

Reversing MSRC Updates: Case Studies of MSRC Bulletins 2004–2007

BlackHat USA 2007

Damian Hasse

Greg Wroblewski

SWI React

Microsoft

Thanks to SWI React team, ChrisW & MSRC

Agenda

- MS06-001 WMF
- MS06-013 IE RCE createTextRange
- MS05-047 RCE in Plug-n-Play via RPC
- MS06-018 MSDTC DoS
- MS05-018 CSRSS EoP
- MS05-018 & MS05-049 CSRSS EoP - again

Our Process

- Mail to secure@microsoft.com
- Team engages & triages the issue
- **Hacking for variations & attack vectors**
 - Works with & educates partners
- Agreement on fix plan
- **Mitigations & Workarounds**
- Ship

Team Goals

- Keep customers secure
 - Zero recalls due to security
- Stay ahead of security trends
 - Participation in at least one security research project
- Ship comprehensive patches
 - Find lots of variants from reported issue
- Use our knowledge to improve SDL and thus all MS products

MS06-001: WMF

Anything interesting in the code below?

```
pldc = GET_PLDC(hdc);
...
while (pMR = (PMETARECORD) GetEvent(pMF, pMR))
{
    if (pMR == (PMETARECORD) -1)
    {
        fStatus = FALSE;
        break;
    }
    ...

    if (pldc && pldc->pfnAbort != NULL)
    {
        if (!(*pldc->pfnAbort)(hdc, 0))
        {
            fStatus = FALSE;
            break;
        }
    }

    // For win3.1 compatability, ignore the return value from PlayMetaFileRecord

    PlayMetaFileRecord(hdc, pht, pMR, noObjs);
}
```



MS06-001: WMF - Legacy code

How it happens?

```
while (pMR = (PMETARECORD) GetEvent(pMF, pMR))
```



```
{  
    if (!(*pldc->pfnAbort)(hdc, 0))
```

```
        ...
```

```
        PlayMetaFileRecord(hdc, pht, pMR, noObjs);
```

```
    }
```



```
    PlayMetaFileRecord(hdc, lpHandleTable, lpMR, noObjs)
```

```
    magic = lpMR->rdFunction;
```

```
    switch (magic & 255) {
```

```
        case (META_ESCAPE & 255):
```

```
        ...
```

```
            int iEscapeSF = (int)(UINT)lpMR->rdParm[0];
```

```
            if (iEscapeSF != MFCOMMENT)
```

```
            {
```

```
                fStatus = Escape(hdc, iEscapeSF, (int)(UINT) lpMR->rdParm[1],  
                                (LPCSTR) &lpMR->rdParm[2], (LPVOID) NULL) != 0;
```



```
                int WINAPI Escape(hdc, iEscape, cjIn, pvIn, pvOut)  
                {
```

```
                    switch (iEscape) {
```

```
                        case SETABORTPROC:
```

```
                            iRet = SetAbortProc(hdc, (ABORTPROC)pvIn);
```

```
                            break;
```



```
                int WINAPI SetAbortProc(HDC hdc, ABORTPROC pfnAbort) {
```

```
                    DC_PLDC(hdc, pldc, iRet);
```

```
                    pldc->pfnAbort = pfnAbort;
```

MS06-001: WMF - Legacy code

Let's look at a WMF file

- Attacker creates WMF file having the following metarecord:
`10 00 00 00 26 06 09 00 16 00 CC CC ...`
- `10 00 00 00` – 0x10 = the 'Size' field of the metarecord
- `26 06` – 0x626 = code of META_ESCAPE function
- `09 00` – 0x9 = code of SETABORTPROC function
- `16 00` – 0x16 = size of the code following this field
- `CC CC ...` – code to be executed (here: 'int 3' breakpoints)
- META_ESCAPE function triggers PlayMetaFileRecord to call into Escape().
- SETABORTPROC function triggers Escape() to call into SetAbortProc().
- SetAbortProc() sets pldc->pfnAbort.
- Existence of pldc->pfnAbort allows for arbitrary code execution.

MS06-001: WMF - Legacy code

What we Learned

- Deprecate old functionality
 - NetDDE, IPX/SPX and NetWare removed from Windows Vista
 - Remove functionality (as in this case)
- Disable old functionality
 - Registry example (for example WMF & EMF) – app compat concious
- Careful code reviews are not bulletproof
 - Backup code reviews with testing & fuzzing

MS06-013: IE RCE createTextRange

Any bugs in the code below?

```
HRESULT
CInput::createTextRange(IHTMLTxtRange * * ppDisp)
{
    HRESULT          hr = S_OK;
    CAutoRange *     pAutoRange = NULL;
    ...
    if (!ppDisp)
    {
        hr = E_INVALIDARG;
        goto Cleanup;
    }

    if (!HasSlavePtr())
    {
        goto Cleanup;
    }
    ...
    *ppDisp = pAutoRange;
    pAutoRange->AddRef();

Cleanup:
    ...
    RRETURN( SetErrorInfo( hr ) );
}
```

MS06-013: IE RCE createTextRange Returning incorrect error code

How does it get exploited?

```
hr = (*pHandler)(this, pSrvProvider, pDisp, wEntry,  
    (PROPERTYDESC_BASIC_ABSTRACT *)pDesc, wFlags,  
    pdispparams, pvarResult);
```



```
HRESULT  
CInput::createTextRange( IHTMLTxtRange * * ppDisp )
```



```
if (hr == S_OK && pvarResult &&  
    V_VT(pvarResult) == VT_DISPATCH &&  
    V_DISPATCH(pvarResult))  
{  
    IDispatch *pdisptemp = V_DISPATCH(pvarResult);  
    hr = pdisptemp->QueryInterface(IID_IDispatch,  
        (LPVOID*)&V_DISPATCH(pvarResult));
```

 Calling un-initialized
memory on the heap

MS06-013: IE RCE createTextRange Returning incorrect error code

What we Learned

- Static analysis tools improvement
 - Detection of un-init memory usage
 - Using SAL to ensure error check upon return

MS05-047: RCE in Plug-n-Play via RPC

wsprintf looks suspicious, but RegOpenKeyEx should block invalid data, right?

```
#define MAX_CM_PATH          360
GetInstanceList(
    IN LPCWSTR pszDevice, IN OUT LPWSTR *pBuffer,
    IN OUT PULONG pulLength)
{
    WCHAR          RegStr[MAX_CM_PATH], szInstance[MAX_DEVICE_ID_LEN];
    ...
    // Validate that passed in pszDevice is an actual registry entry
    // If lookup for the key fails, reject call and cleanup.
    // ghEnumKey points to HKLM\System\CurrentControlSet\Enum
    if (RegOpenKeyEx(ghEnumKey, pszDevice, 0,
        KEY_ENUMERATE_SUB_KEYS, &hKey) != ERROR_SUCCESS) {
        Status = CR_REGISTRY_ERROR;
        goto Clean0;
    }
    ...
    ulLen = MAX_DEVICE_ID_LEN; // size in chars
    ...
    // Query szInstance from registry
    RegStatus = RegEnumKeyEx(hKey, ulIndex, szInstance, &ulLen, ...);
    if (RegStatus == ERROR_SUCCESS) {
        // Build lookup string given a valid registry root key and
        // valid instance ID
        wsprintf(RegStr, TEXT("%s\\%s"), pszDevice, szInstance);
    }
}
```

MS05-047: RCE in Plug-n-Play via RPC

How can we get wsprint to blow-up?

Windows System Information

Registry Element Size Limits

The following table identifies the size limits for the various registry elements.

Registry Element	Size Limit
Key name	255 characters
Value name	16,383 characters

```
wsprintf(RegStr, TEXT("%s\\%s"), pszDevice, szInstance);
```

- pszDevice supposed to be less than 255 characters
- pszDevice needs be a valid key in HKLM\System\CurrentControlSet\Enum
- szInstance points to a valid subkey under pszDevice
- RegStr is 360 characters
- Attacker does not have control over registry content

MS05-047: RCE in Plug-n-Play via RPC

Incorrect Assumptions

What we Learned

- Banned APIs effort
 - Unsafe string functions removed from Windows Vista

MS06-018:MSDTC DoS

I can read source code without any problems...



```
void __RPC_FAR * __RPC_API midl_user_allocate(size_t len)
{
...

```

```
    if ( len > CPageInfo::TotalSpaceAvailable() )
    {
        LOG0("Buffer size requested too large for memory manager.
            midl_user_allocate");
        return NULL;
    }

```



```
TotalSpaceAvailable (void){ return (DEFAULT_PAGE_SIZE * sizeof(char)) -
    ALIGNED_SPACE(sizeof(CPageInfo));}

```

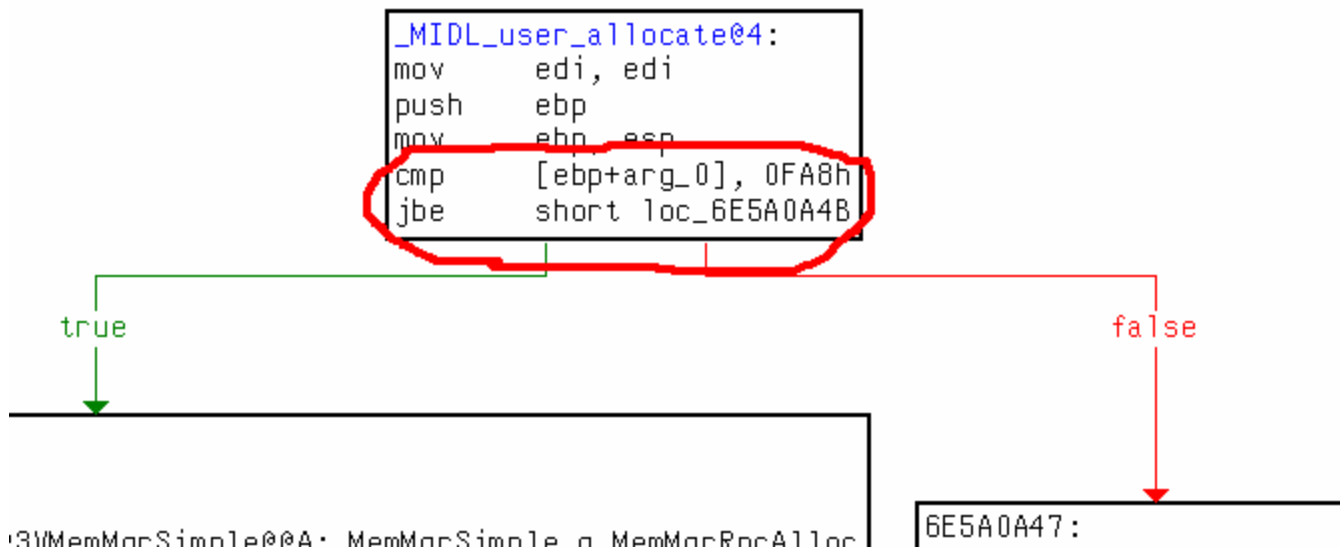


```
inline UINT_PTR ALIGNED_SPACE (INT_PTR x)
{
    UINT_PTR uiTemp;
    if ((x % BYTE_ALIGNMENT) > 0)
        uiTemp = x + (BYTE_ALIGNMENT - (x % BYTE_ALIGNMENT));
    else
        return x;
    return uiTemp;
}
const unsigned long BYTE_ALIGNMENT = 8;
#define DEFAULT_PAGE_SIZE 4096

```

MS06-018: MSDTC DoS Convoluted Code

Assembly does look simpler....



MS06-018: MSDTC DoS

Convoluted Code

What we Learned

- SAL annotation for allocators
- Removal of most custom allocators on Windows Vista
- Verify fix in assembly

MS05-018: CSRSS EoP

Untrusted has been verified, so my code is safe ...

```
VOID
PropertiesUpdate(
    IN PCONSOLE_INFORMATION Console,
    IN HANDLE hClientSection
)
...
/*
 * Get a pointer to the shared memory block.
 */
pStateInfo = NULL;
ulViewSize = 0;
Status = NtMapViewOfSection(hSection,
                            NtCurrentProcess(),
                            &pStateInfo,
                            0,
                            0,
                            NULL,
                            &ulViewSize,
                            ViewUnmap,
                            0,
                            PAGE_READONLY);
...
RtlCopyMemory(pStateInfoLocal, pStateInfo, ulViewSize);
if (!ValidateConsoleProperties(pStateInfoLocal)) {
    RIPMSG0(RIP_WARNING, "Failing console properties update");
    goto ErrorExit;
}
...
FontIndex = FindCreateFont(pStateInfo->FontFamily,
                           pStateInfo->FaceName,
                           pStateInfo->FontSize,
                           pStateInfo->FontWeight);
```

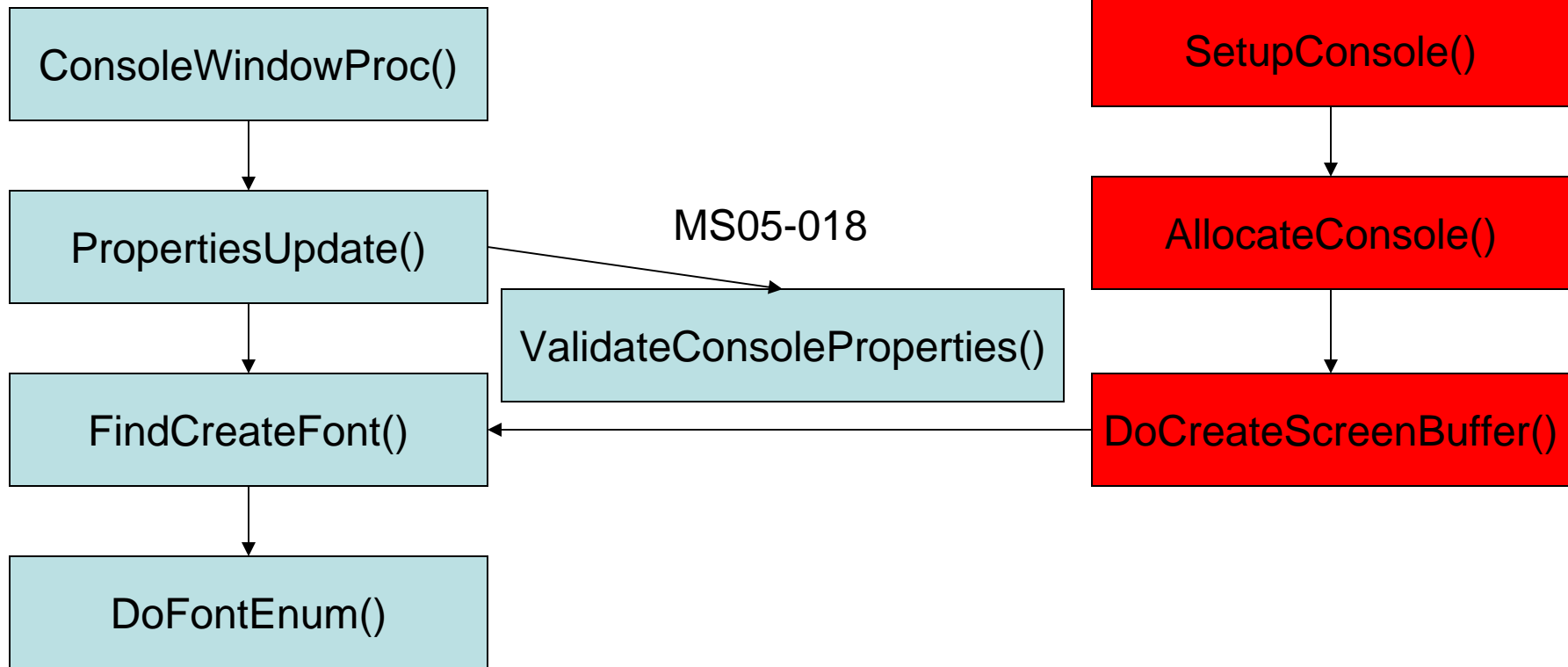


MS05-018 CSRSS EoP - again

MS05-049

Remember – TOCTOU - How would you fix the bug below?

MS05-049



```
if (pwszFace)
    wcscpy(LogFont.IfFaceName, pwszFace);
```



Oops... another way to hit the vulnerable code

MS05-018 CSRSS EoP - again

MS05-049

What we Learned

- Runtime analysis tools to detect shared memory sections
- Use modern IPC techniques
 - No race conditions
 - Some validation
- Fix root cause
 - On security updates
- Search for other callers
- Review threat model & spec

Summary

- Integrated into the security engineering process (i.e. SDL) and feedback loop
- Root-cause-analysis done on cases
- On-going new tools development and/or improvements suggested

Questions?

Have fun :)

Got security bugs?
secure@microsoft.com