

Dynamic Flash instrumentation for fun and profit

Timo Hirvonen
Black Hat USA 2014



Motivation

RSA CVE-2011-060 9

```
var _local8 = "43575309eac70000789cbcbc09601bc515  
var _local9 = new Loader();  
var _local10 = new LoaderContext(false);  
_local9.loadBytes(hexToBin(_local8), _local10);  
childRef = this.addChild(_local9);
```

CosmicDuke CVE-2011-061 1

```
var _local3 = "WINWIIIIINWIIIIINWIIIIIIIIIIIIINWNWIIIIIIIN  
_local3 = this.translit(_local3);  
_arg1.writeBytes(Translate(_local3));  
_local2 = _arg1.length;
```

Youtube ad → Styx EK

```
private function Ex(_arg1:String):void
{
    ExternalInterface.call(((UN3("998a919c8b969091d7d684899e8c
```

Firestar CVE-2014-0497

```
var _local1:uint = 8398021;
var _local2:Array = new Array(198407046, 8403955, 962443453,
var _local3:ByteArray = new ByteArray();
_local3.endian = Endian.LITTLE_ENDIAN;
var _local4:int;
while (_local4 < _local2.length) {
    _local3.writeUnsignedInt((_local2[_local4] ^ _local1));
    _local4++;
};
```

Firestar EK CVE-2014-0497

```
_local3 = new Loader();  
_local4 = (_local3[("content" + "LoaderInfo")] as LoaderInfo);  
_local4.addEventListener(Event.COMPLETE, this.tartv);  
var _local5 = _local3;  
(_local5[("load" + "Bytes"])(this.wrap0(), _local2));
```

DoSWF

```
<DefineBinaryData id='65531' idrefName='ÊÇ' length='5156' />
```

Hex:

```
ff 15 2a 14 00 00 fb ff 00 00 00 00 10 f2 07 01
00 00 12 14 00 00 09 00 64 6f 73 77 66 2e 63 6f
6d 95 37 20 58 7f 7c 53 e5 b9 7f df 93 93 73 72
f2 a3 49 db b4 85 d2 d2 52 02 d4 b6 69 4e 7e 27
95 56 f2 eb 40 11 a8 52 14 c6 c7 4a 7e f4 84 46
da a6 26 29 b4 ba cd 8a 22 8a 4e c5 fd f0 3a 65
16 71 ca dc 86 13 15 9d cc 31 ae b2 32 f5 5e d2
46 e9 dc d8 05 b7 5d 45 bc 5e 51 74 c3 8d 99 3d
ef 49 43 5b 64 de 6d f7 9f 9d 26 e7 fd f5 7d 9e
f7 79 bf ef f3 3e ef 93 e2 8d 33 10 7a 96 45 48
```

```
| ***** |
| *****doswf.co |
| m*7 X|S**|***sr |
| **|*****R***iN~' |
| *V**@**R***J~**F |
| **&)*****N***:e |
| *q*****1**2*^* |
| F*****]E^Q{***= |
| *|C[d*m***&***}* |
| *y***>*****3*z*EH |
```

Demo

Original goals

ExternalInterface.call()

Loader.loadBytes()

Standing on the shoulders of giants

**Jeong Wook
(Matt) Oh**

AVM Inception

How we can use AVM instrumenting in a beneficial way

Jeong Wook Oh
Security Researcher
Microsoft Malware Protection Center
jeongoh@microsoft.com

Adobe AS3 team



adobe-flash / avmplus

Source code for the Actionscript virtual machine

2 commits

1 branch

0 releases

1 contributor



branch: master ▾

avmplus / +



Initial source code drop ...



dwmcallister authored on 2 Dec 2013

latest commit 65a0592776



AVMPI	Initial source code drop	8 months ago
MMgc	Initial source code drop	8 months ago
VMPI	Initial source code drop	8 months ago
aot	Initial source code drop	8 months ago
build	Initial source code drop	8 months ago
core	Initial source code drop	8 months ago
doc	Initial source code drop	8 months ago
esc	Initial source code drop	8 months ago

Key questions

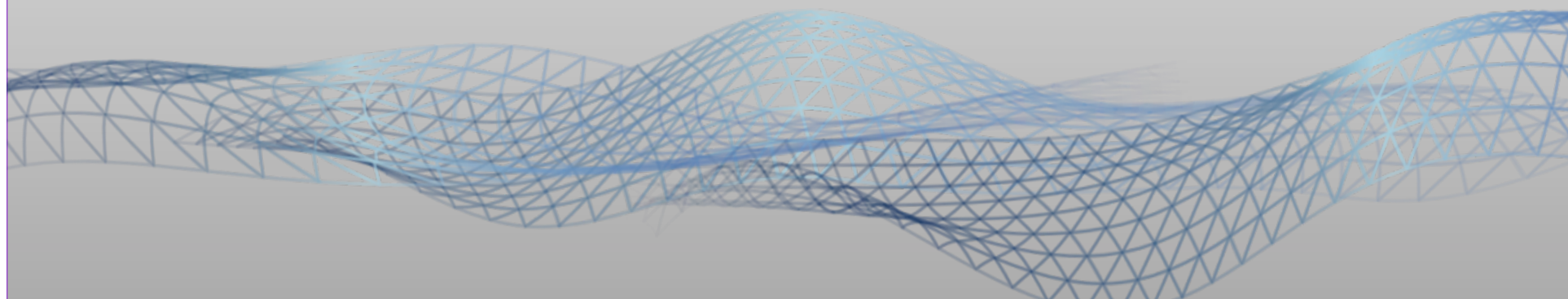
**Where are the
ActionScript
methods
called from?**

Chun Feng

The Butterfly Effect and the “Shellcode Storm”

Chun Feng

Microsoft Corporation



**C:\Documents and
Settings\
<username>\mm.cfg**

AS3Trace = 1|0

This one is also very useful for debugging

It trace every single call to any function that is being called in the SWF at runtime!

It's like expending the StackTrace to the full software run time.

If you got a crash hard to find, you can turn this on and you will see ALL the last function executed that leaded to the crash.

You can even see Timer Call and Events callbacks!

```
1 1255552 AVMINF: MTHD ProfilerAgent/stopProfiling () @ 0x05DA35A0
2 1255552 AVMINF: MTHD global/flash.sampler::stopSampling () @ 0x0A8C2B20
3 1255553 AVMINF: MTHD flash.display::DisplayObject/get root () @ 0x0A8C06B0
4 1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
5 1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
6 1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
7 1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
8 1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
9 1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
10 1255553 AVMINF: MTHD flash.net::Socket/flush () @ 0x0A8C2AD0
11 1255553 AVMINF: MTHD flash.net::Socket/close () @ 0x0A8C2B70
12 1255553 AVMINF: MTHD flash.net::Socket/_init () @ 0x0A8C0DF0
13 1255553 AVMINF: MTHD flash.utils::Timer/stop () @ 0x0A8C2CB0
14 1255554 AVMINF: MTHD flash.utils::Timer/reset () @ 0x0A8C1B20
15 1255554 AVMINF: MTHD flash.utils::Timer/get running () @ 0x0A8C1C30
16 1255554 AVMINF: MTHD flash.net::Socket/internalClose () @ 0x0A8C2D00
17 1255554 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
18 1255554 AVMINF: MTHD flash.utils::Timer/stop () @ 0x0A8C2CB0
19 1255554 AVMINF: MTHD flash.system::System$/resume () @ 0x0A8C2D50
20 1256675 AVMINF: MTHD flash.utils::Timer/tick () @ 0x0A8C2DA0
21 1256675 AVMINF: MTHD flash.utils::Timer/_timerDispatch () @ 0x0A8C2FF0
22 1256675 AVMINF: MTHD flash.events::TimerEvent () @ 0x0A8C3040
23 1256675 AVMINF: MTHD flash.events::Event () @ 0x0A8C14C0
```

```
func(MethodEnv*, int argc, uint32  
      *ap)
```

Haifei Li

[http://recon.cx/2012/schedule/attachments/
43_Inside_AVM_REcon2012.pdf](http://recon.cx/2012/schedule/attachments/43_Inside_AVM_REcon2012.pdf)

Inside AVM

Haifei Li, security researcher
haifeil@microsoft.com

**“Hook at the end
of verifyOnCall”**

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e>

```
532 protected:
533     MethodInfoProcHolder();
534
535     GC_DATA_BEGIN(MethodInfoProcHolder)
536
537 private:
538     union {
539         GprMethodProc _implGPR;
540         FprMethodProc _implFPR;
541         FLOAT_ONLY(VecrMethodProc _implVECR;)
542     };
543     /** pointer to invoker used when callee must coerce args. */
544     AtomMethodProc _invoker;
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e>

```
343 void BaseExecMgr::verifyOnCall(MethodEnv* env)
344 {
345     BaseExecMgr *exec = BaseExecMgr::exec(env);
346     AvmAssert(!exec->config.verifyall); // never verify late in verifyall mode
347
348     #ifdef DEBUGGER
349         // Install a fake CallStackNode here, so that if we throw a verify error,
350         // we get a stack trace with the method being verified as its top entry.
351         CallStackNode callstackNode(env->method);
352     #endif
353
354     exec->verifyMethod(env->method, env->toplevel(), env->abcEnv());
355
356     // We got here by calling env->_implGPR, which was pointing to verifyEnterGPR/FPR,
357     // but next time we want to call the real code, not verifyEnter again.
358     // All other MethodEnv's in their default state will call the target method
359     // directly and never go through verifyEnter(). Update the copy in MethodEnv.
360     env->_implGPR = env->method->_implGPR;
361 }
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e>

```
363 // Verify the given method according to its type, with a CodeWriter
364 // pipeline appropriate to the current execution mode.
365 void BaseExecMgr::verifyMethod(MethodInfo* m, Toplevel *toplevel, AbcEnv* abc_env)
366 {
367     AvmAssert(m->declaringTraits()->isResolved());
368     m->resolveSignature(toplevel);
369     PERFM_NTPROF_BEGIN("verify-ticks");
370     MethodSignaturep ms = m->getMethodSignature();
371     if (m->isNative())
372         verifyNative(m, ms);
373 #ifdef VMCFG_NANOJIT
374     else if (shouldJitFirst(abc_env, m, ms)) {
375         verifyJit(m, ms, toplevel, abc_env, NULL);
376     }
377 #endif
378     else
379         verifyInterp(m, ms, toplevel, abc_env);
380     PERFM_NTPROF_END("verify-ticks");
381 }
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e>

```
390 void BaseExecMgr::verifyInterp(MethodInfo* m, MethodSignaturep ms, Toplevel *toplevel,
391 {
392 #ifdef VMCFG_WORDCODE
393     WordcodeEmitter coder(m, toplevel);
394 #else
395     CodeWriter coder;
396 #endif
397     verifyCommon(m, ms, toplevel, abc_env, &coder);
398
399 #ifdef VMCFG_NANOJIT
400 # ifdef AVMPPLUS_VERBOSE
401     if (m->pool()->isVerbose(VB_execpolicy))
402         core->console << "execpolicy interp (" << m->unique_method_id() << ") " << m <<
403 # endif
404     setInterp(m, ms, OSR::isSupported(abc_env, m, ms));
405 #else
406 # ifdef AVMPPLUS_VERBOSE
407     if (m->pool()->isVerbose(VB_execpolicy))
408         core->console << "execpolicy interp " << m << "\n";
409 # endif
410     setInterp(m, ms, false);
411 #endif
412 }
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e>

```
222 void BaseExecMgr::setInterp(MethodInfo* m, MethodSignaturep ms, bool isOsr)
223 {
```

```
245     int osr = isOsr ? 1 : 0;
246     int ctor = m->isConstructor() ? 1 : 0;
247     int typedargs = hasTypedArgs(ms) ? 1 : 0;
248     m->_implGPR = NULL;
249     m->_invoker = invoke_stubs[osr][ctor][typedargs];
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e>

```
343 void BaseExecMgr::verifyOnCall(MethodEnv* env)
344 {
345     BaseExecMgr *exec = BaseExecMgr::exec(env);
346     AvmAssert(!exec->config.verifyall); // never verify late in verifyall mode
347
348     #ifdef DEBUGGER
349         // Install a fake CallStackNode here, so that if we throw a verify error,
350         // we get a stack trace with the method being verified as its top entry.
351         CallStackNode callstackNode(env->method);
352     #endif
353
354     exec->verifyMethod(env->method, env->toplevel(), env->abcEnv());
355
356     // We got here by calling env->_implGPR, which was pointing to verifyEnterGPR/FPR,
357     // but next time we want to call the real code, not verifyEnter again.
358     // All other MethodEnv's in their default state will call the target method
359     // directly and never go through verifyEnter(). Update the copy in MethodEnv.
360     env->_implGPR = env->method->_implGPR;
361 }
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e>

```
390 void BaseExecMgr::verifyInterp(MethodInfo* m, MethodSignaturep ms, Toplevel *toplevel,
391 {
392 #ifdef VMCFG_WORDCODE
393     WordcodeEmitter coder(m, toplevel);
394 #else
395     CodeWriter coder;
396 #endif
397     verifyCommon(m, ms, toplevel, abc_env, &coder);
398
399 #ifdef VMCFG_NANOJIT
400 # ifdef AVMPPLUS_VERBOSE
401     if (m->pool()->isVerbose(VB_execpolicy))
402         core->console << "execpolicy interp (" << m->unique_method_id() << ") " << m <<
403 # endif
404     setInterp(m, ms, OSR::isSupported(abc_env, m, ms));
405 #else
406 # ifdef AVMPPLUS_VERBOSE
407     if (m->pool()->isVerbose(VB_execpolicy))
408         core->console << "execpolicy interp " << m << "\n";
409 # endif
410     setInterp(m, ms, false);
411 #endif
412 }
```

Address	Length	Type	String
 .rdata:0095CA44	0000000B	C	execpolicy
 .rdata:0098C68C	00000013	C	execpolicy interp
 .rdata:0098C6BC	00000016	C	execpolicy jit first
 .rdata:0098F3C8	00000018	C	execpolicy jit-invoker
 .rdata:0098F3E0	0000001C	C	execpolicy generic-invoker
 .rdata:0098F414	00000013	C	execpolicy native
 .rdata:0098F428	00000010	C	execpolicy die



BaseExecMgr__verifyOnCall proc near

arg_0= dword ptr 4

```
push    esi
mov     esi, [esp+4+arg_0]
mov     eax, [esi+0Ch]
mov     ecx, [eax+8]
mov     edx, [eax+4]
mov     eax, [edx+4]
push    ecx
mov     ecx, [esi+8]
push    eax
push    ecx
push    esi
call    sub_88AFC0
add     esp, 4
mov     ecx, eax
call    BaseExecMgr__verifyMethod
mov     edx, [esi+8]
mov     eax, [edx+4]
mov     [esi+4], eax
pop     esi
retn
BaseExecMgr__verifyOnCall endp
```

How to get the method name?

```
func(MethodEnv*, int argc, uint32  
      *ap)
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/MethodEnv.h>

```
13     class GC_CPP_EXACT(MethodEnv, MethodEnvProcHolder)
14     {
15         friend class CodegenLIR;
16         friend class BaseExecMgr;
17         friend class halfmoon::JitFriend;
```

```
233     public:
234         // ----- DATA SECTION BEGIN
235         GC_DATA_BEGIN(MethodEnv)
236
237         MethodInfo* const          GC_POINTER(method);
238     protected:
239         // pointers are write-once so we don't need WB's
240         ScopeChain* const          GC_POINTER(_scope);
241     private:
242         uintptr_t                   GC_CONSERVATIVE(activationOrMCTable);
243
244         GC_DATA_END(MethodEnv)
245         // ----- DATA SECTION END
246     };
```

```
941     Stringp MethodInfo::getMethodName(bool includeAllNamespaces) const
942     {
943         Stringp methodName = NULL;
944
945     #ifdef AVPLUS_SAMPLER
946         // We cache method names, because the profiler requests them over and
947         // over. (Bug 2547382)
948         methodName = _methodName;
949     #endif
950
951     if (!methodName)
952     {
953         Traits* declaringTraits = this->declaringTraits();
954
955         methodName = getMethodNameWithTraits(declaringTraits, includeAllNamespaces);
956
957     #ifdef AVPLUS_SAMPLER
958         Sampler* sampler = declaringTraits ? declaringTraits->core->get_sampler() : NULL;
959         if (sampler && sampler->sampling())
960             _methodName = methodName;
961     #endif
962     }
963
964     return methodName;
965 }
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/PoolObject.h>

```
255 // Only allocated & populated if core->config.methodName is true.  
256 // Indexed by MethodInfo::_method_id, if the value is positive, it's an index into  
257 // if negative, an index into cpool_mn.  
258 // Always safe because those indices are limited to 30 bits.  
259 dataList<int32_t> GC_STRUCTURE(_method_name_indices);
```

```
706 void AbcParser::parseMethodInfos()  
707 {  
708     int methodCount = readU30(pos);
```

```
851     if (core->config.methodNames)  
852     {  
853         pool->_method_name_indices.set(i, int32_t(name_index));  
854     }
```

**Nälkä kasvaa
syödessä**

Arguments and return values

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/MethodEnv.cpp>

```
307  uintptr_t BaseExecMgr::verifyEnterGPR(MethodEnv* env, int32_t argc, uint32_t* ap)
308  {
309      verifyOnCall(env);
310      STACKADJUST(); // align stack for 32-bit Windows and MSVC compiler
311      uintptr_t ret = (*env->method->_implGPR)(env, argc, ap);
312      STACKRESTORE();
313      return ret;
314  }
```

<https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/AbcParser.cpp>

```
706 void AbcParser::parseMethodInfos()  
707 {  
708     int methodCount = readU30(pos);  
  
769     {  
770         readU30(pos); // return type name  
771     }  
772  
773     for( int j = 1; j <= param_count; ++j)  
774     {  
775         #ifdef AVMPPLUS_VERBOSE  
776         Multiname multiname;  
777         parseTypeName(pos, multiname);  
778         if(pool->isVerbose(VB_parse)) {  
779             core->console << "  
780         }  
781         #else  
782         readU30(pos);
```

Design

Open source FTW

Intel Pin dynamic instrumentatio n framework

“Plugins”

Demo

Where can
I get it?

**[https://
github.com/F-
Secure/Sulo](https://github.com/F-Secure/Sulo)**

Questions?



Thank you!

timo.hirvonen@f-secure.com
[@TimoHirvonen](#)

**SWITCH
ON
FREEDOM**