Linux Incident Reponse - Introduction to rootkits

Taz Wake | Halkyn Consulting Ltd | Originally published 2023-11-20

Rooting out rootkits

Rootkits are an ongoing problem in cybersecurity, particularly within the Linux ecosystem. These surreptitious entities pose a considerable threat by affording unauthorised access and perpetuating control over compromised systems. In this comprehensive exploration, we will delve into the intricacies of Linux rootkits, unravelling their nature, the diverse types they encompass, their intricate construction techniques, and their historical evolution. Additionally, we will scrutinise the steps that malicious actors take to deploy these covert threats and provide meticulous guidance on detection, investigation, and removal.

Understanding Rootkits: The Covert Menace

At its core, a rootkit is a complex assembly of malicious code and tools designed to operate covertly, obscuring the attacker's presence within a compromised system. These applications derive their names from the 'root' user, the omnipotent administrator account in Unix-like operating systems such as Linux. Rootkits are especially menacing owing to their ability to infiltrate the deepest recesses of the operating system, rendering them extraordinarily challenging to detect, identify and ultimately eliminate.

Types of Linux Rootkits: A Multifaceted Threat

Rootkits manifest in a variety of forms, each meticulously tailored to distinct objectives and methodologies:

1. **Loadable Kernel Module Rootkits (LKM)**: These rootkits subvert Linux kernel functionality by injecting malicious code directly into the kernel as loadable kernel modules. LKMs empower malevolent actors to manipulate the kernel's inner workings and execute nefarious code within the kernel space, operating with escalated privileges.

2. **Library/Shared Object Rootkits**: Functioning at a higher level, these rootkits replace system library files with malevolent versions. They intercept system calls and tamper with system behaviour by targeting dynamic link libraries (DLLs) or shared object files (.so), thus maintaining their covert presence.

3. **Application-Level Rootkits**: These rootkits compromise individual user-space applications, enabling them to intercept system calls made by these applications. This manipulation facilitates surreptitious alteration of application behaviour without directly altering core system components.

Constructing a Linux Rootkit: Techniques and Placement

Crafting and deploying rootkits on Linux systems entails a complex tapestry of techniques, each devised to obscure their presence and ensure enduring access. The following are some of the key methods employed by malevolent actors:

1. **File System Manipulation**: Rootkits often tamper with critical system files and directories, either by replacing legitimate binaries with malicious counterparts or by introducing entirely new files into the system. Techniques encompass file overwrites, symbolic link manipulation, and directory cloaking.

2. **Kernel Hooking**: Loadable Kernel Module (LKM) rootkits clandestinely attach themselves to Linux kernel function pointers, diverting system calls to malicious code. This subversion allows them to exert control over system operations while remaining largely undetected.

3. **Process Injection**: Some rootkits inject malicious code directly into running processes, thereby enabling the execution of arbitrary code within the context of those processes. This tactic is employed to evade detection mechanisms and facilitate nefarious activities without raising alarms.

4. **Device Driver Manipulation**: Rootkits may compromise device drivers to gain access to hardware interfaces, establishing covert communication channels with external malicious command and control servers.

Historical Linux Rootkits: Tracing the Evolution

A meticulous examination of historical Linux rootkits furnishes invaluable insights into their evolution and operational intricacies. Notable examples include the '*Knark*' rootkit, an early LKM rootkit. Similarly, the '*Ramen*' rootkit gained notoriety for targeting Red Hat Linux systems. '*Duqu*', another infamous rootkit, stood out for its sophistication and exploitation of zero-day vulnerabilities to infiltrate systems surreptitiously.

Knark

The Knark rootkit was one of the early examples of Loadable Kernel Module (LKM) rootkits, designed to operate on Linux systems. The Knark rootkit operated by exploiting the Linux kernel's capability to load additional modules dynamically. Here's an outline of how it typically worked:

1. **Exploitation and Kernel Module Loading:** The attacker gained initial access to the target Linux system, often through a vulnerability or misconfiguration that allowed them to execute arbitrary code with sufficient privileges. Once inside the system, the attacker would load the Knark rootkit as a kernel module. This action was typically achieved using the or command, allowing the attacker to inject their malicious code into the running kernel.
2. **Kernel Function Hooking:** After loading the Knark rootkit module into the kernel, it would proceed to hook various kernel functions and data structures. The rootkit aimed to intercept and modify system calls and kernel operations to hide its presence

and control the compromised system. For instance, it might intercept file system-related functions to hide specific files or processes from being listed.

3. **Persistence:** Knark was designed to maintain persistence on the compromised system, ensuring that it would be loaded into the kernel each time the system booted. This persistence mechanism allowed the rootkit to maintain control over the system even after a reboot.
4. **Stealth and Evasion:** The Knark rootkit used various techniques to hide its presence and evade detection. This included altering data structures to hide processes, files, or network connections associated with the rootkit. Additionally, it often disabled or manipulated system logging mechanisms to cover its tracks.
5. **Backdoor Access:** Knark often provided a backdoor into the compromised system, allowing attackers to execute arbitrary code with elevated privileges. This backdoor access could be used for various malicious purposes, such as further exploitation, data theft, or launching additional attacks.

Ramen

The Ramen rootkit was a specific type of Linux rootkit that primarily targeted Red Hat Linux systems. It emerged in the early 2000s and gained some notoriety due to its ability to compromise and maintain unauthorized access to Linux servers. The Ramen rootkit employed several techniques to compromise a Linux system and maintain persistence:

1. **Exploitation:** The initial compromise often occurred through known vulnerabilities or misconfigurations in the target system. Attackers would exploit these weaknesses to gain unauthorized access to the system.
2. **Backdoor Installation:** Once inside the system, the attacker would install the Ramen rootkit. The rootkit typically included a backdoor component, which allowed the attacker to access the compromised system remotely and execute commands with elevated privileges.
3. **Replacing System Binaries:** One of the distinctive features of the Ramen rootkit was its practice of replacing legitimate system binaries with malicious versions. These malicious binaries would often have names similar to the original ones, making it difficult to detect their presence. When system administrators or users execute these commands, they unwittingly execute the malicious code.
4. **Kernel-Level Manipulation:** Like many rootkits, the Ramen rootkit operated at the kernel level. It employed techniques to manipulate system calls and kernel data structures to hide its processes, network connections, and files. This made it challenging to identify the rootkit's presence through standard system monitoring tools.
5. **Persistence Mechanisms:** The Ramen rootkit was designed to maintain persistence on the compromised system. It typically achieved this by modifying system startup scripts or configuration files to ensure that the rootkit was loaded into memory each time the system booted.

6. **Concealing Activity:** Ramen also took steps to conceal its activities. It often tampered with system logs or monitoring tools to hide any suspicious entries or events related to its presence.
7. **Payload Delivery:** While the primary goal of the Ramen rootkit was to maintain unauthorized access and control, it could also serve as a platform for launching additional attacks or downloading and executing other malicious payloads.

Duqu

Duqu, often referred to as a sophisticated cyber espionage tool, was not precisely a rootkit but rather a complex malware framework that shared some similarities with Stuxnet. Duqu's primary purpose was to gather intelligence and facilitate targeted attacks on specific organizations. Some threat intelligence vendors believe Duqu is from the same threat actor group as Stuxnet.

Here's an overview of how Duqu operated:

1. **Initial Compromise:** Duqu gained access to a target system through various means, typically leveraging spear-phishing emails or exploiting vulnerabilities in software. The attackers often crafted convincing phishing emails with malicious attachments that, when opened, would execute the initial payload on the victim's machine.
2. **Payload Delivery:** Upon successful execution, Duqu's initial payload would drop and install additional components on the compromised system. These components included a dropper, a loader, and the core Duqu malware.
3. **Kernel Exploitation:** While not a rootkit in the traditional sense, Duqu did use a zero-day Windows kernel vulnerability to elevate its privileges on the system. This allowed it to execute code with higher privileges, making it more challenging to detect and remove.
4. **Data Gathering:** Duqu's primary purpose was to gather intelligence and exfiltrate sensitive data. It had sophisticated modules for data collection, including keylogging, screen capturing, and network traffic interception. It stealthily monitored and recorded user activity and network communications.
5. **Communication and Exfiltration:** Duqu communicated with command and control (C&C) servers controlled by the attackers to receive instructions and transmit stolen data. It used encryption and techniques to hide its network traffic to avoid detection.
6. **Persistence:** Duqu employed various techniques to maintain persistence on the compromised system. It would modify system settings, create registry entries, and hide itself within the system, making it challenging to detect and remove.
7. **Stealth and Evasion:** Duqu was known for its advanced evasion techniques. It would actively try to avoid detection by security software and forensic tools. This included encrypting its payload and using obfuscation techniques to make analysis more difficult.
8. **Modularity and Customizability:** Duqu was designed to be modular and highly customizable, allowing the attackers to adapt it to specific targets and gather

tailored intelligence. The attackers could remotely update and modify the malware's functionality.

It's essential to note that Duqu was attributed to a highly sophisticated threat actor, and its code was meticulously crafted to avoid detection and analysis. It shared similarities with Stuxnet in terms of its complexity and the zero-day vulnerability it exploited.

Building and Deploying a Linux Rootkit: The Operational Landscape

The construction and deployment of a Linux rootkit, though inherently malicious, entail a multifaceted and discreet operational landscape. Malicious actors typically progress through the following phases:

1. **Exploiting Vulnerabilities**: Attackers initiate the process by identifying and exploiting vulnerabilities in the target system, securing initial access through techniques such as remote code execution or privilege escalation.

2. **Privilege Escalation**: To assert control over the compromised system, attackers seek to elevate their privileges to root level. This often involves exploiting known vulnerabilities or misconfigurations that grant root access.

3. **Payload Deployment**: Attackers implant the rootkit payload into the system, ensuring its persistence across reboots. This may entail the manipulation of system binaries or injection of nefarious code into legitimate processes.

4. **Obfuscation Techniques**: To maintain covert presence, rootkits employ a multitude of obfuscation techniques, including process hiding, kernel hooking, and file system manipulation.

Detection, Investigation, and Removal

The battle against rootkits necessitates advanced techniques and dedicated tools. Among the arsenal available, consider the following tools and their command-line arguments:

1. **chkrootkit** and **rkhunter**: These rootkit detection tools are invaluable in identifying known rootkit signatures and anomalies. Execute **chkrootkit -q** or **rkhunter -c** for a system-wide scan.

2. AIDE (Advanced Intrusion Detection Environment): AIDE performs integrity checks on system files and directories. Run **aide --check** to verify file integrity.

Detection and mitigation strategies encompass comprehensive file system analysis, memory examination for irregularities, system integrity checks, and forensic analysis. These techniques ensure thorough scrutiny of digital evidence, aiding in understanding the rootkit's behaviour and its impact on the Linux system.

In conclusion, Linux rootkits persist as an evolving and formidable adversary within the realm of cybersecurity. A deep understanding of their types, construction techniques, historical context, and detection/removal methodologies is imperative for safeguarding Linux-based systems against these elusive threats. While rootkits may be discreet, the

knowledge and tools at the disposal of defenders enable a more vigilant stance in identifying and mitigating their impact on Linux systems.

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858