Dynamic Flash instrumentation for fun and profit

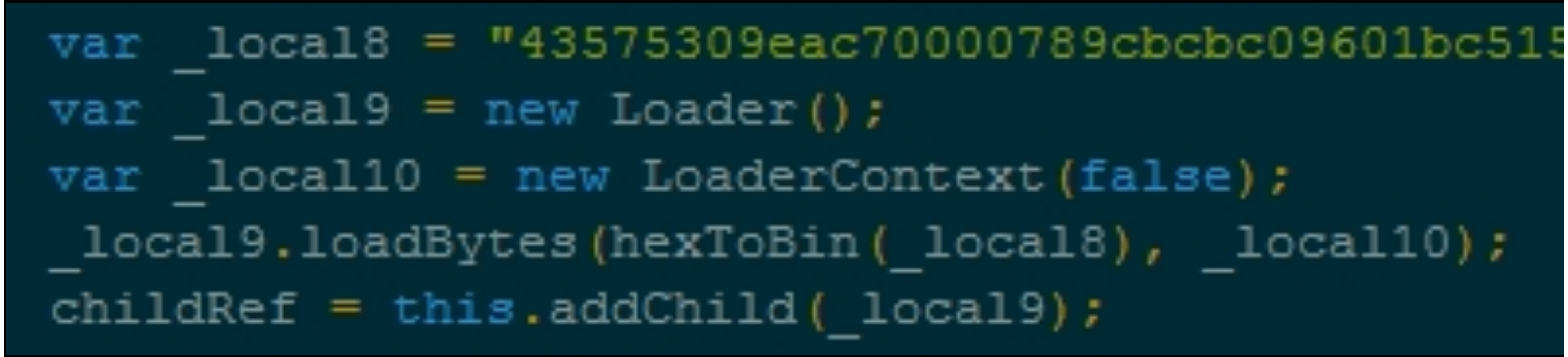
Timo Hirvonen

Black Hat USA 2014

F-Secure

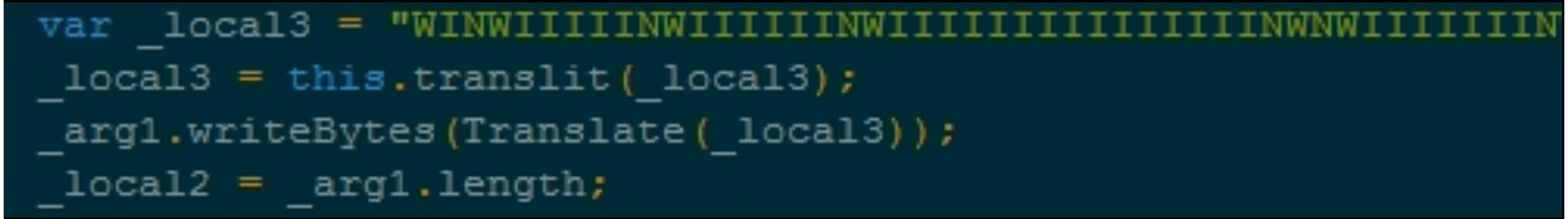
Motivation

RSA
CVE-2011-060
9



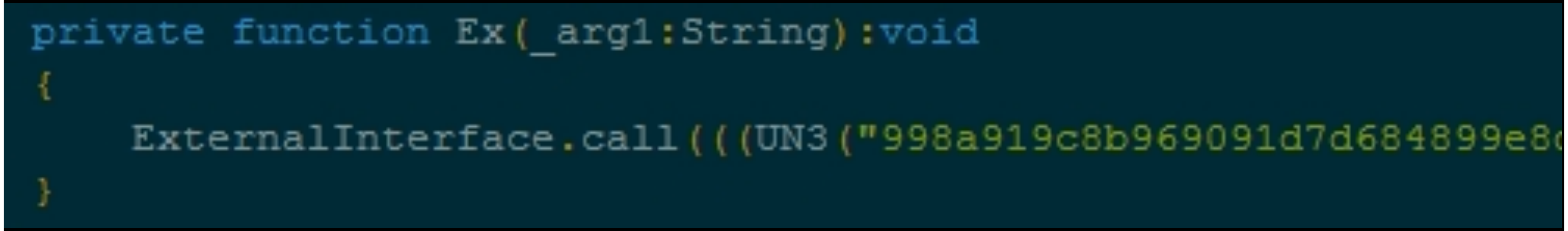
```
var _local8 = "43575309eac70000789cbcbc09601bc515
var _local9 = new Loader();
var _local10 = new LoaderContext(false);
_local9.loadBytes(hexToBin(_local8), _local10);
childRef = this.addChild(_local9);
```

CosmicDuke CVE-2011-0611



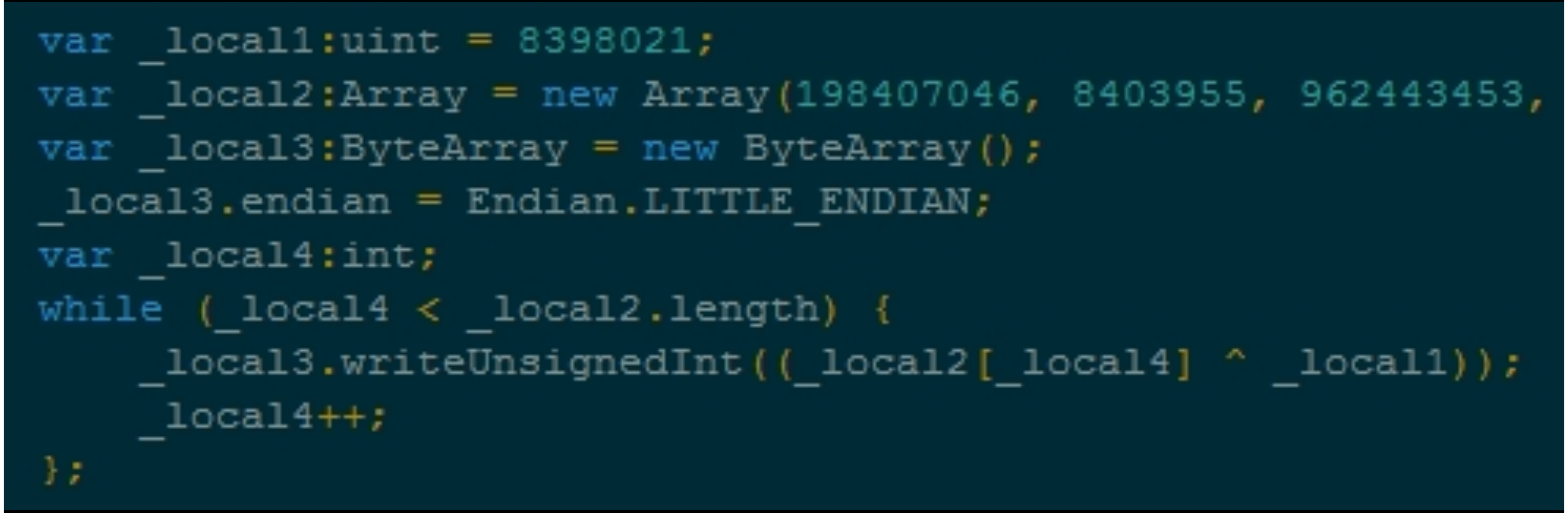
```
var _local3 = "WINWIIIIINWIIIIIIINWIIIIIIIIIIIIIIIIINWNWIIIIIIIIN
_local3 = this.translit(_local3);
_arg1.writeBytes(Translate(_local3));
_local2 = _arg1.length;
```

Youtube ad → Styx EK



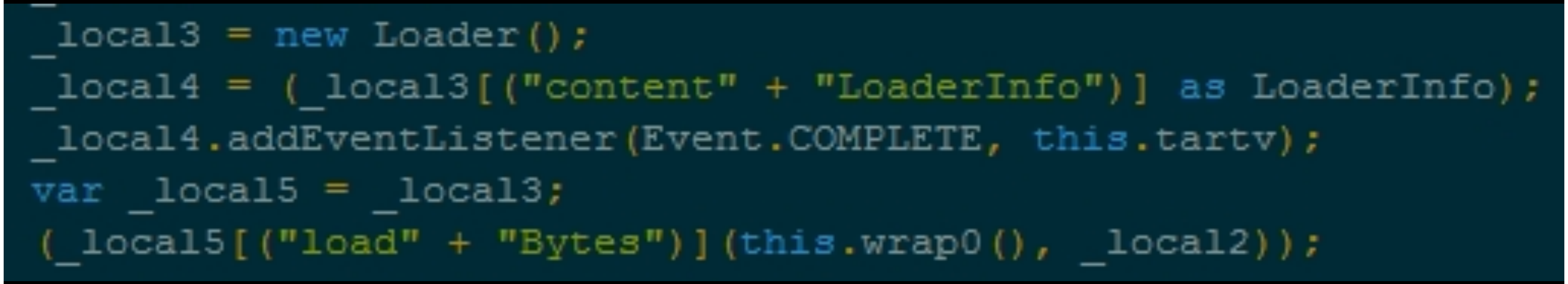
```
private function Ex(_arg1:String):void
{
    ExternalInterface.call(((UN3("998a919c8b969091d7d684899e8
}
```

Fiesta EK CVE-2014-0497



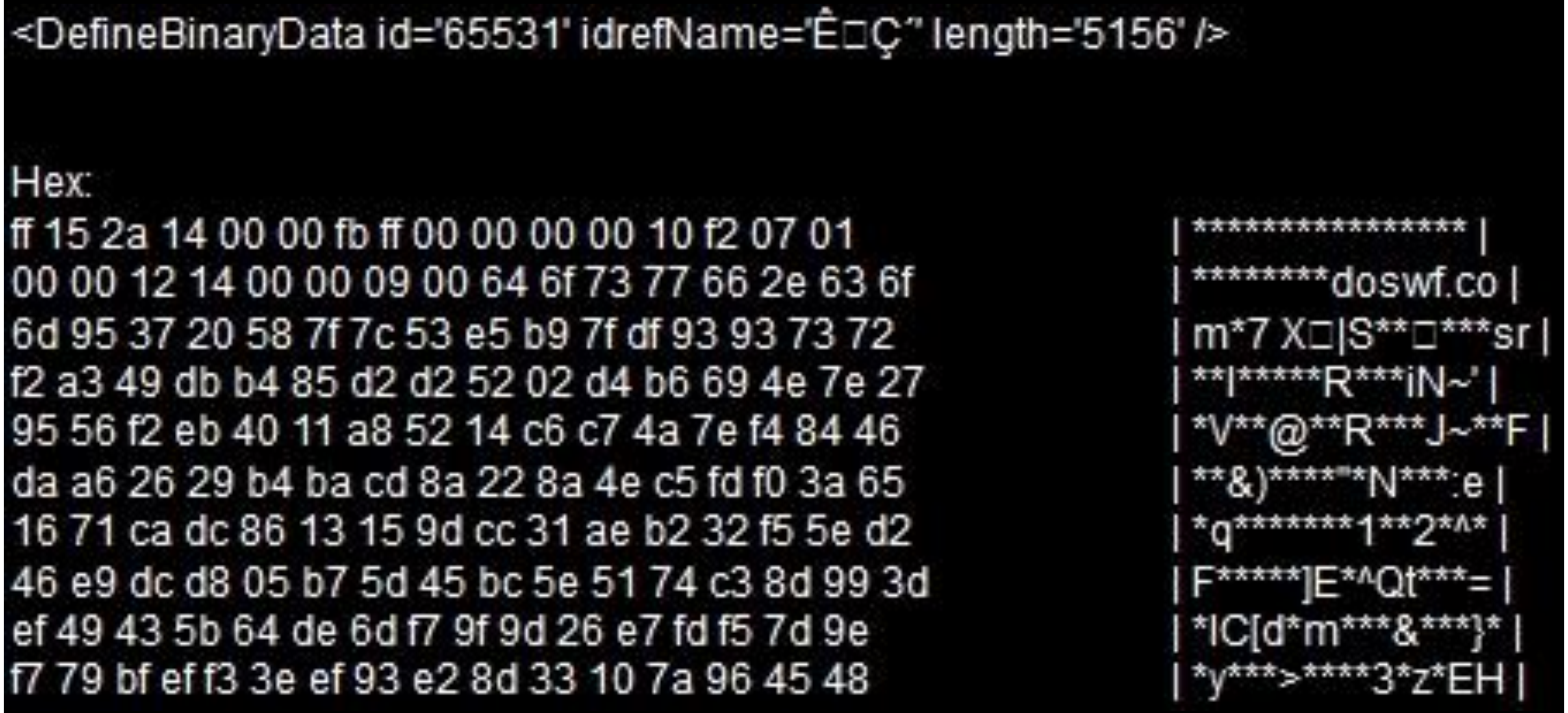
```
var _local1:uint = 8398021;
var _local2:Array = new Array(198407046, 8403955, 962443453,
var _local3:ByteArray = new ByteArray();
_local3.endian = Endian.LITTLE_ENDIAN;
var _local4:int;
while (_local4 < _local2.length) {
    _local3.writeUnsignedInt((_local2[_local4] ^ _local1));
    _local4++;
};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

Fiesta 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='Ê□Ç'' 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      | ********doswf.co |
6d 95 37 20 58 7f 7c 53 e5 b9 7f df 93 93 73 72      | m*7 X□|S**□***sr |
f2 a3 49 db b4 85 d2 d2 52 02 d4 b6 69 4e 7e 27      | **I*****R***iN~' |
95 56 f2 eb 40 11 a8 52 14 c6 c7 4a 7e f4 84 46      | *V**@**R***J~**F |
da a6 26 29 b4 ba cd 8a 22 8a 4e c5 fd f0 3a 65      | **&)****"*N***:e |
16 71 ca dc 86 13 15 9d cc 31 ae b2 32 f5 5e d2      | *q*******1**2*^* |
46 e9 dc d8 05 b7 5d 45 bc 5e 51 74 c3 8d 99 3d      | F*****]E*^Qt***= |
ef 49 43 5b 64 de 6d f7 9f 9d 26 e7 fd f5 7d 9e      | *IC[d*m***&***}* |
f7 79 bf ef f3 3e ef 93 e2 8d 33 10 7a 96 45 48      | *y***>****3*z*EH |
```

F-Secure

Demo

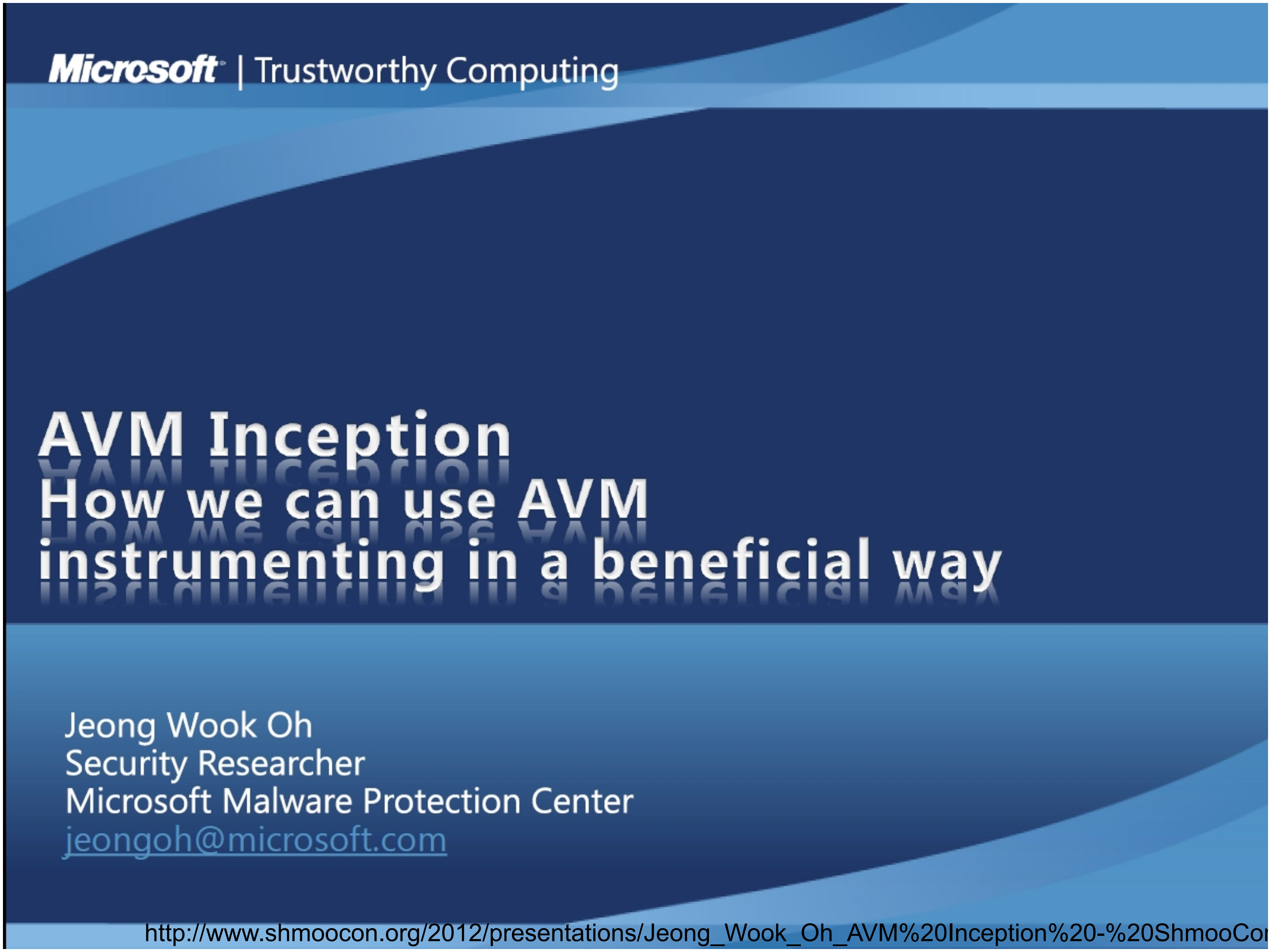
Original goals

ExternalInterface.call()

Loader.loadBytes()

Standing on the shoulders of giants

Jeong Wook (Matt) Oh



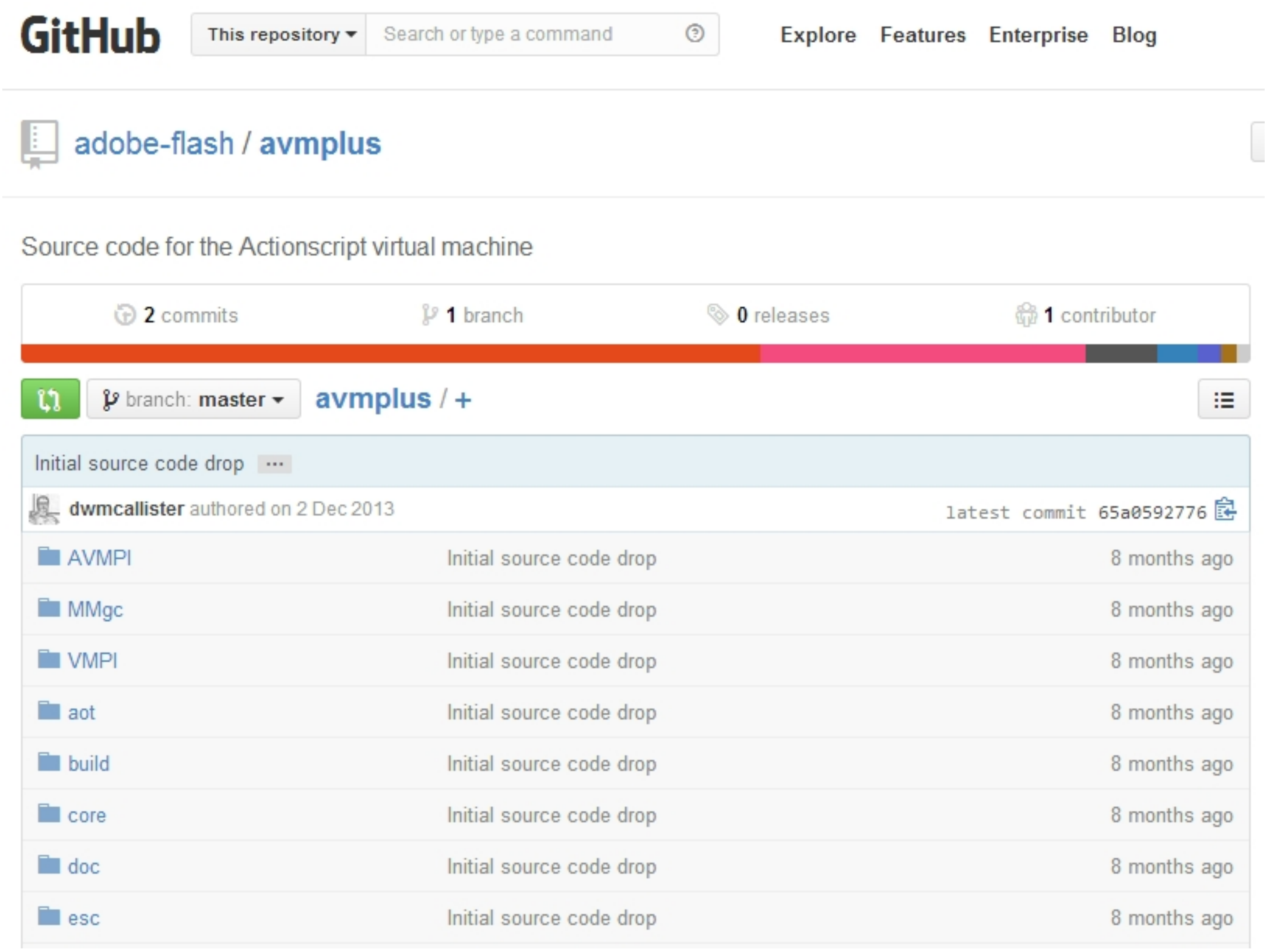
Microsoft | Trustworthy Computing

AVM Inception
How we can use AVM instrumenting in a beneficial way

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

http://www.shmoocon.org/2012/presentations/Jeong_Wook_Oh_AVM%20Inception%20-%20ShmooCo

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

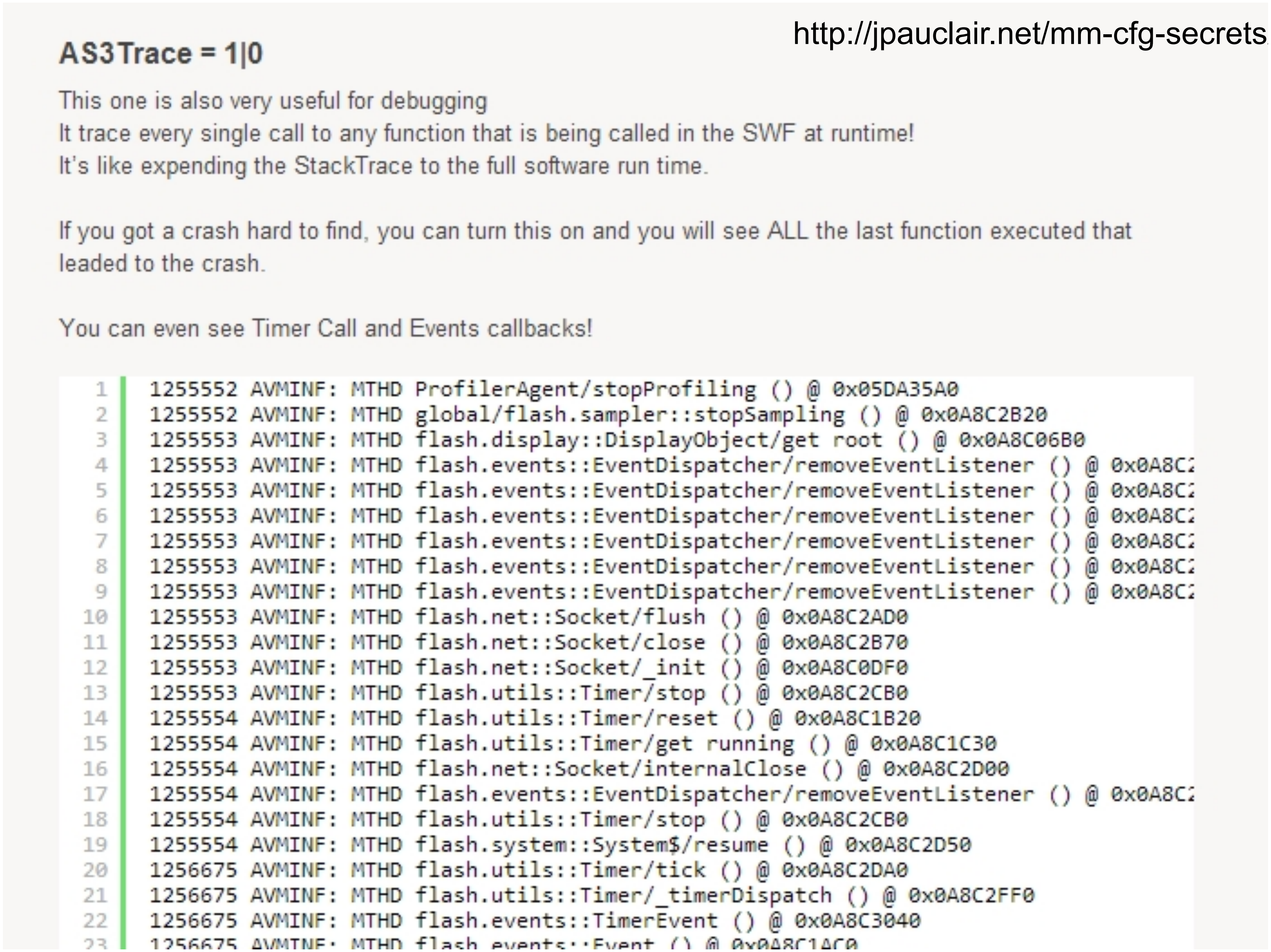
The Butterfly Effect and the “Shellcode Storm”

Chun Feng

Microsoft Corporation

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

http://jpauclair.net/mm-cfg-secrets



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!

```
1255552 AVMINF: MTHD ProfilerAgent/stopProfiling () @ 0x05DA35A0
1255552 AVMINF: MTHD global/flash.sampler::stopSampling () @ 0x0A8C2B20
1255553 AVMINF: MTHD flash.display::DisplayObject/get root () @ 0x0A8C06B0
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
1255553 AVMINF: MTHD flash.net::Socket/flush () @ 0x0A8C2AD0
1255553 AVMINF: MTHD flash.net::Socket/close () @ 0x0A8C2B70
1255553 AVMINF: MTHD flash.net::Socket/_init () @ 0x0A8C0DF0
1255553 AVMINF: MTHD flash.utils::Timer/stop () @ 0x0A8C2CB0
1255554 AVMINF: MTHD flash.utils::Timer/reset () @ 0x0A8C1B20
1255554 AVMINF: MTHD flash.utils::Timer/get running () @ 0x0A8C1C30
1255554 AVMINF: MTHD flash.net::Socket/internalClose () @ 0x0A8C2D00
1255554 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2
1255554 AVMINF: MTHD flash.utils::Timer/stop () @ 0x0A8C2CB0
1255554 AVMINF: MTHD flash.system::System$/resume () @ 0x0A8C2D50
1256675 AVMINF: MTHD flash.utils::Timer/tick () @ 0x0A8C2DA0
1256675 AVMINF: MTHD flash.utils::Timer/_timerDispatch () @ 0x0A8C2FF0
1256675 AVMINF: MTHD flash.events::TimerEvent () @ 0x0A8C3040
1256675 AVMINF: MTHD flash.events::Event () @ 0x0A8C1AC0
```

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

Haifei Li

http://recon.cx/2012/schedule/attachments/43_Inside_AVM_REcon2012.pdf

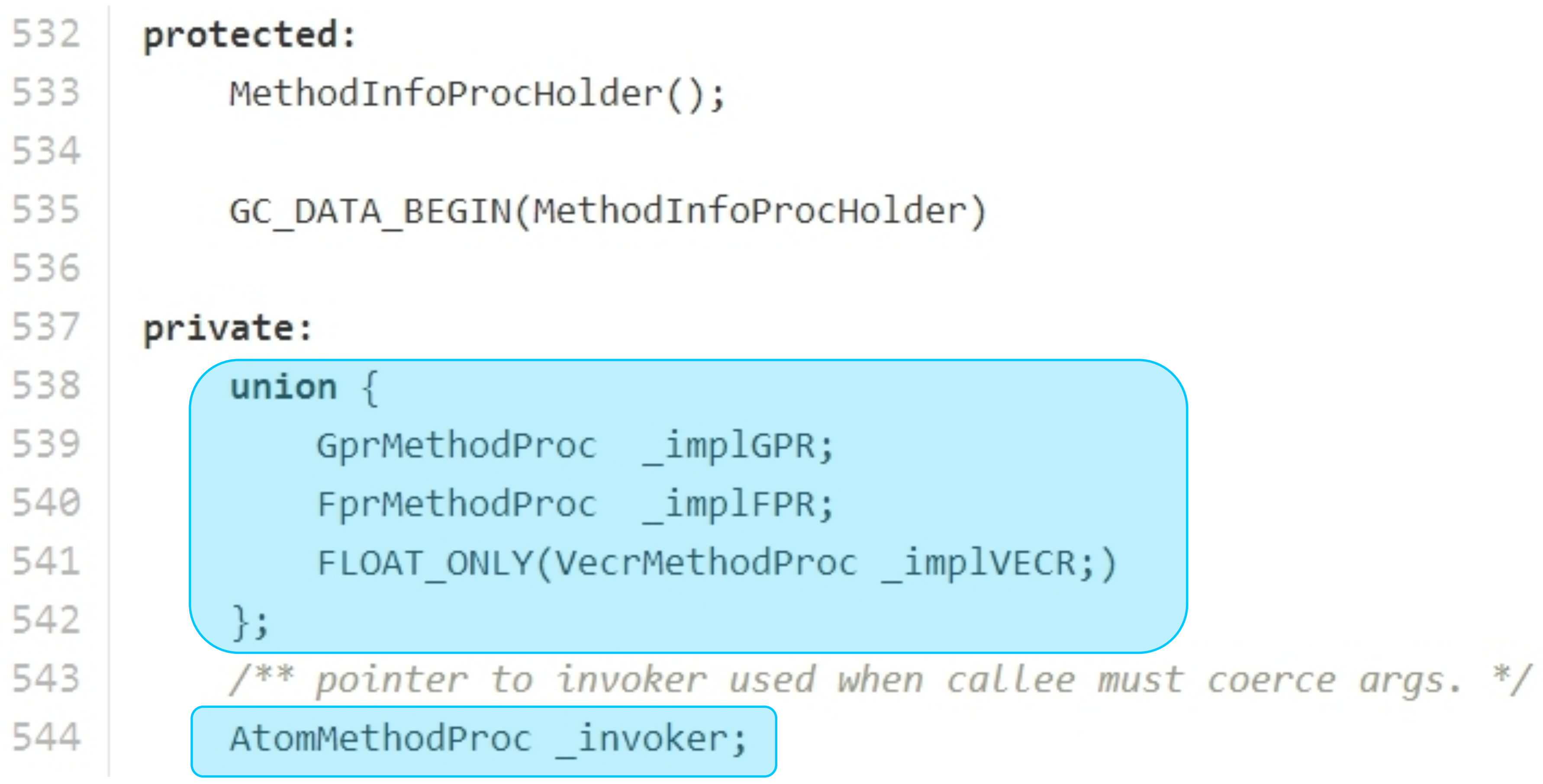
Inside AVM

Haifei Li, security researcher
haifeil@microsoft.com

REcon 2012, Montreal

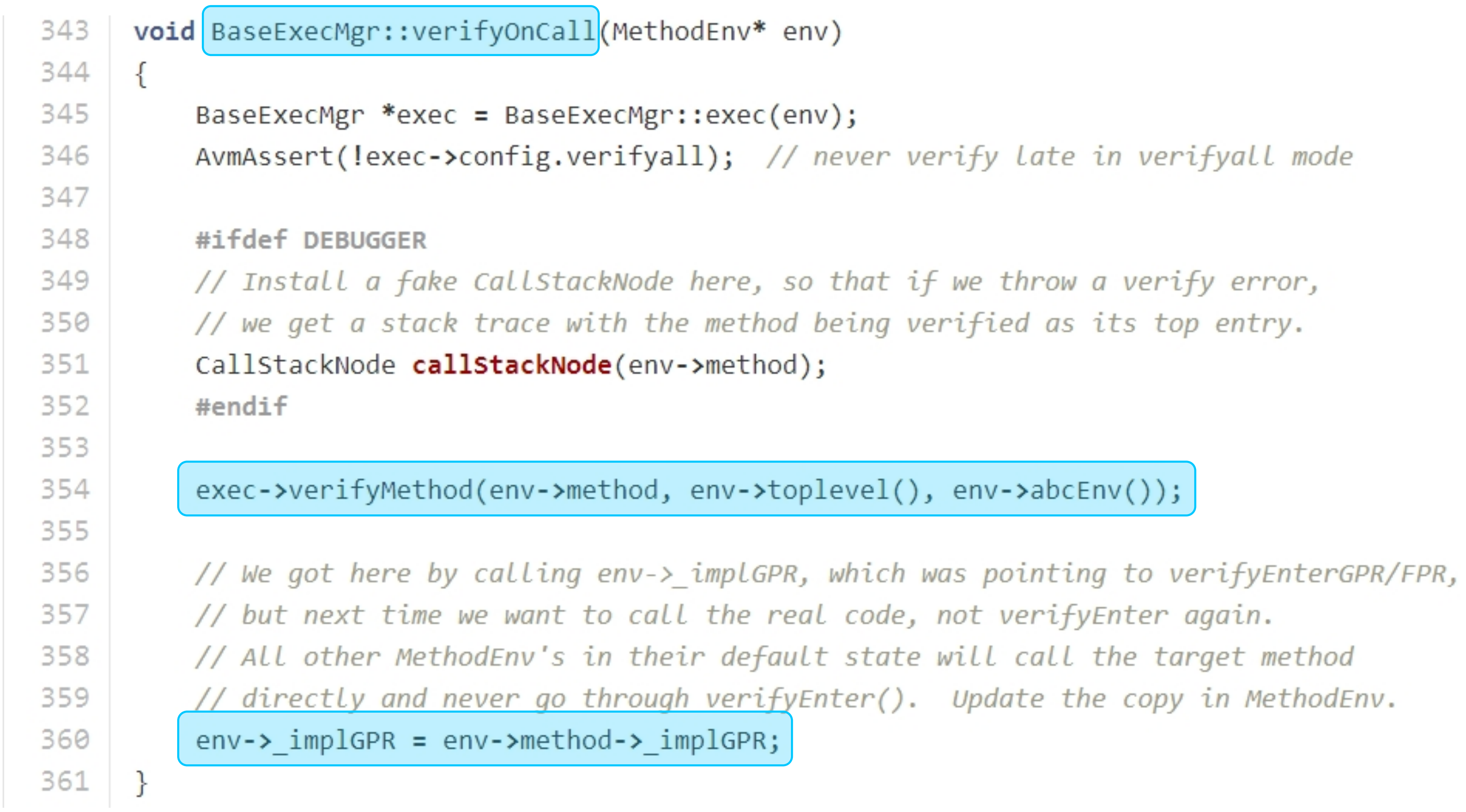
“Hook at the end of verifyOnCall”

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



```
protected:
    MethodInfoProcHolder();

    GC_DATA_BEGIN(MethodInfoProcHolder)

private:
    union {
        GprMethodProc   _implGPR;
        FprMethodProc   _implFPR;
        FLOAT_ONLY(VecrMethodProc _implVECR;)
    };
    /** pointer to invoker used when callee must coerce args. */
    AtomMethodProc _invoker;
```

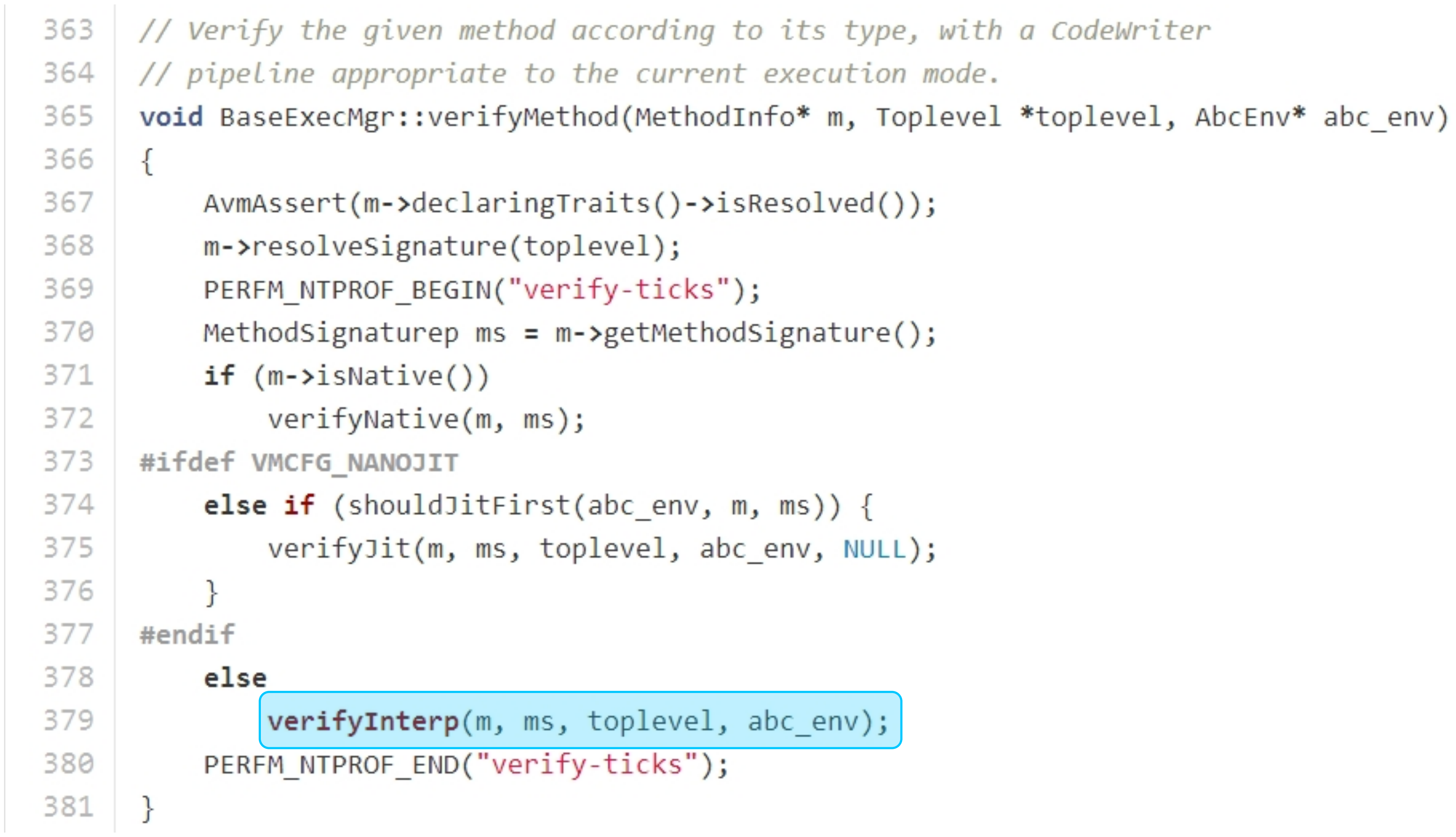
https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e



```
void BaseExecMgr::verifyOnCall(MethodEnv* env)
{
    BaseExecMgr *exec = BaseExecMgr::exec(env);
    AvmAssert(!exec->config.verifyall);  // never verify late in verifyall mode

    #ifdef DEBUGGER
    // Install a fake CallStackNode here, so that if we throw a verify error,
    // we get a stack trace with the method being verified as its top entry.
    CallStackNode callStackNode(env->method);
    #endif

    exec->verifyMethod(env->method, env->toplevel(), env->abcEnv());

    // We got here by calling env->_implGPR, which was pointing to verifyEnterGPR/FPR,
    // but next time we want to call the real code, not verifyEnter again.
    // All other MethodEnv's in their default state will call the target method
    // directly and never go through verifyEnter().  Update the copy in MethodEnv.
    env->_implGPR = env->method->_implGPR;
}
```

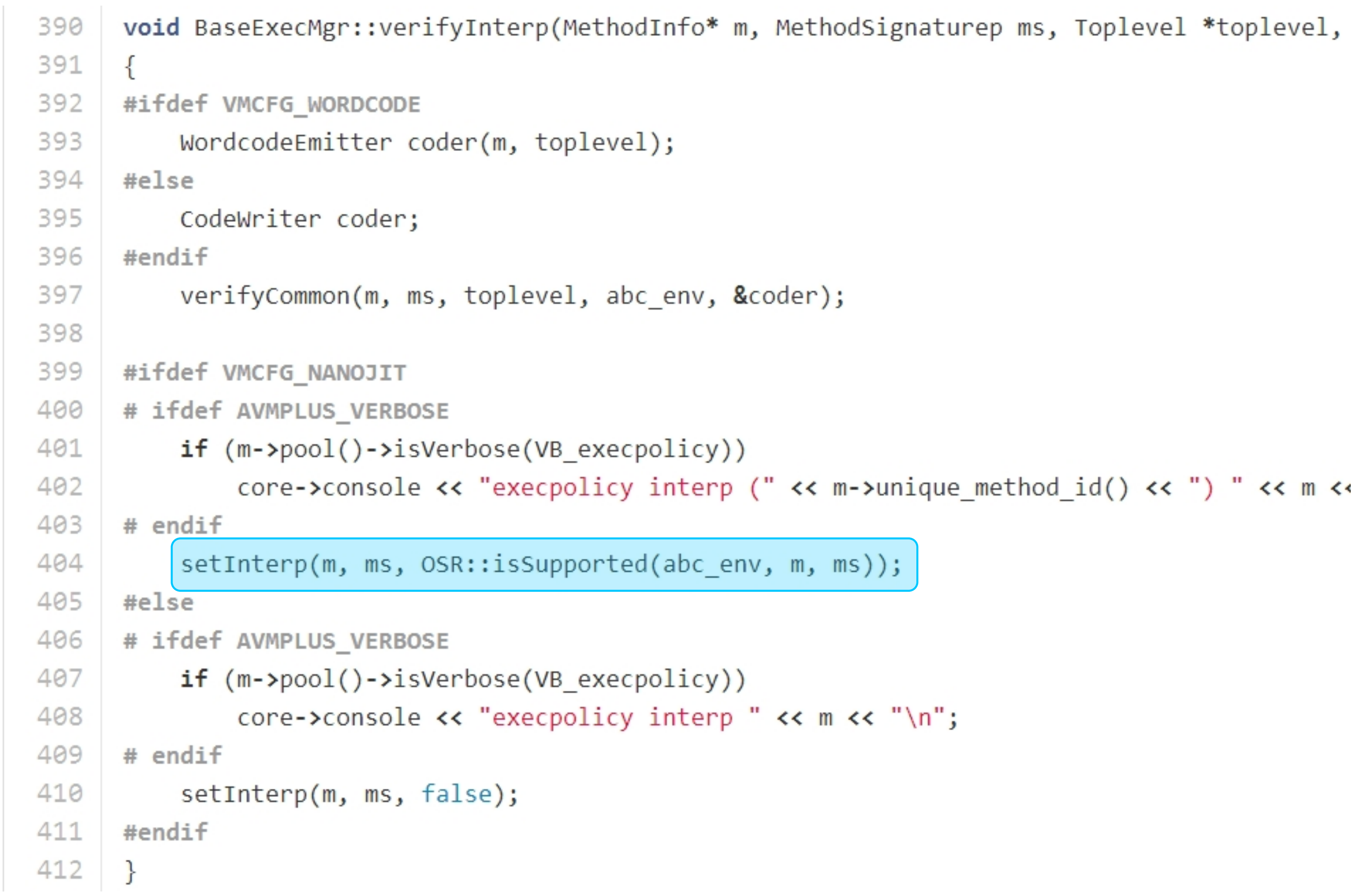
F-Secure

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

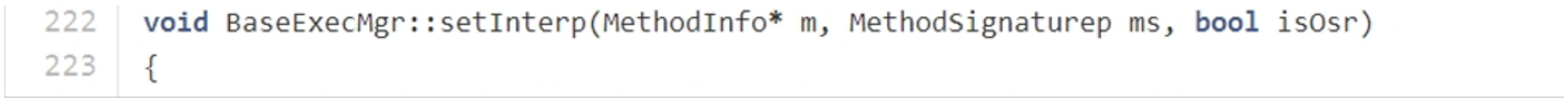


```
// Verify the given method according to its type, with a CodeWriter
// pipeline appropriate to the current execution mode.
void BaseExecMgr::verifyMethod(MethodInfo* m, Toplevel *toplevel, AbcEnv* abc_env)
{
    AvmAssert(m->declaringTraits()->isResolved());
    m->resolveSignature(toplevel);
    PERFM_NTPROF_BEGIN("verify-ticks");
    MethodSignaturep ms = m->getMethodSignature();
    if (m->isNative())
        verifyNative(m, ms);
#ifdef VMCFG_NANOJIT
    else if (shouldJitFirst(abc_env, m, ms)) {
        verifyJit(m, ms, toplevel, abc_env, NULL);
    }
#endif
    else
        verifyInterp(m, ms, toplevel, abc_env);
    PERFM_NTPROF_END("verify-ticks");
}
```

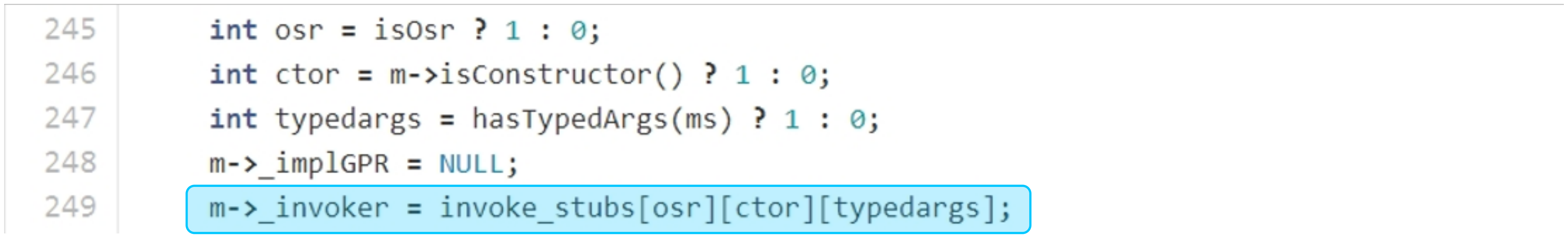
https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e›


```
void BaseExecMgr::verifyInterp(MethodInfo* m, MethodSignaturep ms, Toplevel *toplevel,
{
#ifdef VMCFG_WORDCODE
    WordcodeEmitter coder(m, toplevel);
#else
    CodeWriter coder;
#endif
    verifyCommon(m, ms, toplevel, abc_env, &coder);

#ifdef VMCFG_NANOJIT
# ifdef AVMPLUS_VERBOSE
    if (m->pool()->isVerbose(VB_execpolicy))
        core->console << "execpolicy interp (" << m->unique_method_id() << ") " << m <‹
# endif
    setInterp(m, ms, OSR::isSupported(abc_env, m, ms));
#else
# ifdef AVMPLUS_VERBOSE
    if (m->pool()->isVerbose(VB_execpolicy))
        core->console << "execpolicy interp " << m << "\n";
# endif
    setInterp(m, ms, false);
#endif
}
```

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

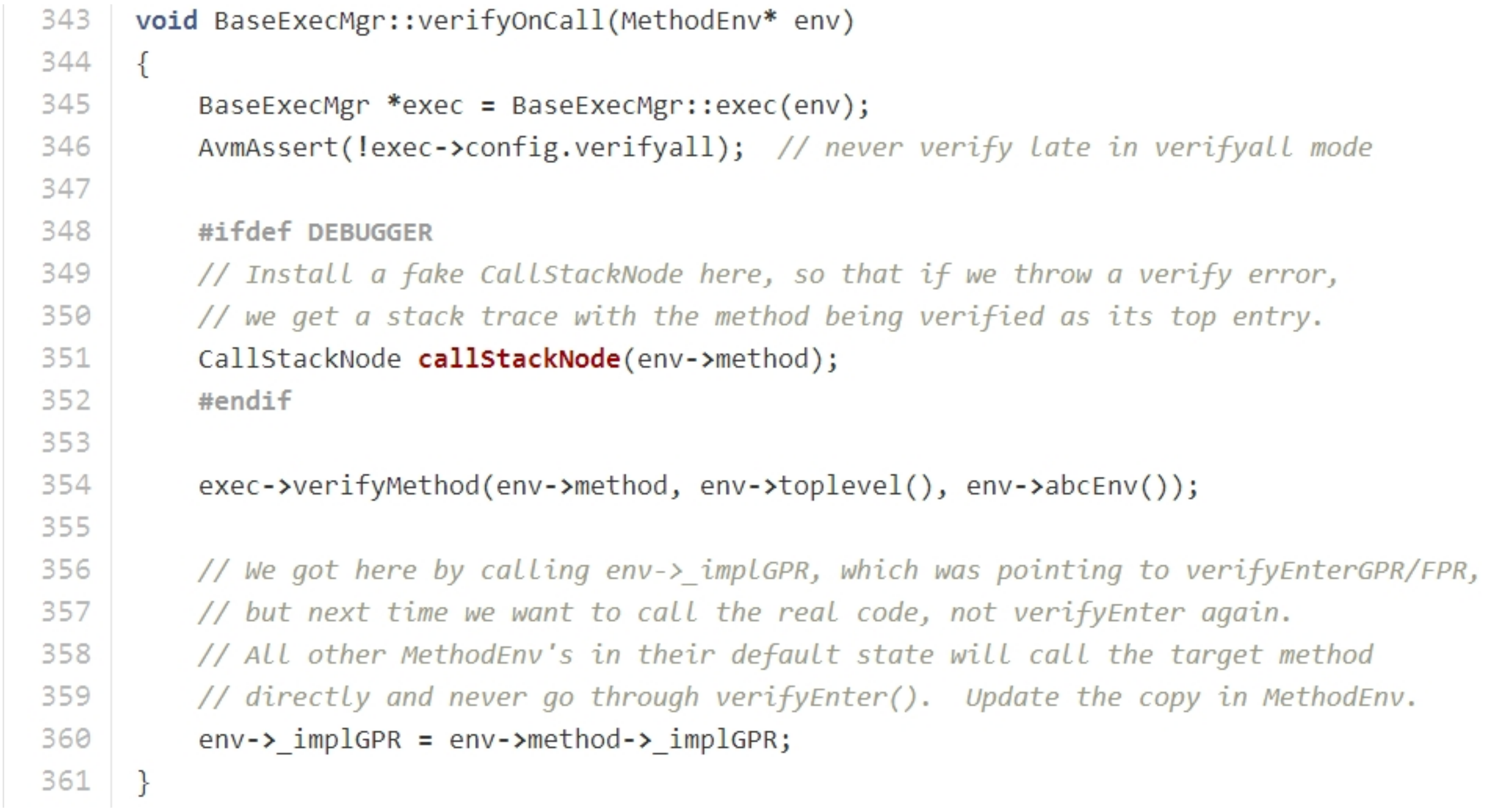


```
void BaseExecMgr::setInterp(MethodInfo* m, MethodSignaturep ms, bool isOsr)
{
```



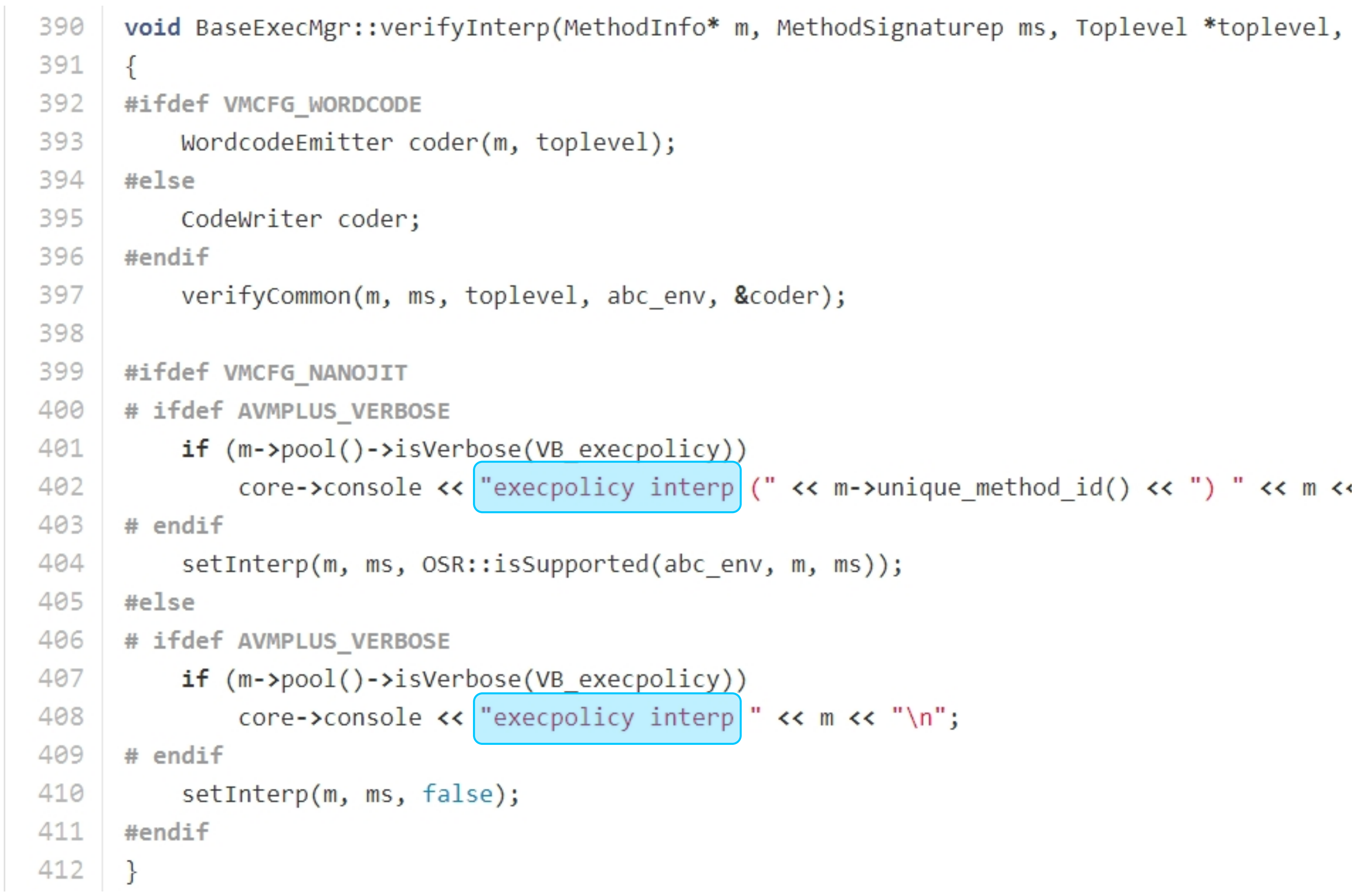
```
    int osr = isOsr ? 1 : 0;
    int ctor = m->isConstructor() ? 1 : 0;
    int typedargs = hasTypedArgs(ms) ? 1 : 0;
    m->_implGPR = NULL;
    m->_invoker = invoke_stubs[osr][ctor][typedargs];
```

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

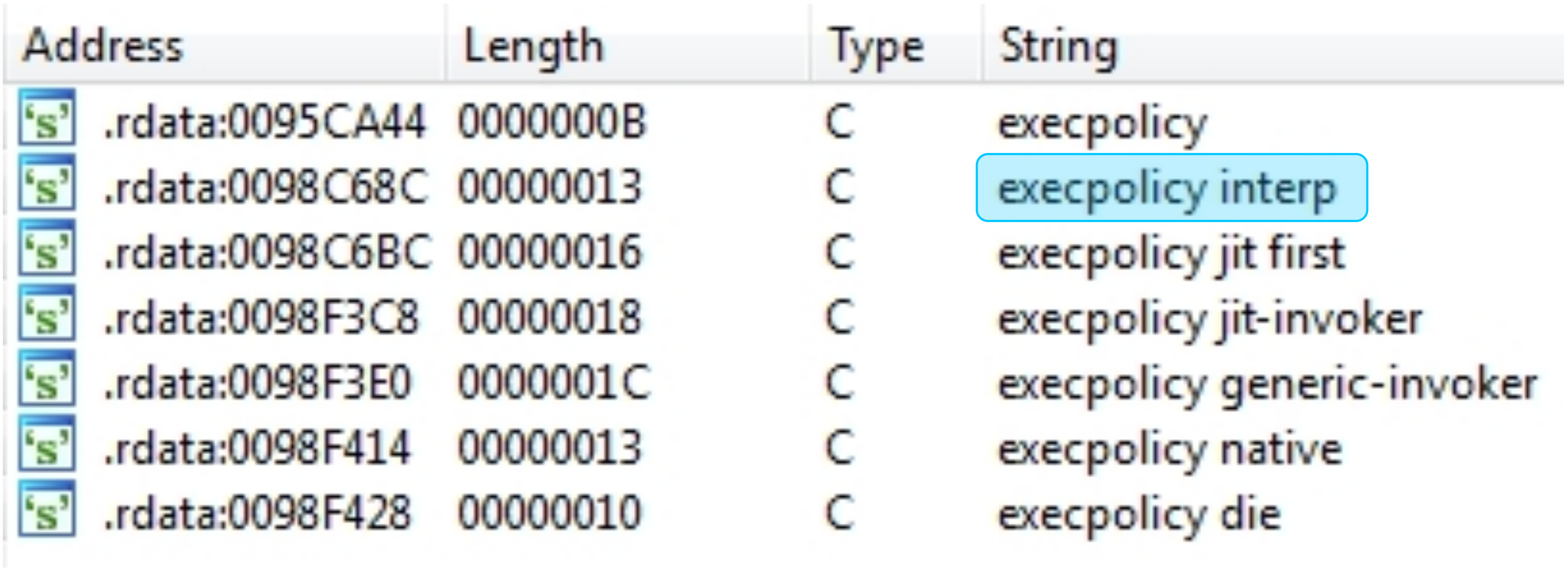


```
void BaseExecMgr::verifyOnCall(MethodEnv* env)
{
    BaseExecMgr *exec = BaseExecMgr::exec(env);
    AvmAssert(!exec->config.verifyall);  // never verify late in verifyall mode

    #ifdef DEBUGGER
    // Install a fake CallStackNode here, so that if we throw a verify error,
    // we get a stack trace with the method being verified as its top entry.
    CallStackNode callStackNode(env->method);
    #endif

    exec->verifyMethod(env->method, env->toplevel(), env->abcEnv());

    // We got here by calling env->_implGPR, which was pointing to verifyEnterGPR/FPR,
    // but next time we want to call the real code, not verifyEnter again.
    // All other MethodEnv's in their default state will call the target method
    // directly and never go through verifyEnter().  Update the copy in MethodEnv.
    env->_implGPR = env->method->_implGPR;
}
```

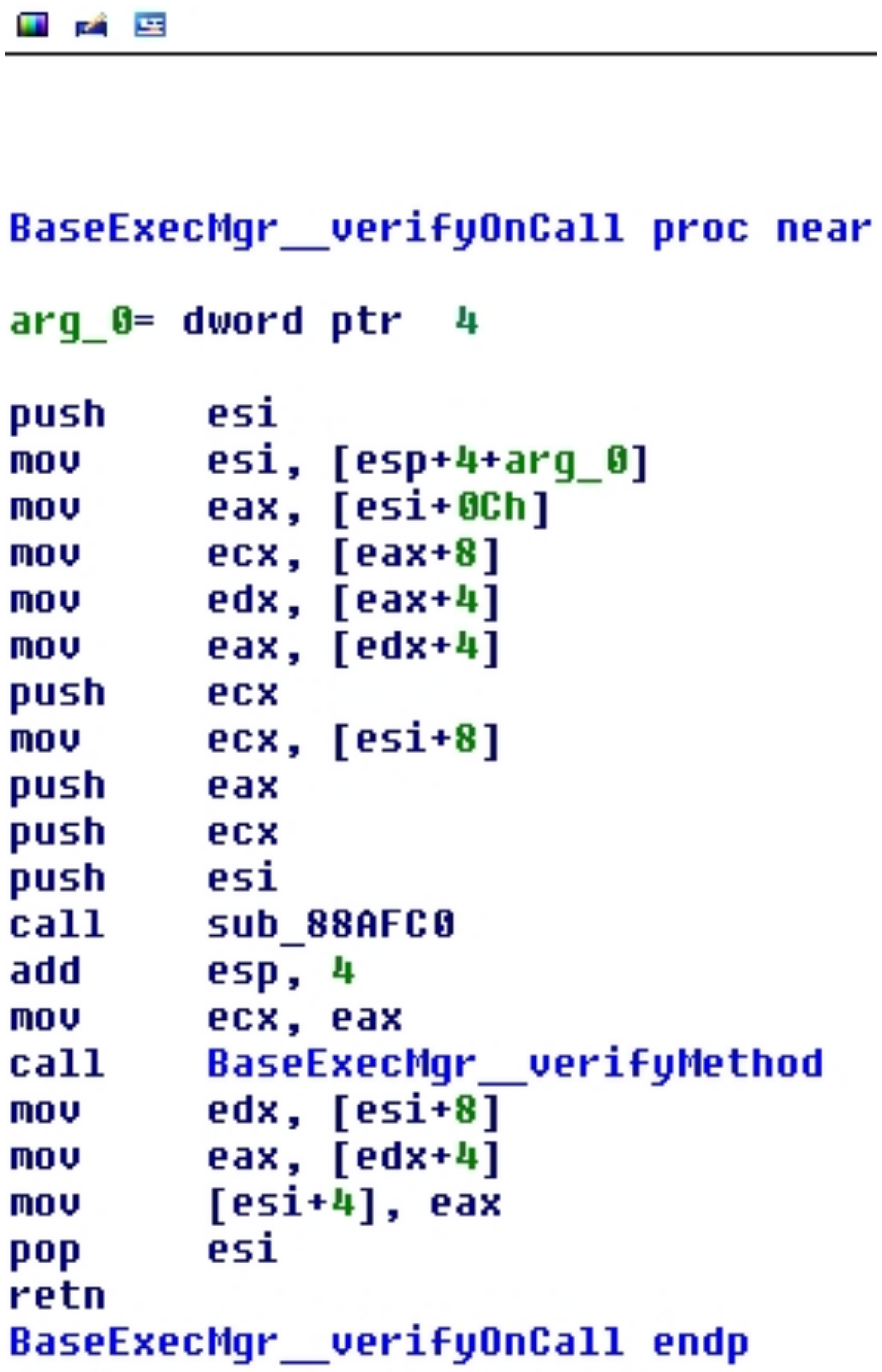
https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/e



```
void BaseExecMgr::verifyInterp(MethodInfo* m, MethodSignaturep ms, Toplevel *toplevel,
{
#ifdef VMCFG_WORDCODE
    WordcodeEmitter coder(m, toplevel);
#else
    CodeWriter coder;
#endif
    verifyCommon(m, ms, toplevel, abc_env, &coder);

#ifdef VMCFG_NANOJIT
# ifdef AVMPLUS_VERBOSE
    if (m->pool()->isVerbose(VB_execpolicy))
        core->console << "execpolicy interp (" << m->unique_method_id() << ") " << m <<
# endif
    setInterp(m, ms, OSR::isSupported(abc_env, m, ms));
#else
# ifdef AVMPLUS_VERBOSE
    if (m->pool()->isVerbose(VB_execpolicy))
        core->console << "execpolicy interp " << m << "\n";
# endif
    setInterp(m, ms, false);
#endif
}
```



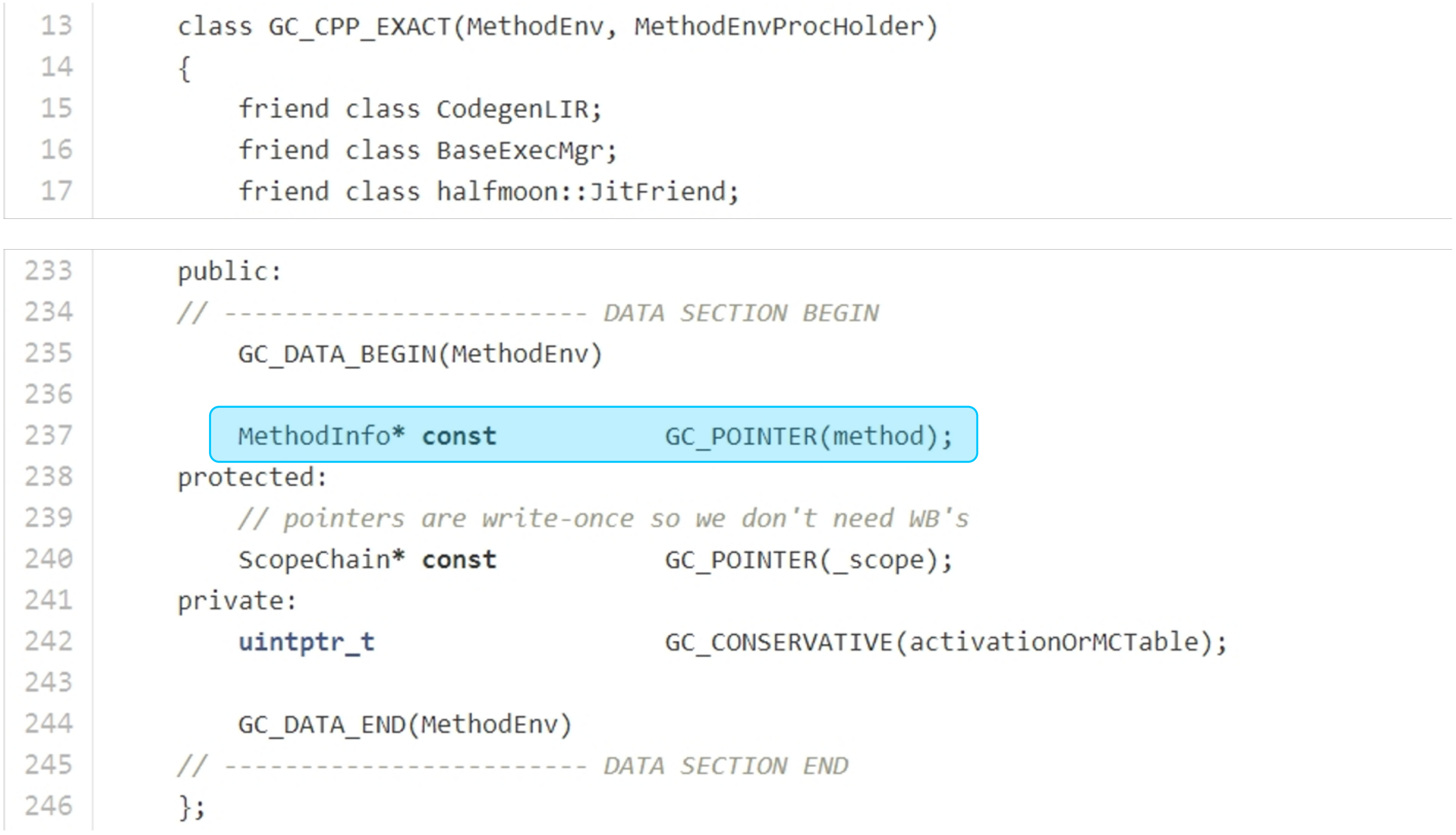
Address	Length	Type	String
.rdata:0095CA44	0000000B	C	execpolicy
.rdata:0098C68C	00000013	C	execpolicy interp
.rdata:0098C6BC	00000016	C	execpolicy jit first
.rdata:0098F3C8	00000018	C	execpolicy jit-invoker
.rdata:0098F3E0	0000001C	C	execpolicy generic-invoker
.rdata:0098F414	00000013	C	execpolicy native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

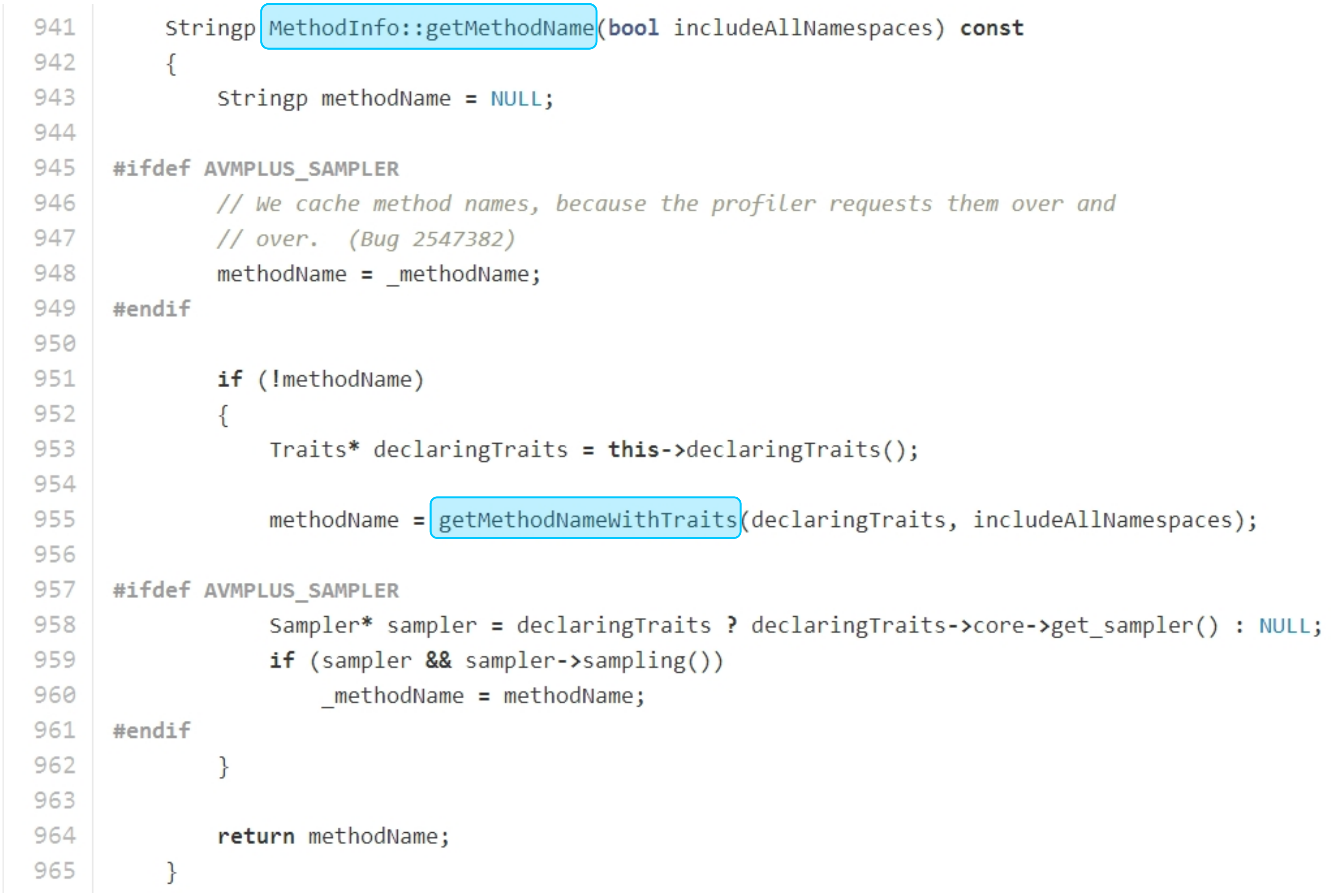
https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/MethodEnv.h



```
    class GC_CPP_EXACT(MethodEnv, MethodEnvProcHolder)
    {
        friend class CodegenLIR;
        friend class BaseExecMgr;
        friend class halfmoon::JitFriend;
```

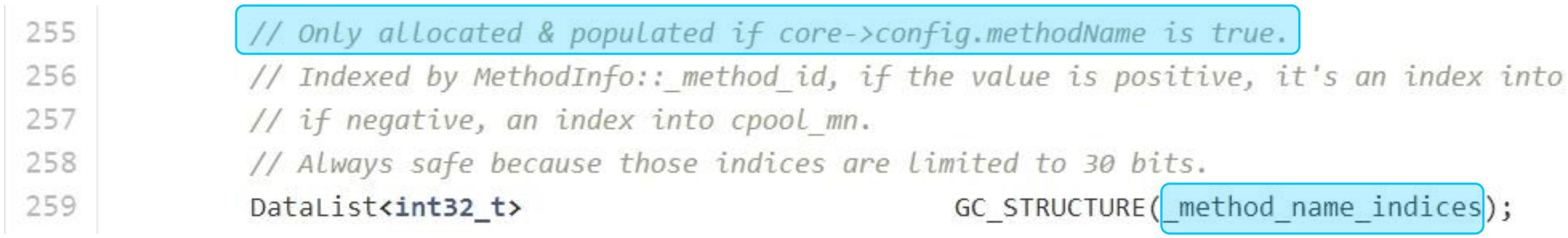
```
    public:
    // ------------------------ DATA SECTION BEGIN
        GC_DATA_BEGIN(MethodEnv)

        MethodInfo* const             GC_POINTER(method);
    protected:
        // pointers are write-once so we don't need WB's
        ScopeChain* const             GC_POINTER(_scope);
    private:
        uintptr_t                     GC_CONSERVATIVE(activationOrMCTable);

        GC_DATA_END(MethodEnv)
    // ------------------------ DATA SECTION END
    };
```

https://github.com/adobe-flash/avmplus/blob/65a05927767f3735db37823eebf7d743531f5d37/core/MethodInfo.pp



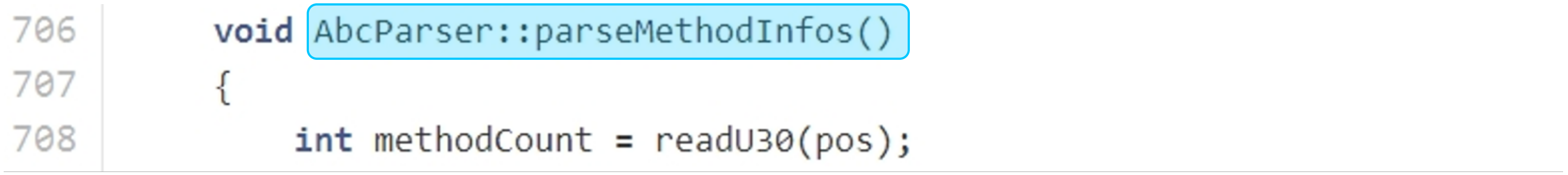
```
    Stringp MethodInfo::getMethodName(bool includeAllNamespaces) const
    {
        Stringp methodName = NULL;

#ifdef AVMPLUS_SAMPLER
        // We cache method names, because the profiler requests them over and
        // over.  (Bug 2547382)
        methodName = _methodName;
#endif

        if (!methodName)
        {
            Traits* declaringTraits = this->declaringTraits();

            methodName = getMethodNameWithTraits(declaringTraits, includeAllNamespaces);

#ifdef AVMPLUS_SAMPLER
            Sampler* sampler = declaringTraits ? declaringTraits->core->get_sampler() : NULL;
            if (sampler && sampler->sampling())
                _methodName = methodName;
#endif
        }

        return methodName;
    }
```

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

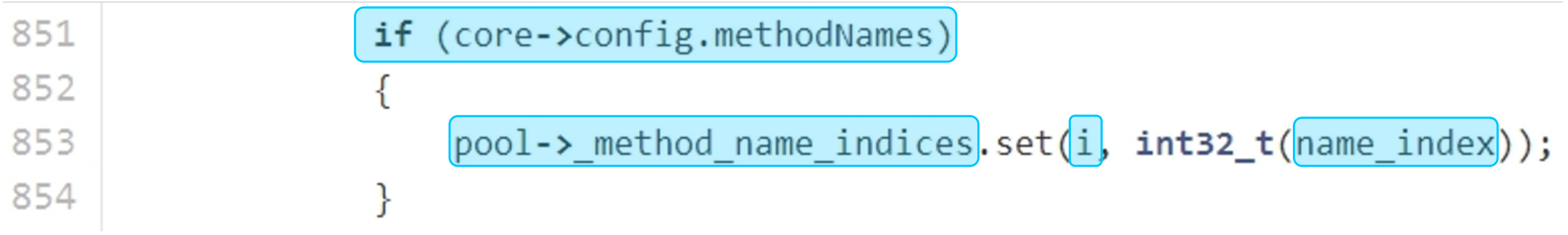


```
        // Only allocated & populated if core->config.methodName is true.
        // Indexed by MethodInfo::_method_id, if the value is positive, it's an index into
        // if negative, an index into cpool_mn.
        // Always safe because those indices are limited to 30 bits.
        DataList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

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



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

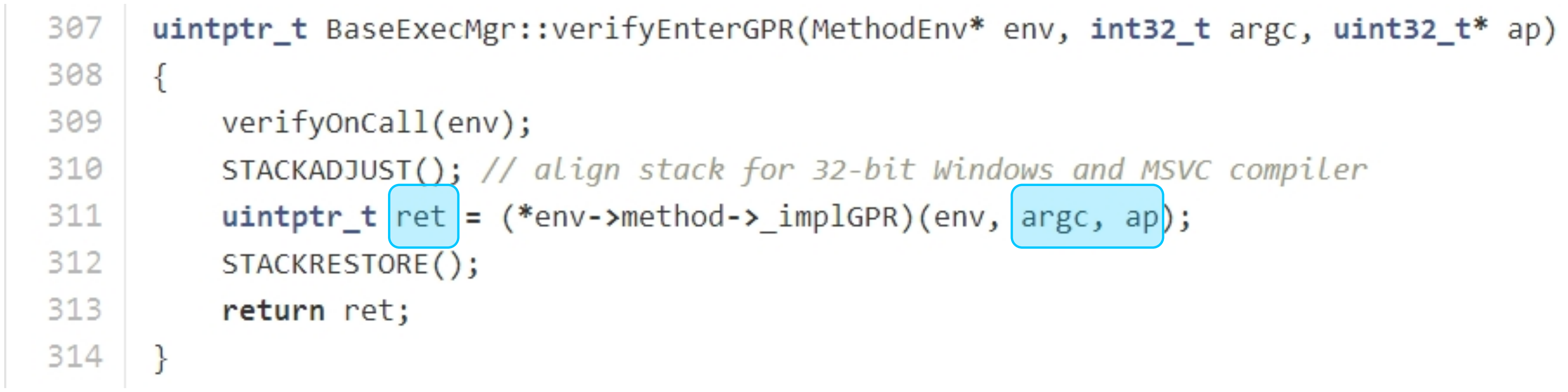


```
            if (core->config.methodNames)
            {
                pool->_method_name_indices.set(i, int32_t(name_index));
            }
```

Nälkä kasvaa syödessä

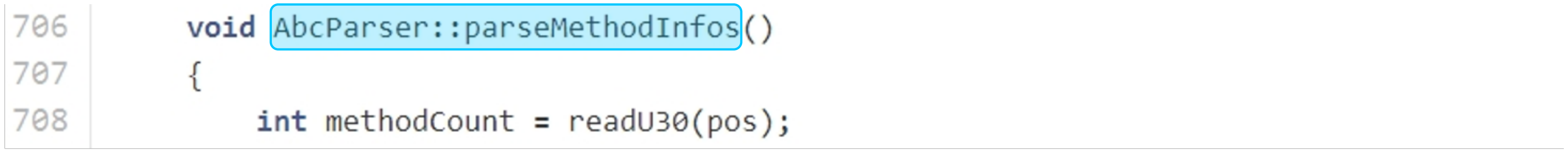
Arguments and return values

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

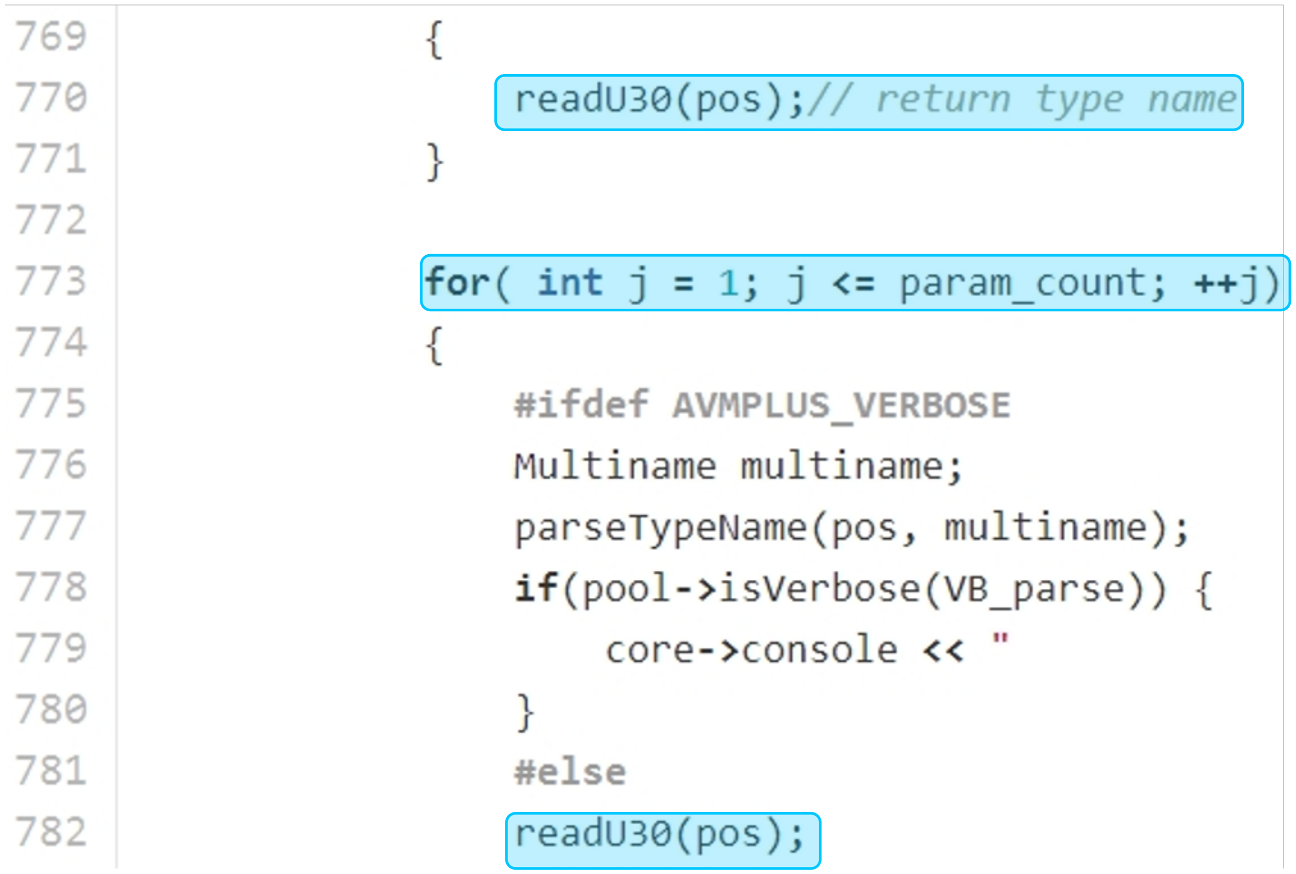


```
uintptr_t BaseExecMgr::verifyEnterGPR(MethodEnv* env, int32_t argc, uint32_t* ap)
{
    verifyOnCall(env);
    STACKADJUST(); // align stack for 32-bit Windows and MSVC compiler
    uintptr_t ret = (*env->method->_implGPR)(env, argc, ap);
    STACKRESTORE();
    return ret;
}
```

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



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



```
                {
                    readU30(pos);// return type name
                }

                for( int j = 1; j <= param_count; ++j)
                {
                    #ifdef AVMPLUS_VERBOSE
                    Multiname multiname;
                    parseTypeName(pos, multiname);
                    if(pool->isVerbose(VB_parse)) {
                        core->console << "
                    }
                    #else
                    readU30(pos);
```

Design

Open source FTW

Intel Pin dynamic instrumentation framework

“Plugins”

Demo

Where can I get it?

F-Secure.

https://github.com/F-Secure/Sulo

Questions?

Thank you!

timo.hirvonen@f-secure.com

@TimoHirvonen

SWITCH ON FREEDOM