Как EDR видит угрозы — и где он слепнет

Posted on 28 марта, 2026 by AkaTor

Категория: Red Team / Blue Team / Purple Team
Уровень: Intermediate — Advanced
Автор: Aka Tor

Введение

EDR (Endpoint Detection and Response) — это основной рубеж защиты на конечных точках. Но чтобы обходить EDR или защищаться от обхода, нужно понимать **как именно** он работает внутри. В этой статье разберём архитектуру EDR на Windows, что он видит, а что — нет.

---

1. Архитектура EDR: из чего состоит

EDR работает на двух уровнях — **User Mode** и **Kernel Mode**.

1.1 User Mode компоненты

API Hooking (Inline Hooks в ntdll.dll)

Большинство EDR перехватывают вызовы в `ntdll.dll` — последнем уровне перед переходом в ядро. EDR вставляет **JMP-инструкцию** в начало функции, перенаправляя выполнение в свою DLL.

```
; Оригинальная NtAllocateVirtualMemory:
    mov r10, rcx
    mov eax, 0x18        ; syscall number
    syscall
    ret
```

```
; После хука EDR:
    jmp 0x7FF812340000   ; -> EDR_hook.dll
```

Что перехватывается:

Функция	Зачем EDR следит
`NtAllocateVirtualMemory`	Выделение памяти (шеллкод)
`NtWriteVirtualMemory`	Запись в чужой процесс
`NtCreateThreadEx`	Создание потока (инжект)
`NtMapViewOfSection`	Маппинг секций (process hollowing)
`NtProtectVirtualMemory`	Смена прав памяти (RWX)
`NtQueueApcThread`	APC injection
`NtCreateFile`	Доступ к файлам

Загрузка DLL агента

EDR инжектит свою DLL в каждый новый процесс. Это позволяет:

- Устанавливать хуки в user-mode
- Мониторить поведение процесса изнутри
- Отправлять телеметрию в kernel-mode драйвер

1.2 Kernel Mode компоненты

Kernel Callbacks

EDR регистрирует callback-функции через API ядра:

```
PsSetCreateProcessNotifyRoutineEx()  // создание/завершение процессов
PsSetCreateThreadNotifyRoutine()     // создание потоков
PsSetLoadImageNotifyRoutine()        // загрузка DLL/EXE
ObRegisterCallbacks()                // доступ к хэндлам объектов
CmRegisterCallbackEx()              // операции с реестром
```

Пример: когда процесс вызывает `OpenProcess()` с доступом `PROCESS_ALL_ACCESS` к `lsass.exe`, `ObRegisterCallbacks` срабатывает и EDR может:

- Залогировать событие

- Урезать права доступа (удалить `PROCESS_VM_READ`)
- Заблокировать вызов полностью

Minifilter Drivers

Файловая система проходит через стек minifilter-драйверов. EDR регистрирует minifilter для мониторинга:

- Создания/записи файлов (дроп payload)
- Чтения sensitive файлов (SAM, NTDS.dit)
- Переименования/удаления (ransomware-поведение)

```
Приложение
    ↓
I/O Manager
    ↓
[EDR Minifilter]  ← перехват здесь
    ↓
Файловая система (NTFS)
    ↓
Диск
```

ETW (Event Tracing for Windows)

EDR подписывается на ETW-провайдеры для получения телеметрии:

```
Microsoft-Windows-Threat-Intelligence  — memory operations (kernel-level)
Microsoft-Windows-DotNETRuntime        — .NET Assembly.Load
Microsoft-Antimalware-Scan-Interface   — AMSI events
Microsoft-Windows-PowerShell           — PowerShell logging
Microsoft-Windows-DNS-Client           — DNS queries
```

Threat Intelligence ETW — особенно важен. Это kernel-level провайдер, который нельзя отключить из user-mode. Он даёт EDR видимость в:

- Выделение RWX-памяти
- Маппинг в чужой процесс
- Создание remote threads

---

2. Что EDR реально видит (телеметрия)

2.1 Полная цепочка видимости

```
[Пользователь запускает payload.exe]
        |
        ▼
Kernel Callback: PsSetCreateProcessNotifyRoutineEx
→ EDR видит: родительский процесс, командная строка, путь к файлу
        |
        ▼
Image Load Callback: PsSetLoadImageNotifyRoutine
→ EDR видит: загрузка ntdll.dll, kernel32.dll, payload зависимостей
        |
        ▼
EDR DLL инжектится в процесс
→ Устанавливаются хуки на ntdll.dll
        |
        ▼
Payload вызывает NtAllocateVirtualMemory(RWX)
→ EDR хук срабатывает → логирование + анализ
        |
        ▼
Payload вызывает NtWriteVirtualMemory (пишет шеллкод)
→ EDR хук срабатывает → логирование + анализ
        |
        ▼
Payload вызывает NtCreateThreadEx (запуск шеллкода)
→ EDR хук + Thread Creation Callback → ALERT!
```

2.2 Поведенческие паттерны, которые триггерят EDR

```
Allocate RWX memory → Write shellcode → Create thread
    = Classic injection → HIGH ALERT

Open handle to lsass.exe → Read memory
```

```
    = Credential dumping → CRITICAL ALERT

cmd.exe → whoami / net user / ipconfig
    = Recon activity → MEDIUM ALERT

PowerShell → IEX (New-Object Net.WebClient).DownloadString(...)
    = Fileless malware → HIGH ALERT

svchost.exe spawns cmd.exe
    = Anomalous parent-child → HIGH ALERT
```

---

3. Где EDR слепнет — слабые места

3.1 Direct Syscalls

Проблема для EDR: хуки стоят в `ntdll.dll`, но можно вызвать syscall напрямую, минуя ntdll.

```
; Direct syscall — NtAllocateVirtualMemory
mov r10, rcx
mov eax, 0x18        ; syscall number
syscall              ; прямой переход в ядро, минуя хуки EDR
```

Инструменты атакующих:

- **SysWhispers** — генерация syscall stubs
- **HellsGate / HalosGate** — динамическое определение syscall номеров
- **RecycledGate** — использование существующих syscall инструкций в ntdll

Red Team: Direct syscalls обходят user-mode хуки, но НЕ kernel callbacks и ETW-TI.

Blue Team — как детектить:

- Анализ call stack: легитимный вызов идёт через `ntdll.dll` → `kernel32.dll`. Syscall напрямую из .exe или неизвестной DLL — аномалия
- ETW Threat Intelligence провайдер: работает на уровне ядра, видит операции независимо от хуков

- Мониторинг `unbacked memory` — код исполняется из памяти, не связанной с файлом на диске

```
Легитимный call stack:
    ntdll!NtAllocateVirtualMemory
    kernel32!VirtualAlloc
    program.exe!main

Подозрительный call stack (direct syscall):
    program.exe!main        ← syscall прямо отсюда, ntdll пропущен
```

3.2 Unhooking — снятие хуков

Атака: перезагрузка чистой копии `ntdll.dll` поверх хукнутой.

```
// Концепт: загрузка чистой ntdll с диска
HANDLE hFile = CreateFileA("C:\\Windows\\System32\\ntdll.dll", ...);
HANDLE hMapping = CreateFileMapping(hFile, ...);
LPVOID cleanNtdll = MapViewOfFile(hMapping, ...);

// Перезапись .text секции хукнутой ntdll чистой версией
memcpy(hookedNtdllText, cleanNtdllText, sizeOfTextSection);
```

Вариации:

- Чтение ntdll с диска (`C:\Windows\System32\ntdll.dll`)
- Чтение из `\KnownDlls\ntdll.dll`
- Чтение из другого процесса (fork & read)
- Загрузка из suspended process

Blue Team — как детектить:

- Мониторинг `CreateFileMapping` + `MapViewOfFile` для ntdll.dll
- Периодическая проверка целостности хуков (hook integrity checks)
- Kernel callbacks остаются активны даже после unhooking
- Сравнение .text секции ntdll в памяти с оригиналом

3.3 ETW Patching

Атака: патчинг функции `EtwEventWrite` в ntdll, чтобы отключить user-mode ETW.

```
// Патчинг EtwEventWrite для отключения логирования
void PatchETW() {
    FARPROC addr = GetProcAddress(
        GetModuleHandleA("ntdll.dll"),
        "EtwEventWrite"
    );

    DWORD oldProtect;
    VirtualProtect(addr, 1, PAGE_READWRITE, &oldProtect);

    // xor eax, eax; ret — функция просто возвращает 0 (SUCCESS)
    char patch[] = { 0x33, 0xC0, 0xC3 };
    memcpy(addr, patch, sizeof(patch));

    VirtualProtect(addr, 1, oldProtect, &oldProtect);
}
```

Что это ломает:

- PowerShell ScriptBlock Logging
- .NET Assembly Load логирование
- AMSI (частично)

Что НЕ ломает:

- ETW Threat Intelligence (kernel-mode, нельзя пропатчить из user-mode)
- Kernel Callbacks
- Minifilter Drivers

Blue Team — как детектить:

- Мониторинг `VirtualProtect` на адреса внутри ntdll.dll
- Периодическая проверка первых байт `EtwEventWrite`
- Kernel-mode ETW (TI) продолжает работать — используйте его

3.4 AMSI Bypass

Атака: патчинг `AmsiScanBuffer` в `amsi.dll`.

```
# Концепт (обфусцированные версии гуляют в дикой природе)
$patch = [Byte[]](0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
# Перезапись начала AmsiScanBuffer → возврат AMSI_RESULT_CLEAN
```

Blue Team — как детектить:

- AMSI логирует попытку ДО патчинга (первый вызов виден)
- Мониторинг загрузки `amsi.dll` + `VirtualProtect` на её адреса
- Sysmon Event ID 7 (Image Load) для `amsi.dll`
- Hardware breakpoints вместо software hooks

3.5 Callback Removal (Kernel Level)

Атака: загрузка уязвимого драйвера (BYOVD — Bring Your Own Vulnerable Driver) для удаления kernel callbacks EDR.

```
1. Загрузить легитимный, но уязвимый драйвер (например, старый RTCore64.sys)
2. Через уязвимость получить arbitrary kernel read/write
3. Найти массив PspCreateProcessNotifyRoutine в ядре
4. Обнулить callback EDR-драйвера
```

Известные инструменты: EDRSilencer, EDRSandblast, Terminator

Blue Team — как детектить:

- Мониторинг загрузки драйверов (Sysmon Event ID 6)
- Blocklist уязвимых драйверов (Microsoft Vulnerable Driver Blocklist)
- HVCI (Hypervisor-protected Code Integrity) — запрет неподписанного кода в ядре
- Мониторинг Device Guard policy violations

---

4. Матрица: атака vs. защита

Техника обхода	Уровень	Обходит	НЕ обходит	Детект
Direct Syscalls	User	ntdll hooks	Kernel callbacks, ETW-TI	Stack trace analysis, unbacked memory

Техника обхода	Уровень	Обходит	НЕ обходит	Детект
Unhooking	User	ntdll hooks	Kernel callbacks, ETW-TI	Hook integrity checks, file access to ntdll
ETW Patching	User	User-mode ETW, PS logging	Kernel ETW-TI, callbacks	VirtualProtect мониторинг, byte check
AMSI Bypass	User	AMSI scanning	ETW, hooks, callbacks	Load amsi.dll + VirtualProtect monitoring
BYOVD Callback Removal	Kernel	Kernel callbacks	HVCI, Secure Boot	Driver load monitoring, blocklist
Full Kernel Compromise	Kernel	Всё	Secure Boot + HVCI + VBS	Attestation, TPM measurements

---

5. Рекомендации

Для Red Team

- Один bypass недостаточен — EDR использует **многослойную** телеметрию
- Direct syscalls обходят хуки, но kernel callbacks всё равно видят операции
- Комбинируй техники: unhooking + direct syscalls + indirect syscalls
- Всегда тестируй против конкретного EDR — у каждого свои blind spots

Для Blue Team

- **Не полагайся только на user-mode hooks** — их можно снять
- Включи **ETW Threat Intelligence** (требует PPL или kernel consumer)
- Используй **Kernel Callbacks** как основной источник телеметрии
- Включи **HVCI** для защиты от BYOVD атак
- Мониторь anomalous call stacks (syscalls из небэкнутых регионов памяти)
- **Sysmon** — минимальный набор Event ID: 1, 3, 6, 7, 8, 10, 11, 13, 25

Для Purple Team

- Валидируй каждый детект: воспроизведи технику → проверь алерт
- Стройте детекты по MITRE ATT&CK: T1055 (Injection), T1003 (Credential Dumping), T1562.001 (Disable/Modify Tools)
- Документируйте blind spots вашего EDR — знание слабостей = сила

Заключение

EDR — мощный инструмент, но не серебряная пуля. Его архитектура имеет фундаментальные ограничения: user-mode компоненты можно обойти, kernel-mode — сложнее, но тоже возможно через BYOVD. Понимание этих механизмов одинаково важно как для атакующих, так и для защитников.

В следующей статье разберём **Unhooking и Direct Syscalls** на практике — с кодом, PoC и детект-правилами.

Дисклеймер: Материал предоставлен исключительно в образовательных целях для специалистов по информационной безопасности. Используйте полученные знания только в рамках авторизованного тестирования на проникновение и защиты инфраструктуры.