From BinDiff to Zero-Day: A Proof of Concept Exploiting CVE-2019-1208 in Internet Explorer

Technical Brief
Last June, I disclosed a use-after-free (UAF) vulnerability in Internet Explorer (IE) to Microsoft. It was rated as critical, designated as CVE-2019-1208, and then addressed in Microsoft’s September Patch Tuesday. I discovered this flaw through BinDiff (a binary code analysis tool) and wrote a proof of concept (PoC) showing how it can be fully and consistently exploited in Windows 10 RS5.

UAF vulnerabilities like CVE-2019-1208 are a class of security flaws that can corrupt valid data, crash a process, and, depending on when it is triggered, can enable an attacker to execute arbitrary or remote code. In the case of CVE-2019-1208, an attacker successfully exploiting this vulnerability could gain the same rights as the current user in the system. If the current user has administrative privileges, the attacker can hijack the affected system — from installing or uninstalling programs and viewing and modifying data to creating user accounts with full privileges.

A more tangible attack scenario would entail attackers sending socially engineered phishing emails to unwitting users and tricking them into accessing a malicious website (containing an exploit for CVE-2019-1208) via Internet Explorer. Alternatively, an attacker can send spam emails with attachments containing an exploit for the vulnerability. These attachments can be a Microsoft Office document that has the IE rendering engine enabled, or application files embedded with an ActiveX control that, in turn, contains an exploit for the vulnerability. Attackers could also compromise and host an exploit on legitimate websites, like those that accept content or input (i.e., advertisements) from users.

**Starting from BinDiff**

When using BinDiff to compare changes and updates made on vbscript.dll between May and June, there are some interesting changes in the functions VbsJoin and VbsFilter. As shown in Figure 2, the functions rtJoin and rtFilter are surrounded with SafeArrayAddRef, SafeArrayReleaseData, and SafeArrayReleaseDescriptor, which seems to be for addressing a bug.

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Confidence</th>
<th>Address</th>
<th>Primary Name</th>
<th>Type</th>
<th>Address</th>
<th>Secondary Name</th>
<th>Type</th>
<th>Basic Blocks</th>
<th>Jumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.83</td>
<td>0.95</td>
<td>1600101C</td>
<td>VbsJoin</td>
<td>Normal</td>
<td>1600101C</td>
<td>VbsJoin</td>
<td>Normal</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>0.86</td>
<td>0.95</td>
<td>1600101C</td>
<td>VbsFilter</td>
<td>Normal</td>
<td>1600101C</td>
<td>VbsFilter</td>
<td>Normal</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>0.44</td>
<td>0.99</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0.99</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0.99</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0.99</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0.99</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0.99</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0.99</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>102128C</td>
<td>rtJoin</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0.99</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>102128C</td>
<td>rtFilter</td>
<td>Normal</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 1. Snapshot of updates made in vbscript.dll between May and June via BinDiff](image-url)
Figure 2. Differences in VbsJoin (top) and VbsFilter (bottom)
SAFEARRAY specifies a multidimensional array of OLE Automation types. Here is the syntax of SAFEARRAY:

```c
typedef struct tagSAFEARRAY {
    USHORT cDims;
    USHORT fFeatures;
    ULONG cbElements;
    ULONG cLocks;
    PVOID pvData;
    SAFEARRAYBOUND rgsabound[1];
} SAFEARRAY;
```

The following are the functions:

- `cDims` — Specifies the number of dimensions
- `fFeatures` — Specifies the feature of an array, like how it is allocated or which element it saves
- `cbElements` — Specifies the size of an array element (generally, the size is 10 bytes)
- `cLocks` — Saves the number of times the array has been locked without a corresponding unlock
- `pvData` — Saves the pointer to the array buffer
- `rgsabound` — A SAFEARRAYBOUND structure that saves the bound information for each dimension

Here is the syntax of SAFEARRAYBOUND:

```c
typedef struct tagSAFEARRAYBOUND {
    ULONG cElements;
    LONG lLbound;
} SAFEARRAYBOUND, *LPSAFEARRAYBOUND;
```

The following are the functions:

- `cElements` — Specifies the number of elements in the dimension
- `lLbound` — Specifies the lower bound of the dimension

Figure 3 shows a simple example of the memory layout of one SAFEARRAY:

```
0:000> dd 15778fe8  L8
15778fe8  09200010 00000010 00000000 1577afd0
dim arr(2)
1577ff0  00000000 00000000 00000000 00000000
arr(0) = &h11223344
0:000> dt ole32!safearray 15778fe0
+0x000  cDims : 1
+0x002  fFeatures : 0x892
+0x004  cbElements : 0x10
+0x008  cLocks : 0
+0x00c  pvData : 0x1577afd0 Void
+0x010  rgsabound : [1] tagSAFEARRAYBOUND
0:000> dd 0x1577afd0 Lc
1577af0  c0000003 c0000009 11233444 c0000009
1577afe0  c0000003 c0000009 55667788 c0000009
1577aff0  00000000 00000000 00000000 00000000
```

Figure 3. Example of a memory layout of SAFEARRAY

From the syntax of SAFEARRAY, it can be seen that SafeArray doesn’t have a reference count attribute. Therefore, the functions SafeArrayAddRef, SafeArrayReleaseData, and SafeArrayReleaseDescriptor add the ability to use reference counting to pin the SafeArray into memory before calling from an untrusted
script into an **IDispatch** method that may not expect the script to free that memory before the method returns.

**SafeArrayAddRef** increases the pinning reference count of the descriptor for the specified SafeArray by one. The pinning reference count of the data for the specified SafeArray may increase by one if that data was dynamically allocated, as determined by the descriptor of the SafeArray (shown in Figure 3). **SafeArrayReleaseData** decreases the pinning reference count for the specified SafeArray data by one. When its reference count reaches 0, the memory for that data is no longer prevented from being freed. **SafeArrayReleaseDescriptor** decreases the pinning reference count for the descriptor of the specified SafeArray by one. When its reference count reaches 0, the memory for that descriptor is no longer prevented from being freed.

![Code snippets of SafeArrayAddRef (top), SafeArrayReleaseData (center), and SafeArrayReleaseDescriptor (bottom)](image_url)

Figure 4. Code snippets of SafeArrayAddRef (top), SafeArrayReleaseData (center), and SafeArrayReleaseDescriptor (bottom)
When these are all put together, with a focus on the reference count addition/subtraction operation of SafeArrayDescriptor and SafeArrayData, a code flow can be generated, as shown in Figure 5. VbsJoin is used as an example.

There seems to be no problem about the reference count addition/subtraction operation of SafeArrayDescriptor and SafeArrayData in native code. But if a VBScript callback in rJoin is made and the reference count of SafeArrayDescriptor and SafeArrayData is modified by script, will this code flow still have no problem?
Giving a callback in rtJoin

Inspired by the previous vulnerability (CVE-2018-8373) I found in 2018, I used VBScriptClass' 'Public Default Property Get' function to give me a callback in VbsJoin. Figure 6 shows the PoC.

<script type="text/vbscript">
Dim arr

Class MyClass
Public Default Property Get P
    arr = Array(0)
End Property
End Class

arr = Array(New MyClass)
'crash
arr(0) = Join(arr)
</script>

Figure 6. Initial PoC of CVE-2019-1208

This vulnerability's trigger flow can be simplified through these steps, as shown in Figure 7:

1. **arr = Array(New MyClass)** — Create a SafeArray and save the VBScriptClass: MyClass in arr[0];
2. **Callback: arr = Array(0)** — Join(arr) will trigger MyClass 'Public Default Property Get' function callback. In this callback, create a new SafeArray to variant arr and, as shown in Figure 7, this new SafeArray is not protected by function SafeArrayAddRef. Thus, the normal code flow assumption in Figure 5 is broken by this callback, meaning something will go wrong later.
3. **arr(0) = Join(arr)** — When back from the 'Public Default Property Get' callback, the code flow in VbsJoin will call SafeArrayReleaseData and SafeArrayReleaseDescriptor to decrease the reference count of SafeArrayData and SafeArrayDescriptor. But since the new SafeArray is not protected by SafeArrayAddRef, the reference count of SafeArrayData and SafeArrayDescriptor is 0. Therefore, the new SafeArray's SafeArrayData and SafeArrayDescriptor will be freed in the functions SafeArrayReleaseData and SafeArrayReleaseDescriptor.

When saving the VbsJoin return value to arr(0), however, the PoC crashes in vbscript!AccessArray because the SafeArrayDescriptor is freed (shown in Figure 8) and the Variant arr still saves the pointer of the freed SafeArrayDescriptor.
Figure 7. Code snippets showing: arr = Array(New MyClass) in memory (top); arr = Array(0) in memory (center); and the callback (highlighted, bottom)
Figure 8. Code snippet showing the initial PoC crashing in vbscript!AccessArray

From limited UAF to Read/Write primitive

From the previous process, a dangling pointer `arr` results from the PoC. However, it is a limited UAF because the dangling pointer `arr` must point to SafeArrayDescriptor structure (see Figure 3) and the free memory is 0x18 bytes. Some data is needed to reuse the freed 0x18 bytes memory hole and make a fake SafeArrayDescriptor like this: 08800001 00000001 00000000 00000000 7fffffff 00000000.

The data structure I chose is basic string/binary string (BSTR). But it doesn’t work when BSTR is used to occupy the freed 0x18 bytes because the memory size of BSTR is a multiple of 0x10 bytes (shown in Figure 9).

Figure 9. Code snippet of oleaut32!SysAllocStringLen
The memory size of BSTR should be 0x10 bytes, 0x20 bytes, or 0xX0, and cannot be 0x18 bytes. However, I was still able to make 0x20 bytes of freed memory and get it reused by BSTR.

As already mentioned, SafeArray is a multidimensional array. The offset 0x10 of SAFEARRAY is an array that saves every SAFEARRAYBOUND structure of the dimensions. For example, a one-dimensional array has one SAFEARRAYBOUND structure whose memory size is 0x8 bytes while a two-dimensional array has two SAFEARRAYBOUND structures whose memory size is 0x10 bytes. Hence, the memory size of two-dimensional SafeArray is 0x20 bytes. Since VbsJoin can process only one-dimensional arrays, I tried to change the SafeArray dimensions in the callback, as shown in a modified PoC in Figure 10. Unfortunately, that doesn’t work. It throws a runtime error saying the array type does not match in Join. Because VbsJoin can process only a one-dimensional array, there will be a runtime error even if the arr in the two-dimensional array in the callback is modified.

```vbs
<meta http-equiv="x-ua-compatible" content="IE=10">
<script type="text/vbscript">
  Dim arr
  Class MyClass
  Public Default Property Get P
    ReDim arr(1, 1)
  End Property
  End Class
  ReDim arr(1)
  Set arr(0) = New MyClass
  arr(0) = Join(arr)
</script>
```

![Figure 10. Snippets showing the PoC with the modified SafeArray dimensions (top) and the runtime error (bottom)](image)

To bypass the runtime error, I used **On Error Resume Next**, which specifies that when a runtime error occurs, control immediately goes to the statement where the error occurred, and execution continues from that point. Using On Error Resume Next bypassed the runtime error and resulted in a 0x20-byte dangling pointer arr, as shown in Figure 11.
After getting 0x20 bytes of freed memory, I used BSTR with a size of 0x20 bytes to fake a big-size SafeArray:

```
Unescape("%u4141%u4141%u4141%u4141%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u0000%u0000%u7fff%u0000%u0000")
```

By using heap feng shui, this BSTR can stably reuse the 0x20-byte freed memory. Figure 11 (bottom) shows how I was able to get a fake, one-dimensional SafeArray whose element number is 0x7fffffff and element size is 1 byte.

So far, I have shown a fake SafeArray that can be used to read or write memory from 0x00000000 to 0x7fffffff. To leak some read/write address for exploitation, I applied Simon Zuckerbraun's research on CVE-2019-0752 (another vulnerability in IE already patched). I used heap spray in order to have some fixed read/write address (0x28281000), as shown in Figure 12. By using the fixed read/write memory address named ‘util_memory’ (0x28281000) and faked SafeArray named ‘fake_array’, the read/write memory function is easily made.
Just pop out a calculator

To demonstrate and carry out remote code execution (RCE), I used the Scripting.Dictionary object as introduced in Simon Zuckerbraun’s research. Unlike in the case of CVE-2019-0752, however, this vulnerability can’t be used to write memory 1 byte by 1 byte because every Variant in vbscript.dll occupies 0x10 bytes.
To get around this, I used BSTR to make a fake Dictionary through these steps:

1. Use read/write memory function to read the original Dictionary memory, save its data to one BSTR, and replace VBADictionary::Exists to kernel32!Winexec.
2. Write the Winexec parameter (\..\calc.exe) to this BSTR.
3. Save this BSTR to util_memory + 0x1000, and modify 'util_memory + 0x1000 - 8 = 9' to make fake_array(util_memory + 0x1000) an object.
4. Use fake_array(util_memory + &h1000).Exists "dummy" to trigger the function Winexec, as shown in Figure 13.

![Figure 13. Screenshot showing the faked Dictionary's memory layout](image-url)
What does this vulnerability mean?

On August 13, 2019, VBScript, which has already been disabled in Windows 10, was disabled for Internet Explorer 11 in Windows 7, 8, and 8.1. Therefore, the PoC detailed here was developed in local mode. But as Microsoft says, this setting can still be enabled via Registry or Group Policy. All the same, users and organizations should always adopt best practices: Keep systems patched and updated, disable components if they are not needed (or restrict use), and foster cybersecurity awareness on vectors that may be used by attackers, such as spam emails and other socially engineered threats.
TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threats techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com