Exploiting Windows Driver Signature Enforcement Loopholes for Malware Persistence

Felix

Introduction **2**
1. Body of Research **2**
1.1 Windows Driver Signature Enforcement 2
Loophole Exploitation: 3
Methods: 3
Impact of the Loophole: 4
Microsoft's Response: 4
1.2 Exploitation Techniques 5
a) Vulnerable Driver Abuse: 5
b) Certificate Theft: 5
c) Timestamp Abuse: 7
d) Exploiting Legacy Systems: 7
1.3 Malware Persistence Methods 7
a) Boot-time Loading: 7
b) Registry Modifications: 8
c) Hooking System Calls: 8
d) Kernel Patching: 8
1.4 Impact and Risks 9
a) Rootkit Installation: 9
b) Data Exfiltration: 9
c) System Control: 9
d) Antivirus Evasion: 9
e) Persistent Access: 9
1.5 Mitigation Strategies 9
Conclusion 10
References 10

Introduction

The security of operating systems is of critical concern in our digital age. Windows, being one of the most used operating systems, is a prime target for cybercriminals seeking to exploit vulnerabilities for malicious purposes. One area of particular concern is the exploitation of Windows driver signature enforcement loopholes, which can allow attackers to gain persistent access to a system at the kernel level.

This report examines the techniques used by malicious actors to bypass Windows driver signature enforcement, the methods employed to achieve malware persistence, and the potential impact of such attacks. Additionally, we will explore mitigation strategies and best practices for defending against these threats.

1. Body of Research

1.1 Windows Driver Signature Enforcement

Windows driver signature enforcement is a critical security mechanism designed to ensure that only drivers signed by trusted authorities can be loaded into the Windows kernel. This process aims to prevent unauthorized or malicious code from gaining low-level access to the system, which could lead to severe security breaches.

The driver signing process involves several steps:
1. Developers create a driver and submit it to Microsoft for testing and certification.
2. Microsoft verifies the driver's compatibility and security.
3. If approved, Microsoft digitally signs the driver.
4. The signed driver can then be distributed and installed on Windows systems.

However, this process has evolved over time, with Microsoft implementing stricter policies to enhance security:

- Windows Vista introduced the requirement for all kernel-mode drivers to be digitally signed by a certificate authority (CA).
- Windows 10 version 1607 further tightened security by requiring all kernel drivers to be signed through Microsoft's own Developer Program.

Despite these measures, cybercriminals have discovered and exploited loopholes in the signature enforcement mechanism, allowing them to load unsigned or maliciously signed drivers.

Loophole Exploitation:

The primary loophole being exploited stems from exceptions Microsoft implemented to accommodate existing drivers during the transition to stricter policies. Specifically, one exception allows drivers signed with a valid user certificate before July 29, 2015, to be accepted if the certificate was issued by a certificate authority trusted in Windows.

Methods:

Timestamp Manipulation: Hackers developed methods to sign new drivers and alter the signature timestamp, making it appear that the certificate was signed before July 29, 2015.

Use of Expired Certificates: Windows accepts the installation of drivers signed with expired code-signing certificates. This allows attackers to use old, expired certificates that meet the pre-July 29, 2015 requirement.

Open-Source Tools: Tools like HookSignTool and FuckCertVerifyTimeValidity have been developed and made publicly available. These tools can forge signature timestamps, bypassing Windows' verification process.

Certificate Theft and Misuse: Attackers have stolen or purchased legitimate code-signing certificates issued before the cutoff date. For example, the LAPSUS$ hacking group leaked two Nvidia certificates (expired since 2014 and 2018) that were subsequently used to sign malware.

API Hooking: The aforementioned tools use the Microsoft Detours library to hook into the Windows API. They intercept calls to the CertVerifyTimeValidity function, redirecting them to a modified version that allows custom timestamps.

Impact of the Loophole:

This vulnerability has been exploited for various malicious purposes:

Malware Persistence:
Attackers can deploy highly persistent malware that operates at the kernel level, making it extremely difficult to detect and remove.

Game Cheating:
The loophole has been used to defeat anti-cheating and digital rights management (DRM) features in games.

Browser Hijacking:
Malware like RedDriver uses this technique to hijack browser traffic through a driver that interacts with the Windows Filtering Platform (WFP).

Evasion of Security Controls:
Signed malicious drivers can bypass endpoint detection systems and manipulate both system and user-mode processes.

Microsoft's Response:

In light of these exploits, Microsoft has taken several steps:
Advisory Publication:
Microsoft released an advisory acknowledging the issue.

Driver Blacklisting:
Updates have been released to blacklist reported malicious drivers.

Certificate Blacklisting:
Signing certificates used for creating malicious drivers have been blacklisted.

Enhanced Detection:
Microsoft Defender has been updated with new detections for these threats.

Account Suspensions: Accounts submitting malicious drivers to the Microsoft Partner Center have been suspended.

1.2 Exploitation Techniques

Several techniques have been observed in the wild for exploiting Windows driver signature enforcement:

a) Vulnerable Driver Abuse:

Attackers may use legitimately signed but vulnerable drivers to load their malicious code. These vulnerable drivers often have excessive privileges that can be exploited to execute arbitrary code in kernel mode.
Example:

```
// Pseudo-code for exploiting a vulnerable driver
HANDLE hDevice = CreateFile("\\\\.\\VulnerableDriver", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
if (hDevice != INVALID_HANDLE_VALUE) {
    DWORD bytesReturned;
    char payload[] = "malicious_code_here";
    DeviceIoControl(hDevice, IOCTL_VULNERABLE_FUNCTION, payload, sizeof(payload), NULL, 0, &bytesReturned, NULL);
    CloseHandle(hDevice);
}
```

CreateFile: Opens a handle to the vulnerable driver using the CreateFile function. The driver is identified by the path \\.\VulnerableDriver. The handle allows read and write access.
DeviceIoControl: Sends an I/O control code (IOCTL_VULNERABLE_FUNCTION) to the driver. This function is used to communicate with the driver and pass the payload (malicious_code_here).
Payload: Contains the malicious code intended to exploit the vulnerability in the driver.
CloseHandle: Closes the handle to the driver after the operation is complete.

b) Certificate Theft:

Attackers may steal or purchase legitimate code-signing certificates to sign their malicious drivers, bypassing the signature check.
The theft or misuse of legitimate code-signing certificates is a significant concern in cybersecurity, particularly when it comes to driver signing. The following are examples of incidents involving Driver signatures:

Comodo Incident (2011):
In 2011, a hacker compromised a Registration Authority (RA) of Comodo, a major certificate authority. The attacker managed to issue nine fraudulent certificates for domains including Google, Yahoo, and Microsoft. While this incident didn't directly involve driver signing, it highlighted the vulnerabilities in the certificate issuance process that could be exploited for malicious purposes, including driver signing.
Source: Independent Iranian Hacker Claims Responsibility for Comodo Hack | WIRED

DigiNotar Breach (2011):
In 2011, DigiNotar, a Dutch certificate authority, was breached. The attackers issued hundreds of fraudulent certificates, including one for *.google.com. This incident led to the complete collapse of DigiNotar and highlighted how stolen certificates could be used to sign malicious software, potentially including drivers.
Source: One year after DigiNotar breach, Fox-IT details extent of compromise – Computerworld

Stuxnet Worm (2010):
While not specifically about driver signing, the Stuxnet worm used stolen certificates from Realtek Semiconductor and JMicron Technology to sign its driver components. This allowed the malware to appear legitimate and bypass Windows driver signing requirements.
Source: What we really know about the Stuxnet worm | InfoWorld

eSentire Threat Intelligence (2022):
In 2022, eSentire reported on a malware campaign where threat actors were using stolen Nvidia code-signing certificates to sign malicious Windows drivers and loader components. These signed drivers were then used to bypass security controls and deploy Cobalt Strike beacons.
Source: eSentire | eSentire Threat Intelligence Malware Analysis: Resident…

SolarWinds Supply Chain Attack (2020):
While not specifically about driver signing, the SolarWinds incident demonstrated how attackers could compromise a software supply chain to distribute maliciously signed updates. This technique could potentially be applied to driver updates as well.
Source: SolarWinds Supply Chain Attack Uses SUNBURST Backdoor | Google Cloud Blog

Dark Web Certificate Markets:
There have been reports of underground markets where stolen or fraudulently obtained code-signing certificates are sold. These certificates can be used by attackers to sign malicious drivers. Prices for these certificates can range from a few hundred to several thousand dollars, depending on the perceived legitimacy of the certificate.

Example of how such a market listing might appear:
CopyFOR SALE: Extended Validation (EV) Code Signing Certificate
Issuer: [Redacted Major CA]
Valid until: 2025-12-31
Price: $3,000 BTC
Contact: [Encrypted messaging app handle]

Malware-as-a-Service Operations:
Some sophisticated cybercriminal groups operate Malware-as-a-Service platforms where they provide pre-signed malicious drivers to their customers. These drivers are often signed with stolen certificates, allowing less technically proficient attackers to deploy kernel-mode malware.

c) Timestamp Abuse:

By manipulating the timestamp of a driver, attackers can exploit the fact that Windows may accept drivers signed before a certain date, even if the signing certificate has been revoked.

Windows Policy: Microsoft implemented a policy that allows drivers signed before July 29, 2015, to be accepted if they were signed with a valid certificate from a trusted CA.
Timestamp Manipulation: Attackers use tools like HookSignTool and FuckCertVerifyTimeValidity to manipulate the timestamp of newly created malicious drivers.
API Hooking: These tools hook into the Windows API, specifically targeting the CertVerifyTimeValidity function.
Custom Timestamp Injection: By intercepting calls to CertVerifyTimeValidity, attackers can inject a custom timestamp that predates July 29, 2015.
Bypassing Revocation Checks: Even if the certificate used for signing has been revoked, Windows may still accept the driver if it believes it was signed before the cutoff date.

d) Exploiting Legacy Systems:

Older versions of Windows or systems with outdated security patches may have weaker driver signature enforcement, making them more susceptible to attack.

Outdated Policies: Older Windows versions may have less stringent driver signing requirements.
Unpatched Vulnerabilities: Systems that haven't received recent security updates may have known vulnerabilities in their driver signing verification process.
Compatibility Mode: Some legacy systems may run in compatibility modes that relax certain security checks to support older software.
Limited Hardware Support: Older systems may not support advanced security features like Secure Boot or Hypervisor-protected Code Integrity (HVCI).

1.3 Malware Persistence Methods

Once a malicious driver is loaded, attackers can use various methods to maintain persistence:

a) Boot-time Loading:

Malicious drivers can be configured to load early in the boot process, ensuring they are active before many security mechanisms initialize.
Early Load Drivers: Malicious drivers can be configured as "boot-start" drivers, which load before the Windows kernel fully initializes.
Manipulating Boot Configuration Data (BCD): Attackers may modify the BCD to ensure their driver loads early in the boot process.

Firmware Persistence: Advanced attackers might even implant their malicious code in the system's UEFI firmware, ensuring it loads before the OS.

b) Registry Modifications:

Attackers may modify registry keys to ensure their malicious drivers are loaded automatically on system startup.
Using Registry Run Keys: Attackers can add their driver to various run keys to ensure it loads at startup.
Modifying Existing Services: Instead of creating new entries, malicious code might modify existing legitimate service entries to point to the malicious driver.

Example:

```
# PowerShell command to add a malicious driver to the registry
New-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\MaliciousDriver" -Name
"ImagePath" -Value "C:\Windows\System32\drivers\malicious.sys"
-PropertyType String
```

c) Hooking System Calls:

By intercepting and modifying system calls, malicious drivers can hide their presence and manipulate system behavior.
Inline Hooking: Attackers may modify the first few bytes of a system call to redirect execution to their malicious code.
SSDT Hooking: By modifying the System Service Descriptor Table, attackers can intercept and modify system calls.
IDT Hooking: Interception of interrupt handlers can allow malware to capture and modify system behavior at a low level.

d) Kernel Patching:

Advanced malware may directly modify kernel memory to alter system functionality and evade detection.
Direct Kernel Object Manipulation (DKOM): Attackers can directly modify kernel objects to hide processes, files, or registry keys.
Patching Kernel Functions: Critical kernel functions can be modified to alter system behavior or bypass security checks.
Memory Paging Attacks: By manipulating page tables, attackers can redirect code execution or hide malicious code.

1.4 Impact and Risks

The successful exploitation of driver signature enforcement loopholes can have severe consequences:

a) Rootkit Installation:

Attackers can install rootkits that operate at the kernel level, making them extremely difficult to detect and remove.

b) Data Exfiltration:

Malicious drivers can intercept and steal sensitive data directly from kernel memory.

c) System Control:

Attackers may gain complete control over the compromised system, potentially using it as a launchpad for further attacks.

d) Antivirus Evasion:

Kernel-level malware can disable or bypass antivirus software, leaving the system vulnerable to additional threats.

e) Persistent Access:

The ability to load malicious drivers provides attackers with a resilient method of maintaining access to compromised systems.

1.5 Mitigation Strategies

To defend against these threats, organizations and individuals should consider the following strategies:

1. Regular System Updates: Keep Windows and all drivers up to date with the latest security patches.

2. Use of Windows Defender Application Control (WDAC): Implement WDAC policies to restrict which drivers can be loaded based on their attributes and signatures.

3. Driver Block Rules: Utilize Windows Security Center to create and enforce driver block rules for known vulnerable or malicious drivers.

4. Hypervisor-Protected Code Integrity (HVCI): Enable HVCI on supported systems to provide additional protection against kernel-level attacks.

5. Monitoring and Logging: Implement robust logging and monitoring solutions to detect suspicious driver loading activities.

6. Least Privilege Principle: Limit user and application privileges to reduce the potential impact of successful exploits.

7. Security Information and Event Management (SIEM): Use SIEM tools to correlate events and identify potential driver-based attacks.

8. User Education: Train users to be cautious when installing drivers from untrusted sources.

Conclusion

The exploitation of Windows driver signature enforcement loopholes represents a significant and evolving threat to system security. As attackers continue to develop sophisticated techniques to bypass these protections, it is crucial for organizations and security professionals to stay informed and implement comprehensive defense strategies.

By understanding the methods used to exploit driver signature enforcement and the ways in which malware can persist using these techniques, we can better prepare our systems and networks to resist such attacks. The mitigation strategies outlined in this report provide a starting point for enhancing system security, but they must be part of a larger, holistic approach to cybersecurity that includes continuous monitoring, rapid incident response, and ongoing security education.

As the threat landscape continues to evolve, so too must our defensive measures. Staying vigilant and adapting our security practices will be key to protecting against the ever-changing tactics of cybercriminals seeking to exploit the core of our operating systems.

References

1. Microsoft. (2021). Driver signing policy.
https://docs.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-policy--windows-vista-and-later-

2. Eclypsium. (2022). Bring Your Own Vulnerable Driver.
https://therecord.media/sneaky-malware-blacklotus-can-bypass-important-windows-boot-functions

3. ESET Research. (2023). BlackLotus UEFI bootkit: Myth confirmed.
https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed/

4. Hackers exploit Windows driver signature enforcement. (2023)
https://www.csoonline.com/article/645585/hackers-exploit-windows-driver-signature-enforcement-loophole-for-malware-persistence.html

5. Microsoft. (2023). Windows Defender Application Control.
https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/windows-defender-application-control

6. MITRE ATT&CK. (2023). Boot or Logon Autostart Execution: Driver.
https://attack.mitre.org/techniques/T1547/008/

7. Mandiant. (2022). Bring Your Own Vulnerable Driver: Exploiting a Vulnerable Signed Driver.
https://www.mandiant.com/resources?f%5B0%5D=layout%3Aarticle_blog

8. Microsoft. (2023). Hypervisor-Protected Code Integrity (HVCI).
https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-hvci-enablement

9. Rapid7 annual Vulnerability Intelligence Report
https://www.rapid7.com/globalassets/_pdfs/research/rapid7_2024_attack_intelligence_report.pdf