programmers to interact with the underlying operating system. These syscalls are implemented in the ntdll.dll file. In addition to this, they also serve as the primary interface for accessing system resources and functionalities.

However, most syscalls still need to be officially documented by Microsoft, making it challenging for developers and security solutions to utilize them directly. Instead, they rely on higher-level Windows API functions, which internally call the corresponding syscalls.

When it comes to EDR Solutions, API hooking specifically targets these high-level API functions, like those found in ntdll.dll, to intercept and monitor their behavior. By hooking these functions, EDRs can detect malicious activities like process injection, which often involve the use of these higher functions.

Limitations of API Hooking and the Need for EDR Evasion

API hooking is a valuable method for identifying and preventing malicious activities; however, it has certain constraints. Competent attackers can employ evasion techniques in order to circumvent EDR solutions that heavily depend on API hooking.

One of these limitations involves the dependence on specific API functions. Attackers can evade detection by directly invoking the underlying system calls (syscalls) instead of using the higher-level API functions. By doing so, they can avoid triggering the hooks established by EDR solutions.

Additionally, EDR solutions typically rely on static detection mechanisms that search for recognizable signatures or patterns of malicious behavior. Attackers can overcome this challenge by encrypting or encoding their shellcode, making it arduous for EDR solutions to identify them based on static signatures.

Evading EDR Detection with Direct System Calls

One effective technique for evading EDR detection is the use of direct system calls. Rather than relying on the higher-level API functions, attackers can directly invoke the underlying syscalls to interact with the operating system. By bypassing the API functions, they can evade the hooks set up by EDR solutions and execute their malicious code without detection.

To exploit direct system calls, attackers must collect addresses of syscall functions during runtime. These addresses can be resolved from the ntdll.dll file, which will allow the attackers to call the syscalls directly without relying on the higher-level API functions.

By exploiting direct system calls, attackers can bypass EDR detection, and they can carry out various malicious activities such as process injection, memory allocation, and file manipulation without setting off hooks in EDR solutions.

Implementing Userland Hooking for EDR Evasion

- Usually, EDR solutions employ userland hooking in order to monitor and intercept API calls made by programs. Userland hooking involves placing hooks on specific API functions to redirect the execution flow and analyze the behavior of the program.
- By exploiting direct system calls, attackers can bypass EDR detection and carry out various malicious activities in the form of process injection, memory allocation and file manipulation without setting off hooks in EDR solutions.
- Using specialized hooking methods, attackers can alter the behavior of API calls in a way that avoids detection by Endpoint Detection and Response (EDR) systems.
- One way of using userland hooking for EDR evasion is to develop a custom DLL that applies hooks on specific API functions. This DLL can be injected into the target process, effectively bypassing the EDR's hooks and allowing the attacker to modify the execution flow of the program.
- By utilizing userland hooking techniques, attackers can evade EDR detection and execute their malicious code without triggering alarms or alerts.

Understanding Kernel Mode Hooking

While userland hooking is a commonly employed technique for EDR evasion, some advanced EDR solutions may also utilize kernel mode hooking. Kernel mode hooking involves placing hooks in the kernel, which allows for more comprehensive monitoring and analysis of system activities.

However, kernel mode hooking is more complex and challenging to implement compared to userland hooking. Placing hooks in the kernel can lead to stability issues and unexpected behavior, making it less common in modern EDR solutions.

Due to the complexity and potential risks associated with kernel mode hooking, attackers often focus on evading userland hooking techniques to bypass EDR detection.

Evading EDR Detection with APC Queue Injection

One of the most effective techniques for process injection and EDR evasion is APC (Asynchronous Procedure Call) Queue Injection. APC Queue Injection takes advantage of the APC mechanism in Windows, which allows for the asynchronous execution of code within a thread's context.

By queuing an APC to a target thread, attackers can inject their malicious code into the target process without triggering EDR detection. APC Queue Injection is particularly useful for executing shellcode in local processes and, with certain modifications, can also be used for remote process injection.

To perform APC Queue Injection, attackers spray APCs across multiple threads of the target process. By injecting their shellcode into the APC queue of each thread, attackers increase the chances of successful execution without triggering EDR detection.

APC Queue Injection can be further enhanced by targeting processes with a large number of threads or processes that frequently perform I/O operations. This increases the likelihood of successful injection and execution of the malicious code.

Implementing a Shellcode Loader for EDR Evasion

To effectively evade EDR detection, attackers often utilize a shellcode loader to load and execute their malicious code in the target process. A shellcode loader is a program or script that facilitates the injection and execution of shellcode, bypassing various security mechanisms, including EDR solutions.

The shellcode loader typically follows a series of steps to ensure successful execution:

1) Allocate Memory in the Target Process

The shellcode loader allocates memory in the target process to store the shellcode. This memory is usually allocated with specific permissions to allow for code execution.

2) Write the Shellcode to Allocated Memory

The Shellcode Loader writes encrypted or encoded shellcode into memory space in the target process. This step ensures that the shellcode remains undetected by static signature-based detection mechanisms.

3) Modify Memory Protection

The shellcode loader must modify memory protection so as to allow for code execution in order to execute the shellcode. This step ensures that the shellcode can be executed without triggering memory protection mechanisms.

4) Execute the Shellcode

Finally, the shellcode loader initiates the execution of the shellcode in its target process. By bypassing EDR hooks and evading detection, the shellcode can perform its intended malicious actions.

By implementing a robust and efficient shellcode loader, attackers can successfully evade EDR detection and execute their malicious code within the target process.

Limitations and Caveats of EDR Evasion Techniques

While EDR evasion techniques can be effective in bypassing detection and executing malicious code, they also have certain limitations and caveats that attackers must consider.

One limitation is the reliance on specific versions of the Windows operating system. Often, EDR evasion techniques depend upon specific syscalls or API functions. Moreover, they differ across different Windows versions, service packs, or patch levels. Attackers must ensure that their techniques are compatible with the target system to achieve successful evasion.

One drawback of basic EDR solutions is their inferiority to sophisticated EDR solutions with more advanced detection mechanisms that may detect evasion techniques; more sophisticated solutions have mechanisms in place that detect and prevent EDR evasion altogether.

Attackers must also consider how their attacks could potentially impact system stability and user experience. Evasion techniques like injecting shellcode into multiple threads or processes could consume significant system resources, potentially leading to system instability or performance issues.

Attackers must carefully assess the risks and benefits associated with each evasion technique they consider using, adapting their approach based on specific circumstances and target environments.

TL;DR

Responding to EDR evasion techniques remains an ongoing challenge in cybersecurity. EDR solutions are intended to detect and stop malicious activities; however, attackers continue devising novel tactics for bypassing detection systems so as to deploy their malicious code without setting off alarms in EDR software.

To effectively address this ongoing challenge, both security professionals and attackers must have an in-depth knowledge of EDR solutions, Windows syscalls, and EDR evasion techniques. With this understanding in hand, defenders can create countermeasures tailored specifically to EDR evasion attempts while attackers adapt and refine their methods as necessary – creating an ongoing cycle of adaptation that provides both parties a competitive edge in cybersecurity.

As the cybersecurity landscape continues to shift, EDR evasion techniques will remain a top priority for both defenders and attackers alike. Staying vigilant, developing advanced detection and prevention mechanisms, and being vigilant with our own actions are paramount for mitigating risks associated with EDR evasion techniques while effectively combating sophisticated cyber threats.

Redfox Security is a diverse network of expert security consultants with a global mindset and a collaborative culture. If you are looking to improve your organization's security posture, **contact us** today to discuss your security testing needs. Our team of security professionals can help you **identify vulnerabilities and weaknesses in your systems and provide recommendations to remediate them**.

"Join us on our journey of growth and development by signing up for our comprehensive **courses**."

PreviousHardware Hacking for IoT Devices – Offensive IoT Exploitation
NextDecoding JTAG: Unveiling and Validating Headers on Your Device

Recent Blog

September 09, 2025
Is APK Decompilation Legal? What You Need To Know
September 06, 2025
When Hackers Hit the Road: The Jaguar Land Rover Cyberattack
September 05, 2025
This Is the Hacker's Swiss Army Knife. Have You Heard About It?