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

@heisecode
Agenda

- **Who am I**
- Background
- Sample Channels
- Tool to identify 0-day
About me

• Core Member of Trend Micro Zero-Day Discovery Team
• Trend Micro Anti-APT engine developer
• Interested in discovering vulnerabilities and writing exploit.
• Flash/Android/OS X
Agenda

• Who am I
• **Background**
• Sample Channels
• Tool to identify 0-day
Flash Year

- JAVA Click-to-Play
- Browsers’ UAF mitigations
- So Flash Player boom in 2015
Flash Year

• Zero-day attacks’ targets are mostly Flash Player in 2015
  CVE-2015-0310
  CVE-2015-0311
  CVE-2015-0313
  CVE-2015-3043
  CVE-2015-3113
  CVE-2015-5119
  CVE-2015-5122
  ...
  ...
Flash Year

• In late 2014, I decided to catch Flash Player zero-day attacks in 2015.

• There were two questions need to solve to achieve the goal.
Two questions

• How to get effective samples in the wild?
  Try any possible source channel to get effective samples.

• How to identify 0-day from these samples?
  Need a processing tool, fast, low false alert.
Agenda

• Who am I
• Background
• **Sample Channels**
• Tool to identify 0-day
Sourcing Channels

• Channel 1 - Products’ feedback
  
  > Large number of SWF samples from products or engines’ detection feedback

  > Most effective channel
Sourcing Channels

• Channel 2 - URL Crawl

  > Several exploits integrated in one URL

  > Trigger which exploit depends on software version installed in victim’s system

  > Crawl this kind of URLs may catch other software exploits
Sourcing Channels

• Channel 3 - VT intelligence
  
  > SWF samples downloaded from https://www.virustotal.com/intelligence/

  > 0-day sample may be submitted to VT before it is discovered
Sourcing Channels

• Channel 4 - URL Pattern

  > Exploit Kit or Campaign URLs may have some pattern.

  > Find these kinds of URLs, visit them to detect
Agenda

• Who am I
• Background
• Sample Channels
• Tool to identify 0-day
Need a tool

- Need a tool to identify SWF files or URLs can exploit target version of Flash Player.
  - Low False Alert.
  - Logger for automation.
  - Record exploit event when detect.
  - High performance.
Advanced Flash Exploit Detector (AFED)

• IE BHO written by C++

• Hook Flash OCX to detect.

• Hook IE event to get current URL name.

• Write detections and behaviors to log.
Automation Process

• Simple Python code

• Register AFED using regsvr32.exe

• Every time load a URL in IE, AFED hook Flash OCX to detect

• Kill IE processes to load next URL

• When finished all URLs, parse log file with rules
How to implement AFED?

• Before vector.<*> mitigation introduced, all Flash Exploits used corrupted vector.<*> to exploit.
Typical Exploit Flow Before Mitigation

- Simplified Exploit Flow

- vectorAllocate();
- triggerVulnerability();
- findCorruptVector();
- buildRopAndShellCode();
- execRopAndShellCode();
Detect Flow Before Mitigation

- Ideally

1. `vectorAllocate();`
2. `findCorruptVector();`
3. `execRopAndShellCode();`
4. `triggerVulnerability();`
5. `CheckVectorLen();`
6. `LogExploit();`
7. `buildRopAndShellCode();`
Hook JIT

• Almost each AS3 method will be JITed before called

• So I hook the JIT point of AVM2

• In hook point, check Vector object length
Detect Flow Before Mitigation

• So, Practically

vectorAllocate();

triggerVulnerability();

findCorruptVector();

buildRopAndShellCode();

execRopAndShellCode();

JIT_HOOK();

CheckVectorLen();
Hook JIT

- Key function

> In AVM2(https://github.com/adobe-flash/avmplus), BaseExecMgr::verifyJit is the function to verify and emit native code.

```c
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());
    }
}
How to check vector length?

• Hook Vector Creating

> Vector.<int>, Vector.<uint>, Vector.<Number> and Vector.<Object>
> Template function, 4 instances in flash binary.

```cpp
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;
}
```
How to check vector length?

- Check Vector length before mitigation

> vec_obj_addr + 0x18  is the ListData which save data.
> poi(poi(vec_obj_addr + 0x18) ) is vector length

```
template<class STORAGE, uint32_t slop>
struct ListData {
  uint32_t  len;          // Invariant: Must *never* exceed kListMaxLength
  void* _gc;              // Lying: Really holds capacity()
  STORAGE entries[1];     // Lying: Really holds capacity()

  // add an empty, inlined ctor to avoid spurious warnings in MSVC2008
  REALLY_INLINE explicit ListData() {}
```
Detect Flow Before Mitigation

• So, Finally

- vectorAllocate();
- triggerVulnerability();
- findCorruptVector();
- buildRopAndShellCode();
- execRopAndShellCode();

- SaveVectorObj();
- JIT_HOOK();
- CheckVectorLen();
After mitigation introduced

• Use other objects to exploit.

  > CVE-2015-7645 used ByteArray based object.
  > Overwrite ByteArray length to achieve arbitrary read and write.
  > No general exploit object like Vector
Detect Based on Behaviors

- JIT native code prologues are almost like this:
  
  > Hook first 9 bytes of JIT native code to record each call.

```
058254f9 55 push ebp
058254fa 8bec mov ebp,esp
058254fc 81e8000000 sub esp,0E8h
05825502 899d7cfffffff mov dword ptr [ebp-84h],ebx
05825508 89b53cfffffff mov dword ptr [ebp-0C4h],esi
05825510 89bd38fffffff mov dword ptr [ebp-0C8h],edi
0582551c c300

05825033 55 push ebp
05825034 8bec mov ebp,esp
05825036 83ec70 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,word ptr [ebp+8]
```
Detect Based on Behaviors

• AFED can get AS3 method name and JIT native code address by hooking JIT.

• So we can get something like this in log:

```
Call [Function$/createEmptyFunction]
Call [Object$/_dontEnumPrototype]
Call [Object$/__init]
Call [flash.geom::Rectangle]
Call [flash.display::Stage]
Call [flash.display::DisplayObjectContainer]
Call [flash.display::InteractiveObjectVector.<flash.display::Stage3D>]
Call [flash.display::DisplayObject]
Call [flash.events::EventDispatcher]
Call [Main]
Call [flash.display::Sprite]
Call [Main/init]
Call [flash.text::TextField]
Call [flash.display::LoaderInfo]
Call [flash.display::Loader]
Call [Main/HexString2Bin]
Call [Array]
Call [flash.utils::ByteArray]
Call [flash.system::LoaderContext]
Call [flash.display::Loader/loadBytes]
Call [flash.display::Loader/_buildLoaderContext]
Call [ext_fla::MainTimeline]
Call [flash.display::MovieClip_b1]
Call [flash.accessibility::AccessibilityProperties]
Call [ext_fla::MainTimeline/frame1]
Call [flash.events::Event]
Call [MyClass$/OnLoadEmbedFlashComplete]
Call []
Call [Traits@5ef91f0]
Call [Main$/LogToText]
Call [flash.text::TextField/appendText]
Call [MyOwnBá]
Call [MyExt2]
Call [MyExt1]
Call [flash.utils::ObjectOutput]]
```
Heuristic rules based on behaviors

• For example, ByteArray heapspray.

• AFED will print lots of “Call [flash.utils::ByteArray]” to log. Add this rule when parsing the log.

• Other heuristic rule can be added by analysis from recent exploits or experience.

• Recently, exploits also used BitmapData heapspray.
Forget one thing

- Hook Flash OCX loading, like Windbg’s module load event.
- Hook CoGetClassObject function in urlmon.dll
- IsEqualCLSID(rclsid, CLSID_Flash) to identify Flash OCX is being loaded or not.
Reference

• “Inside AVM,” Haifei Li
• Google Project zero,
  http://googleprojectzero.blogspot.tw/2015/07/significant-flash-exploit-mitigations_16.html
Thank you!