Today, we talk about circumventing Endpoint Detection & Response (EDR) systems

Agenda

- How EDRs work
- Effective techniques to circumvent them
- How to compensate for EDR protection gaps

Related work

- We are not the first to look at EDR evasion. Plenty of information is available online, including on the techniques presented herein
- Check out this paper for a summary and references: www.mdpi.com/2624-800X/1/3/21
Nice to meet you :)
We run a small EDR test lab

### Background
- SRLabs regularly conducts red team exercises
- The prepare and test EDR evasion for these exercises, we run our own mini EDR test lab
- Each EDR is running in an isolated virtual machine
- All EDR features are enabled with one exception: Cloud uploads
- The results shared in this presentation were generated in the test lab in August 2022

### EDR Test Lab

Inject malware samples and see if they get detected

- **EDR: MS Defender for Endpoint**
  - Win 10 VM

- **EDR: Symantec EDR**
  - Win 10 VM

- **EDR: Sentinel One**
  - Win 10 VM

- **Antivirus (as reference)**
  - Win 10 VM

- **VMWare Hypervisor**
Agenda

How EDRs work

- Effective techniques to circumvent them
- How to compensate for EDR protection gaps
EDRs conduct three types of analyses to detect endpoint detection and abuse

<table>
<thead>
<tr>
<th>A. Static analysis</th>
<th>B. Dynamic analysis</th>
<th>C. Behavioral analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Extract information from binary</td>
<td>▪ Execute binary in a sandbox environment and observe it</td>
<td>▪ Observe the binary as its executing on the computer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Hook into important functions/syscalls to learn in realtime about behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Analyzes not only the binary, but everything that surrounds the execution</td>
</tr>
</tbody>
</table>

**Looks for**

**Common patterns:**
- Known malicious strings
- Threat actor IP or domains
- Malware binary hashes

**Malicious behavior in sandbox:**
- Network connections
- Registry changes
- Memory Access
- File creation/deletion

**Malicious behavior when running without sandbox:**
- User actions
- System calls
- Commands executed in the command line
- Which process is executing the code

---

**Antivirus tools** are based on static and dynamic analysis

**EDRs add behavioral analysis** — our focus today
A. Static Analysis – your good ol’ antivirus engine

**Static analysis** detects malware through known indicators of compromise

User downloads file

**EDR or Antivirus**

- Extract static values
  - String
  - Domain
  - Assembly snippet
  - IP Address
  - File Hash
  - ...

- Compare against database of known malicious values\(^1\)

- Do nothing if legit

- Delete file if malicious

Database with Indicators of Compromise (IOCs)

**Static analysis evasion** allows malware to stay undetected by avoiding static signatures, using two techniques

- **Obfuscation**
  - Change function and variable names
  - Applying encoding mechanisms such as Caesar ciphers

- **Encryption**
  - Apply encryption to potentially-flagged code parts (“packer”/ “loader”)
  - Then obfuscate the decryption routine to avoid additional signatures

---

\(^1\) Checking for exact values can also be augmented by heuristics that are applied on the collected data
B. Dynamic Analysis – controlled detonation in a sandbox

**Dynamic analysis** observes malware in sandbox

- **User downloads file**
  - `clickme.exe`

- **EDR or Antivirus**
  - Execute file in virtual machine (“sandbox”) and observe behavior
    - Network Connections
    - Registry Changes
    - Memory access
    - File Creation

  ![Database](image)
  - Database with Indicators of Compromise (IOCs)

- **Compare against database of known malicious values**
  - **Do nothing if legit**
  - **Delete file if malicious**

**Dynamic analysis evasion** tries to detect the sandbox and stop the malware before being detected

<table>
<thead>
<tr>
<th>Check</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of processors</td>
<td>Sandbox environments usually run with a limited number of processors</td>
</tr>
<tr>
<td>memory size</td>
<td>Sandbox environments usually do not have much RAM memory available</td>
</tr>
<tr>
<td>filename</td>
<td>Check if the malware name changed when bring copied into the sandbox</td>
</tr>
<tr>
<td>non-virtualized APIs</td>
<td>Some WinAPIs are not emulated by most sandboxes. For example, the return value of VirtualAllocExNuma() will be NULL</td>
</tr>
<tr>
<td>user/domain</td>
<td>For targeted attacks, the malware can check whether the targeted user account or domain name exists in the sandbox</td>
</tr>
<tr>
<td>Sleep</td>
<td>Delaying the execution of the malicious routine can help to exhaust the EDR engine</td>
</tr>
</tbody>
</table>

The more sandbox checks are used in parallel, the more suspicious the malware might appear.

[1] Checking for exact values can also be augmented by heuristics that are applied on the collected data.
C. Behavioral Analysis – playing with fire

Behavioral analysis closely monitors malware while it is executing on the actual computer

Malware starts, checks for sandbox, exits if found

Malware allocates memory

Decrypts payload and writes to memory

Launches payload as a new thread

Windows processes load dll’s to access OS features

kernel32.dll

VirtualAlloc

WriteProcessMemory

CreateThread

ntdll.dll

NtAllocateVirtualMemory

NtWriteVirtualMemory

NtCreateThread

EDR solutions typically overwrite the loaded dll’s in each processes’ memory, thereby diverting execution of key functions[1]

Check and inspect if required

Correlate info

NtAllocateVirtualMemory

NtWriteVirtualMemory

NtCreateThread

User space

Kernel

EDR

Malware

[1] Different EDRs might apply different hooking methods and also choose other DLLs or functions to hook
Agenda

- How EDRs work
- Effective techniques to circumvent them
- How to compensate for EDR protection gaps
Evasion techniques can render EDRs ineffective – We discuss three options

We are finding out EDR effectiveness by testing different versions of our encrypted malware loader

1. Apply **base evasion** to all samples

   - **Evade Static Analysis**
     - Payload AES Encryption
   - **Evade Dynamic Analysis / Sandbox**
     - Sleep Timer
     - VirtualAllocExNuma()
     - Check If >2GB RAM

2. Experiment with different **Behavioral Analysis Evasion techniques**

   - **No evasion attempt** (just WinAPI calls)
   - **Evasion through Unhooking**
     - Malware uses code fragments in kernel dll’s without calling the hooked functions in those dll’s
   - **Evasion through Direct System Calls**
     - Malware circumvents hook in system dll’s by directly sys-calling into kernel
   - **Evasion through Indirect System Calls**
     - Malware overwrites EDR hooks before executing payload

1. Details on next slides
Evasion technique 1 – Unhook EDR by overwriting ntdll.dll with a clean version

Malware

Normal

- EDR hooks into NTDLL to analyze and correlate the data
- Common API calls go through NTDLL

Kernel

EDR

EDR catches syscall

ntdll.dll with hooks

NtAllocateVirtualMemory

kernel32

VirtualAlloc

NtAllocateVirtualMemory
Evasion technique 1 – Unhook EDR by overwriting ntdll.dll with a clean version

Normal

- EDR hooks into NTDLL to analyze and correlate the data
- Common API calls go through NTDLL

```
kernel32 VirtualAlloc
ntdll.dll with hooks NtAllocateVirtualMemory
```

“Unhooking” the EDR

- Obtain original ntdll.dll without EDR hooks (e.g. read from disk)
- Overwrite ntdll.dll in own process memory with original one

```
kernel32 VirtualAlloc
ntdll.dll without hooks NtAllocateVirtualMemory
```

Different methods exist to obtain a “clean” ntdll.dll

1. Read ntdll.dll from disk
2. Start process in suspended state and copy it from there
3. Get creative!

EDR

```n
EDR catches syscall
```

Kernel

```n
NtAllocateVirtualMemory
```

EDR sees nothing

```n
NtAllocateVirtualMemory
```

Security Research Labs
Evasion technique 1 – Unhook EDR by overwriting ntdll.dll with a clean version

**Normal**
- EDR hooks into NTDLL to analyze and correlate the data
- Common API calls go through NTDLL

**“Unhooking” the EDR**
- Obtain original ntdll.dll without EDR hooks (e.g. read from disc)
- Overwrite ntdll.dll in own process memory with original one

---

**Might not work:**
- Accessing ntdll.dll from disk is often flagged by EDRs, as it is a common way of unhooking a process.
- The API calls to overwrite ntdll.dll are probably hooked as they reside within the hooked ntdll.dll

---

**EDR**
- EDR catches syscall
- EDR sees nothing

**Kernel**
- NtAllocateVirtualMemory

---

**Malware**
- NtAllocateVirtualMemory
- VirtualAlloc

---

Different methods exist to obtain a “clean” ntdll.dll
Evasion technique 2 – Avoid EDR hooks by directly calling kernel system calls

**Normal**
- EDR hooks into NTDLL to analyze and correlate the data
- Common API calls go through NTDLL

**Direct Syscalls**
- Implement own syscall in assembly
- Call syscall directly and bypass NTDLL hooks

**SSN (System Service Number)**
- It identifies which syscall executes
- The syscall number varies between Windows versions
- It can be obtained dynamically – conveniently automated by SysWhispers2
- Can trigger static analysis since the syscall assembly instruction might be flagged

---

Security Research Labs
Evasion technique 2 – Avoid EDR hooks by directly calling kernel system calls

### Malware

<table>
<thead>
<tr>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>- EDR hooks into NTDLL to analyze and correlate the data</td>
</tr>
<tr>
<td>- Common API calls go through NTDLL</td>
</tr>
</tbody>
</table>

### Direct Syscalls

<table>
<thead>
<tr>
<th>Direct Syscalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Implement own syscall in assembly</td>
</tr>
<tr>
<td>- Call syscall <strong>directly</strong> and bypass NTDLL hooks</td>
</tr>
</tbody>
</table>

### SSN (System Service Number)

- It identifies **which syscall** executes
- The syscall number varies between Windows versions
- It can be obtained

---

**Might not work:**

- **Having syscall assembly instructions compiled into an executable is unusual and can be flagged as suspicious / malicious**
- **Heads up: Only the loader evades the EDR. You need to be careful since the C2 malware might still use the hooked functions**

---

Security Research Labs
Evasion technique 3 – Further increase stealth through indirect system calling

Normal
- EDR hooks into NTDLL to analyze and correlate the data
- Common API calls go through NTDLL

Direct Syscalls
- Implement own syscall in assembly
- Call syscall directly and bypass NTDLL hooks

Indirect Syscalls
- Prepare syscall in assembly (as with direct syscalls)
- Then find a syscall instruction in ntdll.dll and jumps to that location

EDR catches syscall

EDR sees nothing

Import assembly

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

VirtualAlloc

kernel32

NtAllocateVirtualMemory

ntdll.dll

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

NtAllocateVirtualMemory10 proc
  mov r10, rcx
  mov eax, 18h
  syscall
  ret
NtAllocateVirtualMemory10 endp

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory

ntdll.dll

NtAllocateVirtualMemory
One more thing: You can boost any of the evasion techniques by hiding inside a .dll

- **.exe**
  - Is designed to run independently
  - Has its own memory space
  - Allows EDR to tightly observe execution of suspicious files, for example Internet downloads

- **.dll**
  - The Windows implementation of “shared libraries”
  - Need a host process loading them and shares memory space with the host process
  - Harder to follow suspicious downloads
The 3 simple injection techniques work surprisingly well against common EDR systems

Step 1: System Infection. We tested three different evasion techniques (and two base cases) against three leading EDR solutions, and one antivirus solution. All experiments were run in August 2022.

<table>
<thead>
<tr>
<th>No behavioral analysis or sandbox evasion</th>
<th>EDR1</th>
<th>EDR2</th>
<th>EDR3</th>
<th>AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>.exe</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
</tr>
<tr>
<td>.dll</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
</tr>
<tr>
<td>Only sandbox evasion</td>
<td>EDR1</td>
<td>EDR2</td>
<td>EDR3</td>
<td>AV</td>
</tr>
<tr>
<td>.exe</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
</tr>
<tr>
<td>.dll</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
</tr>
<tr>
<td>1 Unhooking</td>
<td>EDR1</td>
<td>EDR2</td>
<td>EDR3</td>
<td>AV</td>
</tr>
<tr>
<td>.exe</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
</tr>
<tr>
<td>.dll</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
</tr>
<tr>
<td>2 Direct syscalls</td>
<td>EDR1</td>
<td>EDR2</td>
<td>EDR3</td>
<td>AV</td>
</tr>
<tr>
<td>.exe</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
</tr>
<tr>
<td>.dll</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
</tr>
<tr>
<td>3 Indirect syscalls</td>
<td>EDR1</td>
<td>EDR2</td>
<td>EDR3</td>
<td>AV</td>
</tr>
<tr>
<td>.exe</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
</tr>
<tr>
<td>.dll</td>
<td>![Undetected]</td>
<td>![Detected]</td>
<td>![Undetected]</td>
<td>![Detected]</td>
</tr>
</tbody>
</table>

Cobalt Strike and Sliver are popular C&C tools to control infected computers

Base case. A malware that does not try to evade behavioral analysis

EDR evasion techniques. Three approaches to circumvent EDR behavioral analysis (as explained on previous slides)

Take aways.
- EDRs are more likely to trigger based on well-known abuse tools like Cobalt Strike, suggesting some level of fingerprinting
- Malware hiding in .dll’s is less likely to get detected by EDRs
- EDRs differ in their effectiveness, however some evasion techniques successfully circumvent most (all?) of them
- Our experiments so far only use well-known techniques. Better evasion is possible should it become necessary
The 3 simple injection techniques work surprisingly well against common EDR systems

### Step 1: System Infection.
We tested three different evasion techniques (and two base cases) against three leading EDR solutions, and one antivirus solution. All experiments were run in August 2022.

<table>
<thead>
<tr>
<th>EDR1</th>
<th>EDR2</th>
<th>EDR3</th>
<th>AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cobalt</td>
<td>Sliver</td>
<td>Cobalt</td>
<td>Sliver</td>
</tr>
<tr>
<td><strong>No behavioral analysis or sandbox evasion</strong></td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td><strong>Only sandbox evasion</strong></td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td><strong>Unhooking</strong></td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td><strong>Direct syscalls</strong></td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td><strong>Indirect syscalls</strong></td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
</tbody>
</table>

**Take aways.**
- EDRs are more likely to trigger based on well-known abuse tools like Cobalt Strike, suggesting some level of fingerprinting.
- Malware hiding in .dll’s is less likely to get detected by EDRs.
- EDRs differ in their effectiveness, however some evasion techniques successfully circumvent most (all?) of them.
- Our experiments so far only use well-known techniques. Better evasion is possible should it become necessary.

Cobalt Strike and Sliver are popular C&C tools to control infected computers.

Base case. A malware that does not try to evade behavioral analysis.

EDR evasion techniques. Three approaches to circumvent EDR behavioral analysis (as explained on previous slides).

Hiding inside a .dll really helps.
The 3 simple injection techniques work surprisingly well against common EDR systems

Step 1: System Infection. We tested three different evasion techniques (and two base cases) against three leading EDR solutions, and one antivirus solution. All experiments were run in August 2022.

<table>
<thead>
<tr>
<th>EDR1</th>
<th>EDR2</th>
<th>EDR3</th>
<th>AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cobalt</td>
<td>Sliver</td>
<td>Cobalt</td>
<td>Sliver</td>
</tr>
<tr>
<td>Cobalt</td>
<td>Sliver</td>
<td>Cobalt</td>
<td>Sliver</td>
</tr>
</tbody>
</table>

- **No behavioral analysis or sandbox evasion**
  - .exe
  - .dll

- **Only sandbox evasion**
  - .exe
  - .dll

- **1 Unhooking**
  - .exe
  - .dll

- **2 Direct syscalls**
  - .exe
  - .dll

- **3 Indirect syscalls**
  - .exe
  - .dll

<table>
<thead>
<tr>
<th>Detection</th>
<th>Undetected</th>
</tr>
</thead>
<tbody>
<tr>
<td>.exe</td>
<td></td>
</tr>
<tr>
<td>.dll</td>
<td></td>
</tr>
</tbody>
</table>

**Take aways.**
- EDRs are more likely to trigger based on well-known abuse tools like Cobalt Strike, suggesting some level of fingerprinting
- Malware hiding in .dll’s is less likely to get detected by EDRs
- EDRs differ in their effectiveness, however some evasion techniques successfully circumvent most (all?) of them
- Our experiments so far only use well-known techniques. Better evasion is possible should it become necessary

*Cobalt Strike and Sliver* are popular C&C tools to control infected computers.

Detection seems partly based on known Cobalt Strike behavior signatures.

A malware that does not try to evade behavioral analysis.

Three approaches to circumvent EDR behavioral analysis (as explained on previous slides).
The 3 simple injection techniques work surprisingly well against common EDR systems

Step 1: System Infection. We tested three different evasion techniques (and two base cases) against three leading EDR solutions, and one antivirus solution. All experiments were run in August 2022.

<table>
<thead>
<tr>
<th>EDR1</th>
<th>EDR2</th>
<th>EDR3</th>
<th>AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cobalt</td>
<td>Sliver</td>
<td>Cobalt</td>
<td>Sliver</td>
</tr>
<tr>
<td>No behavioral analysis or sandbox evasion</td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td>Only sandbox evasion</td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td>Unhooking</td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td>Direct syscalls</td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
<tr>
<td>Indirect syscalls</td>
<td>.exe</td>
<td>.dll</td>
<td>.exe</td>
</tr>
</tbody>
</table>

Two evasion techniques work universally against all tested EDRs

Take aways.
- EDRs are more likely to trigger based on well-known abuse tools like Cobalt Strike, suggesting some level of fingerprinting
- Malware hiding in .dll’s is less likely to get detected by EDRs
- EDRs differ in their effectiveness, however some evasion techniques successfully circumvent most (all?) of them
- Our experiments so far only use well-known techniques. Better evasion is possible should it become necessary

Cobalt Strike and Sliver are popular C&C tools to control infected computers

Base case. A malware that does not try to evade behavioral analysis

EDR evasion techniques. Three approaches to circumvent EDR behavioral analysis (as explained on previous slides)
After successful injection, the EDR might still detect the hacker based on suspicious actions

Chain of events from malware download to execution and system abuse

- User interacts with infected file, e.g. .lnk, Office Macro
- Malware is executed – either in the delivery script or deferred with .dll hijacking
- Hacker interacts with the malware remotely (“command and control”)
- Hacker collects more information from system and Active Directory
- Finally, hacker performs malicious actions, like stealing or encrypting files

What we covered so far
- Potential malware get downloaded/executed
- EDR analyses
- We use evasion techniques not to get detect

Let’s look at the next steps in the hacking chain ...
- Once the malware is running, we can trigger different malicious actions
- These, too, can get detected by the EDR
- But mostly they are not – see next slide
EDR systems only trigger on few suspicious actions

<table>
<thead>
<tr>
<th>Abuse vector</th>
<th>EDR1</th>
<th>EDR2</th>
<th>EDR3</th>
<th>AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use malware built-in capabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C&amp;C channel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open SOCS tunnel, e.g. for Network scanning</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data exfiltration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KeyLogger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamically add new capabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run C# binary (through execute-assembly)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run code (in process: beacon object file)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e.g. Sharphound, NanoDump: dumping LSASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run C# code (in process: through inline-execute-assembly) e.g. certify</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Core functionality** of Cobalt+ Sliver. Should be easier to detect based on behavior signatures

**Community extensions.** Harder to detect. Some extensions come in form of BOFs. For other tools that have not yet been prepared as BOF, you can instead use the generic ‘inline-execute-assembly’ as a wrapper and execute pretty much any tool

**Take aways.**
- EDRs are highly ineffective at detecting abuse actions after injection
- When adding new capabilities, red teamers should avoid the built-in ‘execute-assembly’ option that might trigger an EDR

---

**Step 2: System Abuse.** After successfully starting the malware (in step 1), we are now executing malicious actions of the target. All tests in this overview are based on the *indirect syscall.dll* injection technique (from step 1).
Putting the pieces together: By combining the right injection and abuse strategies, hackers can fully circumvent common EDR solutions.

Detailed chain of events from malware download to execution:

1. **ZIP**
   - The **user downloads a zip file** that contains some `.lnk` files.

2. **.lnk**
   - **.lnk executes** `mshta.exe` with the malware location as argument.

3. **HTA**
   - `mshta.exe` downloads and executes a `.hta` malicious file from our server.

EDR system does not detect infection when using the evasion techniques discussed in this presentation.

**Applocker bypass**
- `mshta.exe` is a system default binary that can be misused; this successfully bypasses Applocker.

**Browser bypass**
- Modern browsers refuse to download `.lnk` files. This protection is bypassed by putting the `.lnk` inside a `.zip`.
- Due to `.dll hijacking`, our payload is executed every time Teams is opened.
All hackers gain hidden agenda
[*] Server v1.5.16 - 23eeef3d15cd
c116a1b9936cf392fda3e7d93ad4
[*] Client v1.5.19 - c46e193153e
eea236b2e7898c0cd45ba4e3b4244
[*] Welcome to the sliver shell,
please type 'help' for options
sliver > sessions
[*] No sessions 😔

sliver > [ ]
Agenda

- How EDRs work
- Effective techniques to circumvent them
- How to compensate for EDR protection gaps
Do we even need EDRs on endpoints?

Final experiment: Endpoint-based vs cloud-based detection.
- We uploaded the samples that every EDR in our test lab missed to VirusTotal (indirect system calls, .dll)
- 13/16 engines in VirusTotal successfully detected the malware, without any behavioral analysis on the target endpoint
- This suggests that it is possible to find well-obfuscated malware by building better sandboxes that are harder to detect

Cobalt Strike sample upload to VirusTotal

Sliver sample upload to VirusTotal
Some complimentary controls are available to make up for the protection gaps in EDRs

**EDR make corporations “12%” harder to compromise**

- **8 weeks hacking baseline.** A red team exercise to take over a large corporate takes an average of 4 experts and 8 weeks, including preparation (this varies widely by company, of course)
- Knowing that an EDR is used makes red teaming much slower since testers become very careful not to trigger anomaly detection, and avoid servers that run EDRs
- **1 more week to evade EDR.** When the company uses an EDR on user endpoints and Windows servers, the red team requires about one more week of preparation and execution – “12% more”
- For smaller or easier-to-hack companies, the relative security uplift from using an EDR is larger

**Other controls are needed to further increase hacking resilience**

- Additional security measures further increase the resilience to malware injections:
  - App allow-listing
  - Heavy monitoring on common external compromise vectors (.lnk, ISO, Word...)
  - Tier-0 / zero-trust architecture
  - Threat hunting, that is: Deeper analysis on EDR telemetry
  - Prevent LSASS dumping by running it as protected process light (RunAsPPL)
Security software can introduce software bugs, further decreasing their protection contribution.

EDR systems can have bugs, too

We found issues in a modern EDR system:

- Through default credentials we gained full access to the popular EDR backend, its privileges, and functions (on-premise only)
- Additionally, we discovered three high-severity vulnerabilities in the EDR, arising from weak access control on API endpoints: CVE-2022-27968 and -27969
- All issues have been fixed in the latest versions

EDR management interface, accessible over network with default credentials
Security software can introduce software bugs, further decreasing their protection contribution.

Details of CVE-2022-27968 and -27969

```curl
https://<cynet-server>:8443/WebApp/DeceptionUser/GetAllDeceptionUsers
[
    {
        "Id":2,
        "UserName":"DecoyUser A",
        "DomainName":"company.local",
        "UserType":2,
        "GroupId":3,
        "GroupName":"Main"
    },
    {
        "Id":4,
        "UserName":"DecoyUser B",
        "DomainName":"company.local",
        "UserType":2,
        "GroupId":3,
        "GroupName":"Manually Installed Agents - Linux"
    },
    {
        "Id":3,
        "UserName":"DecoyUser C",
        "DomainName":"company.local",
        "UserType":2,
        "GroupId":3,
        "GroupName":"Manually Installed Agents"
    }
]
```
## Take aways

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>EDR systems can be circumvented</strong> with well-documented techniques</td>
</tr>
<tr>
<td>2</td>
<td><strong>The EDR slows down hackers</strong>, instead of preventing endpoint hacking</td>
</tr>
<tr>
<td>3</td>
<td><strong>Complementary controls are needed</strong> to reach high hacking resilience, in particular system hardening and threat hunting,</td>
</tr>
</tbody>
</table>

**Questions?**

Jorge Gimenez <jorge@srlabs.de>
Karsten Nohl <nohl@srlabs.de>