RBS-2015-002

EverFocus Electronics Corp
EverNet3 ActiveX Control Multiple Vulnerabilities
Table of Contents

Table of Contents 2
About Risk Based Security 3
  Company History 3
  Solutions 3
Vulnerable Program Details 4
References 4
Credits 4
Vulnerability Details 5
  DestroyOcx() Method Uninitialized Value Use 5
  MoveWindow() Method Uninitialized Value Use 7
  ProbeDevice() Method Stack Buffer Overflow 8
  ReadUnicodeText() Method Stack Buffer Overflow 10
  SendHttpRequest() Method Buffer Overflows 11
  SendHttpRequest2() Method Heap Buffer Overflow 13
  SendHttpRequestEx() Method Buffer Overflows 14
  SetRegedit() Method Stack Buffer Overflows 16
  SetROIEraser() Method Arbitrary Code Execution 17
  SetUnicodeFontInfo() Method Stack Buffer Overflow 20
  String Encoding Routine Heap Buffer Overflow 21
  RemoveMultiClient() Method Arbitrary Code Execution 24
  CreateObjectSizeGrids() Method Stack Buffer Overflow 25
Solution 28
Timeline 28
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 Lifecyle (SDL) - Consulting, auditing, and verification specialized in breaking code, which in turn greatly increases the security of products.
Vulnerable Program Details

Details for tested products and versions:

Vendor: EverFocus Electronics Corporation
Product: EZN3260 and EDN3340
Version: 1.2.3-130815

Component: EverNet3 ActiveX Control (EverNet3.ocx)
Version: 1.0.2.48

NOTE: It is suspected that all IP cameras in the EverFocus EZN and EDN product lines bundle the vulnerable ActiveX control and possibly any products in the NeVio series. Other models and versions than the ones listed above are likely affected.

References

RBS: RBS-2015-002
OSVDB / VulnDB: 116603, 116604, 116605, 116606, 116607, 116608, 116609, 116610, 116611, 116612, 116613, 116614, 116615, 116616, 116617, 116618, 116619, and 116620
CVE: N/A

Credits

Carsten Eiram, Risk Based Security

Twitter: @CarstenEiram
Twitter: @RiskBased

Vulnerability Details

IP cameras in the EverFocus EZN and EDN product lines provide a web-based interface for e.g. configuring device settings and streaming footage. When accessed for the first time using Internet Explorer, an ActiveX control, EverNet3.ocx, is installed on the client system. This bundled ActiveX control has been determined to be affected by a large number of critical vulnerabilities, all allowing attackers to compromise the client system when viewing a malicious web page.

DestroyOcx() Method Uninitialized Value Use

The ActiveX control supports the DestroyOcx() method, which accepts no arguments. When calling the method prior to properly initializing the ActiveX control, the this pointer references an only partially initialized CEverNet3Ctrl object.

Later in the function, some of these uninitialized values are used to make branching decisions. More importantly, an uninitialized value is eventually treated as an object pointer and dereferenced in a virtual function call. This allows gaining control of the execution flow.
MoveWindow() Method Uninitialized Value Use

The ActiveX control supports the `MoveWindow()` method, which accepts four arguments:

```c
void MoveWindow(
    long x,
    long y,
    long divX,
    long divY);
```

When calling the method prior to proper initialization of the ActiveX control, the `this` pointer references an only partially initialized `CEverNet3Ctrl` object, which is eventually used when
calling another function.

Within the called function some of the uninitialized values in memory may be used e.g. as an object pointer dereferenced in a virtual function call. This allows gaining control of the execution flow.
**ProbeDevice() Method Stack Buffer Overflow**

The ActiveX control supports the `ProbeDevice()` method, which accepts five arguments:

```c
long ProbeDevice(
    BSTR szID,
    BSTR szIp,
    BSTR szUsr,
    BSTR szPwd,
    BSTR szXmlCmd);
```

Before using the “`szXmlCmd`” argument, it is passed to a conversion function as source string along with a destination pointer to a 32,772 byte stack buffer.

```c
... mov eax, [ebp+pwzXmlCmd]
push eax                      ; pwzXmlStr
lea   ecx, [ebp+szDstBuf]     ; char[32772]
push ecx                      ; pszDstBuf
mov   ecx, [ebp+pThis]
call ConvertXmlString ; b0f!
```

The conversion function only accepts two arguments: Pointers to source and destination buffers. It calculates the length of the source buffer and uses this as a while loop counter. For each iteration a character is retrieved from the source buffer, converted if required, and written to the destination buffer.
As the conversion function is not supplied the size of the destination buffer to perform bounds checks, it is the responsibility of the calling function to ensure that the destination buffer is properly sized prior to conversion. However, in this case no such checks are performed, leading to a stack-based buffer overflow. This allows gaining control of the execution flow.
**ReadUnicodeText() Method Stack Buffer Overflow**

The ActiveX control supports the `ReadUnicodeText()` method, which accepts a single argument:

```c
long ReadUnicodeText(BSTR strText);
```

When the method is called, a debug string is created by copying the `"strText"` argument to a 30 wchar stack buffer using `_swprintf()`.

```c
As no bounds checks are performed, this allows triggering a stack-based buffer overflow and gaining control of the execution flow.
```
SendHttpRequest() Method Buffer Overflows

The ActiveX control supports the `SendHttpRequest()` method, which accepts five arguments:

```c
long SendHttpRequest(
    BSTR szID,
    BSTR szIp,
    BSTR szUsr,
    BSTR szPwd,
    BSTR szXmlCmd);
```

Prior to being used, the "szXmlCmd" argument is passed as source string to the conversion function previously described in the "ProbeDevice() Method Stack Buffer Overflow" section along with a destination pointer to a heap buffer. As no bounds checks are performed prior to calling the function, a heap-based buffer overflow may occur.

```asm
.text:100132D0 ; int __stdcall M_SendHttpRequest(int, wchar_t *, wchar_t *, wchar_t *, wchar_t *)
.text:100132D0 M_SendHttpRequest proc near           ; DATA XREF: .rdata:1007A9580
.text:100132D0 ...
.text:100132D0 wzDest = word ptr -22Ch          ; wchar[256]
.text:100132D0 var_2C = dword ptr -2Ch
.text:100132D0 var_28 = dword ptr -28h
.text:100132D0 var_24 = word ptr -24h
.text:100132D0 var_20 = dword ptr -20h
.text:100132D0 var_1C = byte ptr -1Ch
.text:100132D0 var_10 = dword ptr -10h
.text:100132D0 var_1C = dword ptr -0Ch
.text:100132D0 var_4 = dword ptr -4
.text:100132D0 arg_0 = dword ptr 8
.text:100132D0 arg_4 = dword ptr 0Ch
.text:100132D0 arg_8 = dword ptr 10h
.text:100132D0 arg_1C = dword ptr 14h
.text:100132D0 pwzXmlCmd = dword ptr 18h

... .text:1001338D mov edx, [ebp+pwzXmlCmd]
.text:1001339A push edx
.text:1001339B mov eax, [ebp+var_10]
.text:1001339E add eax, 10h
.text:10013391 push eax
.text:10013392 mov ecx, [ebp+pThis]
.text:10013398 call ConvertXmlString ; b0f!
```

2015-07-31
If a buffer overflow is not triggered within the conversion function, the converted string is eventually passed to another function to craft an HTTP request to `/cgi-bin/cgiStream` on a specified IP camera.

The called function creates the parameters as part of the request ("video=2&xml=`%s`"), passing the converted XML command as value to the "xml" parameter. As the string is written to a 16,388 byte stack buffer using `sprintf()` without performing any boundary checks, this may lead to a stack-based buffer overflow and gaining control of the execution flow.
SendHttpRequest2() Method Heap Buffer Overflow

The ActiveX control supports the SendHttpRequest2() method, which accepts eight arguments:

```c
long SendHttpRequest2(
    short method,
    BSTR szIp,
    BSTR szPort,
    BSTR szUsr,
    BSTR szPwd,
    BSTR szXmlCmd,
    BSTR szXmlUrl,
    BSTR szFileName);
```

Prior to being used, the “szXmlCmd” argument is passed as source string to the conversion function previously described in the “ProbeDevice() Method Stack Buffer Overflow” section along with a destination pointer to a heap buffer. As no bounds checks are performed prior to calling the function, a heap-based buffer overflow may occur.
SendHttpRequestEx() Method Buffer Overflows

The ActiveX control supports the SendHttpRequestEx() method, which accepts 13 arguments:

```c
long SendHttpRequestEx(
    short sMethod,
    short sBehave,
    BSTR szID,
    BSTR szIp,
    BSTR szPort,
    short sEncode,
    BSTR szUsr,
    BSTR szPwd,
    BSTR szXmlHeader,
    BSTR szXmlCmd,
    BSTR szXmlUrl,
    BSTR szFileName1,
    BSTR szFileName2);
```

When the method is called, a debug string ("SendHttpRequest: id=%s ip=%s usr=%s pwd=%s xml=%s") is crafted based on five of the supplied arguments: "szID", "szIp", "szUsr", "szPwd", and "szXmlUrl".

```
.text:10013580 ; int __stdcall M_SendHttpRequestEx(__int16 sMethod, __int16 sBehave, int pwzID, wchar_t *pwzIp, wchar_t *pwzPort, __int16 sEncode, LPCWSTR pwzUsr, LPCWSTR pwzPwd, wchar_t *pwzXmlHeader, wchar_t *pwzXmlCmd, wchar_t *pwzXmlUrl, wchar_t *pwzFileName1, wchar_t *pwzFileName2)
```

```
.text:10013580 M_SendHttpRequestEx proc near ; DATA XREF: .rdata:1007B2180
```

```
...  
.text:10013580 wzDest = word ptr -234h ; wchar[256]
.text:10013580 var_34 = dword ptr -34h
.text:10013580 var_30 = word ptr -30h
.text:10013580 var_2C = dword ptr -2Ch
.text:10013580 var_28 = byte ptr -28h
.text:10013580 var_1C = dword ptr -1Ch
.text:10013580 var_18 = byte ptr -18h
.text:10013580 var_14 = dword ptr -14h
.text:10013580 var_10 = byte ptr -10h
.text:10013580 var_C = dword ptr -0Ch
.text:10013580 var_4 = dword ptr -4
.text:10013580 sMethod = word ptr 8
.text:10013580 sBehave = word ptr 0Ch
.text:10013580 pwzID = dword ptr 10h
```
As the string is written into a 256 wchar stack buffer using _swprintf() without performing any boundary checks, this leads to a stack-based buffer overflow and allows gaining control of the execution flow.

If a buffer overflow is not triggered, the code eventually passes the “szXmlUrl” argument as source string to the conversion function previously described in the “ProbeDevice() Method Stack Buffer Overflow” section along with a destination pointer to a 8,192 byte heap buffer. As no bounds checks are performed prior to calling the function, a heap-based buffer overflow may occur.
Similar vulnerable conversion occurs for the “szXmlHeader” and “szXmlCmd” arguments, also leading to heap-based buffer overflows.

```
.text:100136A2    push  eax           ; pwzXmlStr
.text:100136A3    mov   ecx, [ebp+var_1C]
.text:100136A6    add   ecx, 2010h
.text:100136AC    push  ecx           ; pszDstBuf
.text:100136AD    mov   ecx, [ebp+pThis]
.text:100136B3    call  ConvertXmlString ; b0f!
.text:100136B8    mov   edx, [ebp+pwzXmlCmd]
.text:100136BB    push  edx           ; pwzXmlStr
.text:100136BC    mov   eax, [ebp+var_1C]
.text:100136BF    add   eax, 10h
.text:100136C2    push  eax           ; pszDstBuf
.text:100136C3    mov   ecx, [ebp+pThis]
.text:100136C9    call  ConvertXmlString ; b0f!
```

**SetRegedit() Method Stack Buffer Overflows**

The ActiveX control supports the `SetRegedit()` method, which accepts three arguments:

```c
short SetRegedit(
    BSTR szPath,
    BSTR szName,
    BSTR szValue);
```

When called, the responsible function, `CEverNet3Ctrl::SetRegedit()`, creates the full path for the registry key to set ("Software\Ext\Nevio\%s") based on the supplied "szPath" argument.

```
.text:10018AE0 ; int __stdcall M_SetRegedit(int pwzPath, LPCWSTR pwzValueName, wchar_t *pwzValue)
.text:10018AE0 M_SetRegedit  proc near           ; DATA XREF: .rdata:1007AAB80
.text:10018AE0
.text:10018AE0 var_218   = dword ptr -218h
.text:10018AE0 dwDisposition = dword ptr -214h
.text:10018AE0 wzDest    = word ptr -210h         ; wchar[258]
.text:10018AE0 var_C    = dword ptr -0Ch
.text:10018AE0 var_8    = dword ptr -8
.text:10018AE0 phkResult = dword ptr -4
.text:10018AE0 pwzPath   = dword ptr 8
.text:10018AE0 pwzValueName = dword ptr 0Ch
.text:10018AE0 pwzValue  = dword ptr 10h
.text:10018AE0
.text:10018AE0 push  ebp
.text:10018AE1 mov   ebp, esp
.text:10018AE3 sub   esp, 218h
```

2015-07-31  16 of 28
As the path is written into a 258 wchar buffer using _swprintf() without performing any boundary checks, this allows triggering a stack-based buffer overflow and gaining control of the execution flow.

Later in the function a similar problem exists for the “szValue” argument when copied into a 258 wchar stack buffer.

SetROIerasure() Method Arbitrary Code Execution

The ActiveX control supports the SetROIerasure() method, which accepts two arguments:

```c
void SetROIerasure(
    short sCh,
    short sSeq);
```

When calling the method prior to properly initializing the ActiveX control, the this pointer references an only partially initialized CEverNet3Ctrl object. This results in uninitialized memory being used in various locations within the function depending on the value of the “sSeq” argument.
In cases where “sSeq” is not 99, the code branches to the following location where uninitialized memory is used as an object pointer when calling a virtual function.

```assembly
.text:1002308D loc_1002308D: ; CODE XREF: M_SetROIEraser+53j
.text:1002308D movsx eax, [ebp+sSeq]
.text:10023091 cmp eax, 63h
.text:10023094 jnz loc_1002316C ; sSeq != 99?
```

```assembly
.text:1002316C
.text:1002316C loc_1002316C: ; CODE XREF: M_SetROIEraser+74j
.text:1002316C movsx edx, [ebp+sSeq]
.text:10023170 shl edx, 2
.text:10023173 mov [ebp+var_1C], dx
.text:10023177 push offset Data
.text:1002317C push 0
.text:1002317E push 0
.text:10023180 push 0
.text:10023182 push 0
.text:10023184 push 0
.text:10023186 push 0
.text:10023188 push 0
.text:1002318A push 0
.text:1002318C movsx eax, [ebp+sSeq]
.text:10023190 add eax, 0CDh
.text:10023195 push eax
.text:10023196 mov ecx, [ebp+lpoCUnk1]
.text:10023199 mov edx, [ecx] ; read from unit mem
.text:1002319B mov eax, [ebp+lpoCUnk1]
.text:1002319E push eax
.text:1002319F mov ecx, [edx+0F8h]
.text:100231A5 call ecx
```
If “sSeq” is 99, the code branches to the following location where a similar thing happens.

Both of these cases allows gaining control of the execution flow.

Another problem exists in the function when handling the “sCh” argument, since it’s sign-extended and used as index into an array without performing any bounds checks. This eventually leads to the referenced memory being treated as an object pointer and stored in a local variable.
Depending on the value of the “sSeg” argument, as mentioned in the previous section, one of two code locations may eventually dereference the value when calling a virtual function.

```
.text:10023196      mov    ecx, [ebp+1lp0CUnk1]
.text:10023199      mov    edx, [ecx]
.text:1002319B      mov    eax, [ebp+1lp0CUnk1]
.text:1002319E      push   eax
.text:1002319F      mov    ecx, [edx+0F8h]
.text:100231A5      call   ecx
```

And...

```
.text:100230EC      mov    ecx, [ebp+1lp0CUnk1]
.text:100230EF      mov    edx, [ecx]
.text:100230F1      mov    eax, [ebp+1lp0CUnk1]
.text:100230F4      push   eax
.text:100230F5      mov    ecx, [edx+0F8h]
.text:100230FB      call   ecx
```

This allows gaining control of the execution flow.

**SetUnicodeFontInfo() Method Stack Buffer Overflow**

The ActiveX control supports the `SetUnicodeFontInfo()` method, which accepts four arguments:

```c
long SetUnicodeFontInfo(
    short sFontH,
    short sIsShift,
    long lFontSize,
    BSTR strFontType);
```

When the method is called, the “strFontType” argument is copied into a 30 wchar stack buffer using `_swprintf()` without performing any boundary checks.
This leads to a stack-based buffer overflow and allows gaining control of the execution flow.

**String Encoding Routine Heap Buffer Overflow**

The ActiveX control supports the `SimpleEncDecStr()` method, which accepts two arguments:

```
BSRT SimpleEncDecStr(
    short method,
    BSTR strName);
```

When the `SimpleEncDecStr()` method is called, the string supplied via the “strName” argument is converted using `WideCharToMultiByte()` into a 128 byte destination buffer.
Checks then confirm that the value passed via the “method” argument is either 0 or 1. This defines whether the function should decode (0) or encode (1) the supplied string. In case the value is 1, the function calls SimpleEncStr() to encode the string.

Inside SimpleEncStr() the length of the string is calculated using strlen(). Memory is then allocated based on the length * 4 + 2. This is to ensure sufficient room for each byte being encoded into 4 bytes along with the NULL terminator.
The encoding routine is then entered. The string is encoded one byte at a time and the result appended to the previously allocated output buffer. The first step of the encoding process changes the value of the current byte based on its position in the string. This resulting byte value is then sign-extended and passed to `swprintf()` using the "%.2x" format specifier before being written to the output buffer using `wcscat()`.
The developer seemingly expected the precision specifier to limit the output to two wide-characters, but precision used with the 'X' specifier specifies the \textit{minimum} number of digits. As the encoded byte value is sign-extended, the result of calling \texttt{swprintf()} with the ".2x" format specifier may be a 16 byte string. As the heap-based buffer was allocated on the assumption that a byte is encoded to 4 bytes, a heap-based buffer overflow can be triggered.

\textit{RemoveMultiClient()} Method Arbitrary Code Execution

The ActiveX control supports the \texttt{RemoveMultiClient()} method, which accepts a single argument:

\begin{verbatim}
void RemoveMultiClient(short sChannel);
\end{verbatim}

Two problems exist in the \texttt{CEverNet2Ctrl::RemoveMultiClient()} function handling calls to this method. Firstly, the supplied "sChannel" argument is sign-extended and used as an index into an array in the \texttt{CEverNet2Ctrl} object to retrieve an object pointer without performing any bounds checks.

Secondly, when calling the method without properly initializing the control, the mentioned array contains uninitialized values that are treated as object pointers and dereferenced when performing a virtual function call.

Both cases allow gaining control of the execution flow.
These issues are also triggerable via the `CEverNet2Ctrl::AddFilledClient()` function handling calls to the `AddFilledClient()` method as well as the `CEverNet2Ctrl::AddMultiClient()` function handling calls to the `AddMultiClient()` method. In both cases, the methods also accept a "sChannel" argument eventually passed to the `CEverNet2Ctrl::RemoveMultiClient()` function.

```c
short AddFilledClient(
    short sChannel,
    BSTR szSource,
    BSTR szEncUsr,
    BSTR szEncPwd);

short AddMultiClient(
    short sChannel,
    BSTR szSource,
    BSTR szEncUsr,
    BSTR szEncPwd);
```

**CreateObjectSizeGrids() Method Stack Buffer Overflow**

The ActiveX control supports the `CreateObjectSizeGrids()` method, which accepts five arguments:

```c
short CreateObjectSizeGrids(
    short sCh,
    short sEnable,
    short sWNum,
    short sHNum,
    BSTR lpSelGrid);
```

When the method is called, a debug string is crafted based on the supplied arguments.
As the string is written into a 258 wchar stack buffer using _swprintf() without performing any boundary checks, it’s possible to trigger a stack-based buffer overflow and gain control of the execution flow.

CreateAreaLine() Method Uninitialized Value Use Arbitrary Code Execution

The ActiveX control supports the CreateAreaLine() method, which accepts six arguments:

```plaintext
short CreateAreaLine(
    short sCh,
    short sEnable,
    short sColor,
    short sPenBold,
    short sWNum,
    short sHNum);
```

When calling the method prior to properly initializing the ActiveX control, the this pointer references an only partially initialized CEverNet3Ctrl object. Eventually, the code checks the value of the supplied “sCh” argument.
If not set to 99, two additional checks ensure that the sign-extended result is neither negative nor 4 or greater.

If passing the check, the "sCh" argument is used as index into an array inside the CEverNet3Ctrl object. However, the values in this array may be uninitialized, resulting in one of these being treated as an object pointer and dereferenced when calling a virtual function. This allows gaining control of the execution flow.
If the “sCh” argument was set to 99, another similar dereference of an uninitialized value occurs.

```
.text:1001F67D   mov   cx, [ebp+shNum]
.text:1001F681   push  ecx
.text:1001F682   mov   dx, [ebp+shNum]
.text:1001F686   push  edx
.text:1001F687   mov   ax, [ebp+spenBold]
.text:1001F68B   push  eax
.text:1001F68C   mov   cx, [ebp+scolor]
.text:1001F690   push  ecx
.text:1001F691   mov   dx, [ebp+sEnable]
.text:1001F695   push  edx
.text:1001F696   mov   eax, [ebp+iCount]
.text:1001F699   push  edx
.text:1001F69C   mov   edx, [ecx+eax*4+0B064h]
.text:1001F6A3   mov   eax, [ebp+iCount]
.text:1001F6A6   mov   ecx, [ebp+lpThis]
.text:1001F6A9   mov   eax, [ecx+eax*4+0B064h]
.text:1001F6B0   mov   ecx, [edx]
.text:1001F6B2   push  eax
.text:1001F6B3   mov   edx, [ecx+0F4h]
.text:1001F6B9   call  edx
```

Solution

The vendor was not responsive, and we are not aware of any available fix for these vulnerabilities.

Timeline

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-10-23</td>
<td>Vulnerabilities discovered.</td>
</tr>
<tr>
<td>2014-12-08</td>
<td>Emailed vendor to obtain details for security contact.</td>
</tr>
<tr>
<td>2015-01-01</td>
<td>OSVDB entries published and details made available on VulnDB².</td>
</tr>
<tr>
<td>2015-07-31</td>
<td>Publication of this vulnerability report.</td>
</tr>
</tbody>
</table>