Talos Vulnerability Report

TALOS-2020-1210

SoftMaker Office PlanMaker Excel document CEscherObject::ReadNativeProperties multiple heap buffer overflow vulnerabilities

FEBRUARY 3, 2021

CVE NUMBER

CVE-2020-27247, CVE-2020-27248, CVE-2020-27249,CVE-2020-27250,CVE-2020-28587

Summary

An exploitable heap-based buffer overflow vulnerability exists in the Office Art record-parsing functionality of SoftMaker Office 2021's PlanMaker application. A specially crafted document can cause the document parser to copy data from a particular record type into a static-sized buffer within an object that is smaller than the size used for the copy, which will cause a heap-based buffer overflow. An attacker can entice the victim to open a document to trigger this vulnerability.

Tested Versions

SoftMaker Software GmbH SoftMaker Office PlanMaker 2021 (Revision 1014)

Product URLs

https://www.softmaker.com/en/softmaker-office

CVSSv3 Score

8.8 - CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

CWE

CWE-122 - Heap-based Buffer Overflow

Details

SoftMaker Software GmbH is a German software company that develops office software. Their flagship product, SoftMaker Office, is supported on a variety of platforms and contains a handful of components that can allow the user to perform a multitude of tasks such as word processing, spreadsheets, presentation design, and even allows for scripting. The SoftMaker Office suite supports a variety of common office file formats, as well as a number of internal formats that the user may choose to use when performing their necessary work.

The PlanMaker application of SoftMaker's suite is designed as an all-around spreadsheet tool, and supports of a number of features that allow it to remain competitive with similar office suites that are developed by its competitors. This application includes a number of parsers that enable the user to interact with a variety of document types or templates that are common within this type of software. One supported file format, which is relevant to the vulnerability described within this document, is the Microsoft Excel file format. This format is based on Microsoft's Compound Document file format and is primarily contained within either the "Workbook" stream for later versions of the format, or the "Book" stream for earlier versions.

When opening up a Microsoft Excel Document, the following function will be executed in order to load the document. Before the document is loaded, the application will open up the file temporarily in order to fingerprint the document and determine which handler is used to load the document's contents. At [1], the application will call a function that is responsible for parsing the different streams that may be found within the document.

```
0x7ea017:    push   %rbp
0x7ea018:    mov    %rsp,%rbp
0x7ea01b:    sub    $0x260,%rsp
0x7ea022:    mov    %rdi,-0x248(%rbp)
0x7ea029:    mov    %rsi,-0x250(%rbp)   ; object
0x7ea030:    mov    %rdx,-0x258(%rbp)   ; document path
0x7ea037:    mov    %ecx,-0x25c(%rbp)
...
0x7ea24b:    mov    -0x234(%rbp),%ecx
0x7ea251:    mov    -0x258(%rbp),%rdx   ; document path
0x7ea258:    mov    -0x250(%rbp),%rsi   ; object
0x7ea25f:    mov    -0x248(%rbp),%rax
0x7ea266:    mov    %rax,%rdi
0x7ea269:    callq  0x695c7c            ; [1]
0x7ea26e:    test   %eax,%eax
0x7ea270:    setne  %al
0x7ea273:    test   %al,%al
0x7ea275:    je     0x7ea2a3
```

Eventually the following function will be called from an array of functions used to handle the different record types within the workbook stream belonging to the Microsoft Excel document. This function is given a handle to the document stream as one of its parameters in order to read records from the stream. After initializing some of the local variables within the function, the loop at [2] will be executed. This loop is directly responsible for reading a record from the stream at [3], and then looking at the record type to in order to determine how to actually parse the record.

```
0x634eeb:    push   %rbp
0x634eec:    mov    %rsp,%rbp
0x634eef:    sub    $0x90,%rsp
0x634ef6:    mov    %rdi,-0x78(%rbp)    ; object
0x634efa:    mov    %rsi,-0x80(%rbp)    ; document stream
0x634efe:    mov    %edx,-0x84(%rbp)    ; flags
0x634f04:    mov    %fs:0x28,%rax
...
0x634f92:    mov    -0x78(%rbp),%rax    ; [2] beginning of loop
0x634f96:    mov    0x88(%rax),%eax
0x634f9c:    and    $0xff000000,%eax
0x634fa1:    test   %eax,%eax
0x634fa3:    jne    0x635e02            ; exit loop
...
0x634fa9:    mov    -0x40(%rbp),%r9     ; record data
0x634fad:    lea    -0x64(%rbp),%r8     ; result code
0x634fb1:    lea    -0x6e(%rbp),%rcx    ; result record length
0x634fb5:    lea    -0x6c(%rbp),%rdx    ; result record type
0x634fb9:    mov    -0x80(%rbp),%rsi    ; stream object
0x634fbd:    mov    -0x78(%rbp),%rax    ; object
0x634fc1:    sub    $0x8,%rsp
0x634fc5:    lea    -0x68(%rbp),%rdi
0x634fc9:    push   %rdi
0x634fca:    mov    %rax,%rdi           ; object
0x634fcd:    callq  0x61e4a8            ; [3] parse the next record from the stream
0x634fd2:    add    $0x10,%rsp
0x634fd6:    mov    %rax,-0x40(%rbp)    ; record data from stream
0x634fda:    cmpq   $0x0,-0x40(%rbp)
0x634fdf:    je     0x635e01
...
0x635dfb:    nop
0x635dfc:    jmpq   0x634f92            ; [2] continue loop
```

Once successfully parsing the record within each iteration of the loop, the loop will use the record type in order to determine which handler to use for parsing the record's contents at [4]. The vulnerability described by this advisory is for record type 0x00eb which represents the `MsoDrawingGroup` record which is a container for records which conform to the Microsoft Office Art file format specification. At [5], a traversal is done to locate the correct handler for the mentioned record type. Once the handler has been determined, the stream object, the record's data, and a state is passed as the parameter to the function call at [6].

```
0x635075:    movzwl -0x6c(%rbp),%eax    ; [4] read record type
0x635079:    movzwl %ax,%eax
0x63507c:    cmp    $0xc1,%eax
0x635081:    je     0x635588
0x635087:    cmp    $0xc1,%eax
0x63508c:    jg     0x6351e8            ; [5] find record type 0x00eb
...
0x6351e8:    cmp    $0x161,%eax
0x6351ed:    je     0x6355ad
0x6351f3:    cmp    $0x161,%eax
0x6351f8:    jg     0x6352ab
0x6351fe:    cmp    $0xe2,%eax
0x635203:    jg     0x635258            ; [5] find record type 0x00eb
...
0x635258:    cmp    $0xfc,%eax
0x63525d:    je     0x635b84
0x635263:    cmp    $0xfc,%eax
0x635268:    jg     0x635285
0x63526a:    cmp    $0xea,%eax
0x63526f:    je     0x635d83
0x635275:    cmp    $0xeb,%eax          ; record type referenced by this advisory
0x63527a:    je     0x635c90            ; [5] find record type 0x00eb
...
0x635c90:    lea    -0x68(%rbp),%rcx
0x635c94:    mov    -0x40(%rbp),%rdx    ; record data
0x635c98:    mov    -0x80(%rbp),%rsi    ; stream object
0x635c9c:    mov    -0x78(%rbp),%rax    ; record object
0x635ca0:    mov    %rax,%rdi
0x635ca3:    callq  0x646136            ; [6] parse record
0x635ca8:    mov    %rax,-0x40(%rbp)
0x635cac:    jmpq   0x635dfc
```

When parsing the 0x00eb record type, the following function is responsible for aggregating related records into an object in order to identify the Office Art record types contained therein. The first thing the function will do is to assign its parameters into variables located within the function's frame. After this is done, the function will read the `uint16` length from the record header at [7], and then pass it along with a pointer to the record's contents at [8] to a function call responsible for appending the record's contents into an object at [9].

```
0x646137:    mov    %rsp,%rbp
0x64613a:    push   %r12
0x64613c:    push   %rbx
0x64613d:    sub    $0x190,%rsp
0x646144:    mov    %rdi,-0x188(%rbp)   ; record object
0x64614b:    mov    %rsi,-0x190(%rbp)   ; stream object
0x646152:    mov    %rdx,-0x198(%rbp)   ; record data from stream
0x646159:    mov    %rcx,-0x1a0(%rbp)   ; result
...
0x64617e:    mov    -0x198(%rbp),%rax   ; record data
0x646185:    movzwl 0x2(%rax),%eax      ; [7] uint16 record length
0x646189:    mov    %ax,-0x176(%rbp)    ; store it
...
0x646190:    movzwl -0x176(%rbp),%edx   ; uint16 record length
0x646197:    mov    -0x198(%rbp),%rax   ; record data from stream
0x64619e:    lea    0x4(%rax),%rcx      ; [8] shift past record header into contents
0x6461a2:    lea    -0x130(%rbp),%rax   ; aggregation object
0x6461a9:    mov    %rcx,%rsi
0x6461ac:    mov    %rax,%rdi
0x6461af:    callq  0xe4381e            ; [9] append record contents into aggregation object
0x6461b4:    movq   $0x0,-0x158(%rbp)
```

After the first record has been appended into the aggregation object, the following loop will iterate through all of the records that follow which retain the record type 0x00eb. This record type represents the `MsoDrawingGroup` record. At [10], the loop will call a function which checks if the type of the next record corresponds to the 0x00eb type. At the end of each iteration of the loop, the `uint16` record length will be stored at [11], and then used with a pointer to the contents of the record at [12] as a parameter toe the function call at [13] which will each record with the type 0x00eb to an object.

```
0x6461bf:    movl   $0x0,-0x174(%rbp)
...
0x6461c9:    lea    -0x174(%rbp),%rcx   ; result record length
0x6461d0:    mov    -0x158(%rbp),%rdx   ; current record data from stream
0x6461d7:    mov    -0x190(%rbp),%rsi   ; stream object
0x6461de:    mov    -0x188(%rbp),%rax   ; record object
0x6461e5:    mov    $0xeb,%r8d          ; look for record type 0x00eb
0x6461eb:    mov    %rax,%rdi
0x6461ee:    callq  0x620bdc            ; [10] check the record type matches
0x6461f3:    mov    %rax,-0x158(%rbp)   ; store
0x6461fa:    cmpq   $0x0,-0x158(%rbp)
0x646202:    setne  %al
0x646205:    test   %al,%al
0x646207:    je     0x646241
...
0x646209:    mov    -0x158(%rbp),%rax   ; record data
0x646210:    movzwl 0x2(%rax),%eax      ; [11] uint16 record length
0x646214:    mov    %ax,-0x176(%rbp)    ; store it
...
0x64621b:    movzwl -0x176(%rbp),%edx   ; uint16 record length
0x646222:    mov    -0x158(%rbp),%rax   ; pointer to record data from stream
0x646229:    lea    0x4(%rax),%rcx      ; [12] shift past header into record contents
0x64622d:    lea    -0x130(%rbp),%rax   ; aggregation object
0x646234:    mov    %rcx,%rsi
0x646237:    mov    %rax,%rdi
0x64623a:    callq  0xe4381e            ; [13] append record contents to aggregation object
0x64623f:    jmp    0x6461c9
```

If the record type is not of the type 0x00eb, then the following loop will be executed as an alternative case. Similar to the prior loop, this loop is responsible for aggregating all records of the 0x003c type which represents a `Continue` record into the same object. At [14], the current record data, the stream object, a record object, and the desired 0x003c type is passed to a function call which filters the records being iterated through by the loop for the desired record type. At [15], the record length and a pointer that points to the record contents at [16] is passed to the function call at [17]. This function will append all `Continue` records of type 0x003c into the same prior-mentioned aggregation object. After both of these loops have been executed, the aggregation object will contain all records of type 0x003c representing any `Continue` records, and records of type 0x00eb which contains the `MsoDrawingGroup` record used to contain any Office Art records.

```
0x646241:    lea    -0x174(%rbp),%rcx   ; result record length
0x646248:    mov    -0x158(%rbp),%rdx   ; current record data from stream
0x64624f:    mov    -0x190(%rbp),%rsi   ; stream object
0x646256:    mov    -0x188(%rbp),%rax   ; record object
0x64625d:    mov    $0x3c,%r8d          ; look for record type 0x003c
0x646263:    mov    %rax,%rdi
0x646266:    callq  0x620bdc            ; [14] check the record type matches
0x64626b:    mov    %rax,-0x158(%rbp)   ; store
0x646272:    cmpq   $0x0,-0x158(%rbp)
0x64627a:    setne  %al
0x64627d:    test   %al,%al
0x64627f:    je     0x6462b9
...
0x646281:    mov    -0x158(%rbp),%rax   ; record data
0x646288:    movzwl 0x2(%rax),%eax      ; [15] uint16 record length
0x64628c:    mov    %ax,-0x176(%rbp)    ; store it
...
0x646293:    movzwl -0x176(%rbp),%edx   ; uint16 record length
0x64629a:    mov    -0x158(%rbp),%rax   ; pointer to record data from stream
0x6462a1:    lea    0x4(%rax),%rcx      ; [16] shift past header into record contents
0x6462a5:    lea    -0x130(%rbp),%rax   ; aggregation object
0x6462ac:    mov    %rcx,%rsi
0x6462af:    mov    %rax,%rdi
0x6462b2:    callq  0xe4381e            ; [17] append record contents to aggregation object
0x6462b7:    jmp    0x646241
```

After collecting the boundaries of all of the `Continue` and `MsoDrawingGroup` records, the contents of the aggregation object will be sorted into a tree and then stored within a different object. This object will then be passed into the recursive function at [18] in order to parse the contents of all of the records that have been aggregated.

```
0x6465cf:    mov    -0x188(%rbp),%rax   ; record object
0x6465d6:    mov    (%rax),%rax
0x6465d9:    mov    0x4c0(%rax),%rsi
0x6465e0:    mov    -0x164(%rbp),%ecx   ; offset
0x6465e6:    lea    -0x110(%rbp),%rdx
0x6465ed:    mov    -0x150(%rbp),%rax   ; object with linked list of records
0x6465f4:    sub    $0x8,%rsp
0x6465f8:    pushq  $0x0
0x6465fa:    pushq  -0x188(%rbp)        ; record object
0x646600:    pushq  $0x0
0x646602:    mov    $0x0,%r9d
0x646608:    mov    $0x0,%r8d
0x64660e:    mov    %rax,%rdi           ; object with linked list of records
0x646611:    callq  0xe7fe6e            ; [18] call recursive function
0x646616:    add    $0x20,%rsp
```

Once inside the recursive function, the implementation will proceed to recursively traverse the records within the object calling back into itself when necessary. Upon identifying a record with the required contents, at [19] the function will copy the necessary fields into an object and then later pass this object along with the offset from its parameter, and the current parsing state to the function call at [21] to begin parsing Office Art records within their record container.

```
0xe7fe6e:    push   %rbp
0xe7fe6f:    mov    %rsp,%rbp
0xe7fe72:    push   %r12
0xe7fe74:    push   %rbx
0xe7fe75:    sub    $0xb0,%rsp
0xe7fe7c:    mov    %rdi,-0x78(%rbp)    ; object with linked list of records
0xe7fe80:    mov    %rsi,-0x80(%rbp)    ; object
0xe7fe84:    mov    %rdx,-0x88(%rbp)    ; object
0xe7fe8b:    mov    %ecx,-0x8c(%rbp)    ; offset
0xe7fe91:    mov    %r8,-0x98(%rbp)
0xe7fe98:    mov    %r9,-0xa0(%rbp)
...
0xe803df:    mov    -0x78(%rbp),%rax
0xe803e3:    mov    -0x80(%rbp),%rdx
0xe803e7:    mov    %rdx,0x8(%rax)      ; [19] initialize field in object
...
0xe803eb:    mov    -0x78(%rbp),%rax
0xe803ef:    mov    -0x98(%rbp),%rdx
0xe803f6:    mov    %rdx,0x30(%rax)     ; [19] initialize field in object
...
0xe803fa:    mov    -0x78(%rbp),%rax
0xe803fe:    mov    -0xa0(%rbp),%rdx
0xe80405:    mov    %rdx,0x50(%rax)     ; [19] initialize field in object
...
0xe80409:    mov    -0x78(%rbp),%rax
0xe8040d:    mov    -0xa8(%rbp),%rdx
0xe80414:    mov    %rdx,0x38(%rax)     ; [19] initialize field in object
...
0xe80418:    mov    -0x78(%rbp),%rax
0xe8041c:    mov    -0xb0(%rbp),%rdx
0xe80423:    mov    %rdx,0x48(%rax)     ; [19] initialize field in object
...
0xe80427:    mov    -0x78(%rbp),%rax
0xe8042b:    mov    -0xb8(%rbp),%rdx
0xe80432:    mov    %rdx,0x40(%rax)     ; [19] initialize field in object
...
0xe80436:    mov    -0x78(%rbp),%rax
0xe8043a:    lea    0xd0(%rax),%rcx
0xe80441:    mov    -0x78(%rbp),%rdx
0xe80445:    mov    -0x8c(%rbp),%esi    ; [20] chunk offset size
0xe8044b:    mov    -0x88(%rbp),%rax    ; object
0xe80452:    mov    %rax,%rdi
0xe80455:    callq  0xe7f9fc            ; [21] call function to parse contents of office art records
0xe8045a:    test   %eax,%eax
0xe8045c:    sete   %al
0xe8045f:    test   %al,%al
0xe80461:    je     0xe8046a
```

When parsing the contents of the current Office Art record, the following function will be executed. As `MsoDrawingGroup` records can contain a number of particular record types, this function contains a loop which is responsible for iterating through all of the records. This loop is terminated by the current record's size which is calculated at [22]. For each iteration, the loop checks to ensure that the art records within the `MsoDrawingGroup` are still within its bounds.

```
0xe7f9fc:    push   %rbp
0xe7f9fd:    mov    %rsp,%rbp
0xe7fa00:    push   %r12
0xe7fa02:    push   %rbx
0xe7fa03:    sub    $0x50,%rsp
0xe7fa07:    mov    %rdi,-0x48(%rbp)    ; object
0xe7fa0b:    mov    %esi,-0x4c(%rbp)    ; offset
0xe7fa0e:    mov    %rdx,-0x58(%rbp)    ; object with items
0xe7fa12:    mov    %rcx,-0x60(%rbp)    ; parsing state
...
0xe7fa73:    mov    -0x48(%rbp),%rax    ; object
0xe7fa77:    mov    (%rax),%rax         ; vtable
0xe7fa7a:    add    $0x18,%rax
0xe7fa7e:    mov    (%rax),%rax         ; 3rd method of vtable
0xe7fa81:    mov    -0x48(%rbp),%rdx
0xe7fa85:    mov    %rdx,%rdi
0xe7fa88:    callq  *%rax               ; [22] return difference between two records (size)
0xe7fa8a:    cmp    -0x40(%rbp),%eax
0xe7fa8d:    setb   %al
0xe7fa90:    test   %al,%al
0xe7fa92:    je     0xe7fbfa
...
0xe7fbda:    mov    $0x2,%ebx
0xe7fbdf:    lea    -0x30(%rbp),%rax    ; record art object
0xe7fbe3:    mov    %rax,%rdi
0xe7fbe6:    callq  0xe43f10            ; does nothing
0xe7fbeb:    test   %ebx,%ebx
0xe7fbed:    je     0xe7fbfa
0xe7fbef:    cmp    $0x2,%ebx
0xe7fbf2:    jne    0xe7fc00
0xe7fbf4:    nop
0xe7fbf5:    jmpq   0xe7fa73
```

For each iteration (and thus each art record) in this loop, the following code will be executed. As Microsoft Office Art records have a different header type, the header is copied into a temporary object and then its fields are extracted in order to parse the record properly. At [23], the `uint32` record length is extracted from the record's contents, along with the record's type at [24]. This type is then checked against either the `uint16` 0xf002 or 0xf000. As this vulnerability described by this document is specific to the 0xf000 record type, the branch at [25] is taken. Once the type has been identified, an object specific to this record container is allocated at [26] followed by its construction at [27]. After validating the object has been properly constructed, the function call at [28] will be used to dispatch into a virtual method belonging to the object that was just constructed. Eventually within the function call at [28], its descendant will extract the 4th method of the object from its virtual-method table at [29] and then pass the object along with the stream object to the function call at [30]. This method for the object that is being used, is used to parse a variety of record types such as 0xf001, 0xf00b, 0xf006, 0xf016, 0xf11e, and 0xf150.

```
0xe7faca:    lea    -0x30(%rbp),%rax
0xe7face:    mov    %rax,%rdi
0xe7fad1:    callq  0xe45b0c            ; [23] return uint32 art record length
0xe7fad6:    mov    %eax,-0x3c(%rbp)
...
0xe7fad9:    lea    -0x30(%rbp),%rax
0xe7fadd:    mov    %rax,%rdi
0xe7fae0:    callq  0xe45ade            ; [24] return uint16 art record type
...
0xe7fae5:    cmp    $0xf000,%eax
0xe7faea:    je     0xe7faf8            ; [25] branch for record type 0xf000
...
0xe7faf8:    mov    $0x8b18,%edi
0xe7fafd:    callq  0x10cb580 <_Znwm>   ; [26] allocate 0x8b18 object
0xe7fb02:    mov    %rax,%rbx
...
0xe7fb05:    mov    -0x58(%rbp),%rdx
0xe7fb09:    mov    -0x3c(%rbp),%eax    ; record length
0xe7fb0c:    mov    %eax,%esi
0xe7fb0e:    mov    %rbx,%rdi
0xe7fb11:    callq  0xe7c3e4            ; [27] construct object
...
0xe7fb16:    mov    %rbx,%rdi
0xe7fb19:    callq  0xab875d            ; check pointer
0xe7fb1e:    mov    %rax,%rdx
...
0xe7fb28:    mov    -0x60(%rbp),%rdx
0xe7fb2c:    mov    -0x48(%rbp),%rax    ; stream object
0xe7fb30:    mov    %rdx,%rsi
0xe7fb33:    mov    %rax,%rdi
0xe7fb36:    callq  0xe44281            ; [28] call parser of 0x8b18 object \\
0xe7fb3b:    test   %eax,%eax
0xe7fb3d:    sete   %al
0xe7fb40:    test   %al,%al
0xe7fb42:    je     0xe7fbad
\\
0xe440ed:    mov    -0x18(%rbp),%rax    ; object
0xe440f1:    mov    (%rax),%rax         ; [29] vtable
0xe440f4:    add    $0x20,%rax
0xe440f8:    mov    (%rax),%rax         ; 4th method of object
0xe440fb:    mov    -0x20(%rbp),%rcx    ; stream object
0xe440ff:    mov    -0x18(%rbp),%rdx
0xe44103:    mov    %rcx,%rsi           ; stream object
0xe44106:    mov    %rdx,%rdi
0xe44109:    callq  *%rax               ; [30] call parser for sub-record types
```

The following code represents a disassembly of the 4th virtual method from the object that was constructed to parse the contents of the 0xf000 record type. As the contents of the 0xf000 record type represents a container of a variety of Office Art records, this method is primarily constructed of a similar loop to the method used for parsing the contents of the `MsoDrawingGroup` record. At [31] the size of the current record is read, and used to bound the loop when extracting the sub-records within the record's contents.

```
0xe7c46e:    push   %rbp
0xe7c46f:    mov    %rsp,%rbp
0xe7c472:    push   %r13
0xe7c474:    push   %r12
0xe7c476:    push   %rbx
0xe7c477:    sub    $0xa8,%rsp
0xe7c47e:    mov    %rdi,-0xb8(%rbp)    ; object
0xe7c485:    mov    %rsi,-0xc0(%rbp)    ; stream object
...
0xe7c4cc:    mov    -0xc0(%rbp),%rax    ; stream object
0xe7c4d3:    mov    (%rax),%rax
0xe7c4d6:    add    $0x18,%rax
0xe7c4da:    mov    (%rax),%rax
0xe7c4dd:    mov    -0xc0(%rbp),%rdx    ; stream object
0xe7c4e4:    mov    %rdx,%rdi
0xe7c4e7:    callq  *%rax               ; [31] return size of current record
0xe7c4e9:    cmp    -0xa4(%rbp),%eax
0xe7c4ef:    setb   %al
0xe7c4f2:    test   %al,%al
0xe7c4f4:    je     0xe7c8fa
...
0xe7c8db:    lea    -0x80(%rbp),%rax
0xe7c8df:    mov    %rax,%rdi
0xe7c8e2:    callq  0xe43f10            ; do nothing
0xe7c8e7:    test   %ebx,%ebx
0xe7c8e9:    je     0xe7c8fa
0xe7c8eb:    cmp    $0x2,%ebx
0xe7c8ee:    jne    0xe7c9bf
0xe7c8f4:    nop
0xe7c8f5:    jmpq   0xe7c4cc
```

For each iteration of this record, the loop will execute the following code to check which handler to use for which particular record type. The record type is extracted from the current record's header by calling the function at [32]. The branches that follow will then dispatch to the correct block of code used to handle each individual record type. As this vulnerability is specifically with regards to the 0xf150 record type, the branches at [33] will dispatch to the handler for the record type containing the vulnerability. Once at the correct hander, the function call at [34] will extract the 12-bit instance from the record header and store it into a variable. Afterwards, at [35], the record instance, an object containing the art object, and the stream object are each passed as parameters to the function call to the `CEscherObject::ReadNativeProperties`.

```
0xe7c52f:    lea    -0x80(%rbp),%rax    ; record art object
0xe7c533:    mov    %rax,%rdi
0xe7c536:    callq  0xe45ade            ; [32] return art record type
0xe7c53b:    cmp    $0xf00b,%eax
0xe7c540:    je     0xe7c7cb
0xe7c546:    cmp    $0xf00b,%eax
0xe7c54b:    ja     0xe7c564            ; [33] check for record type 0xf150
...
0xe7c564:    cmp    $0xf11e,%eax
0xe7c569:    je     0xe7c755
0xe7c56f:    cmp    $0xf150,%eax
0xe7c574:    je     0xe7c848            ; [33] check for record type 0xf150
...
0xe7c848:    lea    -0x80(%rbp),%rax    ; record art object
0xe7c84c:    mov    %rax,%rdi
0xe7c84f:    callq  0xe45af4            ; [34] extract the art record instance from the record header
0xe7c854:    mov    %eax,-0x98(%rbp)    ; store art record instance
...
0xe7c85a:    mov    -0xb8(%rbp),%rax
0xe7c861:    lea    0x28(%rax),%rcx     ; offset +0x28 of 0x8b18 object
0xe7c865:    mov    -0x98(%rbp),%edx    ; art record instance
0xe7c86b:    mov    -0xc0(%rbp),%rax    ; stream object
0xe7c872:    mov    %rax,%rsi
0xe7c875:    mov    %rcx,%rdi
0xe7c878:    callq  0xe4d1ba            ; [35] CEscherObject::ReadNativeProperties
0xe7c87d:    test   %eax,%eax
0xe7c87f:    sete   %al
0xe7c882:    test   %al,%al
0xe7c884:    je     0xe7c893
```

Inside the `CEscherObject::ReadNativeProperties` method, offset +0x28 of the 0x8b18 `CEscherObject`, combined with the stream object, and the instance from the record header is stored within the frame of the method. Afterwards, the implementation enters a loop which will continue until an index that is incremented reaches the 12-bit integer that was read from the record's instance. At [36], the 8-byte header is then read from the current position in the stream in order to determine how the contents of it must be parsed.

```
0xe4d1ba:    push   %rbp
0xe4d1bb:    mov    %rsp,%rbp
0xe4d1be:    push   %r12
0xe4d1c0:    push   %rbx
0xe4d1c1:    sub    $0xd0,%rsp
0xe4d1c8:    mov    %rdi,-0xc8(%rbp)    ; escher object
0xe4d1cf:    mov    %rsi,-0xd0(%rbp)    ; stream object
0xe4d1d6:    mov    %edx,-0xd4(%rbp)    ; record's instance
...
0xe4d1eb:    movl   $0x0,-0xb8(%rbp)
...
0xe4d1f5:    mov    -0xb8(%rbp),%eax    ; current loop index
0xe4d1fb:    cmp    -0xd4(%rbp),%eax    ; record's instance
0xe4d201:    jge    0xe4ecc1
...
0xe4d207:    mov    -0xd0(%rbp),%rax    ; stream object
0xe4d20e:    mov    (%rax),%rax         ; stream object vtable
0xe4d211:    add    $0x30,%rax          ; 6th method of stream object
0xe4d215:    mov    (%rax),%rax
0xe4d218:    mov    $0x8,%edx           ; record header length
0xe4d21d:    lea    -0xa0(%rbp),%rsi    ; record header destination
0xe4d224:    mov    -0xd0(%rbp),%rcx
0xe4d22b:    mov    %rcx,%rdi
0xe4d22e:    callq  *%rax               ; [36] read record header
0xe4d230:    test   %eax,%eax
0xe4d232:    sete   %al
0xe4d235:    test   %al,%al
0xe4d237:    je     0xe4d243
...
0xe4ecb4:    nop
0xe4ecb5:    addl   $0x1,-0xb8(%rbp)
0xe4ecbc:    jmpq   0xe4d1f5
```

In the loop, each iteration will first do some boundary checks on the contents of the record. At [37], the `uint16` containing the Version/Instance from the record header is extracted and checked against a constant. If it corresponds to the constant then at [38], the length will be checked for underflow along with an overflow check on the length at [39]. After verifying the length will not affect any of the following parsing of the record, at [40] the Version/Instance from the record header will be extracted again. Once checking its bounds, this value will then be used to branch to a handler that will be used to parse the current record the loop is processing.

```
0xe4d283:    mov    -0xa0(%rbp),%eax            ; [37] check version/instance from record header
0xe4d289:    cmp    $0x20,%eax
0xe4d28c:    jne    0xe4d2cf
...
0xe4d28e:    mov    -0x9c(%rbp),%eax            ; [38] length from record header
0xe4d294:    mov    %eax,%edx
0xe4d296:    mov    -0xb4(%rbp),%eax
0xe4d29c:    sub    $0x4,%eax
0xe4d29f:    cltq                               ; [38] underflow check of length
0xe4d2a1:    add    $0x4,%rax
0xe4d2a5:    cmp    %rax,%rdx
0xe4d2a8:    jbe    0xe4d2cf
...
0xe4d2aa:    mov    -0x9c(%rbp),%eax            ; [39] length from record header
0xe4d2b0:    mov    %eax,%edx
0xe4d2b2:    mov    -0xb4(%rbp),%eax
0xe4d2b8:    cltq
0xe4d2ba:    add    $0x4,%rax                   ; add 4 to length for total length of record
0xe4d2be:    cmp    %rax,%rdx                   ; [39] overflow check of length
0xe4d2c1:    ja     0xe4d2cf
0xe4d2c3:    movl   $0x21,-0xa0(%rbp)
0xe4d2cd:    jmp    0xe4d2f6
...
0xe4d2f6:    mov    -0xa0(%rbp),%eax            ; [40] version/instance from record header
0xe4d2fc:    cmp    $0x30,%eax
0xe4d2ff:    ja     0xe4ec3d                    ; [40] bounds check
...
0xe4d305:    mov    %eax,%eax
0xe4d307:    mov    0x14db5e0(,%rax,8),%rax     ; [41] use version/instance to branch to handler
0xe4d30f:    jmpq   *%rax
```

The handler for each iteration of the loop used to parse the contents of the 0xf150 record will generally use the fields within the record header in order to parse the contents of each sub-record. Typically, the contents of each sub-record is read into a field located within the 0x8b18 object that was previously allocated. Due to this object being of a static size, any kind of copy operation that writes within this object and trusts the length field from the record header can be used to overflow the contents of said object.

CVE-2020-27247 - Version/Instance 0x0002

The following code shows an instance when the Version/Instance field is of the value 0x0002. At [42], the stream object will be used to read the contents from the stream into the 0x8b18 object that was previously allocated. At [43], the `uint32` length from the record header is trusted, and then at [44] the contents of the stream is read at offset 0x90c of the object. It is suspected that this member is 0x90 bytes in size. Due to the `uint32` length being trusted when writing directly into this statically allocated member of the target object, this can cause a heap-based buffer overflow. These types of overflows can trigger heap corruption which can lead to code execution under the context of the application.

```
0xe4d718:    mov    -0xc8(%rbp),%rax
0xe4d71f:    movl   $0x1,0x908(%rax)
...
0xe4d729:    mov    -0xd0(%rbp),%rax    ; [42] stream object
0xe4d730:    mov    (%rax),%rax         ; virtual method table of object
0xe4d733:    add    $0x30,%rax
0xe4d737:    mov    (%rax),%rax         ; 6th method of stream object
0xe4d73a:    mov    -0x9c(%rbp),%edx    ; [43] length from record header
0xe4d740:    mov    -0xc8(%rbp),%rcx    ; object written into
0xe4d747:    lea    0x90c(%rcx),%rsi    ; offset 0x90c of object
0xe4d74e:    mov    -0xd0(%rbp),%rcx    ; [42] stream object
0xe4d755:    mov    %rcx,%rdi
0xe4d758:    callq  *%rax               ; [44] read data from stream into object
0xe4d75a:    test   %eax,%eax
0xe4d75c:    sete   %al
0xe4d75f:    test   %al,%al
0xe4d761:    je     0xe4ec7e
```

Within the provided proof-of-concept, the “Workbook” stream was made contiguous and begins at sector 7 (offset 0x1000) of the file. The `MsoDrawingGroup` excel record with the type 0x00eb is at offset 0x364c within the file. This record type has a maximum size of 8228 bytes, and thus if an attacker wishes to write more than this number of bytes when triggerring the overflow, either more contiguous `MsoDrawingGroup` records or `Continue` records with the type 0x003c need to follow the initial `MsoDrawingGroup` record.

Within the `MsoDrawingGroup` record, the Microsoft Office Records are contained which contain a different header corresponding to the Microsoft Office Art File Format Specification. Of the records containing within the `MsoDrawingGroup` record, there must be an `OfficeArtDggContainer` record of type 0xf000 which is located at offset 0x3650 of the provided proof-of-concept. This record type is also a container record which contains a number of sub-records. Within these sub-records, at offset 0x3658, a record of type 0xf150 must be contained.

Once the 0xf150 record container has been identified, the uint32 length is used to read the record’s contents, and Version/Instance field must be set to 0x0002. If the length is larger than 0x90, then this vulnerability is being triggerred.

Crash Information

Once running the application in a debugger, set the following breakpoint at the construction of the 0x8b18 object prior to opening up the provided proof-of-concept.

```
(gdb) bp e7fafd
Breakpoint 4 at 0xe7fafd
(gdb) continue
```

Once the breakpoint is triggered at the time of the allocation, step over it in order to examine the pointer that has been returned in the `%rax` register.

```
Thread 1 "planmaker" hit Breakpoint 4, 0x0000000000e7fafd in ?? ()
(gdb) h

-=[registers]=-
[rax: 0x000000000000f000] [rbx: 0x0000000002ed2900] [rcx: 0x000000c2f000000f]
[rdx: 0x00000000000000ca] [rsi: 0x000000c2f000000f] [rdi: 0x0000000000008b18]
[rsp: 0x00007fffffffb9a0] [rbp: 0x00007fffffffba00] [ pc: 0x0000000000e7fafd]
[ r8: 0x0000000000000000] [ r9: 0x0000000000000000] [r10: 0x0000000001b82010]
[r11: 0x00007ffff77a4be0] [r12: 0x0000000000000001] [r13: 0x00007fffffffeb00]
[r14: 0x0000000000000000] [r15: 0x0000000000000000] [efl: 0x00000246]
[flags: +ZF -SF -OF -CF -DF +PF -AF +IF R1]

-=[stack]=-
7fffffffb9a0 | 0000000002ed29d0 0000000002ed2900 | .).......)......
7fffffffb9b0 | 0000dfe4ffffba00 00007fffffffbb90 | ................
7fffffffb9c0 | 000000c20000dfe4 0000000000000000 | ................
7fffffffb9d0 | 000000c2f000000f 00000001000000ca | ................

-=[disassembly]=-
   0xe7faf8:    mov    $0x8b18,%edi
=> 0xe7fafd:    callq  0x10cb580 <_Znwm>
   0xe7fb02:    mov    %rax,%rbx
   0xe7fb05:    mov    -0x58(%rbp),%rdx
   0xe7fb09:    mov    -0x3c(%rbp),%eax
   0xe7fb0c:    mov    %eax,%esi

(gdb) n
0x0000000000e7fb02 in ?? ()

(gdb) p/x $rax
$1 = 0x2e33210
```

Afterwards, set the next breakpoint at the beginning of the `CEscherObject::ReadNativeProperties` function and continue execution. When the program breaks, examine the first parameter within the `%rdi` register and note that it is +0x28 bytes from the 0x8b18 allocation that was previously examined.

```
(gdb) bp e4d1ba
Breakpoint 5 at 0xe4d1ba

(gdb) c
Continuing.

Thread 1 "planmaker" hit Breakpoint 5, 0x0000000000e4d1c8 in ?? ()
(gdb) p/x $rdi
$2 = 0x2e33238

(gdb) p/x $1+0x28
$3 = 0x2e33238
```

The provided proof-of-concept uses a Version/Instance of 3 which results in the handler at 0xe4d771 being used to read the contents of the stream into offset +0x99c of the object. If we set a breakpoint at address 0xe4d7b1, we will break on the call to the stream reading method prior to it writing outside the bounds of the object member at offset +0x99c. At this function call, the `uint32` length that is used is stored in the `%edx` register and the offset into the object is within the `%rsi` register. With the provided proof-of-concept, the length is set to 0xdfcc which when written to offset +0x99c of the object will result in 0xdf8c bytes being written into the member which is suspected to be only 0x40 bytes in size. This large value will also result in writing 0x8154 bytes after the object which will corrupt memory outside the object's bounds.

```
(gdb) bp e4d7b1
Breakpoint 7 at 0xe4d7b1
(gdb) c
Continuing.

Thread 1 "planmaker" hit Breakpoint 7, 0x0000000000e4d7b1 in ?? ()
(gdb) x/i $pc
=> 0xe4d7b1:    callq  *%rax

(gdb) i r edx rsi
edx            0xdfcc              0xdfcc
rsi            0x2e33bd4           0x2e33bd4
```

At this point, resuming execution will read the contents of the record into the member and corrupt memory that will likely contain heap metadata or objects belonging to the `CEscherObject`. Upon usage of the heap, the platform's heap implementation will either raise a signal, or receive a signal from the operating system due to the corruption of the memory that follows the object.

```
(gdb) c
Continuing.
free(): corrupted unsorted chunks

Thread 1 "planmaker" received signal SIGABRT, Aborted.
__GI_raise (sig=sig@entry=0x6) at ../sysdeps/unix/sysv/linux/raise.c:50
50      ../sysdeps/unix/sysv/linux/raise.c: No such file or directory.
```

CVE-2020-27248 - Version/Instance 0x0003 and 0x0014

The prior mentioned code is actually a common pattern within the implementation of the `CEscherObject::ReadNativeProperties` method. As such, there are a number of cases that contain a similar vulnerability where only the offset into the statically sized object is different. Specifically the following code is for cases where the Version/Instance field is either 3 or 0x14. This handler will then trust the `uint32` length at [43] to read the contents of the stream into offset 0x99c of the object at [44]. It is suspected that the size of this member is 0x40 bytes in size.

```
0xe4d771:    mov    -0xc8(%rbp),%rax
0xe4d778:    movl   $0x1,0x998(%rax)
...
0xe4d782:    mov    -0xd0(%rbp),%rax    ; [42] stream object
0xe4d789:    mov    (%rax),%rax         ; virtual method table of object
0xe4d78c:    add    $0x30,%rax
0xe4d790:    mov    (%rax),%rax         ; 6th method of stream object
0xe4d793:    mov    -0x9c(%rbp),%edx    ; [43] length from record header
0xe4d799:    mov    -0xc8(%rbp),%rcx    ; object written into
0xe4d7a0:    lea    0x99c(%rcx),%rsi    ; offset 0x99c of object
0xe4d7a7:    mov    -0xd0(%rbp),%rcx    ; [42] stream object
0xe4d7ae:    mov    %rcx,%rdi
0xe4d7b1:    callq  *%rax               ; [44] read data from stream into object
0xe4d7b3:    test   %eax,%eax
0xe4d7b5:    sete   %al
0xe4d7b8:    test   %al,%al
0xe4d7ba:    je     0xe4d7c6
```

Within the provided proof-of-concept, the “Workbook” stream was made contiguous and begins at sector 7 (offset 0x1000) of the file. The `MsoDrawingGroup` excel record with the type 0x00eb is at offset 0x364c within the file. This record type has a maximum size of 8228 bytes, and thus if an attacker wishes to write more than this number of bytes when triggerring the overflow, either more contiguous `MsoDrawingGroup` records or `Continue` records with the type 0x003c need to follow the initial `MsoDrawingGroup` record.

Within the `MsoDrawingGroup` record, the Microsoft Office Records are contained which contain a different header corresponding to the Microsoft Office Art File Format Specification. Of the records containing within the `MsoDrawingGroup` record, there must be an `OfficeArtDggContainer` record of type 0xf000 which is located at offset 0x3650 of the provided proof-of-concept. This record type is also a container record which contains a number of sub-records. Within these sub-records, at offset 0x3658, a record of type 0xf150 must be contained.

Once the 0xf150 record container has been identified, the uint32 length is used to read the record’s contents, and Version/Instance field must be set to either 0x0003 or 0x0014. If the length is larger than 0x40, then this vulnerability is being triggerred.

CVE-2020-27249 - Version/Instance 0x0004 and 0x0015

This next snippet is specifically for cases where the Version/Instance field is either 4 or 0x15. This handler will then trust the `uint32` length at [43] to read the contents of the stream into offset 0x9dc of the object at [44]. It is suspected that this member is 0x64 bytes in size.

```
0xe4d83d:    mov    -0xc8(%rbp),%rax
0xe4d844:    movl   $0x1,0x9d8(%rax)
...
0xe4d84e:    mov    -0xd0(%rbp),%rax    ; [42] stream object
0xe4d855:    mov    (%rax),%rax         ; virtual method table of object
0xe4d858:    add    $0x30,%rax
0xe4d85c:    mov    (%rax),%rax         ; 6th method of stream object
0xe4d85f:    mov    -0x9c(%rbp),%edx    ; [43] length from record header
0xe4d865:    mov    -0xc8(%rbp),%rcx    ; object written into
0xe4d86c:    lea    0x9dc(%rcx),%rsi    ; offset 0x9dc of object
0xe4d873:    mov    -0xd0(%rbp),%rcx    ; [42] stream object
0xe4d87a:    mov    %rcx,%rdi
0xe4d87d:    callq  *%rax               ; [44] read data from stream into object
0xe4d87f:    test   %eax,%eax
0xe4d881:    sete   %al
0xe4d884:    test   %al,%al
0xe4d886:    je     0xe4d892
```

Within the provided proof-of-concept, the “Workbook” stream was made contiguous and begins at sector 7 (offset 0x1000) of the file. The `MsoDrawingGroup` excel record with the type 0x00eb is at offset 0x364c within the file. This record type has a maximum size of 8228 bytes, and thus if an attacker wishes to write more than this number of bytes when triggerring the overflow, either more contiguous `MsoDrawingGroup` records or `Continue` records with the type 0x003c need to follow the initial `MsoDrawingGroup` record.

Within the `MsoDrawingGroup` record, the Microsoft Office Records are contained which contain a different header corresponding to the Microsoft Office Art File Format Specification. Of the records containing within the `MsoDrawingGroup` record, there must be an `OfficeArtDggContainer` record of type 0xf000 which is located at offset 0x3650 of the provided proof-of-concept. This record type is also a container record which contains a number of sub-records. Within these sub-records, at offset 0x3658, a record of type 0xf150 must be contained.

Once the 0xf150 record container has been identified, the uint32 length is used to read the record’s contents, and Version/Instance field must be set to either 0x0004 or 0x0015. If the length is larger than 0x64, then this vulnerability is being triggerred.

CVE-2020-27250 - Version/Instance 0x0005 and 0x0016

This next snippet is specifically for cases where the Version/Instance field is either 5 or 0x16. This handler will then trust the `uint32` length at [43] to read the contents of the stream into offset 0xa40 of the object at [44]. It is suspected that this member is 0xec bytes in size.

```
0xe4d92d:    mov    -0xc8(%rbp),%rax
0xe4d934:    movl   $0x1,0xa3c(%rax)
...
0xe4d93e:    mov    -0xd0(%rbp),%rax    ; [42] stream object
0xe4d945:    mov    (%rax),%rax         ; virtual method table of object
0xe4d948:    add    $0x30,%rax
0xe4d94c:    mov    (%rax),%rax         ; 6th method of stream object
0xe4d94f:    mov    -0x9c(%rbp),%edx    ; [43] length from record header
0xe4d955:    mov    -0xc8(%rbp),%rcx    ; object written into
0xe4d95c:    lea    0xa40(%rcx),%rsi    ; offset 0xa40 of object
0xe4d963:    mov    -0xd0(%rbp),%rcx    ; [42] stream object
0xe4d96a:    mov    %rcx,%rdi
0xe4d96d:    callq  *%rax               ; [44] read data from stream into object
0xe4d96f:    test   %eax,%eax
0xe4d971:    sete   %al
0xe4d974:    test   %al,%al
0xe4d976:    je     0xe4d982
```

Within the provided proof-of-concept, the “Workbook” stream was made contiguous and begins at sector 7 (offset 0x1000) of the file. The `MsoDrawingGroup` excel record with the type 0x00eb is at offset 0x364c within the file. This record type has a maximum size of 8228 bytes, and thus if an attacker wishes to write more than this number of bytes when triggerring the overflow, either more contiguous `MsoDrawingGroup` records or `Continue` records with the type 0x003c need to follow the initial `MsoDrawingGroup` record.

Within the `MsoDrawingGroup` record, the Microsoft Office Records are contained which contain a different header corresponding to the Microsoft Office Art File Format Specification. Of the records containing within the `MsoDrawingGroup` record, there must be an `OfficeArtDggContainer` record of type 0xf000 which is located at offset 0x3650 of the provided proof-of-concept. This record type is also a container record which contains a number of sub-records. Within these sub-records, at offset 0x3658, a record of type 0xf150 must be contained.

Once the 0xf150 record container has been identified, the uint32 length is used to read the record’s contents, and Version/Instance field must be set to either 0x0005 or 0x0016. If the length is larger than 0xec, then this vulnerability is being triggerred.

CVE-2020-28587 - Version/Instance 0x001e

This next snippet is specifically for the case where the Version/Instance field is 0x1e. This handler will then trust the `uint32` length at [43] to read the contents of the stream into offset 0xb2c of the object at [44]. It is suspected that this member is 0x1d4 bytes in size.

```
0xe4e469:    mov    -0xc8(%rbp),%rax
0xe4e470:    movl   $0x1,0xb28(%rax)
...
0xe4e47a:    mov    -0xd0(%rbp),%rax    ; [42] stream object
0xe4e481:    mov    (%rax),%rax         ; virtual method table of object
0xe4e484:    add    $0x30,%rax
0xe4e488:    mov    (%rax),%rax         ; 6th method of stream object
0xe4e48b:    mov    -0x9c(%rbp),%edx    ; [43] length from record header
0xe4e491:    mov    -0xc8(%rbp),%rcx    ; object written into
0xe4e498:    lea    0xb2c(%rcx),%rsi    ; offset 0xb2c of object
0xe4e49f:    mov    -0xd0(%rbp),%rcx    ; [42] stream object
0xe4e4a6:    mov    %rcx,%rdi
0xe4e4a9:    callq  *%rax               ; [44] read data from stream into object
0xe4e4ab:    test   %eax,%eax
0xe4e4ad:    sete   %al
0xe4e4b0:    test   %al,%al
0xe4e4b2:    je     0xe4eca5
```

Within the provided proof-of-concept, the “Workbook” stream was made contiguous and begins at sector 7 (offset 0x1000) of the file. The `MsoDrawingGroup` excel record with the type 0x00eb is at offset 0x364c within the file. This record type has a maximum size of 8228 bytes, and thus if an attacker wishes to write more than this number of bytes when triggerring the overflow, either more contiguous `MsoDrawingGroup` records or `Continue` records with the type 0x003c need to follow the initial `MsoDrawingGroup` record.

Within the `MsoDrawingGroup` record, the Microsoft Office Records are contained which contain a different header corresponding to the Microsoft Office Art File Format Specification. Of the records containing within the `MsoDrawingGroup` record, there must be an `OfficeArtDggContainer` record of type 0xf000 which is located at offset 0x3650 of the provided proof-of-concept. This record type is also a container record which contains a number of sub-records. Within these sub-records, at offset 0x3658, a record of type 0xf150 must be contained.

Once the 0xf150 record container has been identified, the uint32 length is used to read the record’s contents, and Version/Instance field must be set to 0x001e. If the length is larger than 0x1d4, then this vulnerability is being triggerred.

Timeline

2020-11-30 - Vendor Disclosure
2021-01-19 - Vendor Patched
2021-02-03 - Public Release

CREDIT

Discovered by a member of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2020-1197

NEXT REPORT

TALOS-2020-1223

© 2022 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our Privacy Policy.