RBS-2015-004

Moxa SoftCMS
Multiple Buffer Overflows

Table of Contents

Vendor / Product Information

Moxa is a Taiwan-based company with offices around the world and customers in over 70 countries. Moxa provides products for industrial networking, computing, and automation. The application for these products include factory automation, smart rail, smart grid, intelligent transportation, oil & gas, marine, and mining.

Moxa SoftCMS is a central management software used for managing large scale CCTV installations from a single interface. It allows live video monitoring, video playback, Emap, remote I/O trigger, and event management. The product is primarily used in USA and Europe.

Vulnerable Program Details

Details for tested products and versions:

Vendor:	Moxa
Product:	SoftCMS
Version:	1.2
Component:	SStreamVideo rtspvideo ActiveX Control (SStreamVideo.ocx)
Version:	0.0.0.65
Component:	IPCam ActiveX Plugin V1 (axvlc.dll)
Version:	0.8.6.0
Component:	libvlc.dll
Version:	N/A

NOTE: Other versions than the one listed above are likely affected. As the vulnerable components are signed by Ningbo HenTek Dragon Electronics[1], these may be bundled with products from other vendors too.

Credits

Carsten Eiram, Risk Based Security
Twitter: @CarstenEiram
Twitter: @RiskBased

[1] http://www.hentek-dragon.com/

Impact

Moxa SoftCMS bundles two ActiveX controls marked safe-for-scripting. These can be instantiated and scripted to within a browser based on the Trident engine e.g. Internet Explorer. The two ActiveX controls as well as linked libraries have been found to be affected by seven critical buffer overflow vulnerabilities.

By tricking a user into visiting a malicious website, these vulnerabilities allow an attacker to compromise the system on which SoftCMS is installed. A successful attack would e.g. give the attacker access to and control over all surveillance cameras managed by the system.

Vulnerability Details

libvlc.dll setAVIPrefix() Function Heap Buffer Overflow

The IPCam ActiveX Plugin V1 (`axvlc.dll`) ActiveX control supports the `setRecordPrefix()` method defined as:

```
void setRecordPrefix([in] BSTR strPrefix);
```

The function handling the method converts the supplied '`strPrefix`' string and passes it as argument to the `setAVIPrefix()` function exported by `libvlc.dll`.

```
.text:06518930 ; int __stdcall VLCControl::setRecordPrefix(int pThis, BSTR strPrefix)
.text:06518930                 public __ZN10VLCControl15setRecordPrefixEPw@8
.text:06518930 __ZN10VLCControl15setRecordPrefixEPw@8 proc near
.text:06518930                                         ; DATA XREF: .rdata:0659967Co
.text:06518930
.text:06518930 var_1C          = dword ptr -1Ch
.text:06518930 var_18          = dword ptr -18h
.text:06518930 var_C           = dword ptr -0Ch
.text:06518930 pThis           = dword ptr  4
.text:06518930 strPrefix       = dword ptr  8
.text:06518930
.text:06518930                 push    esi
.text:06518931                 push    ebx
.text:06518932                 sub     esp, 14h
.text:06518935                 mov     ebx, [esp+1Ch+pThis]
.text:06518939                 mov     eax, [esp+1Ch+strPrefix]
.text:0651893D                 mov     esi, [ebx+4]
.text:06518940                 mov     edx, [esi+58h]
.text:06518943                 mov     [esp+1Ch+var_18], eax ; BSTR
.text:06518947                 mov     [esp+1Ch+var_1C], edx ; CodePage
.text:0651894A                 call    __Z12CStrFromBSTRjPw ; CStrFromBSTR(uint,wchar_t *)
…
```

```
.text:06518994                 mov     [esp+1Ch+var_1C], esi
.text:06518997                 mov     ebx, [esp+1Ch+var_C]
.text:0651899B                 mov     [esp+1Ch+var_18], ebx
.text:0651899F                 call    _setAVIPrefix
```

Within this function, the converted string is copied straight into offset 514h of an object on the heap using `strcpy()`.

```
.text:065DC420 ; int __cdecl setAVIPrefix(char *, int strPrefix)
.text:065DC420                 public _setAVIPrefix
.text:065DC420 _setAVIPrefix   proc near
.text:065DC420
.text:065DC420 var_1C          = dword ptr -1Ch
.text:065DC420 szStrPrefix     = dword ptr -18h
.text:065DC420 var_C           = dword ptr -0Ch
.text:065DC420 var_8           = dword ptr -8
.text:065DC420 var_4           = dword ptr -4
.text:065DC420 arg_0           = dword ptr  4
.text:065DC420 strPrefix       = dword ptr  8
.text:065DC420
.text:065DC420                 sub     esp, 1Ch
.text:065DC423                 mov     [esp+1Ch+var_8], esi
.text:065DC427                 mov     esi, [esp+1Ch+strPrefix]
.text:065DC42B                 mov     [esp+1Ch+var_4], edi
.text:065DC42F                 mov     edi, [esp+1Ch+arg_0]
.text:065DC433                 test    esi, esi
.text:065DC435                 mov     [esp+1Ch+var_C], ebx
.text:065DC439                 jnz     short loc_65DC488 ; string supplied?
.text:065DC488 loc_65DC488:                              ; CODE XREF: _setAVIPrefix+19j
.text:065DC488                 mov     [esp+1Ch+szStrPrefix], esi
.text:065DC48C                 mov     eax, ds:_p_libvlc
.text:065DC491                 mov     [esp+1Ch+var_1C], eax
.text:065DC494                 call    ___vlc_object_get
.text:065DC499                 mov     ebx, eax
.text:065DC49B                 jmp     short loc_65DC441
.text:065DC441 loc_65DC441:                              ; CODE XREF: _setAVIPrefix+7Bj
.text:065DC441                 test    ebx, ebx
.text:065DC443                 jz      short loc_65DC457
.text:065DC445                 mov     [esp+1Ch+szStrPrefix], edi ; char *
.text:065DC449                 lea     edx, [ebx+514h]
.text:065DC44F                 mov     [esp+1Ch+var_1C], edx ; char *
.text:065DC452                 call    _strcpy          ; b0f!
```

As no bounds checks are performed, this leads to a heap-based buffer overflow.

SStreamVideo.ocx AudioRecord() Method ip Argument Stack Buffer Overflow

The SStreamVideo rtspvideo (`SStreamVideo.ocx`) ActiveX control supports the `AudioRecord()` method defined as:

```
VARIANT_BOOL AudioRecord(
                long enable,
                BSTR ip,
                long port,
                long EncodeMethod,
                long subCode,
                long savetofile,
                BSTR fullfilename);
```

When the method is called, the '`enable`' argument is checked if set to 1.

```
.text:1000E97C ; int __stdcall M_AudioRecord(int lEnable, LPCWSTR pwzIP, int lPort, int
lEncodeMethod, int lSubCode, int lSaveToFile, int pwzFullFilename)
.text:1000E97C M_AudioRecord   proc near               ; CODE XREF: sub_1000AF20+F1p
.text:1000E97C                                         ; sub_1000B40D+68p ...
.text:1000E97C
.text:1000E97C var_41C         = dword ptr -41Ch
.text:1000E97C var_418         = dword ptr -418h
.text:1000E97C szIP            = byte ptr -414h        ; char[1040]
.text:1000E97C var_4           = dword ptr -4
.text:1000E97C lEnable         = dword ptr  8
.text:1000E97C pwzIP           = dword ptr  0Ch
.text:1000E97C lPort           = dword ptr  10h
.text:1000E97C lEncodeMethod   = dword ptr  14h
.text:1000E97C lSubCode        = dword ptr  18h
.text:1000E97C lSaveToFile     = dword ptr  1Ch
.text:1000E97C pwzFullFilename = dword ptr  20h
.text:1000E97C
.text:1000E97C lpThis = edi
.text:1000E97C                 push    410h
.text:1000E981                 mov     eax, offset sub_10066BD1
.text:1000E986                 call    __EH_prolog3_GS
.text:1000E98B                 mov     ebx, [ebp+pwzIP]
.text:1000E98E                 mov     esi, [ebp+pwzFullFilename]
.text:1000E991                 mov     lpThis, ecx
.text:1000E993                 call    j_?AfxGetModuleState@@YGPAVAFX_MODULE_STATE@@XZ ;
AfxGetModuleState(void)
.text:1000E998                 push    eax
.text:1000E999                 lea     ecx, [ebp+var_41C]
.text:1000E99F                 call    ??0AFX_MAINTAIN_STATE2@@QAE@PAVAFX_MODULE_STATE@@@Z ;
AFX_MAINTAIN_STATE2::AFX_MAINTAIN_STATE2(AFX_MODULE_STATE *)
.text:1000E9A4                 xor     eax, eax
.text:1000E9A6                 cmp     [ebp+lEnable], 1
.text:1000E9AA                 mov     [ebp+var_4], eax
.text:1000E9AD                 jnz     loc_1000EA4F    ; enable != 1?
```

If so, the '`ip`' argument is converted to multibyte and copied into a 1040 byte stack buffer using `strcpy()`.

```
.text:1000E9DC loc_1000E9DC:                           ; CODE XREF: M_AudioRecord+3Ej
.text:1000E9DC                 push    0FFFFFFFFh      ; cchWideChar
.text:1000E9DE                 push    ebx             ; lpWideCharStr
```

```
.text:1000E9DF                call    ConvertToMultibyte
.text:1000E9E4                mov     ebx, eax
.text:1000E9E6                lea     eax, [ebp+szIP] ; char[1040]
.text:1000E9EC                push    ebx             ; char *
.text:1000E9ED                push    eax             ; char *
.text:1000E9EE                call    _strcpy         ; b0f!
```

As no bounds checks are performed, this leads to a stack-based buffer overflow.

2WayAudio.dll Audio Record Handling Buffer Overflow

The SStreamVideo rtspvideo (`SStreamVideo.ocx`) ActiveX control supports the `AudioRecord()` method defined as:

```
VARIANT_BOOL AudioRecord(
                long enable,
                BSTR ip,
                long port,
                long EncodeMethod,
                long subCode,
                long savetofile,
                BSTR fullfilename);
```

When called, the handler function performs inconsequential processing before eventually calling `AudioRecord_SendClient()` in `2WayAudio.dll` with the arguments supplied to the method. This function checks if the '`enable`' argument is set to 1. If not, the function returns without further processing, else it calls another function to process the input.

Figure 1: AudioRecord_SendClient() function in 2WayAudio.dll

This function copies the '`fullfilename`' argument specifying the name of the file to which audio should be recorded into a 1024 wchar global buffer using `wcscpy()`.

```
.text:059D14E7 sub_59D14E7     proc near               ; CODE XREF:
AudioRecord_SendClient(int,char *,short,int,int,int,wchar_t const *)+20p
.text:059D14E7
.text:059D14E7 var_1C          = byte ptr -1Ch
.text:059D14E7 var_8           = dword ptr -8
.text:059D14E7 ThreadId        = dword ptr -4
.text:059D14E7 pszIP           = dword ptr  8
.text:059D14E7 usPort          = word ptr  0Ch
.text:059D14E7 lEncodeMethod   = dword ptr  10h
.text:059D14E7 lSubCode        = dword ptr  14h
.text:059D14E7 cSaveToFile     = byte ptr  18h
.text:059D14E7 pwzFullFilename = dword ptr  1Ch
.text:059D14E7
.text:059D14E7                 push    ebp
.text:059D14E8                 mov     ebp, esp
.text:059D14EA                 sub     esp, 1Ch
...
.text:059D1510                 push    [ebp+pwzFullFilename] ; wchar_t *
.text:059D1513                 mov     word_5A03FB8, ax
.text:059D1519                 mov     eax, [ebp+lEncodeMethod]
.text:059D151C                 mov     dword_5A03FBC, eax
.text:059D1521                 mov     eax, [ebp+lSubCode]
.text:059D1524                 mov     dword_5A03FC0, eax
.text:059D1529                 mov     al, [ebp+cSaveToFile]
.text:059D152C                 push    offset g_wzBuf1 ; wchar_t *
.text:059D1531                 mov     byte_5A03FC4, al
.text:059D1536                 call    _wcscpy         ; b0f!
```

As no bounds checks are performed, this leads to a global buffer overflow.

SStreamVideo.ocx Multiple Methods URL Handling Heap Buffer Overflow

The SStreamVideo rtspvideo ActiveX control (`SStreamVideo.ocx`) supports the `Open()` and `Open2()` methods defined as:

```
VARIANT_BOOL Open(
                BSTR StrRtspPath,
                BSTR strHttpPath);

VARIANT_BOOL Open2(
                BSTR StrRtspPath,
                BSTR strHttpPath,
                unsigned short nPlayMode);
```

Regardless of which method is invoked, processing is similar. The responsible function eventually calls a function to connect to the provided URL in the '`StrRtspPath`' or '`strHttpPath`' argument.

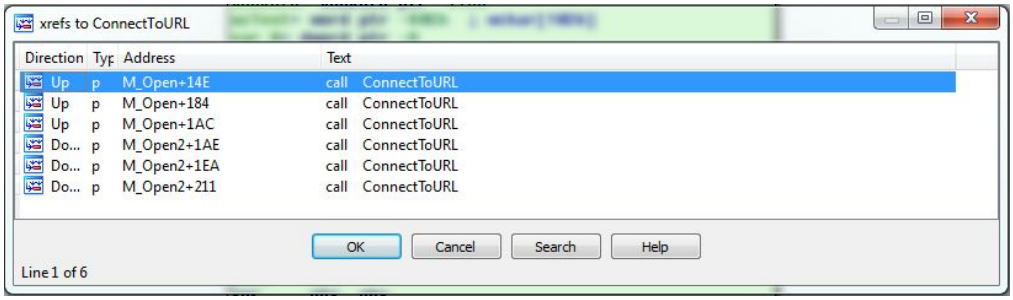

Figure 2: Calls to ConnectToURL() function

Before connecting, the called function calls another function to validate the provided URL.

```
.text:1000F164 ; int __thiscall ConnectToURL(wchar_t *, int, int, int)
.text:1000F164 ConnectToURL    proc near               ; CODE XREF: M_Open+14Ep
.text:1000F164                                         ; M_Open+184p ...
.text:1000F164
.text:1000F164 var_9A0         = dword ptr -9A0h
.text:1000F164 WSAData         = WSAData ptr -99Ch
.text:1000F164 wzText          = word ptr -80Ch        ; wchar[1026]
.text:1000F164 var_8           = dword ptr -8
.text:1000F164 arg_0           = dword ptr  8
.text:1000F164 arg_4           = dword ptr  0Ch
.text:1000F164 arg_8           = dword ptr  10h
.text:1000F164
.text:1000F164                 push    ebp
.text:1000F165                 mov     ebp, esp
.text:1000F167                 sub     esp, 9A0h
.text:1000F16D                 mov     eax, ___security_cookie
.text:1000F172                 xor     eax, ebp
.text:1000F174                 mov     [ebp+var_8], eax
.text:1000F177                 mov     eax, [ebp+arg_0]
.text:1000F17A                 push    ebx
.text:1000F17B                 push    esi
.text:1000F17C                 xor     ebx, ebx
.text:1000F17E                 cmp     [ebp+arg_4], ebx
.text:1000F181                 push    edi
.text:1000F182                 mov     esi, ecx
.text:1000F184                 jnz     short loc_1000F1A4
.text:1000F1A4 loc_1000F1A4:                           ; CODE XREF: ConnectToURL+20j
.text:1000F1A4                 push    800h            ; size_t
.text:1000F1A9                 call    _malloc
.text:1000F1AE                 or      [ebp+var_9A0], 0FFFFFFFFh
.text:1000F1B5                 pop     ecx
.text:1000F1B6                 mov     edi, eax
.text:1000F1B8                 lea     eax, [ebp+var_9A0]
.text:1000F1BE                 push    eax             ; int
.text:1000F1BF                 push    edi             ; pMem800h
.text:1000F1C0                 mov     ecx, esi        ; wchar_t *
.text:1000F1C2                 call    ValidateURL
```

Inside this function, a 1024 wchar buffer is allocated on the heap after which the provided argument is copied into it using `wcscpy()`.

```
.text:1000F2CB ; int __thiscall ValidateURL(wchar_t *, int pMem800h, int)
.text:1000F2CB ValidateURL     proc near               ; CODE XREF: ConnectToURL+5Ep
.text:1000F2CB
.text:1000F2CB var_E4          = dword ptr -0E4h
.text:1000F2CB var_E0          = dword ptr -0E0h
.text:1000F2CB lpMem800h       = dword ptr -0DCh
.text:1000F2CB var_D8          = dword ptr -0D8h
.text:1000F2CB var_D4          = dword ptr -0D4h
.text:1000F2CB var_D0          = dword ptr -0D0h
.text:1000F2CB var_CC          = byte ptr -0CCh
.text:1000F2CB var_CA          = word ptr -0CAh
.text:1000F2CB __security_cookie= dword ptr -4
.text:1000F2CB pMem800h        = dword ptr  8
.text:1000F2CB arg_4           = dword ptr  0Ch
.text:1000F2CB
.text:1000F2CB                 push    ebp
.text:1000F2CC                 lea     ebp, [esp-70h]
.text:1000F2D0                 sub     esp, 0E4h
.text:1000F2D6                 mov     eax, ___security_cookie
.text:1000F2DB                 xor     eax, ebp
.text:1000F2DD                 mov     [ebp+70h+__security_cookie], eax
.text:1000F2E0                 mov     eax, [ebp+70h+pMem800h]
.text:1000F2E3                 push    ebx
.text:1000F2E4                 push    esi
.text:1000F2E5                 push    edi
.text:1000F2E6                 mov     [ebp+70h+lpMem800h], eax
.text:1000F2E9                 mov     eax, [ebp+70h+arg_4]
.text:1000F2EC                 mov     ebx, 800h
.text:1000F2F1                 push    ebx             ; size_t
.text:1000F2F2                 mov     edi, ecx
.text:1000F2F4                 mov     [ebp+70h+var_E4], eax
.text:1000F2F7                 mov     [ebp+70h+var_D8], 3Ah
.text:1000F2FE                 call    _malloc
.text:1000F303                 mov     esi, eax
.text:1000F305                 push    ebx             ; size_t
.text:1000F306                 push    0               ; int
.text:1000F308                 push    esi             ; void *
.text:1000F309                 mov     [ebp+70h+var_D4], esi
.text:1000F30C                 mov     [ebp+70h+var_E0], esi
.text:1000F30F                 call    _memset
.text:1000F314                 push    edi             ; wchar_t *
.text:1000F315                 push    esi             ; wchar_t *
.text:1000F316                 call    _wcscpy         ; b0f!
```

As no bounds checks are performed, this leads to a heap-based buffer overflow.

libvlc.dll VLC_AddTarget() Function Heap Buffer Overflow

The SStreamVideo rtspvideo (`SStreamVideo.ocx`) ActiveX control supports the `Open3()` method defined as:

```
VARIANT_BOOL Open3(
                BSTR StrRtspPath,
                unsigned short nPlayMode);
```

The function in `SStreamVideo.ocx` that handles the `Open3()` method calls the `VLCControl::addTarget()` function in `axvlc.dll` via `CWnd::InvokeHelper()` to add the '`StrRtspPath`' argument to a playlist.

To do so, the function in turn calls `VLC_AddTarget()` in `libvlc.dll`. After inconsequential processing, the function copies the '`StrRtspPath`' argument into offset 148Ch of a playlist object on the heap using `strcpy()` without performing any bounds checks.

```
.text:065DD2F0 _VLC_AddTarget  proc near               ; CODE XREF: _VLC_Init+FB7p
.text:065DD2F0                                         ; _VLC_Init+17C8p ...
...
.text:065DD2F0 arg_0           = dword ptr  4
.text:065DD2F0 arg_4           = dword ptr  8
.text:065DD2F0 arg_8           = dword ptr  0Ch
.text:065DD2F0 arg_C           = dword ptr  10h
.text:065DD2F0 arg_10          = dword ptr  14h
.text:065DD2F0 arg_14          = dword ptr  18h
.text:065DD2F0
.text:065DD2F0                 sub     esp, 3Ch
.text:065DD2F3                 mov     eax, [esp+3Ch+arg_0]
.text:065DD2F7                 mov     [esp+3Ch+var_4], ebp
.text:065DD2FB                 mov     ebp, [esp+3Ch+arg_4]
…
.text:065DD396                 mov     [esp+3Ch+var_38], ebp ; char *
.text:065DD39A                 lea     edx, [esi+148Ch]
.text:065DD3A0                 mov     [esp+3Ch+var_3C], edx ; char *
.text:065DD3A3                 call    _strcpy         ; b0f!
```

This leads to a heap-based buffer overflow.

libvlc.dll setConfigurePathA() Function Heap Buffer Overflow

The IPCam ActiveX Plugin V1 (`axvlc.dll`) ActiveX control supports the `setConfigPath()` method defined as:

```
void setConfigPath([in] BSTR strPATH);
```

The function handling the method converts the supplied '`strPATH`' string and passes it as argument to the `setConfigurePathA()` function exported by `libvlc.dll`.

```
.text:06519090 ; int __stdcall VLCControl::setConfigPath(int pThis, BSTR strPATH)
.text:06519090                 public __ZN10VLCControl13setConfigPathEPw@8
.text:06519090 __ZN10VLCControl13setConfigPathEPw@8 proc near ; DATA XREF: .rdata:06599640o
.text:06519090
.text:06519090 CodePage        = dword ptr -1Ch
.text:06519090 var_18          = dword ptr -18h
.text:06519090 var_C           = dword ptr -0Ch
.text:06519090 var_8           = dword ptr -8
.text:06519090 var_4           = dword ptr -4
.text:06519090 pThis           = dword ptr  4
.text:06519090 strPATH         = dword ptr  8
.text:06519090
.text:06519090                 sub     esp, 1Ch
...
.text:065190D9                 mov     esi, [esp+1Ch+strPATH]
.text:065190DD                 mov     edx, [eax+58h]
.text:065190E0                 mov     [esp+1Ch+var_18], esi ; BSTR
.text:065190E4                 mov     [esp+1Ch+CodePage], edx ; CodePage
.text:065190E7                 call    __Z12CStrFromBSTRjPw ; CStrFromBSTR(uint,wchar_t *)
.text:065190EC                 mov     edx, eax
.text:065190EE                 test    edx, edx
.text:065190F0                 mov     eax, E_OUTOFMEMORY
.text:065190F5                 jz      short loc_65190B0
.text:065190F7                 mov     [esp+1Ch+CodePage], edx
.text:065190FA                 mov     ecx, [esp+1Ch+var_C]
.text:065190FE                 mov     [esp+1Ch+var_18], ecx
.text:06519102                 call    _setConfigurePathA
```

Within this function, the string is copied straight into offset 6DCh of an object on the heap using `strcpy()`.

```
.text:065DC070 ; int __cdecl setConfigurePathA(char *pszSrc, int iUnk2)
.text:065DC070                 public _setConfigurePathA
.text:065DC070 _setConfigurePathA proc near
.text:065DC070
.text:065DC070 var_1C          = dword ptr -1Ch
.text:065DC070 var_18          = dword ptr -18h
.text:065DC070 var_C           = dword ptr -0Ch
.text:065DC070 var_8           = dword ptr -8
.text:065DC070 var_4           = dword ptr -4
.text:065DC070 pszSrc          = dword ptr  4
.text:065DC070 iUnk2           = dword ptr  8
.text:065DC070
.text:065DC070                 sub     esp, 1Ch
.text:065DC073                 mov     [esp+1Ch+var_8], esi
.text:065DC077                 mov     esi, [esp+1Ch+iUnk2]
.text:065DC07B                 mov     [esp+1Ch+var_4], edi
.text:065DC07F                 mov     edi, [esp+1Ch+pszSrc]
.text:065DC083                 test    esi, esi
.text:065DC085                 mov     [esp+1Ch+var_C], ebx
```

```
.text:065DC089                  jnz     short loc_65DC0D8
.text:065DC0D8 loc_65DC0D8:                             ; CODE XREF: _setConfigurePathA+19j
.text:065DC0D8                  mov     [esp+1Ch+var_18], esi
.text:065DC0DC                  mov     eax, ds:_p_libvlc
.text:065DC0E1                  mov     [esp+1Ch+var_1C], eax
.text:065DC0E4                  call    ___vlc_object_get
.text:065DC0E9                  mov     ebx, eax
.text:065DC0EB                  jmp     short loc_65DC091
.text:065DC091 loc_65DC091:                             ; CODE XREF: _setConfigurePathA+7Bj
.text:065DC091                  test    ebx, ebx
.text:065DC093                  jz      short loc_65DC0A7 ; no object retrieved?
.text:065DC095                  mov     [esp+1Ch+var_18], edi ; char *
.text:065DC099                  lea     edx, [ebx+6DCh]
.text:065DC09F                  mov     [esp+1Ch+var_1C], edx ; char *
.text:065DC0A2                  call    _strcpy         ; b0f!
```

As no bounds checks are performed, this leads to a heap-based buffer overflow.

libvlc.dll g_setStreamRecording_FullTargetPath() Function Heap Buffer Overflow

The IPCam ActiveX Plugin V1 (`axvlc.dll`) ActiveX control supports the `setStreamRecordData()` method defined as:

```
void setStreamRecordData(
                [in] BSTR strFullTargetFilename,
                [in] int nFPS);
```

The function handling the method converts the supplied '`strFullTargetFilename`' string and passes it as argument to the `g_setStreamRecording_FullTargetPath()` function exported by `libvlc.dll`.

```
.text:06518C00 ; int __stdcall VLCControl::setStreamRecordData(int pThis, BSTR
strFullTargetFilename, int nFPS)
.text:06518C00                  public __ZN10VLCControl19setStreamRecordDataEPwi@12
.text:06518C00 __ZN10VLCControl19setStreamRecordDataEPwi@12 proc near
.text:06518C00                                          ; DATA XREF: .rdata:0659965Co
.text:06518C00
.text:06518C00 pv               = dword ptr -1Ch
.text:06518C00 var_18           = dword ptr -18h
.text:06518C00 var_C            = dword ptr -0Ch
.text:06518C00 var_8            = dword ptr -8
.text:06518C00 var_4            = dword ptr -4
.text:06518C00 pThis            = dword ptr  4
.text:06518C00 strFullTargetFilename= dword ptr  8
.text:06518C00 nFPS             = dword ptr  0Ch
.text:06518C00
.text:06518C00                  sub     esp, 1Ch
.text:06518C03                  mov     eax, [esp+1Ch+strFullTargetFilename]
.text:06518C07                  mov     [esp+1Ch+var_8], ebx
.text:06518C0B                  mov     ebx, [esp+1Ch+pThis]
```

```
.text:06518C0F                  mov     [esp+1Ch+var_4], esi
.text:06518C13                  mov     ecx, [ebx+4]
.text:06518C16                  mov     edx, [ecx+58h]
.text:06518C19                  mov     [esp+1Ch+var_18], eax ; BSTR
.text:06518C1D                  mov     [esp+1Ch+pv], edx ; CodePage
.text:06518C20                  call    __Z12CStrFromBSTRjPw ; CStrFromBSTR(uint,wchar_t *)
.text:06518C25                  mov     edx, [ebx+4]
.text:06518C28                  mov     esi, eax
.text:06518C2A                  mov     ebx, [edx+54h]
.text:06518C2D                  test    ebx, ebx
.text:06518C2F                  jnz     short loc_6518C50
.text:06518C50 loc_6518C50:                             ; CODE XREF:
__ZN10VLCControl19setStreamRecordDataEPwi@12+2Fj
.text:06518C50                  mov     [esp+1Ch+pv], edx ; this
.text:06518C53                  lea     ebx, [esp+1Ch+var_C]
.text:06518C57                  mov     [esp+1Ch+var_18], ebx ; int *
.text:06518C5B                  call    __ZN9VLCPlugin12getVLCObjectEPi ;
VLCPlugin::getVLCObject(int *)
.text:06518C60                  test    eax, eax
.text:06518C62                  js      short loc_6518C31
.text:06518C64                  mov     [esp+1Ch+pv], esi
.text:06518C67                  mov     ecx, [esp+1Ch+var_C]
.text:06518C6B                  mov     [esp+1Ch+var_18], ecx
.text:06518C6F                  call    _g_setStreamRecording_FullTargetPath
```

Within this function, the string is copied straight into offset 114h of an object on the heap using `strcpy()`.

```
.text:065DC2B0 ; int __cdecl g_setStreamRecording_FullTargetPath(char *, int)
.text:065DC2B0                  public _g_setStreamRecording_FullTargetPath
.text:065DC2B0 _g_setStreamRecording_FullTargetPath proc near
.text:065DC2B0
.text:065DC2B0 var_1C           = dword ptr -1Ch
.text:065DC2B0 var_18           = dword ptr -18h
.text:065DC2B0 var_C            = dword ptr -0Ch
.text:065DC2B0 var_8            = dword ptr -8
.text:065DC2B0 var_4            = dword ptr -4
.text:065DC2B0 arg_0            = dword ptr  4
.text:065DC2B0 arg_4            = dword ptr  8
.text:065DC2B0
.text:065DC2B0                  sub     esp, 1Ch
.text:065DC2B3                  mov     [esp+1Ch+var_8], esi
.text:065DC2B7                  mov     esi, [esp+1Ch+arg_4]
.text:065DC2BB                  mov     [esp+1Ch+var_4], edi
.text:065DC2BF                  mov     edi, [esp+1Ch+arg_0]
.text:065DC2C3                  test    esi, esi
.text:065DC2C5                  mov     [esp+1Ch+var_C], ebx
.text:065DC2C9                  jnz     short loc_65DC318
.text:065DC318 loc_65DC318:                             ; CODE XREF:
_g_setStreamRecording_FullTargetPath+19j
.text:065DC318                  mov     [esp+1Ch+var_18], esi
.text:065DC31C                  mov     eax, ds:_p_libvlc
.text:065DC321                  mov     [esp+1Ch+var_1C], eax
.text:065DC324                  call    ___vlc_object_get
.text:065DC329                  mov     ebx, eax
.text:065DC32B                  jmp     short loc_65DC2D1
```

```
.text:065DC2D1 loc_65DC2D1:                            ; CODE XREF:
_g_setStreamRecording_FullTargetPath+7Bj
.text:065DC2D1                 test    ebx, ebx
.text:065DC2D3                 jz      short loc_65DC2E7
.text:065DC2D5                 mov     [esp+1Ch+var_18], edi ; char *
.text:065DC2D9                 lea     edx, [ebx+114h]
.text:065DC2DF                 mov     [esp+1Ch+var_1C], edx ; char *
.text:065DC2E2                 call    _strcpy         ; b0f!
```

As no bounds checks are performed, this leads to a heap-based buffer overflow.

Solution

Upgrade to Moxa SoftCMS version 1.4.

References

RBS:	RBS-2015-004[2]
VulnDB:	126784, 126785, 126786, 126787, 126788, 126789, 126790
CVE:	CVE-2015-6457, CVE-2015-6458
ICS-CERT:	ICSA-15-239-01[3]

Timeline

2015-05-09	Vulnerabilities discovered.
2015-05-11	Vulnerabilities submitted to ZDI.
2015-05-20	Vulnerabilities accepted by ZDI.
2015-05-28	Vulnerabilities reported to the vendor by ZDI.
2015-06-01	Fix silently released by the vendor.
2015-08-27	Advisory published by ICS-CERT. Details made available to RBS VulnDB customers.
2015-09-08	Public disclosure by ZDI.
2015-12-22	Publication of this vulnerability report.

[2] https://www.riskbasedsecurity.com/research/RBS-2015-004.pdf
[3] https://ics-cert.us-cert.gov/advisories/ICSA-15-239-01

About Risk Based Security

Risk Based Security offers clients fully integrated security solutions, combining real-time vulnerability and threat data, as well as the analytical resources to understand the implications of the data, resulting in not just security, but the right security.

Company History

Risk Based Security, Inc. (RBS) was established in early 2011 to better support the many users and initiatives of the Open Security Foundation - including the OSVDB and DataLossDB projects. RBS was created to transform this wealth of security data into actionable information by enhancing the research available, and providing a first of its kind risk identification and evidence-based security management service.

As a data driven and vendor neutral organization, RBS is able to deliver focused security solutions that are timely, cost effective, and built to address the specific threats and vulnerabilities most relevant to the organizations we serve. We not only maintain vulnerability and data breach databases, we also use this information to inform our entire practice.

Solutions

VulnDB - Vulnerability intelligence, alerting, and third party library tracking based on the largest and most comprehensive vulnerability database. Available as feature-rich SaaS portal or powerful API

Cyber Risk Analytics - Extensive data breach database including interactive dashboards and breach analytics. Clients are able to gather and analyze security threat and data breach information on businesses, industries, geographies, and causes of loss.

YourCISO - Revolutionary service that provides organizations an affordable security solution including policies, vulnerability scans, awareness material, incident response, and access to high quality information security resources and consulting services.

Vulnerability Assessments (VA) and Pentesting - Regularly scheduled VAs and pentests help an organization identify weaknesses before the bad guys do. Managing the most comprehensive VDB puts us in a unique position to offer comprehensive assessments, combining the latest in scanning technology and our own data. Detailed and actionable reports are provided in a clear and easy to understand language.

Security Development Lifecycle (SDL) - Consulting, auditing, and verification specialized in breaking code, which in turn greatly increases the security of products.