



Reversing the Root

Identifying the Exploited Vulnerability in 0-days Used
In-The-Wild



Maddie Stone
@maddiestone
BlackHat USA 2020

Who am I? - Maddie Stone

- Security Researcher on Google Project Zero
 - Focusing on 0-days used in the wild
- Previously, Google's Android Sec team
- Reverse all the things
- Speaker at REcon, OffensiveCon, BlackHat, & more!
- BS in Computer Science, Russian, & Applied Math, MS in Computer Science



@maddiestone

When 0-day exploits are detected in-the-wild, it's the failure case for attackers. We need to learn as much as possible each time this happens.

Root Cause Analyses

- Goal:
 - What is the vulnerability?
 - How do you trigger it?
 - How do you exploit it?
- This then informs actions that should be taken next to prevent attackers from being able to do it again.
 - Structural improvements
 - Variant analysis
 - New detection methods

Root Cause Analyses

- Goal:
 - What is the vulnerability?
 - How do you trigger it?
 - How do you exploit it?
- This then informs actions that should be taken next to prevent attackers from being able to do it again.
 - Structural improvements
 - Variant analysis
 - New detection methods

Root Cause Analyses

[All Project Zero Root Cause Analyses for 0-day Exploits](#)

- CVE-2019-1367 (and CVE-2019-1429)
- CVE-2020-0674
- CVE-2019-1458
- CVE-2019-17026
- CVE-2020-6820
- CVE-2019-2215

Takeaways

1. There are a variety of different ways to reverse engineer the vulnerability being exploited by a 0-day based on information available and the target being exploited.
2. Not every endeavor to do root-cause analysis will be successful or completed quickly enough. Evaluate and use this information to become more efficient and precise in future analyses.
3. Understand not just what the technique is, but how it's done in practice and how it can be slightly different from one 0-day to another.



Maddie Stone

@maddiestone

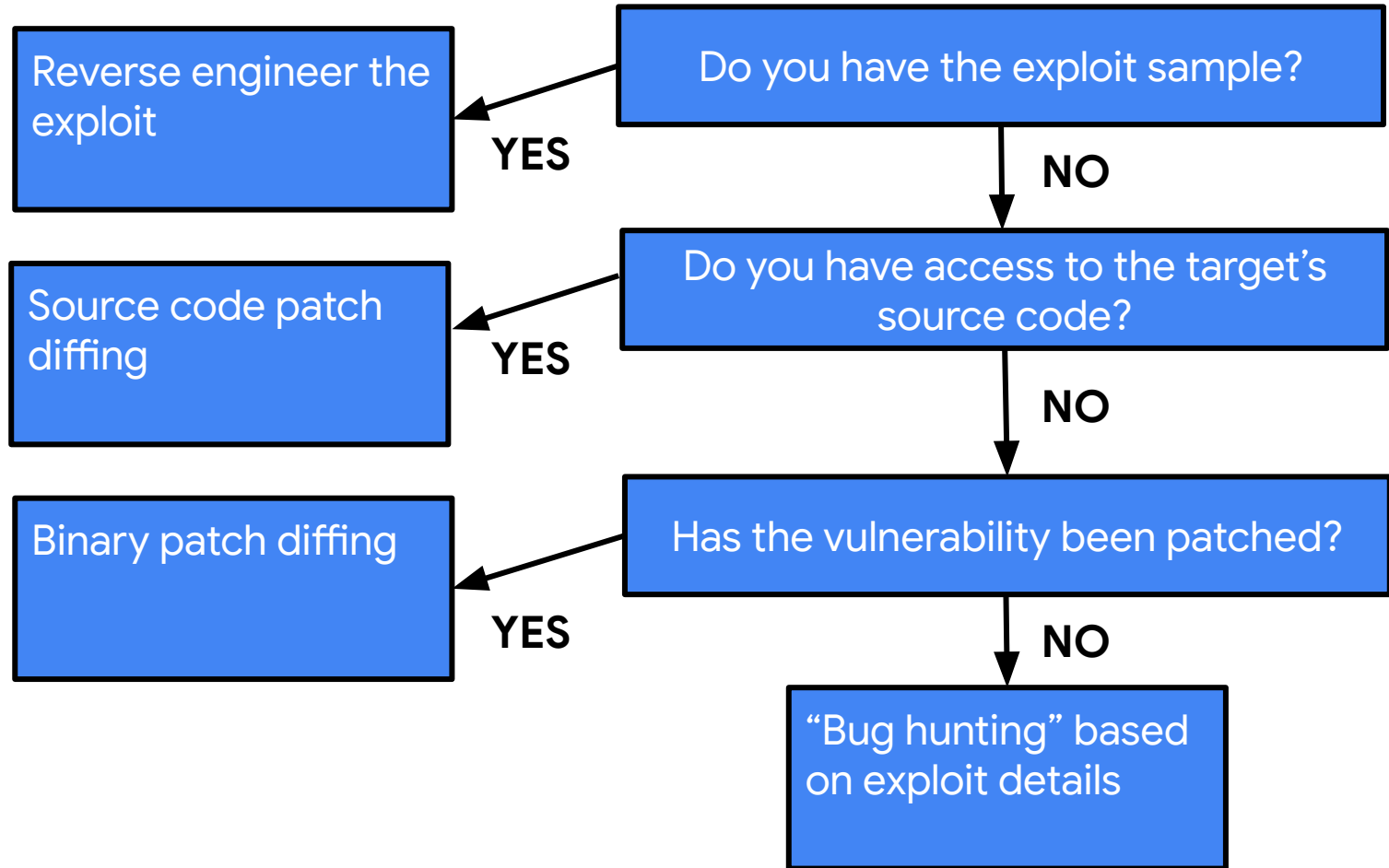


Yo yo yo I'm Maddie Stone
I got these zero days
So let's talk about the ways
That we get to the root
So we can scream woot
And when we root cause that vuln
Life won't be so dull

Practicing for my [@BlackHatEvents](#) talk is going well.

The Techniques

- Reversing the Exploit Sample
- Binary Patch Diffing
- Source Code Patch Diffing
- “Bug Hunting” Based on Exploit Details



What is your role?

- Original discoverer
- Responsible vendor
- 3rd party (users or researchers)

Influences what data you have access to and how much you're willing to invest in getting to the root cause vulnerability.



Case Studies

Reverse Engineering the Exploit

Windows JScript (CVE-2019-1367, CVE-2019-1429, CVE-2020-0674)

- Detected by Clement Lecigne of Google Threat Analysis Group. Analyzed by Clement Lecigne & Ivan Fratric Google Project Zero.
- [Microsoft Security Advisory: CVE-2019-1367](#)
- [Microsoft Security Advisory: CVE-2019-1429](#)
- [Microsoft Security Advisory: CVE-2020-0674](#)
 - *“A remote code execution vulnerability exists in the way that the scripting engine handles objects in memory in Internet Explorer.”*

Process

- Ivan has studied and written tooling extensively for JScript.
- He decided to run the exploit through a custom, simple test case minimizer.

Minimized POC

```
function F(a, b) {
  v.push(arguments);
  y += 2;
  if (y >= (B - A)) {
    CollectGarbage();
    for (var c = 0; c < 100 * 100; c++) q[c] = new Object();
    for (var c = 0; c < z; c++) try {
      throw u[c];
    } catch (d) {
      r[c] = d;
    }
    for (var c = A; c < B; c++) v[((c - A) / 2) | 0][((c - A) % 2)] = r[c];
    for (var c = 0; c < 100 * 100; c++) q[c] = null;
    CollectGarbage();
    for (var c = 0; c < z; c++) r[c] = null;
    CollectGarbage();
    for (var c = 0; c < 0x1000; c++) x[c][E] = 1;
    for (var c = A; c < B; c++) s[c] = v[((c - A) / 2) | 0][((c - A) % 2)];
  } else w[y / 2].sort(F);
  return 0;
}
```

Cleaned Up POC

```
var spray = new Array();

function F() {
  // 2. Create a bunch of objects
  for (var i = 0; i < 20000; i++) spray[i] = new Object();

  // 3. Store a reference to one of them in the arguments array
  //    The arguments array isn't tracked by garbage collector
  arguments[0] = spray[5000];

  // 4. Delete the objects and call the garbage collector
  //    All JScrip variables get reclaimed...
  for (var i = 0; i < 20000; i++) spray[i] = 1;
  CollectGarbage();

  // 5. But we still have reference to one of them in the
  //    arguments array
  alert(arguments[0]);
}

// 1. Call sort with a custom callback
[1,2].sort(F);
```

CVE-2019-1376 Closing Thoughts

- Test case minimizer idea was really cool!
- Expertise in that platforms bug classes and exploit techniques is very important

iOS unc0ver

- We don't consider a 0-day used in the wild (instead it's full disclosure), but still requires going through the similar actions to root cause the vulnerability.
- Analyzed by Brandon Azad of Google Project Zero. Detailed [blogpost](#) of the full analysis process.

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.

Brandon has written iOS kernel exploits before and knew that many memory corruption based exploits have a “critical section” during which kernel state has been corrupted & thus the system would be unstable if the rest of the exploit didn’t continue.

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.

```
panic(cpu 1 caller 0xfffffff020e75424): "Zone cache element was used after free! Element 0xffffffe0033ac810 was corrupted at beginning; Expected 0x87be6c0681be12b8 but found 0xffffffe003059d90; canary 0x784193e68284daa8; zone 0xfffffff021415fa8 (kalloc.16)"
```

```
Debugger message: panic
```

```
Memory ID: 0x6
```

```
OS version: 17B111
```

```
Kernel version: Darwin Kernel Version 19.0.0: Wed Oct 9 22:41:51 PDT 2019; root:xnu-6153.42.1~1/RELEASE_ARM64_T8010
```

```
KernelCache UUID: 5AD647C26EF3506257696CF29419F868
```

```
Kernel UUID: F6AED585-86A0-3BEE-83B9-C5B36769EB13
```

```
iBoot version: iBoot-5540.40.51
```

```
secure boot?: YES
```

```
Paniclog version: 13
```

```
Kernel slide: 0x0000000019cf0000
```

```
Kernel text base: 0xfffffff020cf4000
```

```
mach_absolute_time: 0x3943f534b
```

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.

```
panic(cpu 1 caller 0xfffffff020e75424): "Zone cache element was used after free! Element 0xffffffe0033ac810 was corrupted at beginning; Expected 0x87be6c0681be12b8 but found 0xffffffe003059d90; canary 0x784193e68284daa8; zone 0xfffffff021415fa8
```

```
(kalloc.16)"
```

```
Debugger message: panic
```

```
Memory ID: 0x6
```

```
OS version: 17B111
```

```
Kernel version: Darwin Kernel Version 19.0.0: Wed Oct 9 22:41:51 PDT 2019; root:xnu-6153.42.1~1/RELEASE_ARM64_T8010
```

```
KernelCache UUID: 5AD647C26EF3506257696CF29419F868
```

```
Kernel UUID: F6AED585-86A0-3BEE-83B9-C5B36769EB13
```

```
iBoot version: iBoot-5540.40.51
```

```
secure boot?: YES
```

```
Paniclog version: 13
```

```
Kernel slide: 0x0000000019cf0000
```

```
Kernel text base: 0xfffffff020cf4000
```

```
mach_absolute_time: 0x3943f534b
```

Panic message stating there is a use-after-free in the kalloc.16 allocation zone (general purpose allocations of size up to 16 bytes)

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.
- Determine if the vulnerability relies on re-allocating a `kalloc.16` allocation

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.
- Determine if the vulnerability relies on re-allocating a `kalloc.16` allocation
 - Wrote an app that continuously allocates and free to `kalloc.16`

As soon as the “Jailbreak” button was pressed, caused a kernel panic!

Did not when the app’s zone was changed to `kalloc.32` instead of `kalloc.16`

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.
- Determine if the vulnerability relies on re-allocating a `ka1loc.16` allocation
- Analyze the stack traces from the use-after-free crashes

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.
- Determine if the vulnerability relies on re-allocating a `ka1loc.16` allocation
- Analyze the stack traces from the use-after-free crashes
 - Symbolicate after loading `kernelcache` into IDA

Panicked task 0xffffffe0008a4800: 9619 pages, **230 threads**: pid 222: unc0ver

Panicked thread: 0xffffffe004303a18, backtrace: 0xffffffe00021b2f0, tid: 4884

```
lr: 0xffffffff007135e70
lr: 0xffffffff007135cd0
lr: 0xffffffff0072345c0
lr: 0xffffffff0070f9610
lr: 0xffffffff007135648
lr: 0xffffffff007135990
lr: 0xffffffff0076e1ad4 # _panic
lr: 0xffffffff007185424 # _zcache_alloc_from_cpu_cache
lr: 0xffffffff007182550 # _zalloc_internal
lr: 0xffffffff007140718 # _kalloc_canblock
lr: 0xffffffff0074d5bfc # aio copy in list
lr: 0xffffffff0074d5d90 # _lio_listio
lr: 0xffffffff0075f10d0 # _unix_syscall
lr: 0xffffffff00723468c # _sleh_synchronous
lr: 0xffffffff0070f9610 # _fleh_synchronous
lr: 0x00000001bf085ae4
```

Brandon had recently completed a study of iOS exploits and immediately remembered **_lio_listio** was the vulnerable syscall in Lightspeed-based exploits.

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.
- Determine if the vulnerability relies on re-allocating a `ka1loc.16` allocation
- Analyze the stack traces from the use-after-free crashes
- Investigate how the exploit runs compared to original Lightspeed exploit

Process

- Opened up in IDA, heavily obfuscated → not statically reversing.
- Ran it on a device and stopped it in the middle.
- Determine if the vulnerability relies on re-allocating a `ka1loc.16` allocation
- Analyze the stack traces from the use-after-free crashes
- Investigate how the exploit runs compared to original Lightspeed exploit
 - Patched `lio_listio` in the `kernelcache` using [checkra1n/pongoOS](#)

unc0ver Closing Thoughts

- With the exploit sample, Brandon had identified the vulnerability within 4 hours. And a POC within 10.
 - But his expertise in writing kernel exploits & recently studying previous exploits played a critical role.

unc0ver Closing Thoughts

- With the exploit sample, Brandon had identified the vulnerability within 4 hours. And a POC within 10.
 - But his expertise in writing kernel exploits & recently studying previous exploits played a critical role.
- Switched to dynamic reversing when saw that the exploit was obfuscated.
 - Stopped execution in middle of exploit, causing kernel panic

unc0ver Closing Thoughts

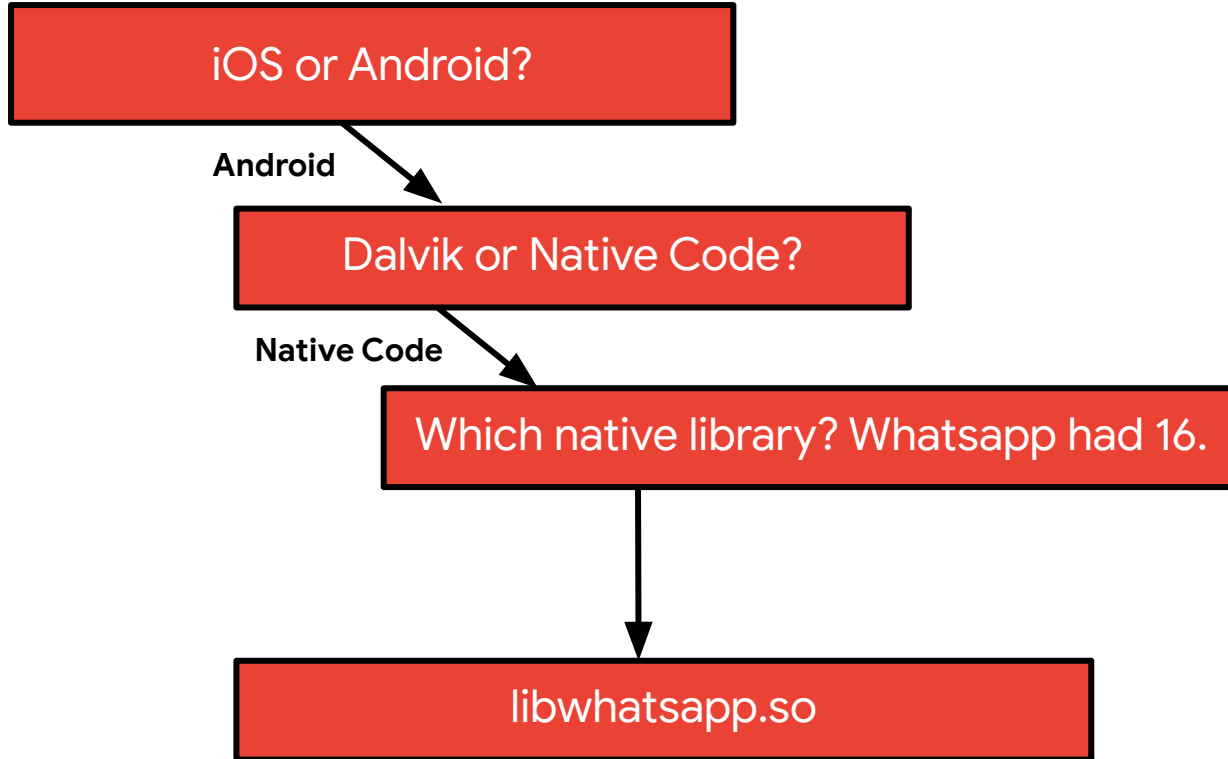
- With the exploit sample, Brandon had identified the vulnerability within 4 hours. And a POC within 10.
 - But his expertise in writing kernel exploits & recently studying previous exploits played a critical role.
- Switched to dynamic reversing when saw that the exploit was obfuscated.
 - Stopped execution in middle of exploit, causing kernel panic
- Another interesting attempt:
 - Many iOS exploits use Mach ports so Brandon wrote an app that churned the `ipc.ports` zone. When the exploit still worked, it suggested that the exploit didn't rely on heap grooming of Mach port allocations.

Binary Patch Diffing

Whatsapp (CVE-2019-3568)

- [Facebook's Advisory for CVE-2019-3568](#)
 - *“A buffer overflow vulnerability in WhatsApp VOIP stack allowed remote code execution via specially crafted series of RTCP packets sent to a target phone number.”*
- Full explanation of binary patch diffing process and tooling in “What’s Up with WhatsApp” presentation. [[video](#), [slides](#)]

My Process



My Process

- Statically reverse engineered to understand the patch
- Used [Frida](#) to understand how patched function is triggered

Windows win32k (CVE-2019-1458)

- Detected by Anton Ivanov and Alexey Kulaev of Kaspersky Lab.
- [Microsoft Security Advisory](#)
 - *“An elevation of privilege vulnerability exists in Windows when the Win32k component fails to properly handle objects in memory.”*
- First time ever looking at Windows and I chose to try to binary patch diff this bug. I reversed the wrong vulnerability.
 - See the [blog post](#) for the full explanation & process.
- But Piotr Florczyk got it right: [His process and root cause analysis](#).

Binary Patch Diffing Conclusions

- Generally the most time consuming of all of the methods.
- Most documented.
- *Usually* only needs to be done by 3rd parties, not vendors and detectors.
- Highest likelihood of getting it wrong due to many other unrelated changes often in the patches.

Source Code Patch Diffing

Firefox (CVE-2019-17026)

- Detected by Qihoo 360 ATA. Analyzed by Samuel Groß of Google Project Zero. [Bug 1607443](#)
- [Mozilla Security Advisory 2020-03](#)
 - *“Incorrect alias information in IonMonkey JIT compiler for setting array elements could lead to a type confusion.”*
- Patch CL:
<https://hg.mozilla.org/mozilla-central/rev/d6e40de88f3defdc12ef27e64ca73e120b1f10e2>

Firefox (CVE-2019-17026) - js/src/jit/AliasAnalysis.cpp

```
--- a/js/src/jit/AliasAnalysis.cpp
+++ b/js/src/jit/AliasAnalysis.cpp
@@ -115,18 +115,16 @@ static inline const MDefinition*
GetObjc
    case MDefinition::Opcode::LoadUnboxedString:
    case MDefinition::Opcode::StoreElement:
    case MDefinition::Opcode::StoreUnboxedObjectOrNull:
    case MDefinition::Opcode::StoreUnboxedString:
    case MDefinition::Opcode::StoreUnboxedScalar:
    case MDefinition::Opcode::SetInitializedLength:
    case MDefinition::Opcode::ArrayLength:
    case MDefinition::Opcode::SetArrayLength:
-   case MDefinition::Opcode::StoreElementHole:
-   case MDefinition::Opcode::FallibleStoreElement:
    case MDefinition::Opcode::TypedObjectDescr:
    case MDefinition::Opcode::Slots:
    case MDefinition::Opcode::Elements:
    case MDefinition::Opcode::MaybeCopyElementsForWrite:
    case MDefinition::Opcode::MaybeToDoubleElement:
    case MDefinition::Opcode::TypedArrayLength:
    case MDefinition::Opcode::TypedArrayByteOffset:
    case MDefinition::Opcode::ArrayPopShift:

@@ -176,16 +174,18 @@ static inline const MDefinition* GetObjc
    case MDefinition::Opcode::WasmAtomicExchangeHeap:
    case MDefinition::Opcode::WasmLoadGlobalVar:
    case MDefinition::Opcode::WasmLoadGlobalCell:
    case MDefinition::Opcode::WasmStoreGlobalVar:
    case MDefinition::Opcode::WasmStoreGlobalCell:
    case MDefinition::Opcode::WasmStoreRef:
    case MDefinition::Opcode::ArrayJoin:
    case MDefinition::Opcode::ArraySlice:
+   case MDefinition::Opcode::StoreElementHole:
+   case MDefinition::Opcode::FallibleStoreElement:
        return nullptr;
    default:
#ifdef DEBUG
        // Crash when the default aliasSet is overridden, but
        when not added in the
        // list above.
        if (!ins->getAliasSet().isStore() ||
            ins->getAliasSet().flags() != AliasSet::Flag::Any) {
            MOZ_CRASH(
```

Firefox (CVE-2019-17026) - js/src/jit/MIR.h

```
--- a/js/src/jit/MIR.h
+++ b/js/src/jit/MIR.h
@@ -7852,22 +7852,16 @@ class MStoreElementHole
    MOZ_ASSERT(index->type() == MIRType::Int32);
}
public:
    INSTRUCTION_HEADER(StoreElementHole)
    TRIVIAL_NEW_WRAPPERS
    NAMED_OPERANDS((0, object), (1, elements), (2, index), (3, value))
-   AliasSet getAliasSet() const override {
-       // StoreElementHole can update the initialized length, the array
length
-       // or reallocate obj->elements.
-       return AliasSet::Store(AliasSet::ObjectFields |
AliasSet::Element);
-   }
-
    ALLOW_CLONE(MStoreElementHole)
};
// Try to store a value to a dense array slots vector. May fail due to
the
// object being non-extensible/sealed/frozen. Cannot be used on an
object that
// has extra indexed properties.
class MFallibleStoreElement
    : public MQuaternaryInstruction,
```

```
@@ -7884,19 +7878,16 @@ class MFallibleStoreElement
    MOZ_ASSERT(index->type() == MIRType::Int32);
}
public:
    INSTRUCTION_HEADER(FallibleStoreElement)
    TRIVIAL_NEW_WRAPPERS
    NAMED_OPERANDS((0, object), (1, elements), (2, index), (3, value))
-   AliasSet getAliasSet() const override {
-       return AliasSet::Store(AliasSet::ObjectFields |
AliasSet::Element);
-   }
    bool needsHoleCheck() const { return needsHoleCheck_; }
    ALLOW_CLONE(MFallibleStoreElement)
};

// Store an unboxed object or null pointer to an elements vector.
class MStoreUnboxedObjectOrNull : public MQuaternaryInstruction,
    public
StoreUnboxedObjectOrNullPolicy::Data {
```

Firefox (CVE-2019-17026) - js/src/jit/MIR.h

```
--- a/js/src/jit/MIR.h
+++ b/js/src/jit/MIR.h
@@ -7852,22 +7852,16 @@ class MStoreElementHole
    MOZ_ASSERT(index->type() == MIRType::Int32),
    }
    public:
    INSTRUCTION_HEADER(StoreElementHole)
    TRIVIAL_NEW_WRAPPERS
    NAMED_OPERANDS((0, object), (1, elements), (2, index), (3, value))
-   AliasSet getAliasSet() const override {
-       // StoreElementHole can update the initialized length, the array
length
-       // or reallocate obj->elements.
-       return AliasSet::Store(AliasSet::ObjectFields |
AliasSet::Element);
-   }
-
    ALLOW_CLONE(MStoreElementHole)
};
// Try to store a value to a dense array slots vector. May fail due to
the
// object being non-extensible/sealed/frozen. Cannot be used on an
object that
```

```
@@ -7884,19 +7878,16 @@ class MFallibleStoreElement
    MOZ_ASSERT(index->type() == MIRType::Int32),
    }
    public:
    INSTRUCTION_HEADER(FallibleStoreElement)
    TRIVIAL_NEW_WRAPPERS
    NAMED_OPERANDS((0, object), (1, elements), (2, index), (3, value))
-   AliasSet getAliasSet() const override {
-       return AliasSet::Store(AliasSet::ObjectFields |
AliasSet::Element);
-   }
    bool needsHoleCheck() const { return needsHoleCheck_; }
    ALLOW_CLONE(MFallibleStoreElement)
};
// Store an unboxed object or null pointer to an elements vector.
class MStoreUnboxedObjectOrNull : public MQuaternaryInstruction,
    public
StoreUnboxedObjectOrNullPolicy::Data {
```

The vulnerability is within the alias information, which describes what side-effects a JIT MIR operation can have, for the StoreElementHole and FallibleStoreElement operations.

Firefox (CVE-2019-17026) - js/src/jit/MIR.h

```
--- a/js/src/jit/MIR.h
+++ b/js/src/jit/MIR.h
@@ -7852,22 +7852,16 @@ class MStoreElementHole
    MOZ_ASSERT(index->type() == MIRType::Int32);
}
public:
    INSTRUCTION_HEADER(StoreElementHole)
    TRIVIAL_NEW_WRAPPERS
    NAMED_OPERANDS((0, object), (1, elements), (2, index), (3, value))
-   AliasSet getAliasSet() const override {
-       // StoreElementHole can update the initialized length, the array
length
-       // or reallocate obj->elements.
-       return AliasSet::Store(AliasSet::ObjectFields |
AliasSet::Element);
-   }
-
    ALLOW_CLONE(MStoreElementHole)
};
// Try to store a value to a dense array slots vector. May fail due to
the
```

```
@@ -7884,19 +7878,16 @@ class MFallibleStoreElement
    MOZ_ASSERT(index->type() == MIRType::Int32);
}
public:
    INSTRUCTION_HEADER(FallibleStoreElement)
    TRIVIAL_NEW_WRAPPERS
    NAMED_OPERANDS((0, object), (1, elements), (2, index), (3, value))
-   AliasSet getAliasSet() const override {
-       return AliasSet::Store(AliasSet::ObjectFields |
AliasSet::Element);
-   }
    bool needsHoleCheck() const { return needsHoleCheck_; }
    ALLOW_CLONE(MFallibleStoreElement)
};

// Store an unboxed object or null pointer to an elements vector.
class MStoreUnboxedObjectOrNull : public MQuaternaryInstruction,
public
StoreUnboxedObjectOrNullPolicy::Data {
```

The operations have been generalized so the compiler now assumes that executing one of these operations can change anything.

“Incorrect side-effect modelling” issue in the JIT engine.

Process

- From the patch, it seems likely that the `StoreElementHole` & `FallibleStoreElement` can cause unexpected execution of arbitrary JavaScript.

Process

- From the patch, it seems likely the `StoreElementHole` & `FallibleStoreElement` can cause unexpected execution of arbitrary JavaScript.
 - Maybe indexed accessors in the prototype chain?

Process

- From the patch, it seems likely the `StoreElementHole` & `FallibleStoreElement` can cause unexpected execution of arbitrary JavaScript.
 - Maybe indexed accessors in the prototype chain?
 - No...the JIT compiler guards against that.

Process

- From the patch, it seems likely the `StoreElementHole` & `FallibleStoreElement` can cause unexpected execution of arbitrary JavaScript.
- `StoreElementHole` and `FallibleStoreElement` operations will happily accept negative numbers as the index. → They write a property and not an element.

Process

- From the patch, it seems likely the `StoreElementHole` & `FallibleStoreElement` can cause unexpected execution of arbitrary JavaScript.
- `StoreElementHole` and `FallibleStoreElement` operations will happily accept negative numbers as the index. → They write a property and not an element.
- A property setter on the property `'-1'` will pass the JITs requirements about the input objects but will also cause unexpected execution of JavaScript during the `StoreElementHole` operation.

CVE-2019-17026 Closing Thoughts

- The bug class was pretty clear immediately from looking at the patch.
- But the exact details of the specific vulnerability, and how to write a trigger, are more difficult to determine.
 - Took Samuel, an expert in the bug class, 2 days to write a POC
 - Would someone who had minimal or no experience with the bug class be able to figure it out?

Firefox (CVE-2020-6820)

- Detected by Francisco Alonso @revskills working with Javier Marcos of @JMPSec [Bug 1626728](#)
- [Mozilla Security Advisory 2020-11](#)
 - *“Under certain conditions, when handling a ReadableStream, a race condition can cause a use-after-free.”*
- Patch CL:
<https://hg.mozilla.org/mozilla-central/rev/6639deb894172375b05d6791f5f8c7d53ca79723>

CVE-2020-6820 - dom/cache/StreamList.cpp

```
--- a/dom/cache/StreamList.cpp
+++ b/dom/cache/StreamList.cpp
@@ -128,17 +128,25 @@ void StreamList::Close(const nsID& aId)
     if (mStreamControl) {
         mStreamControl->Close(aId);
     }
 }
void StreamList::CloseAll() {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    if (mStreamControl) {
-       mStreamControl->CloseAll();
+       auto streamControl = mStreamControl;
+       mStreamControl = nullptr;
+
+       streamControl->CloseAll();
+
+       mStreamControl = streamControl;
+       streamControl = nullptr;
+       mStreamControl->Shutdown();
    }
}
```


CVE-2020-6820 - dom/cache/StreamList.cpp

```
--- a/dom/cache/StreamList.cpp
+++ b/dom/cache/StreamList.cpp
@@ -128,17 +128,25 @@ void StreamList::Close(const nsID& aId)
     if (mStreamControl) {
         mStreamControl->Close(aId);
     }
 }
void StreamList::CloseAll() {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    if (mStreamControl) {
-       mStreamControl->CloseAll();
+       auto streamControl = mStreamControl;
+       mStreamControl = nullptr;
+
+       streamControl->CloseAll();
+
+       mStreamControl = streamControl;
+       streamControl = nullptr;
+       mStreamControl->Shutdown();
    }
}
```

mStreamControl is the parent (browser) side
of the IPDL interface

CVE-2020-6820 - dom/cache/StreamList.cpp

```
--- a/dom/cache/StreamList.cpp
+++ b/dom/cache/StreamList.cpp
@@ -128,17 +128,25 @@ void StreamList::Close(const nsID& aId)
     if (mStreamControl) {
         mStreamControl->Close(aId);
     }
 }
void StreamList::CloseAll() {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    if (mStreamControl) {
-       mStreamControl->CloseAll();
+       auto streamControl = mStreamControl;
+       mStreamControl = nullptr;
+
+       streamControl->CloseAll();
+
+       mStreamControl = streamControl;
+       streamControl = nullptr;
+       mStreamControl->Shutdown();
    }
}
```

If we're setting `mStreamControl` to `nullptr`,
where else is `mStreamControl` checked?

CVE-2020-6820 - dom/cache/StreamList.cpp

```
void StreamList::NoteClosed(const nsID& aId) {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    for (uint32_t i = 0; i < mList.Length(); ++i) {
        if (mList[i].mId == aId) {
            mList.RemoveElementAt(i);
            mManager->ReleaseBodyId(aId);
            break;
        }
    }
    if (mList.IsEmpty() && mStreamControl) {
        mStreamControl->Shutdown();
    }
}
```

```
void StreamList::NoteClosedAll() {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    for (uint32_t i = 0; i < mList.Length(); ++i) {
        mManager->ReleaseBodyId(mList[i].mId);
    }
    mList.Clear();
```

```
    if (mStreamControl) {
        mStreamControl->Shutdown();
    }
}
```

```
void StreamList::Close(const nsID& aId) {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    if (mStreamControl) {
        mStreamControl->Close(aId);
    }
}
```

```
void StreamList::CloseAll() {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    if (mStreamControl) {
        mStreamControl->CloseAll();
    }
    + auto streamControl = mStreamControl;
    + mStreamControl = nullptr;
    +
    + streamControl->CloseAll();
    +
    + mStreamControl = streamControl;
    + streamControl = nullptr;
    + mStreamControl->Shutdown();
    }
}
```

CVE-2020-6820 - dom/cache/StreamList.cpp

```
void StreamList::NoteClosed(const nsID& aId) {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    for (uint32_t i = 0; i < mList.Length(); ++i) {
        if (mList[i].mId == aId) {
            mList.RemoveElementAt(i);
            mManager->ReleaseBodyId(aId);
            break;
        }
    }
}
```

```
if (mList.IsEmpty() && mStreamControl) {
    mStreamControl->Shutdown();
}
```

```
void StreamList::NoteClosedAll() {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    for (uint32_t i = 0; i < mList.Length(); ++i) {
        mManager->ReleaseBodyId(mList[i].mId);
    }
    mList.Clear();
}
```

```
if (mStreamControl) {
    mStreamControl->Shutdown();
}
```

```
void StreamList::Close(const nsID& aId) {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    if (mStreamControl) {
        mStreamControl->Close(aId);
    }
}
```

```
void StreamList::CloseAll() {
    NS_ASSERT_OWNINGTHREAD(StreamList);
    if (mStreamControl) {
        mStreamControl->CloseAll();
    }
    + auto streamControl = mStreamControl;
    + mStreamControl = nullptr;
    +
    + streamControl->CloseAll();
}
```

Calls Send `__delete__`

```
+ mStreamControl->Shutdown();
}
}
```

My Process

1. Figure out how to trigger `StreamList::CloseAll`

My Process

1. Figure out how to trigger `StreamList::CloseAll` 
`cache.delete()`

My Process

1. Figure out how to trigger `StreamList::CloseAll` 
2. Find two ways to trigger `StreamList::CloseAll`

My Process

1. Figure out how to trigger `StreamList::CloseAll` ✓
2. Find two ways to trigger `StreamList::CloseAll` ✓
 - a. Ran the mochitests for `dom/cache` with a breakpoint on `StreamList::CloseAll`.
 - b. Found `QuotaManagerService::ClearStoragesForPrincipal`.
 - c. Didn't realize at first that `StreamList::CloseAll` was in the browser process, not the renderer. So we can assume we already have a compromised renderer. Then I could call it.

My Process




1. Figure out how to trigger `StreamList::CloseAll` ✓
2. Find two ways to trigger `StreamList::CloseAll` ✓
3. Trigger `StreamList::CloseAll` from two different threads

My Process

1. Figure out how to trigger `StreamList::CloseAll` ✓
2. Find two ways to trigger `StreamList::CloseAll` ✓
3. Trigger `StreamList::CloseAll` from two different threads ✗
 - a. With the IPDL, could only get these calls queued, not where they raced with each other.
 - b. And I so I called it. The return on investment was limited when the bug would be derestricted at some point and the reporters said they'd write a blog post.

My Process

Fast forward to Maddie has spent time learning IPDL internals and the bug is derestricted...

1. Figure out how to trigger `StreamList::CloseAll` 
2. Find two ways to trigger `StreamList::CloseAll` 
3. Trigger `StreamList::CloseAll` from two different threads 

My Process

Fast forward to Maddie has spent time learning IPDL internals and the bug is derestricted...

1. Figure out how to trigger `StreamList::CloseAll` ✓
- ~~2. Find two ways to trigger `StreamList::CloseAll` ✓~~
- ~~3. Trigger `StreamList::CloseAll` from two different threads~~

CVE-2019-17026 Closing Thoughts

- I couldn't figure it out from the patch in a timely way.
- Hurdles
 - First, thought it was the renderer exploit, not the sandbox escape
 - Then, got stuck in IPDL internals and “race condition”.
- But when bug report was derestricted and I had spent more time learning IPDL, the vuln was much more obvious.

Source Code Patch Diffing Conclusions

- More exact, and more likelihood of being correct compared to binary patch diffing due to patch being 1 fix rather than all of that update cycle's fixes.
- Knowledge of common bug classes and the platform can still be critical.
- Relies on a semblance of transparency from vendors.
- Have a clear test case to know when you got the vulnerability correct.

“Bug Hunting” based on exploit
details

Android Binder (CVE-2019-2215)

- I reported the bug. [PO 1942](#)
- [Android Security Bulletin - Oct 2019](#)
- Received a descriptive list of characteristics about the exploit.
- If you'd like to hear about the full process and details of finding this vulnerability, check out the [blog post](#) or [presentation](#).

The Details

- It is a kernel privilege escalation using a use-after-free vulnerability, reachable from inside the Chrome sandbox.
- It works on Pixel 1 and 2, but not Pixel 3 and 3a.
- It was patched in the Linux kernel ≥ 4.14 without a CVE.
- `CONFIG_DEBUG_LIST` breaks the primitive.
- `CONFIG_ARM64_UAO` hinders exploitation.
- The vulnerability is exploitable in Chrome's renderer processes under Android's `isolated_app` SELinux domain.
- The exploit requires little or no per-device customization.
- List of affected devices.

My Process

- Combing through changelogs & patches
- Looking for any patches applied to Pixel 3, but not to Pixel 2
- Searched patches for `__list_add` or `list_del`

My Process

- Combing through changelogs & patches
- Looking for any patches applied to Pixel 3/Linux 4.9, but not to Pixel 2/Linux 4.4
- Searched patches for `__list_add` or `list_del`

Thought that I had found a bug, but when I then tried to compare it to all the details, it didn't match one AND it seemed very complex to exploit.

The Details

- It is a kernel privilege escalation using a use-after-free vulnerability, reachable from inside the Chrome sandbox.
- It works on Pixel 1 and 2, but not Pixel 3 and 3a.
- It was patched in the Linux kernel ≥ 4.14 without a CVE.
- `CONFIG_DEBUG_LIST` breaks the primitive.
- `CONFIG_ARM64_UAO` hinders exploitation.
- The vulnerability is exploitable in Chrome's renderer processes under Android's `isolated_app` SELinux domain.
- The exploit requires little or no per-device customization.
- List of affected devices.
- **Look in Binder**

My Process v2

- Combing through changelogs & patches
- When diffing Pixel 2 and Pixel 3 **drivers/android/binder.c**, there were only a few significant changes.
- Commit [550c01d0e051461437d6e9d72f573759e7bc5047](#) stood out in the log because:
 - It discusses fixing a “use-after-free” in the commit message,
 - It is a patch from upstream, and
 - The upstream patch was only applied to 4.14.
 - The “use-after-free” includes a **list_del**

“Bug Hunting” based on exploit details Conclusions

- Highly dependent on the details you receive.
- Are those details are searchable programmatically?
- Worth trying. You never know how easy or not something will be.

Conclusion

Conclusion

- Even within single techniques, there is a lot of room for creativity in tracking down the vulnerability:
 - Test case minimizer, forcing crash while running, etc.
- Challenge your assumptions that come from “descriptions” of the vulnerability.
- The biggest guarantor for success is in-depth knowledge of the target and exploit dev knowledge.
 - Not a specific technique.
 - It’s a good thing we can all grow experience :)

Thank you!
@maddiestone