New Tool for Discovering Flash Player 0-day Attacks in the Wild from Various Channels

@heisecode

1, Background

In 2014, Microsoft introduced isolated heap and memory protector to avoid IE UAF exploits, it is harder to exploit PC using IE as a target. In late 2014 when did a plan for 2015, I thought Flash Player will be most popular attack target in PC, So I did some research for catching flash 0-day attack in 2015 and finally have some results.

There are two important factors to discover 0-day attacks, one is how to get more effective samples in the wild, another is how to identify 0-day from large number of samples.

I found some sample channels which can provide effective samples in the wild and develop a tool to help me identify these samples.

2, Sample Channels

1> Products’ feedback

There are usually many SWF samples from kinds of products or engines’ detection feedback. This channel is most effective channel, because the samples have been filtered by products and engines’ rules, and the number of samples is large. There are many new feedback added every day, so need to process them every day.

2> URL Crawl

Attackers always prepare several exploits in a URL, use which exploit depends on the software version installed in the victims' PC. So products or engines' feedback may contain many old CVEs detection just because some users installed old Java version or IE version.

I get these old CVEs detection URLs from feedback and crawl these URLs to get the whole exploits based in these URLs, and there may have a Flash exploit.

3> Download from VT Intelligence

I can download many SWF samples from https://www.virustotal.com/intelligence

Based on some research, attackers may submit their exploits or POCs to VT to check AV detection. And some 0-days firstly submitted to VT before they were publicly disclosed.

4> URL Pattern

Exploit Kits and some targeted attacks may use URLs which have some pattern. Some URLs have relation to the type of the target and some URLs have same kind based on attackers' habits.

I can use URL pattern to search URLs from our products or engines' feedback to get some suspicious URLs and visit them to check there is a flash 0-day.

3, Tool to identify 0-day

Because the number of samples and URLs is large, so I need a tool to help me to identify. This tool should have fast process speed and low false alert. It also should have a logger to log the URL/Sample name corresponding to the events/behaviors.

So I developed a tool named AFED (Advanced Flash Exploit Detector).

AFED is something:

- An IE BHO written by C++
- Hook Flash OCX when Flash Player loaded to IE tab process.
- Hook IE event to get current URL name.
- Write exploit detection or sample behaviors to log.

And the automation process is:

- Simple Python code
- Register AFED BHO using regsvr32.exe
- Every time load a URL in IE, AFED hook Flash Player OCX to detect
- Kill IE processes to load next URL
- When finished all URLs, parse log file

In July, Adobe introduced Vector object mitigation into Flash Player with help of Google Project Zero.

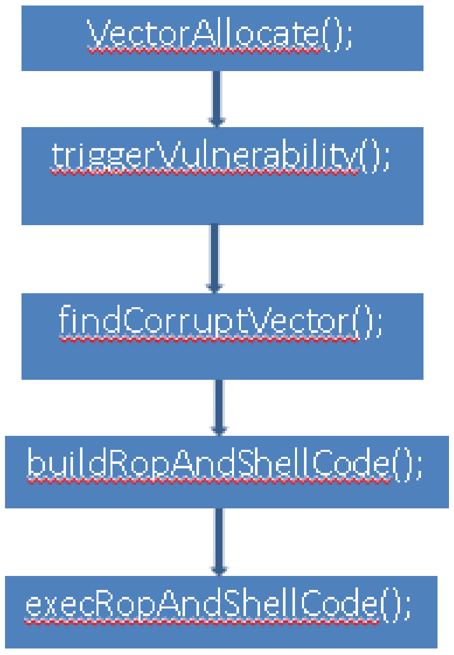
The mitigation is based on the fact that almost every flash exploit used corrupt Vector object to achieve arbitrary read and write. And Vector object was easily to be corrupted.

So there are some differences between before and after Adobe introduced Vector object mitigation.

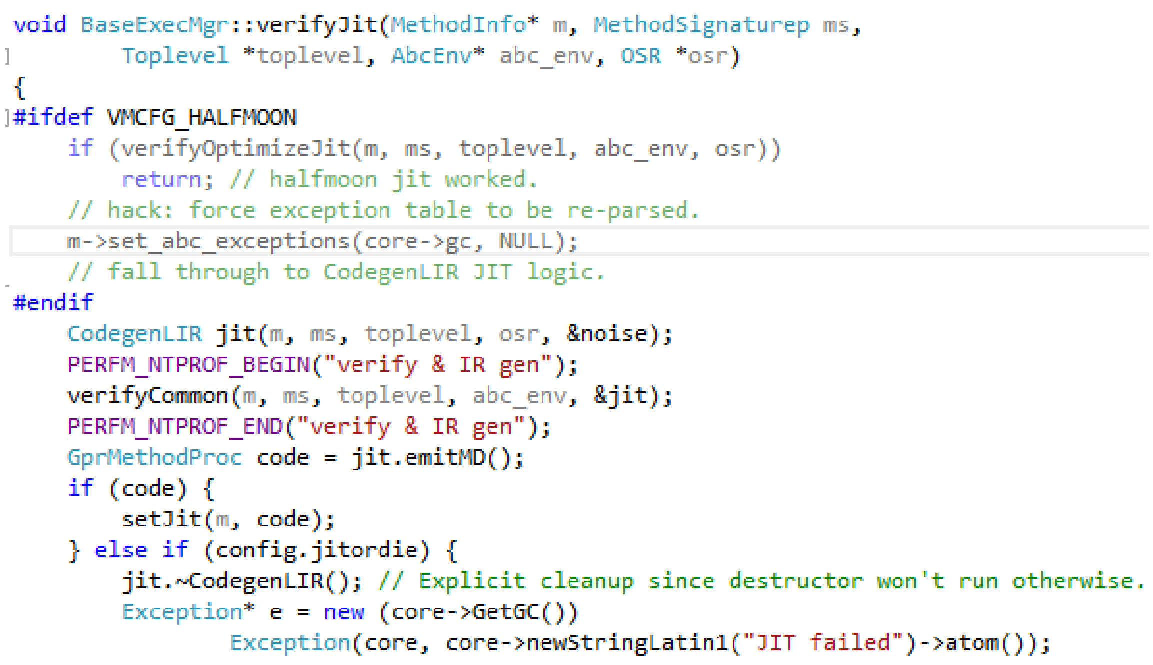
1> Before Adobe introduced Vector.<*> mitigation

Before the mitigation, Flash exploits used corrupted Vector object to achieve arbitrary read and write process memory. So AFED checks the Vector object length to detect Flash exploits.

Simplified flash exploit flow is like this:

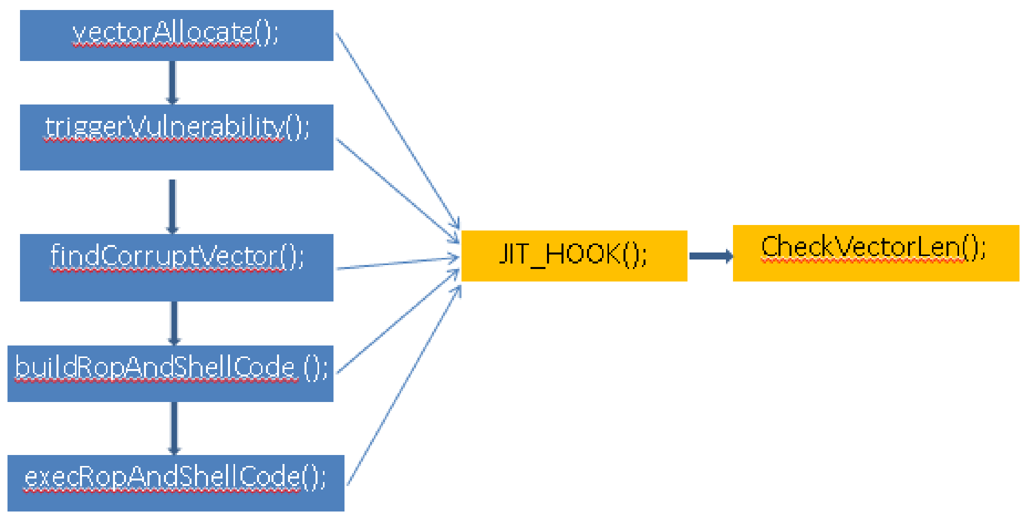


I hook AVM2 JIT flow to get the check point, the hook key function is:

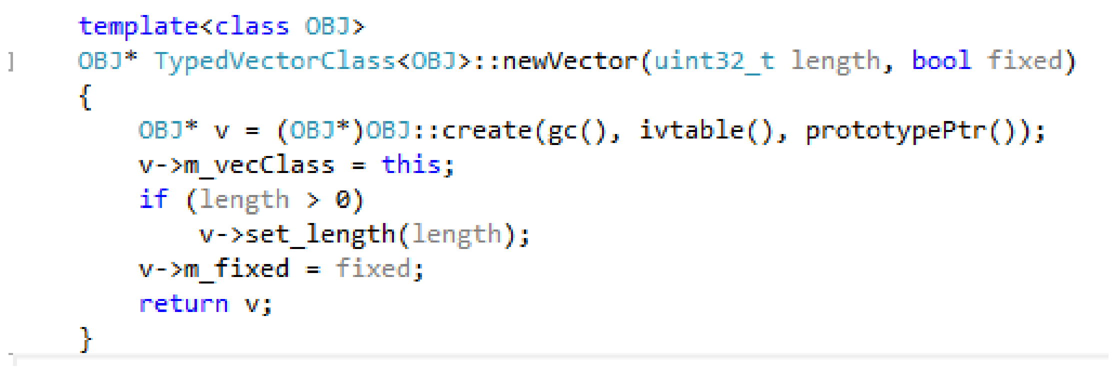


```
void BaseExecMgr::verifyJit(MethodInfo* m, MethodSignaturep ms,
        Toplevel *toplevel, AbcEnv* abc_env, OSR *osr)
{
#ifdef VMCFG_HALFMOON
    if (verifyOptimizeJit(m, ms, toplevel, abc_env, osr))
        return; // halfmoon jit worked.
    // hack: force exception table to be re-parsed.
    m->set_abc_exceptions(core->gc, NULL);
    // fall through to CodegenLIR JIT logic.
#endif
    CodegenLIR jit(m, ms, toplevel, osr, &noise);
    PERFM_NTPROF_BEGIN("verify & IR gen");
    verifyCommon(m, ms, toplevel, abc_env, &jit);
    PERFM_NTPROF_END("verify & IR gen");
    GprMethodProc code = jit.emitMD();
    if (code) {
        setJit(m, code);
    } else if (config.jitordie) {
        jit.~CodegenLIR(); // Explicit cleanup since destructor won't run otherwise.
        Exception* e = new (core->GetGC())
                Exception(core, core->newStringLatin1("JIT failed")->atom());
```

So the detection flow is like this:

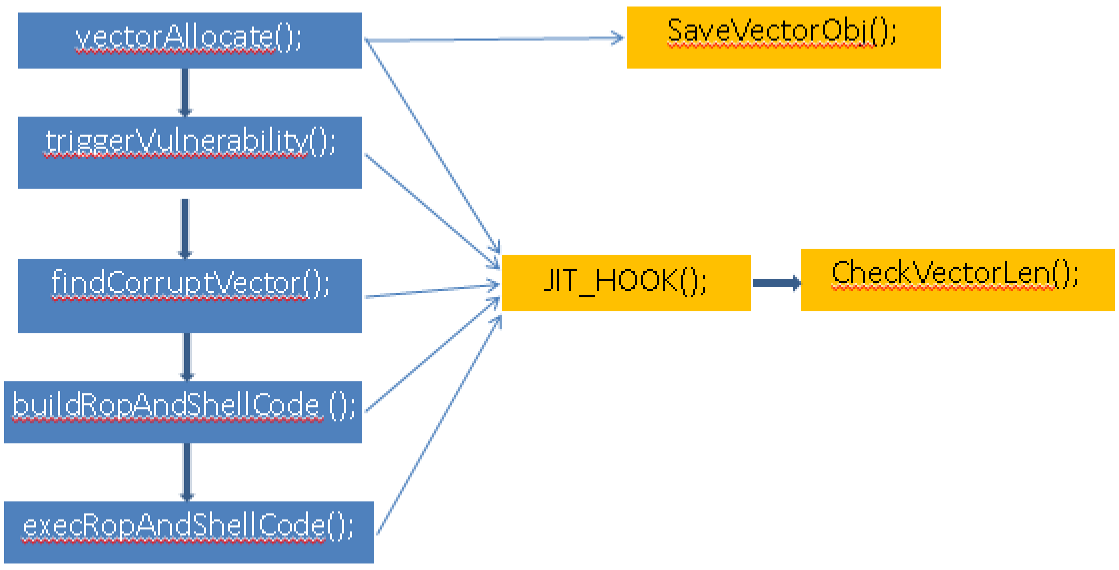


Then I hook the vector object creating, the key function is:



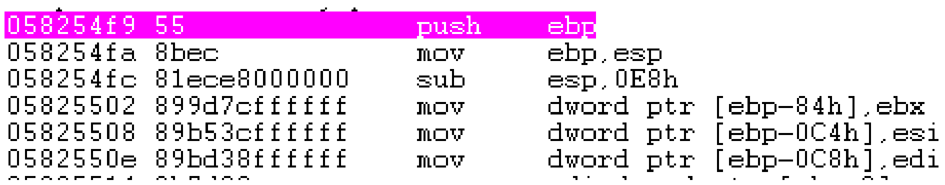
```
template<class OBJ>
OBJ* TypedVectorClass<OBJ>::newVector(uint32_t length, bool fixed)
{
    OBJ* v = (OBJ*)OBJ::create(gc(), ivtable(), prototypePtr());
    v->m_vecClass = this;
    if (length > 0)
        v->set_length(length);
    v->m_fixed = fixed;
    return v;
}
```

So the final detection flow can be this:

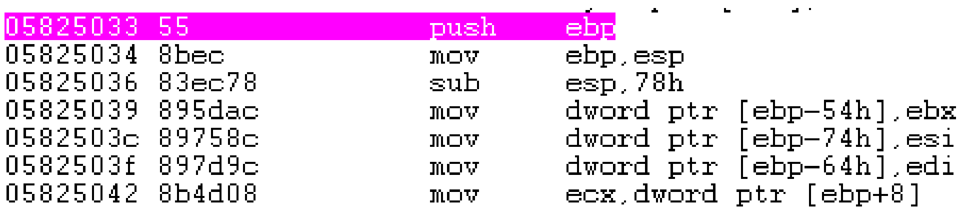


2> After Adobe introduced Vector.<*> mitigation

JIT native code prologues are like this:



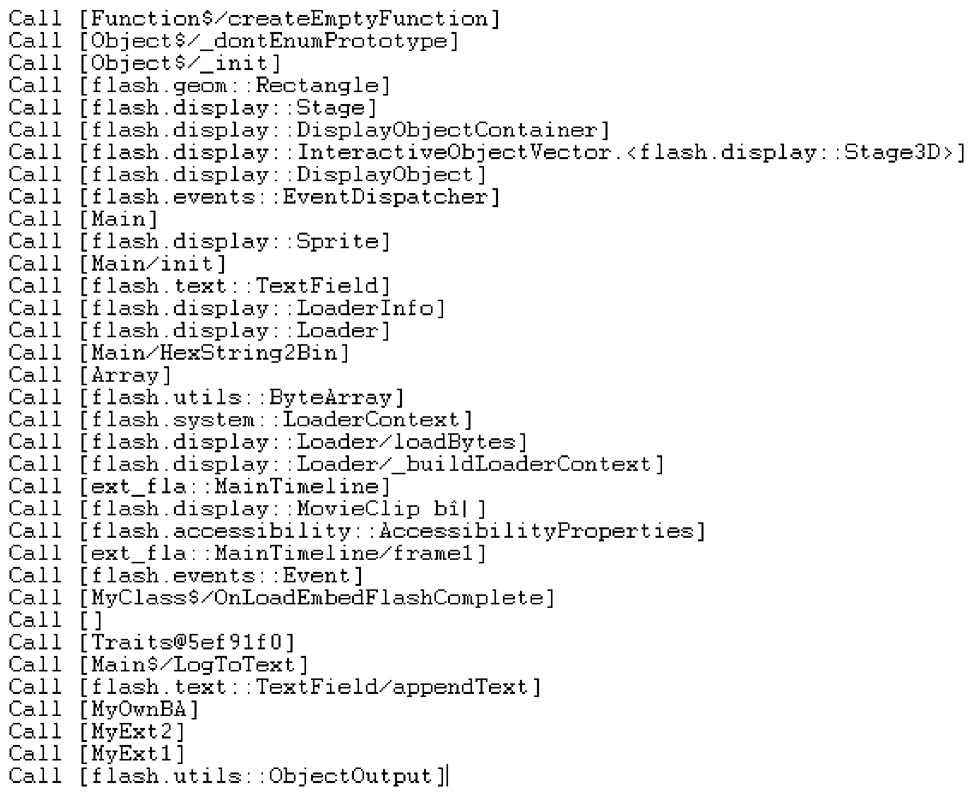
```
058254f9 55              push    ebp
058254fa 8bec            mov     ebp,esp
058254fc 81ece8000000    sub     esp,0E8h
05825502 899d7cffffff    mov     dword ptr [ebp-84h],ebx
05825508 89b53cffffff    mov     dword ptr [ebp-0C4h],esi
0582550e 89bd38ffffff    mov     dword ptr [ebp-0C8h],edi
```



```
05825033 55              push    ebp
05825034 8bec            mov     ebp,esp
05825036 83ec78          sub     esp,78h
05825039 895dac          mov     dword ptr [ebp-54h],ebx
0582503c 89758c          mov     dword ptr [ebp-74h],esi
0582503f 897d9c          mov     dword ptr [ebp-64h],edi
05825042 8b4d08          mov     ecx,dword ptr [ebp+8]
```

And we can get AS3 method name and its JIT native code address in the hook point.

So we can log every AS3 method call:



```
Call [Function$/createEmptyFunction]
Call [Object$/_dontEnumPrototype]
Call [Object$/_init]
Call [flash.geom::Rectangle]
Call [flash.display::Stage]
Call [flash.display::DisplayObjectContainer]
Call [flash.display::InteractiveObjectVector.<flash.display::Stage3D>]
Call [flash.display::DisplayObject]
Call [flash.events::EventDispatcher]
Call [Main]
Call [flash.display::Sprite]
Call [Main/init]
Call [flash.text::TextField]
Call [flash.display::LoaderInfo]
Call [flash.display::Loader]
Call [Main/HexString2Bin]
Call [Array]
Call [flash.utils::ByteArray]
Call [flash.system::LoaderContext]
Call [flash.display::Loader/loadBytes]
Call [flash.display::Loader/_buildLoaderContext]
Call [ext_fla::MainTimeline]
Call [flash.display::MovieClip bî| ]
Call [flash.accessibility::AccessibilityProperties]
Call [ext_fla::MainTimeline/frame1]
Call [flash.events::Event]
Call [MyClass$/OnLoadEmbedFlashComplete]
Call []
Call [Traits@5ef91f0]
Call [Main$/LogToText]
Call [flash.text::TextField/appendText]
Call [MyOwnBA]
Call [MyExt2]
Call [MyExt1]
Call [flash.utils::ObjectOutput]
```

We can add heuristic rules based on behaviors. For example, CVE-2015-7645 used ByteArray heapspray to exploit, so AFED will print lots of "Call [flash.utils::ByteArray]" to the log. We can add rule when parsing log to match the ByteArray heapspray.

We can add other rules based on analysis of recent exploits or your experience such as BitmapData heapspray used in recent exploits.

3> Hook Flash OCX load

Because Flash OCX is a COM component, so I hook CoGetClassObject function in urlmon.dll.

I use IsEqualCLSID(rclsid, CLSID_Flash) to identify Flash OCX is being loaded or not.

4, Reference

1> “Inside AVM”, Haifei Li

2> Google Project Zero Blog,

http://googleprojectzero.blogspot.tw/2015/07/significant-flash-exploit-mitigations_16.html