Butterfly Effect and Program Mistake

Exploit an "Unexploitable" Chrome Bug

Guang Gong(@oldfresher)

About My Team

- Alpha Team @ Qihoo360
- 60 Google vulnerabilities
- 4 Pwn contest winner
  - Pwn2Own 2015 Mobile
  - Pwn0Rama 2016
  - Pwn2Own 2016
  - PwnFest 2016

Agenda

- Invisible Private Property
- The Tiny Logical Bug
- Exploit CVE-2016-9651
- Old trick about sandbox escape

Invisible Private Property

- Every object in JavaScript is an associative array
- An associative array is simply a set of key value pairs

```
var normalObject = {};
normalObject ["string"] = "string";
normalObject [Symbol("d")] = 0.1;
```

- Properties which can be manipulated by normal JavaScript are public properties
- Get all public keys of an object by normal Javascript

```
var ownNames = Object.getOwnPropertyNames(normalObject);
var ownSymbols = Object.getOwnPropertySymbols(normalObject);
var ownKeys = ownNames.concat(ownSymbols)
```

Execution result： ownPublicKeys==["string", Symbol(d)]

Invisible Private Property

- Some special JavaScript objects have special properties in V8.
- These properties can only be accessed by the engine, they are invisible to normal JavaScript.
- This type of property is named as private property.
- Symbols are divided into two types.
  - Public symbol
    - var publicSymbol = Symbol(“360AlphaTeam”);
  - Private Symbol
    - var privateSymbol = %CreatePrivateSymbol(“invisible”);
- Private property often uses private symbol as key.

Invisible Private Property

- Get all properties of an normal object by d8 shell command

```
ggong@ggong-pc:~/ssd1/v8/out/ia32.debug$ ./d8 --allow-natives-syntax
d8> var normalObject  = {};
d8> normalObject ["string"] = "string";
d8> normalObject [Symbol("d")] = 0.1;
d8> %DebugPrint(normalObject )
DebugPrint: 0x30587431: [JS_OBJECT_TYPE]
 - map = 0x53d091c9 [FastProperties]
 - prototype = 0x25605175
 - elements = 0x45384125 <FixedArray[0]> [FAST_HOLEY_ELEMENTS]
 - properties = {
   #string: 0x45384f35 <String[6]: string> (data field at offset 0)
   0x2561ac51 <Symbol: d>: 0x30588d49 <MutableNumber: 0.1> (data field at offset 1)
 }
```

Invisible Private Property

- Get all properties of an special object by d8 shell command

```
d8> var specialObject = new Error("test");
d8> var ownNames = Object.getOwnPropertyNames(specialObject);
d8> var ownSymbols = Object.getOwnPropertySymbols(specialObject);
d8> var ownKeys = ownNames.concat(ownSymbols)
d8> ownKeys
["stack", "message"]                --------> all public properties got by normal JavaScript
d8> %DebugPrint(specialObject)
DebugPrint: 0x3058e8cd: [JS_ERROR_TYPE]
 - map = 0x53d0945d [FastProperties]
 - prototype = 0x2560b9e1
 - elements = 0x45384125 <FixedArray[0]> [FAST_HOLEY_SMI_ELEMENTS]
 - properties = {                   --------> all properties got by DebugPrint
   #stack: 0x453d012d <AccessorInfo> (accessor constant)
   #message: 0x453bb18d <String[4]: test> (data field at offset 0)
   0x453859f1 <Symbol: stack_trace_symbol>: 0x3058e9c1 <JS Array[6]> (data field at offset 1) --------> private property
 }
```

The Butterfly(CVE-2016-9651)

- **19.1.2.1Object.assign (target, ...sources)** in Standard ECMA-262 6th Edition

  The **assign** function is used to copy the values of all of the enumerable own properties from one or more source objects to a *target* object.

- Private property is a feature of V8.
- Other JavaScript engines maybe have no private property.
- Should private property be enumerable?
- Should private property be copied on assignment? No definition!

The Butterfly(CVE-2016-9651)

- The Tiny Logical Bug

```
MUST_USE_RESULT Maybe<bool> FastAssign(Handle<JSReceiver> to,
                                       Handle<Object> next_source) {
 //detect if fast path can be used
 ……
  Handle<DescriptorArray> descriptors(map->instance_descriptors(), isolate);
  int length = map->NumberOfOwnDescriptors();
  bool stable = true;
  for (int i = 0; i < length; i++) {
    Handle<Name> next_key(descriptors->GetKey(i), isolate); ---->hasn't filtered the keys that are private symbols
    Handle<Object> prop_value;
    //copy all properties from next_source to target
    ……
  }
  return Just(true);
}
```

All Private Symbols

```
#define PRIVATE_SYMBOL_LIST(V)          \
  V(array_iteration_kind_symbol)        \
  V(array_iterator_next_symbol)         \
  V(array_iterator_object_symbol)       \
  V(call_site_frame_array_symbol)       \
  V(call_site_frame_index_symbol)       \
  V(class_end_position_symbol)          \
  V(class_start_position_symbol)        \
  V(detailed_stack_trace_symbol)        \
  V(elements_transition_symbol)         \
  V(error_end_pos_symbol)               \
  V(error_script_symbol)                \
  V(error_start_pos_symbol)             \
  V(frozen_symbol)                      \
  V(hash_code_symbol)                   \
  V(home_object_symbol)                 \
  V(intl_initialized_marker_symbol)     \
  V(intl_pattern_symbol)                \
  V(intl_resolved_symbol)               \
  V(megamorphic_symbol)                 \
  V(native_context_index_symbol)        \
  V(nonextensible_symbol)               \
  V(not_mapped_symbol)                  \
  V(premonomorphic_symbol)              \
  V(promise_async_stack_id_symbol)      \
  V(promise_debug_marker_symbol)        \
  V(promise_forwarding_handler_symbol)  \
  V(promise_handled_by_symbol)          \
  V(promise_async_id_symbol)            \
  V(promise_default_resolve_handler_symbol) \
  V(promise_default_reject_handler_symbol)  \
  V(sealed_symbol)                      \
  V(stack_trace_symbol)                 \
  V(strict_function_transition_symbol)  \
  V(uninitialized_symbol)
```

Logical Bug To OOB Read

```
d8> class c {}
d8> %DebugPrint(c)
DebugPrint: 0x30590e99: [Function]
….
 - properties = {
   #length: 0x453cef99 <AccessorInfo> (accessor constant)
   #name: 0x453cefd1 <AccessorInfo> (accessor constant)
   #prototype: 0x453cf009 <AccessorInfo> (accessor constant)
   0x453854c9 <Symbol: home_object_symbol>: 0x30590ebd <a c with map 0x53d098d5> (data field at
offset 0)
   0x45385335 <Symbol: class_start_position_symbol>: 0 (data field at offset 1)  ------->private symbole
   0x453852fd <Symbol: class_end_position_symbol>: 23 (data field at offset 2)  ------->private symbole
 }
```

Logical Bug To OOB Read

```
> class short {}
<· function class short {}
> class longlonglong {}
<· function class longlonglong {}
> Object.assign(short, longlonglong)
<· function class short {}QÕm
```

OOB Read To OOB Write

- var oobStr = short.toString();
- var c = Oobstr[oobIndex]; ✓

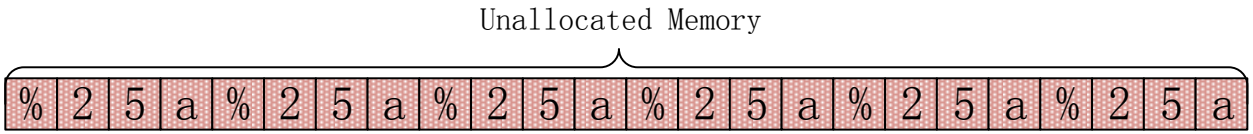
- String in JavaScript is immutable
- oobstr[oobIndex] = newCharactor; ✗

OOB Read To OOB Write

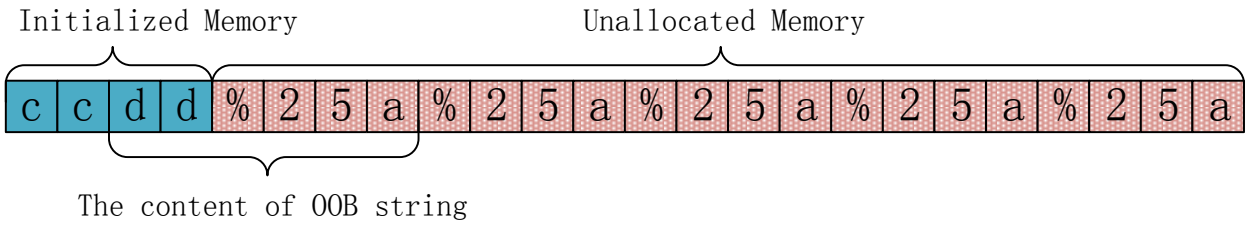
- escape
  - escape("%a");
  - result is "%25a"
- unescape
  - unescape("%25a");
  - result is “%a”, two bytes
  - unescape("aaaa");
  - result is "aaaa“, four bytes
- The length of unescapeed string is depend on the content of the argument string.
- String is immutable but the aforementioned OOB string is mutable.

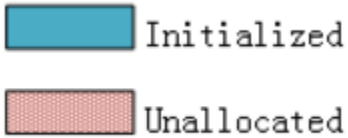
OOB Read To OOB Write

1. Initial memory state, all unallocated memory is sprayed to strings "%25a";



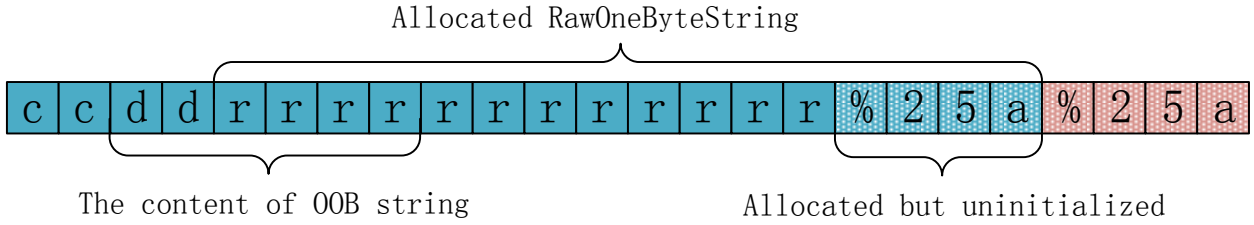
2. memory state after allocating an OOB string ,before executing unescape;



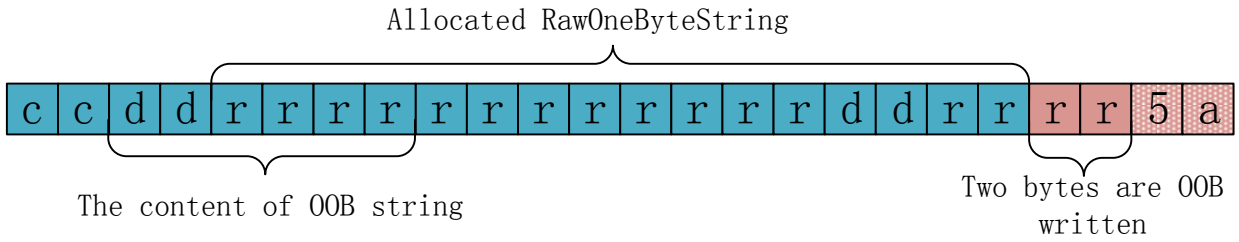


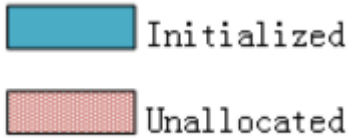
OOB Read To OOB Write

3. Memory state after allocating the destination buffer used by unescape



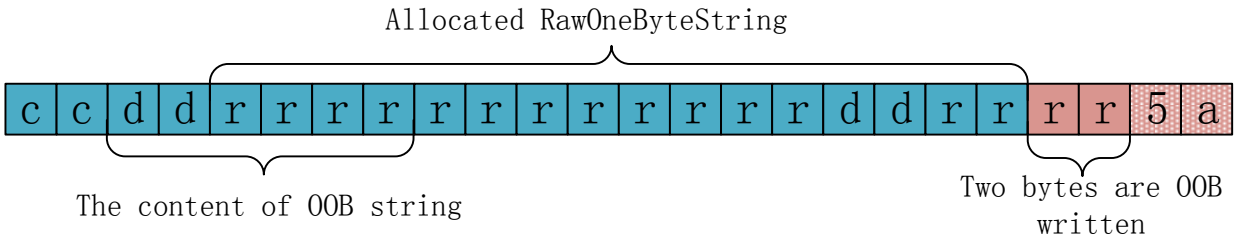
4. memory state after executing unescape;





OOB Write to Arbitrary Memory R/W

- Three conditions to exploit an OOB write Bug
  - The size is controlled easy
  - The content of the source is controlled easy
  - The content to be overwritten is controlled hard



OOB Write to Arbitrary Memory R/W

- Object allocation in v8
  - Regular objects are allocated in new space
  - kMaxRegularHeapObjectSize = 507136
  - Large Objects are allocated in large space
  - Object in new space is allocated sequentially
- The data overwritten in new space is always unallocated.
- OOB write over the boundary of new space
- Overwrite the data in large space following new space
- How to put a large space chunk right below the new space chunk?

Difficulties in Memory Space Shaping

- Chunks of spaces is allocated randomly, not sequentially

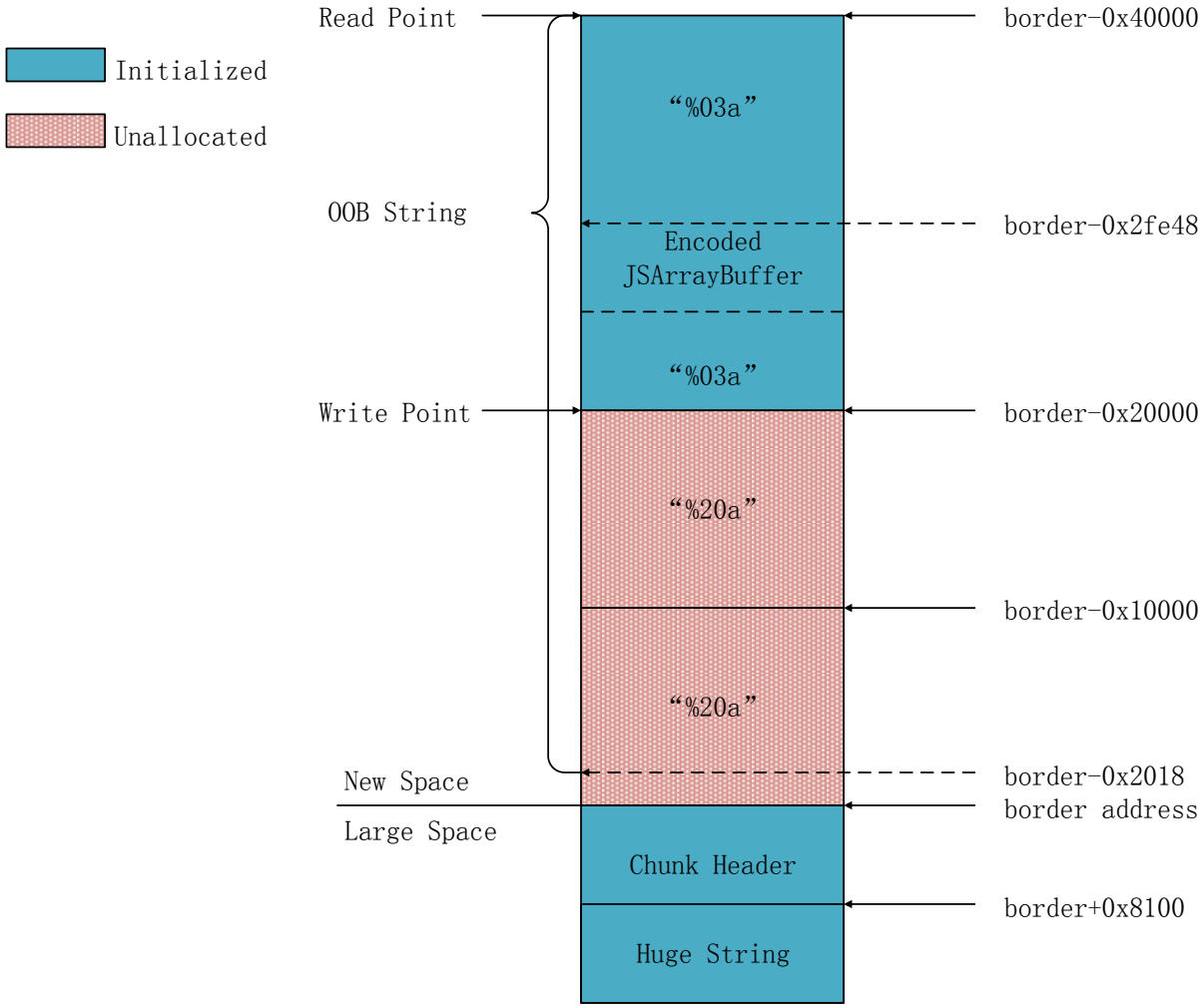
```
VirtualMemory::VirtualMemory(size_t size, size_t alignment)
    : address_(NULL), size_(0) {
  DCHECK((alignment % OS::AllocateAlignment()) == 0);
  size_t request_size = RoundUp(size + alignment,
                                static_cast<intptr_t>(OS::AllocateAlignment()));
  void* reservation = mmap(OS::GetRandomMmapAddr(),  ---->the first arguments isn't NULL
                           request_size,
                           PROT_NONE,
                           MAP_PRIVATE | MAP_ANONYMOUS | MAP_NORESERVE,
                           kMmapFd,
                           kMmapFdOffset);
  //...
}
```

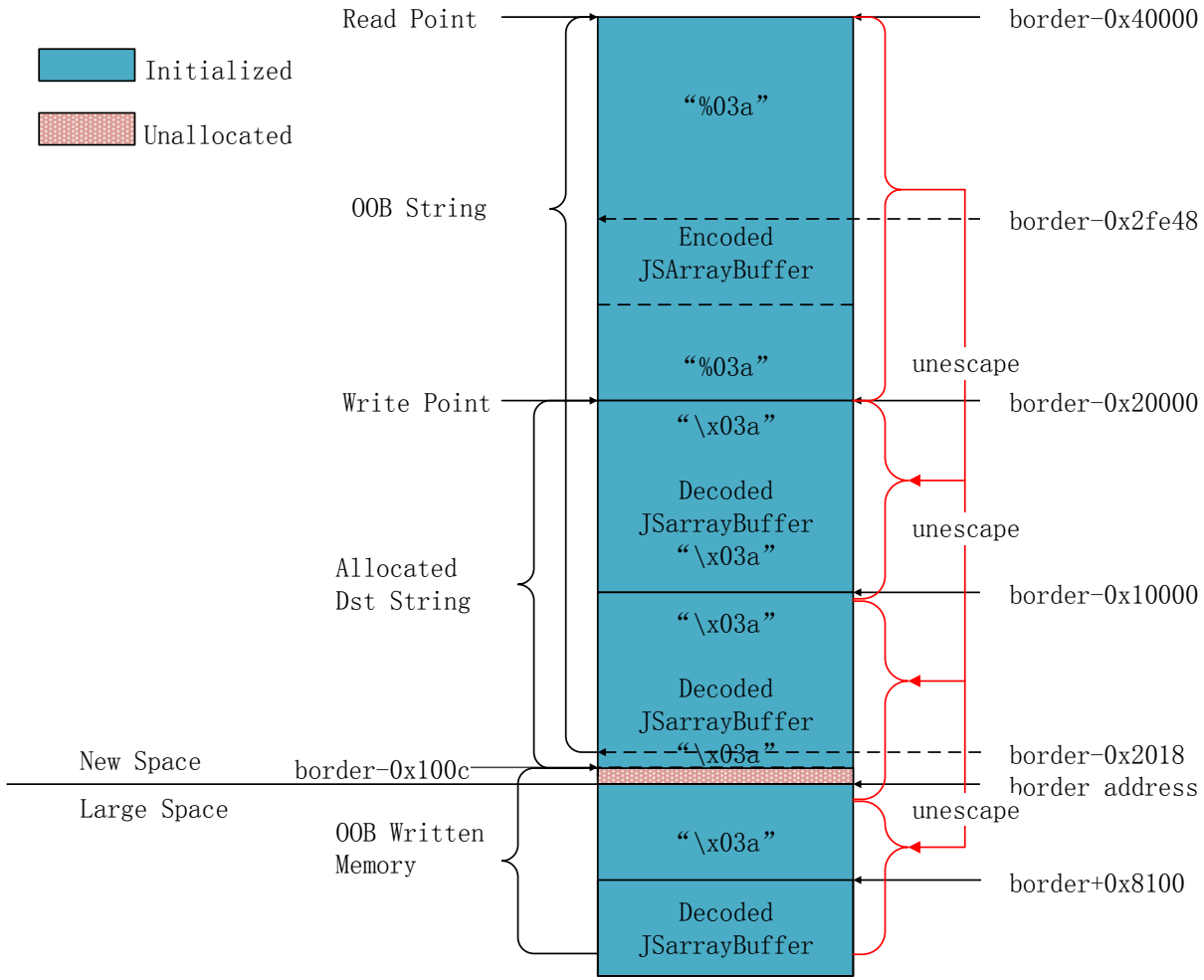
Difficulties in Memory Space Shaping

- The position of New space chunk is immutable once it extends to 16 MB.
- The following layout has to be bypassed.

Solution of Memory Space Shaping

- Web worker is a separate JS instance, each web worker has a separate v8 heap.
- Use the web worker to bypass the address space layout which cannot be shaped.
- brute force address space Feng Shui


Initialized
Unallocated
Read Point
OOB String
Write Point
New Space
Large Space
“%03a”
Encoded
JSArrayBuffer
“%03a”
“%20a”
“%20a”
Chunk Header
Huge String
border-0x40000
border-0x2fe48
border-0x20000
border-0x10000
border-0x2018
border address
border+0x8100


Initialized
Unallocated
Read Point
Write Point
OOB String
Allocated
Dst String
New Space
Large Space
border-0x100c
OOB Written
Memory
"%03a"
Encoded
JSArrayBuffer
"%03a"
"\x03a"
Decoded
JSarrayBuffer
"\x03a"
"\x03a"
Decoded
JSarrayBuffer
"\x03a"
"\x03a"
Decoded
JSarrayBuffer
border-0x40000
border-0x2fe48
unescape
border-0x20000
unescape
border-0x10000
border-0x2018
border address
unescape
border+0x8100

Arbitrary Memory Read/Write to Arbitrary Code Execution

```
(gdb) pt /m JSFunction
type = class v8::internal::JSFunction : public v8::internal::JSObject {
  public:
    static const int kGenerousAllocationCount;
    static const int kPrototypeOrInitialMapOffset;
    static const int kSharedFunctionInfoOffset;
    static const int kContextOffset;
    static const int kLiteralsOffset;
    static const int kNonWeakFieldsEndOffset;
    static const int kCodeEntryOffset;
    static const int kNextFunctionLinkOffset;
    static const int kSize;
}
```

JIT Code in Chrome is writable and executable, overwrite it to execute shellcode.

Sandbox Escape

- An old trick reported more than a year ago in Mobile Pwn2Own 2015
- comprised render process can inject arbitrary JavaScript code to arbitrary web site.
- Apps can be installed from the web site play.google.com
- RCE2UXSS + play.google.com == install any app from google play
- Develops can upload apps include rogue apps to google play
- comprised render process has the permission to install a rogue app

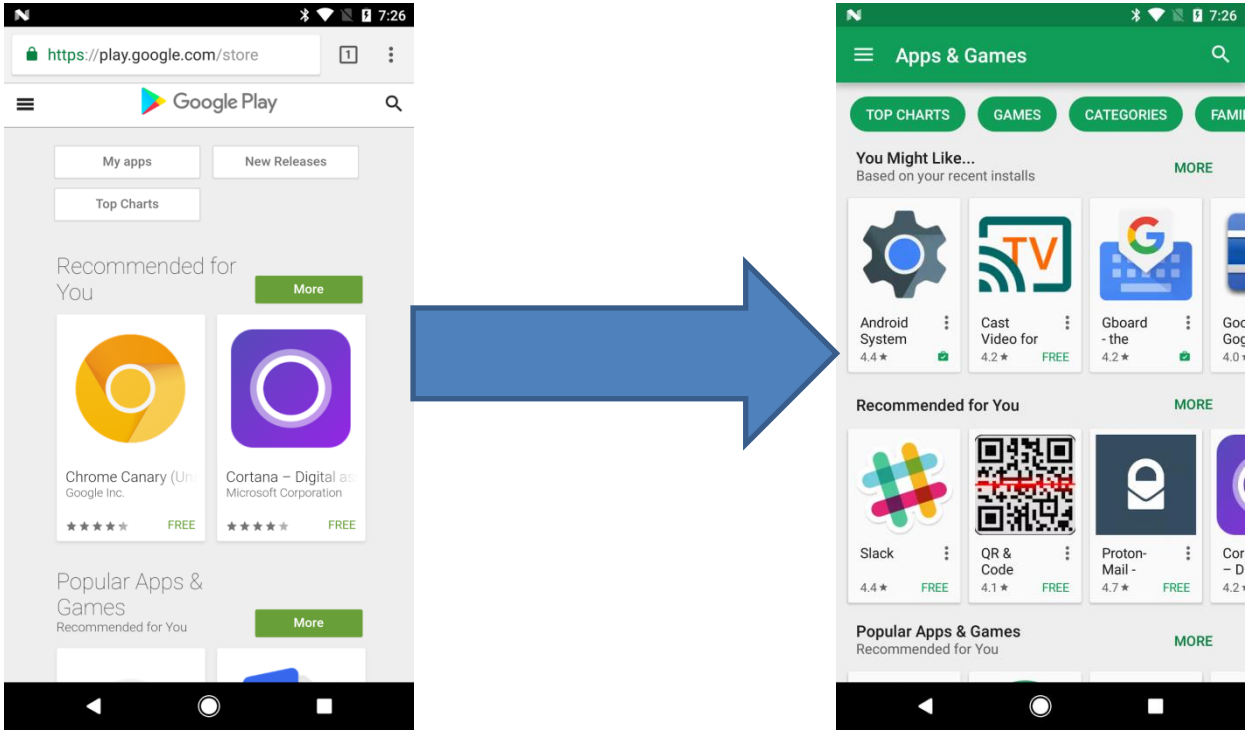
Mitigation

- Descriptions in <Android Security 2016 Year In Review>

PwnFest is a similar hacking contest, which is hosted at the POC security conference. Vulnerability researchers from Qihoo 360 used an exploit chain that took advantage of Google Play's remote app installation feature to install a rogue app. The changes to remote app installation mentioned above also work to disrupt an attacker who might attempt to use this method to install a rogue app.

Mitigation

- top.location = “http://play.google.com”



Demo

Thanks & QA