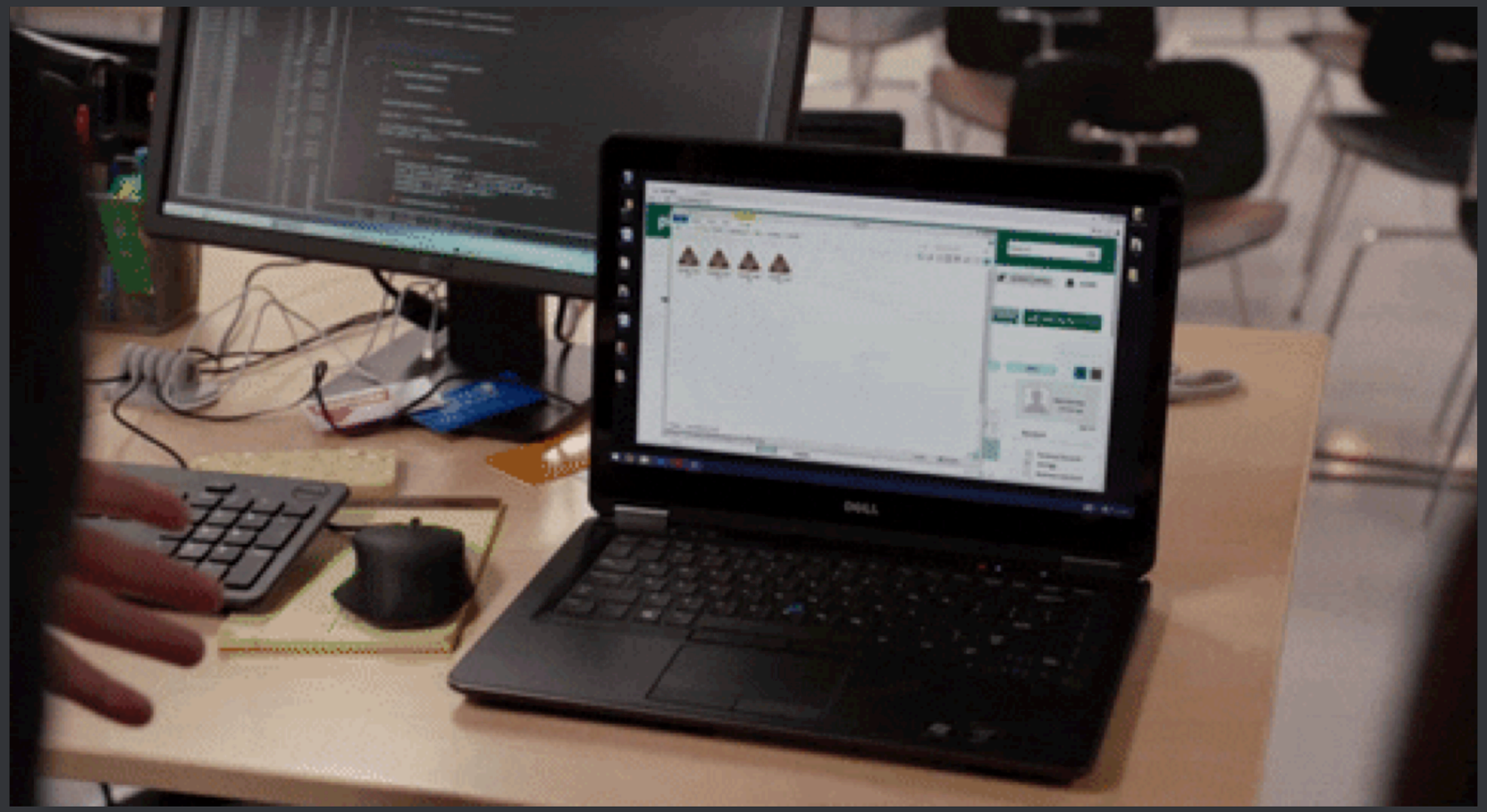
CS 253: Web Security

Browser architecture, Writing secure code

Admin

- Final exam
  - Tuesday, December 10, 3:30pm - 6:30pm in 200-305
  - Sample final will be released tomorrow
  - The alternate exam is Monday, December 9 (time/location TBD)
  - Email accommodations to **feross@stanford.edu** by today
- Office hours after class today, 3-5pm, in Gates 323

Browser architecture



Attacking users through their browsers

- Browsers are complicated
  - Large and complex codebase
  - Network-visible interface
  - Lots of new features added regularly
- Without a solid architecture, this is a recipe for disaster!

An elegant example: Chrome RCE

```
let faultyArrayBuffer = new ArrayBuffer(32)
faultyArrayBuffer.__defineGetter__('byteLength', () => 0xFFFFFFFC)
let faultyBuffer = new Uint32Array(faultyArrayBuffer)

// Read whatever memory we want
console.log(faultyBuffer[1000])

// Write whatever memory we want
faultyBuffer[5000] = 0x42
```

- There is lots of additional code required to get RCE, but you get the idea!
- See: https://bugs.chromium.org/p/chromium/issues/detail?id=351787

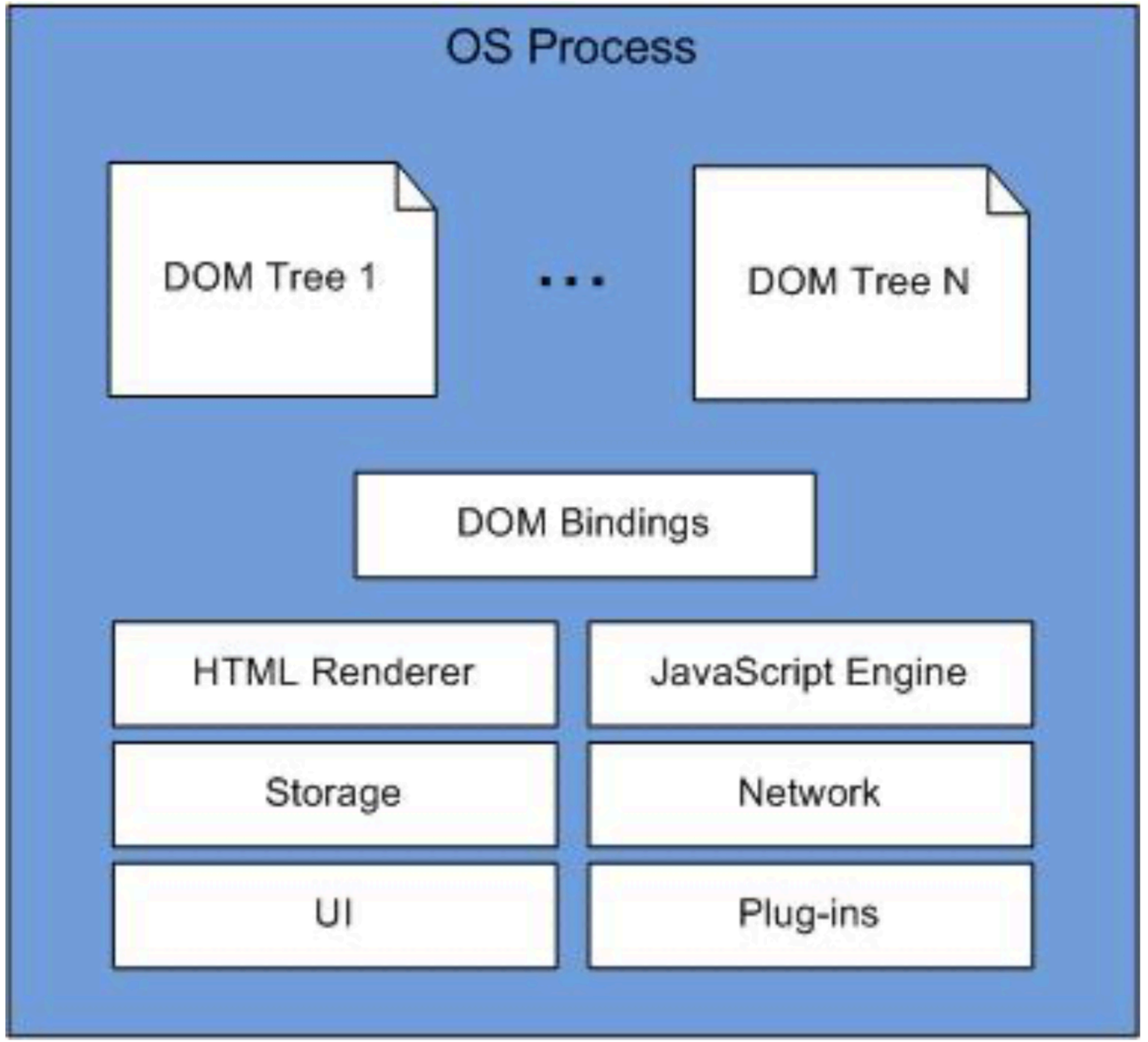
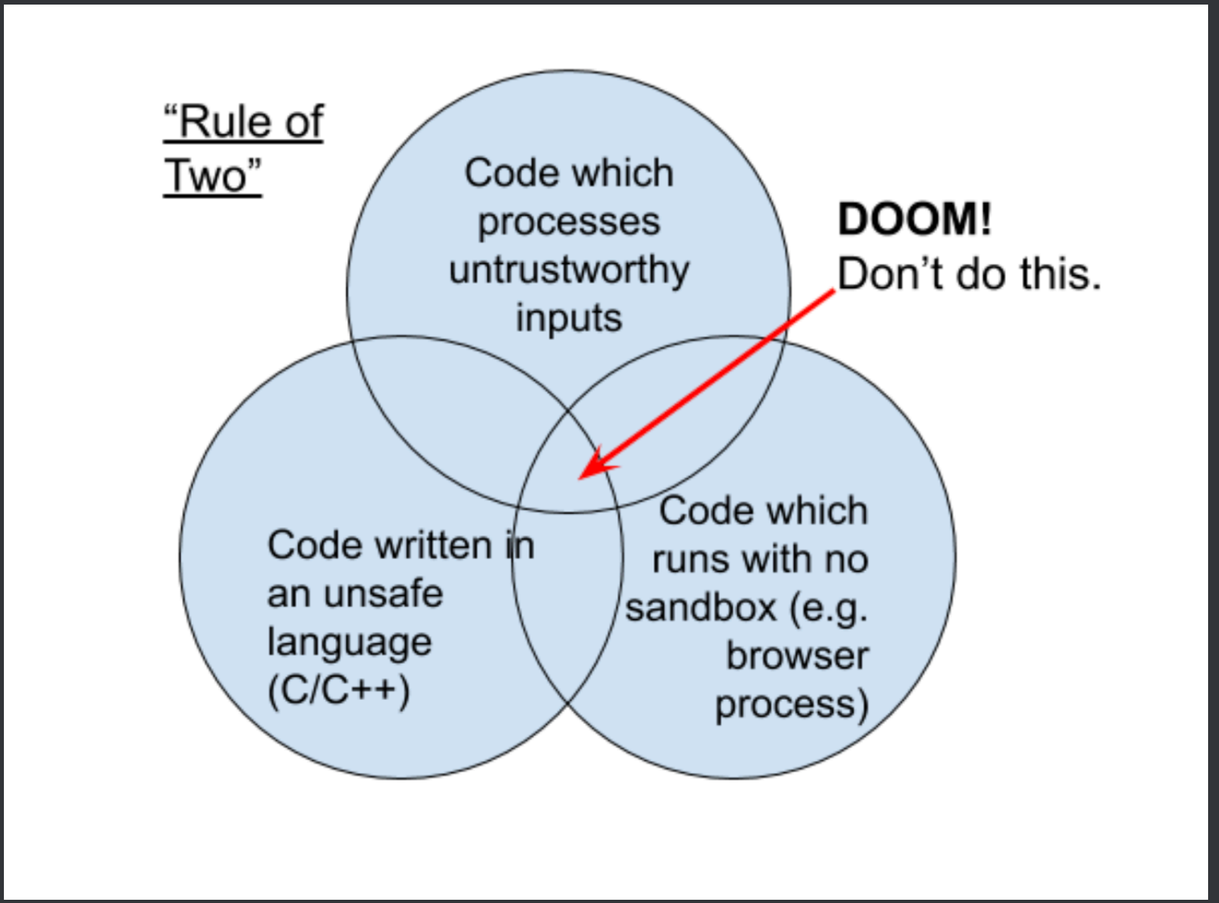


Figure 1. Modular browser architecture

Top goal: Reduce bugs caused by memory unsafety

- Recent Microsoft study: "~70% of the vulnerabilities addressed through a security update each year continue to be memory safety issues"
- As of March 2019, only about 5 of 130 public Critical-severity bugs are not obviously due to memory corruption

The Rule of 2


“Rule of Two”
Code which processes untrustworthy inputs
DOOM!
Don’t do this.
Code written in an unsafe language (C/C++)
Code which runs with no sandbox (e.g. browser process)

Rendering Engine	Browser Kernel
HTML parsing	Cookie database
CSS parsing	History database
Image decoding	Password database
JavaScript interpreter	Window management
Regular expressions	Location bar
Layout	Safe Browsing blacklist
Document Object Model	Network stack
Rendering	SSL/TLS
SVG	Disk cache
XML parsing	Download manager
XSLT	Clipboard

Both
URL parsing
Unicode parsing

Table 1: The assignment of tasks between the rendering engine and the browser kernel.

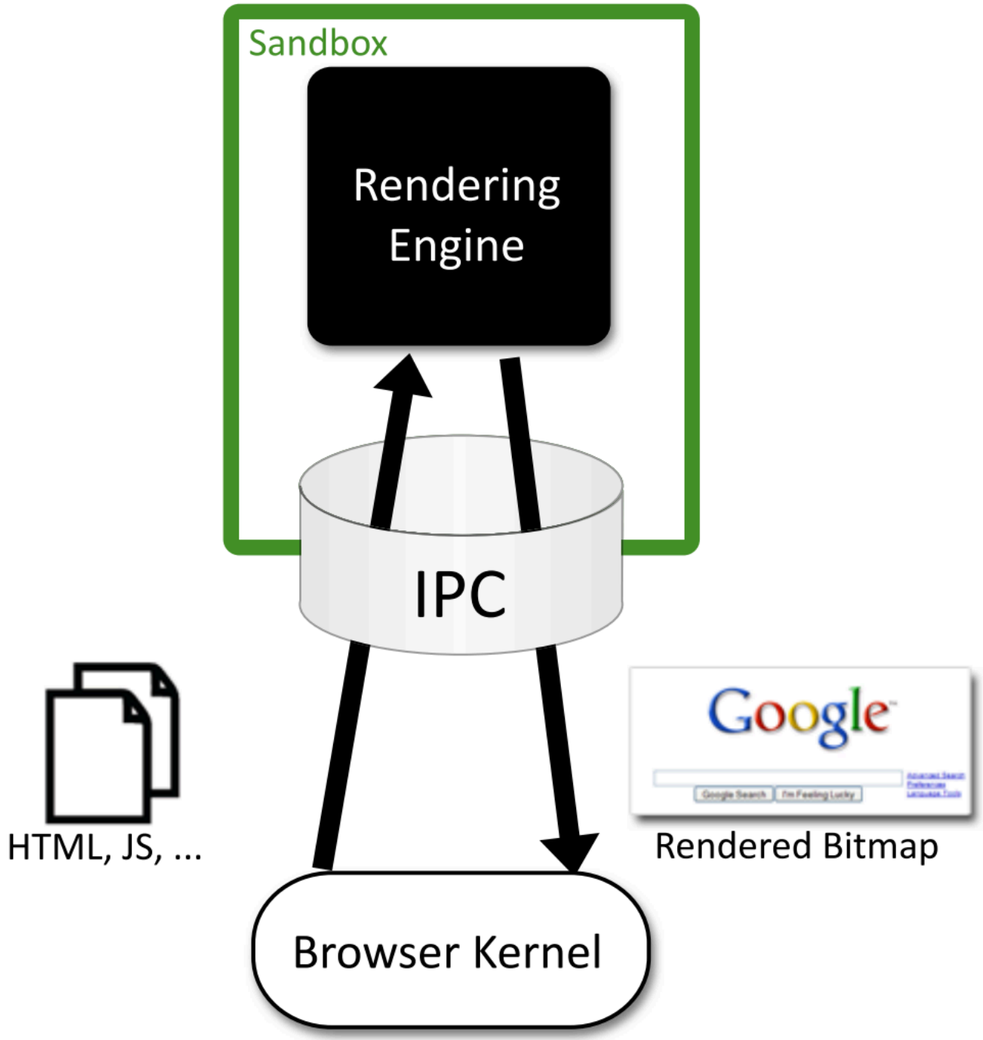


Figure 1: The browser kernel treats the rendering engine as a black box that parses web content and emits bitmaps of the rendered document.

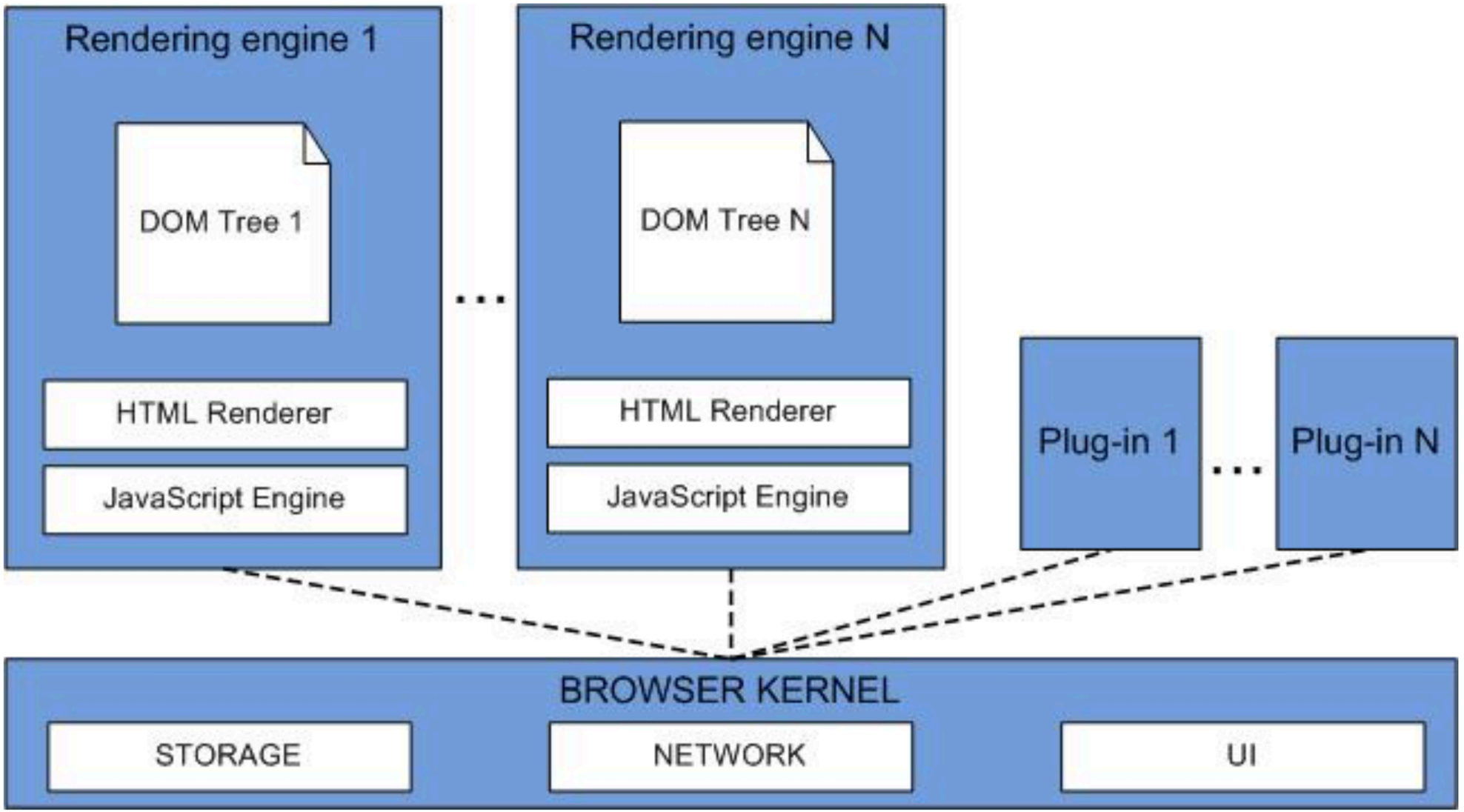
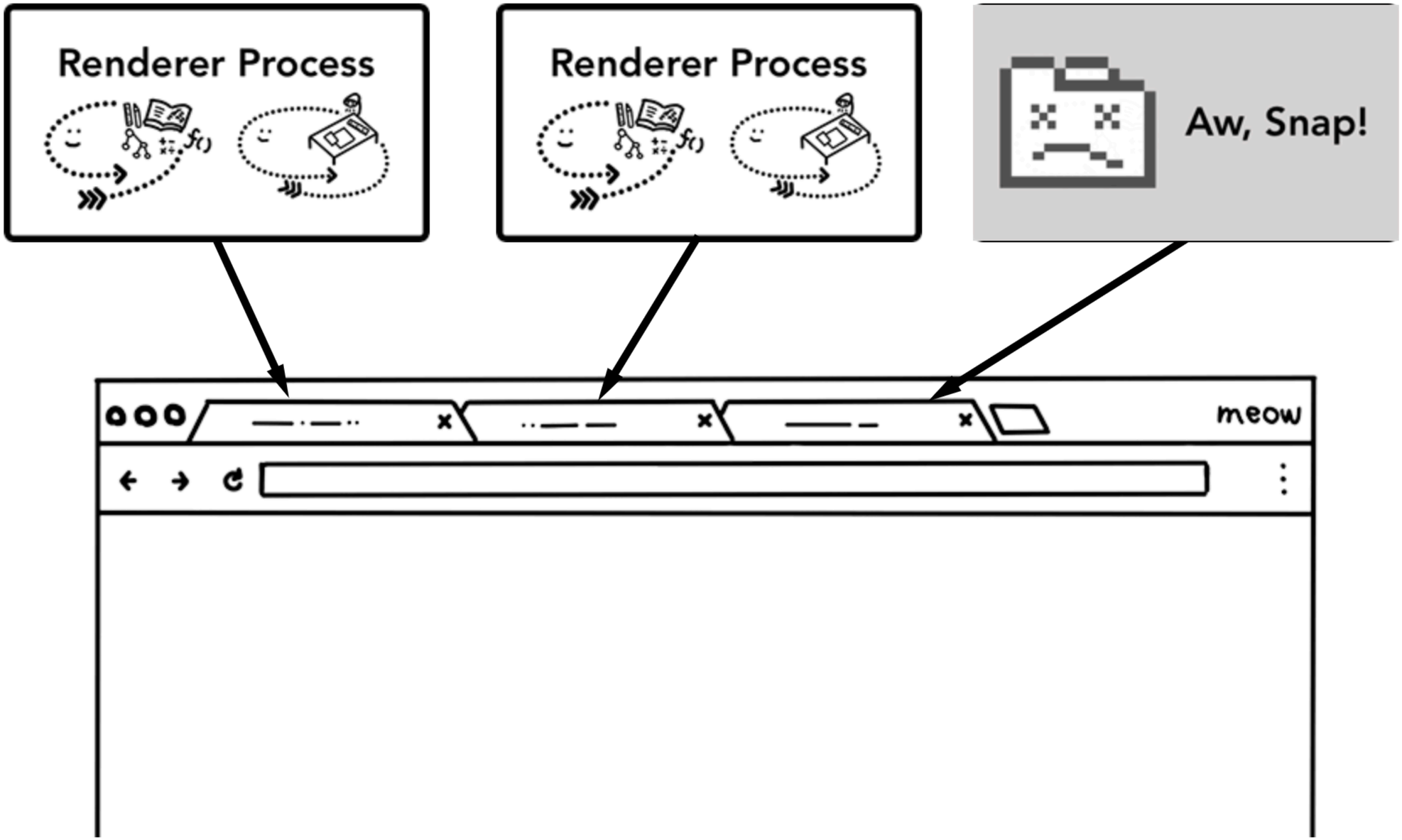
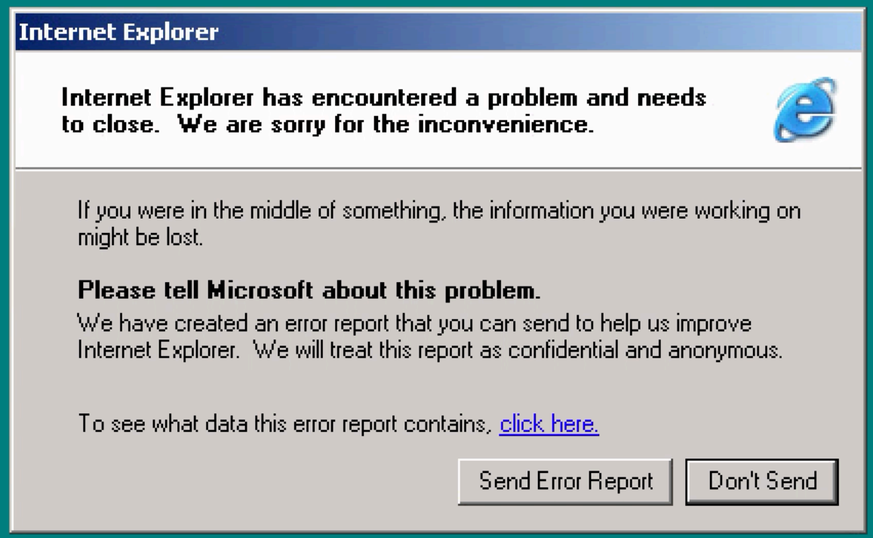


Figure 3. Chrome architecture


Renderer Process
Renderer Process
Aw, Snap!
meow


Internet Explorer
Internet Explorer has encountered a problem and needs to close. We are sorry for the inconvenience.
If you were in the middle of something, the information you were working on might be lost.
Please tell Microsoft about this problem.
We have created an error report that you can send to help us improve Internet Explorer. We will treat this report as confidential and anonymous.
To see what data this error report contains, click here.
Send Error Report
Don't Send

Crash IE with one line

```
<input type crash>
```

Chrome multi-process architecture

- Renderer processes don't require direct access to disk, network, or devices, so Chrome runs them inside a restricted sandbox to limit the damage that attackers can cause if they exploit a vulnerability in the renderer
  - Makes it difficult for attackers to access the user's filesystem, devices, or privileged pages (e.g., settings or extensions) and pages in other profiles (e.g., Incognito mode)
- Allows web pages in unrelated tabs to run in parallel
- Allows users to continue using the browser and other tabs when a renderer process crashes

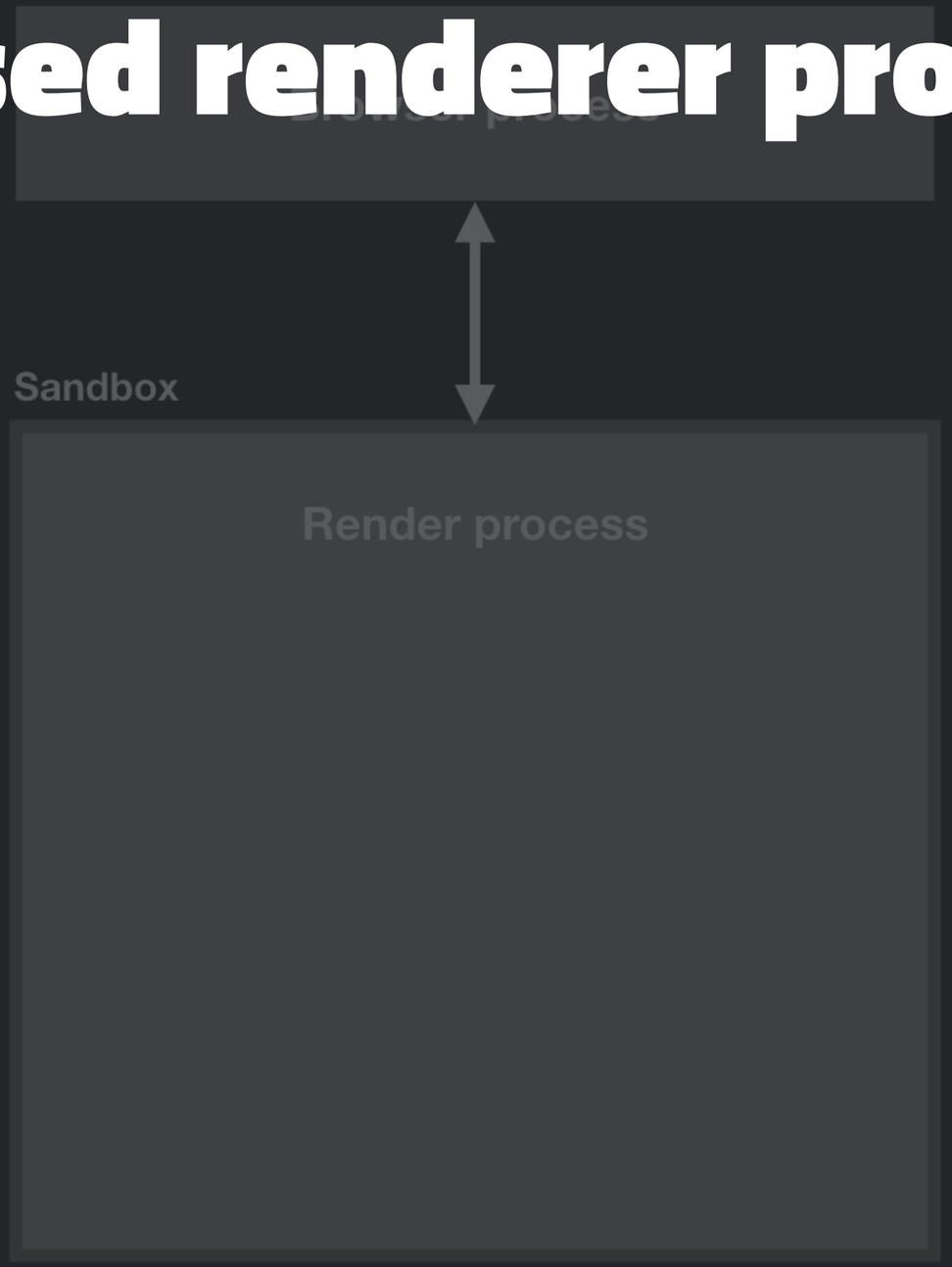
Chrome auto-update

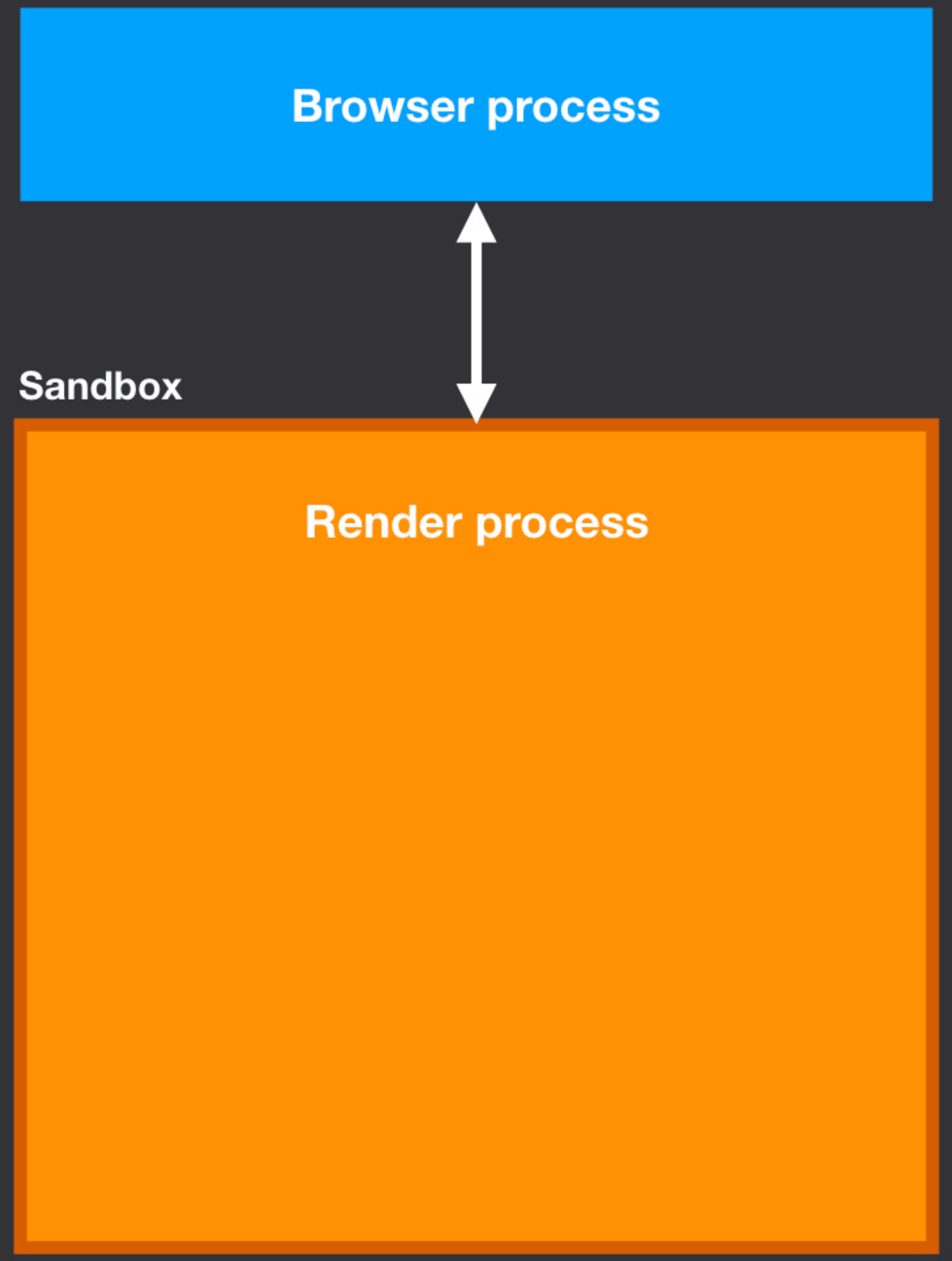
Delving back into project history long before we launched publicly in 2008, the autoupdate project was one of the very first we started working on. **The idea was to give people a blank window with an autoupdater. If they installed that, over time the blank window would grow into a browser.** And today, some five years after our autoupdater started updating a mostly blank window that could barely load webpages, it is now an engine for delivering an incredibly sophisticated web technology platform onto our users' computers, which in turn allows web app developers to build amazing new online experiences.

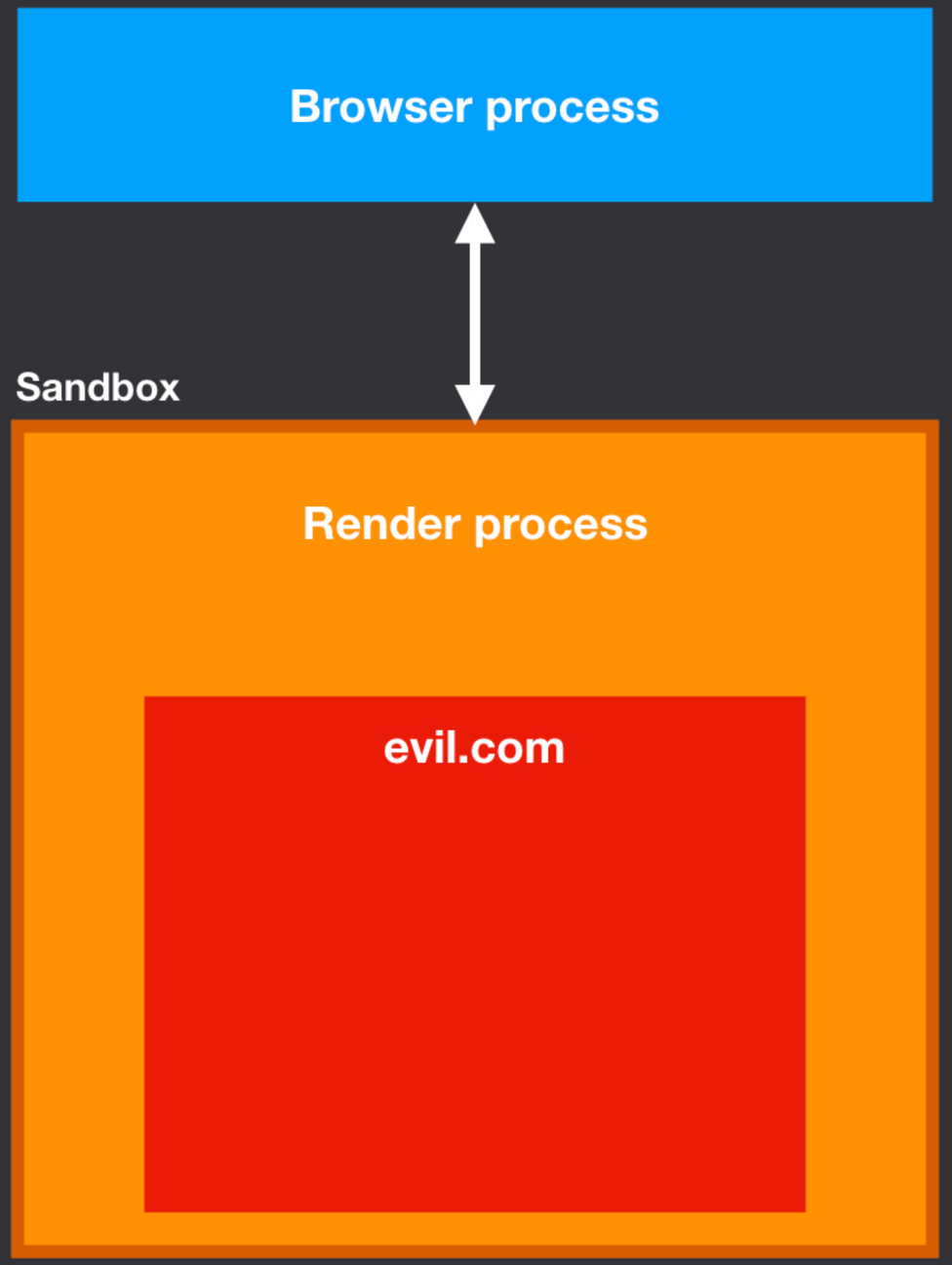
Chrome multi-process architecture limitations

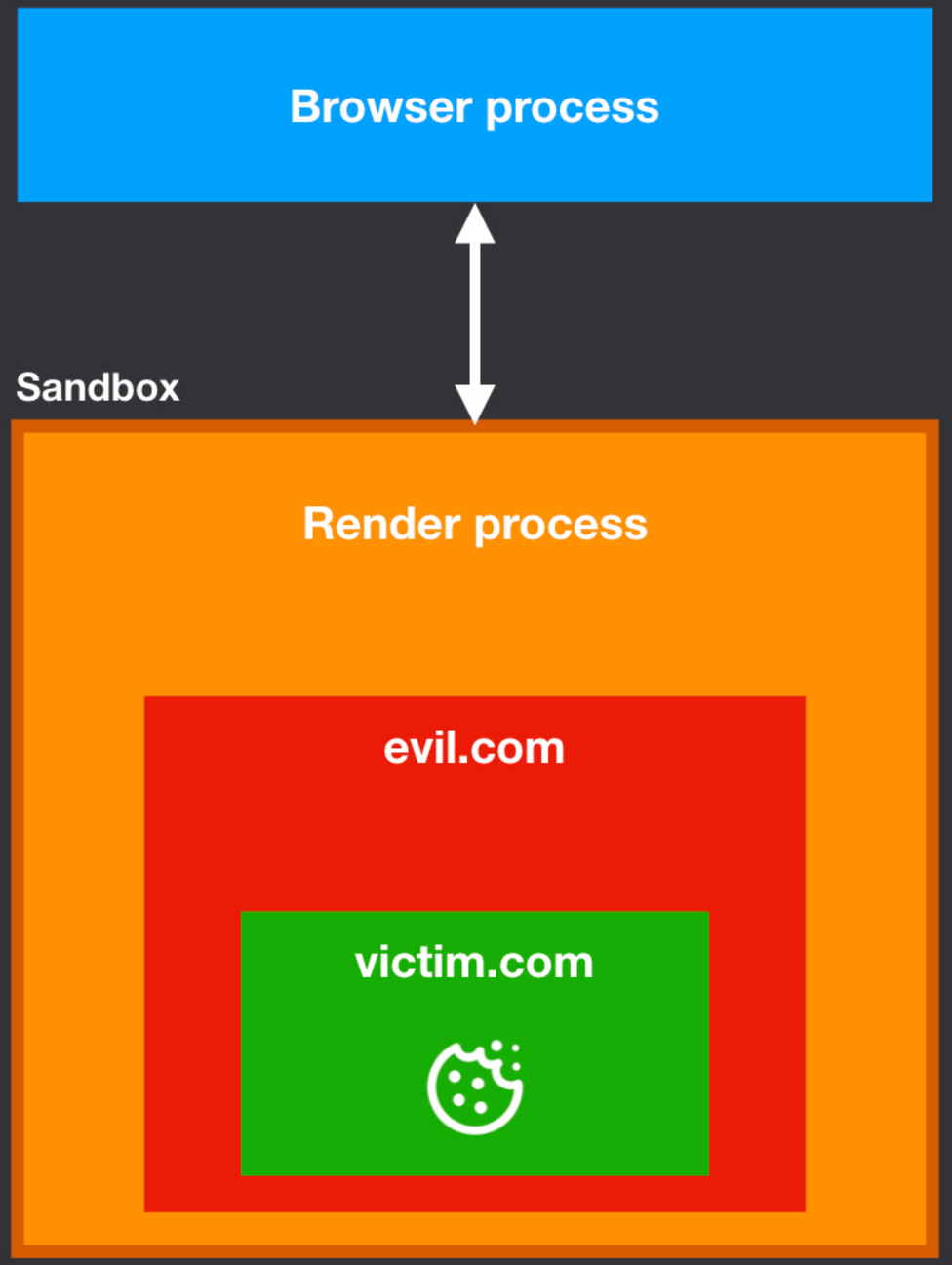
- Chrome would like to place pages from different origins into different renderer processes, but it's difficult in some situations, so it's forced to put pages from different origins in same process
  - Synchronous iframe DOM access (e.g. different domains + `document.domain`)
  - Under memory pressure, lots of processes exhaust device memory
- Every renderer has access to all cookies because a page from one origin can request images, scripts, etc. from other origins

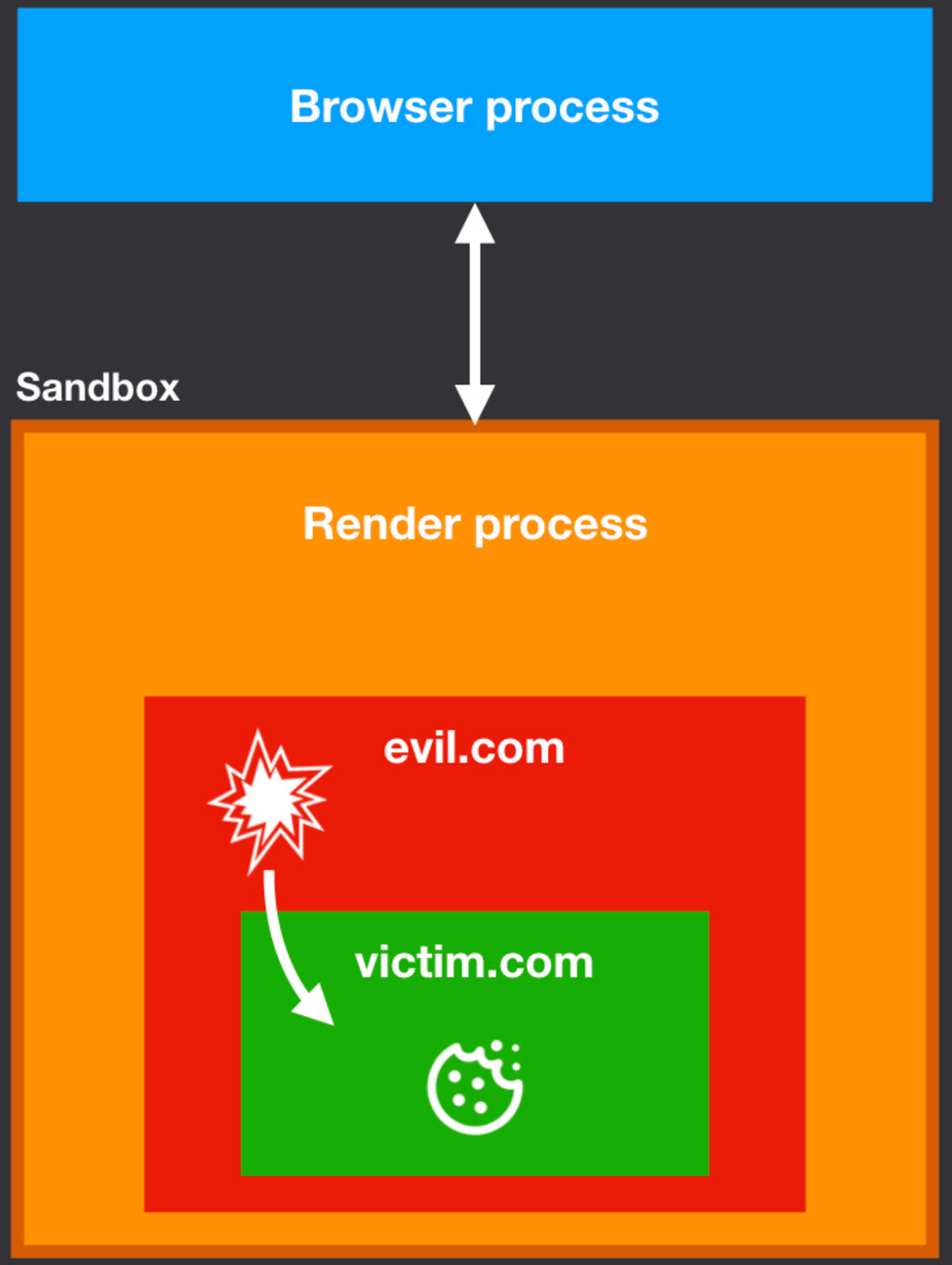
Compromised renderer process



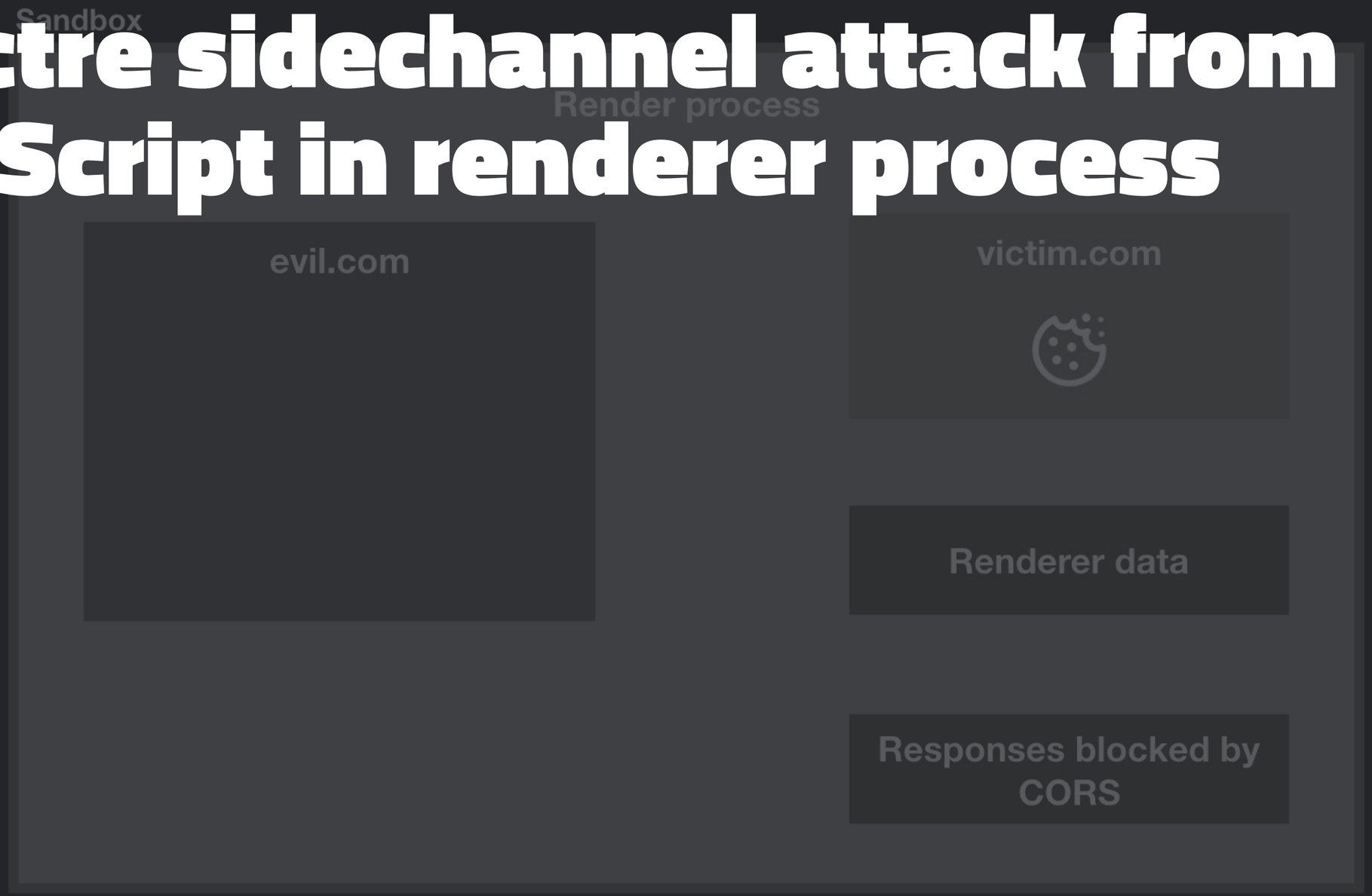

Browser process
Sandbox
Render process

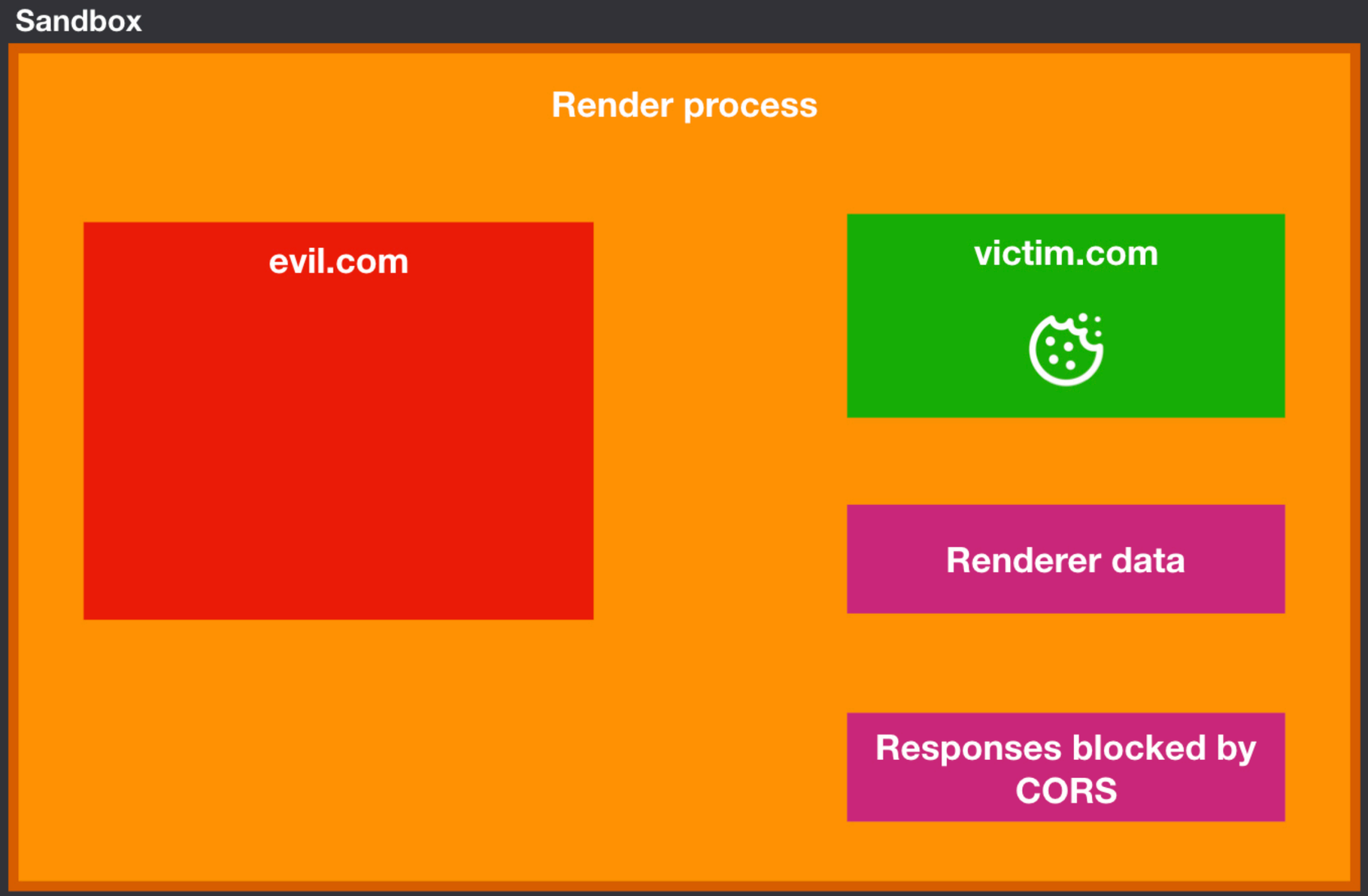

Browser process
Sandbox
Render process
evil.com


Browser process
Sandbox
Render process
evil.com
victim.com


Browser process
Sandbox
Render process
evil.com
victim.com

Spectre sidechannel attack from JavaScript in renderer process




Sandbox
Render process
evil.com
victim.com
Renderer data
Responses blocked by CORS

Sandbox

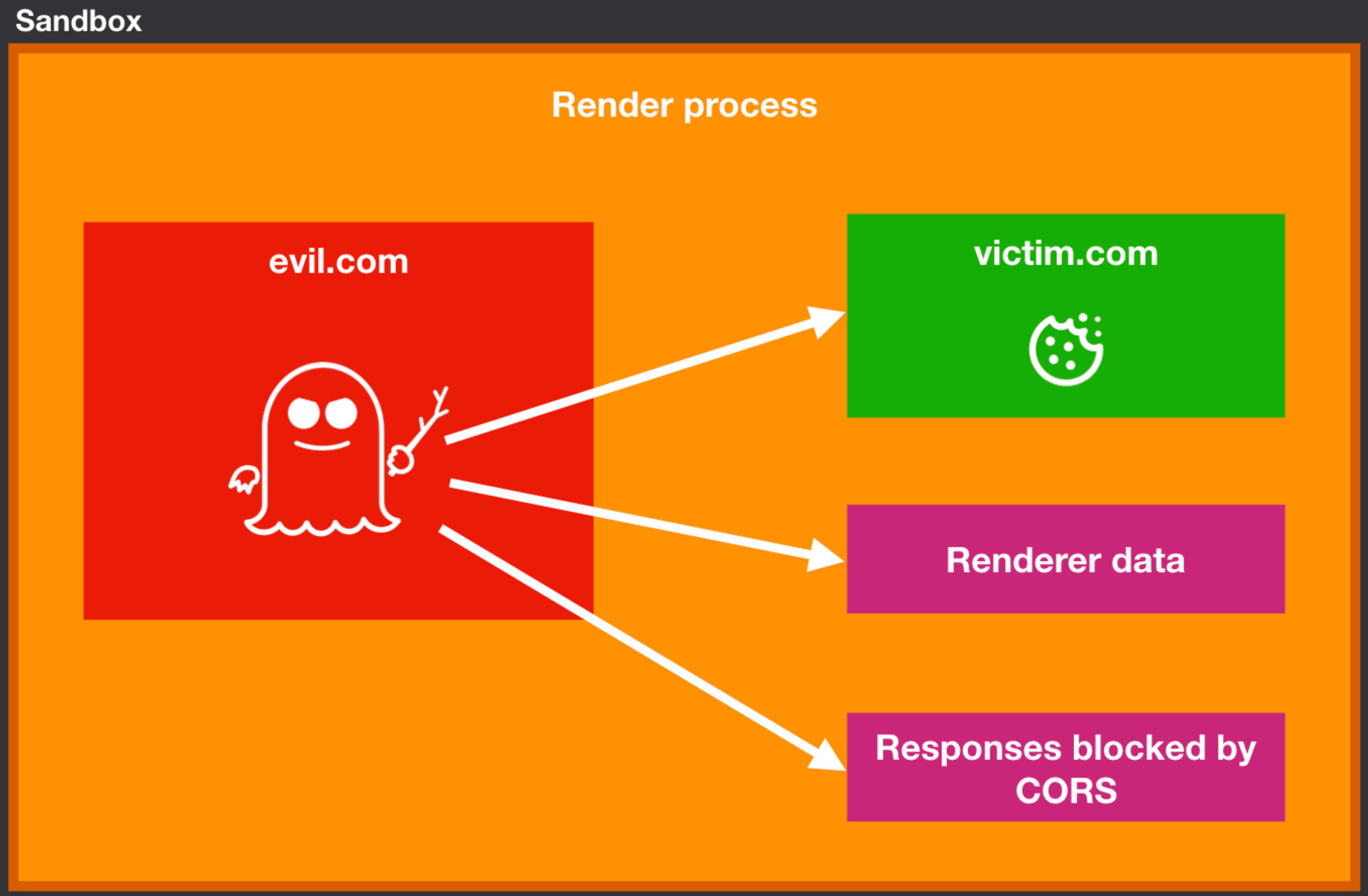
Render process

evil.com

victim.com

Renderer data

Responses blocked by CORS


Sandbox
Render process
evil.com
victim.com
Renderer data
Responses blocked by CORS

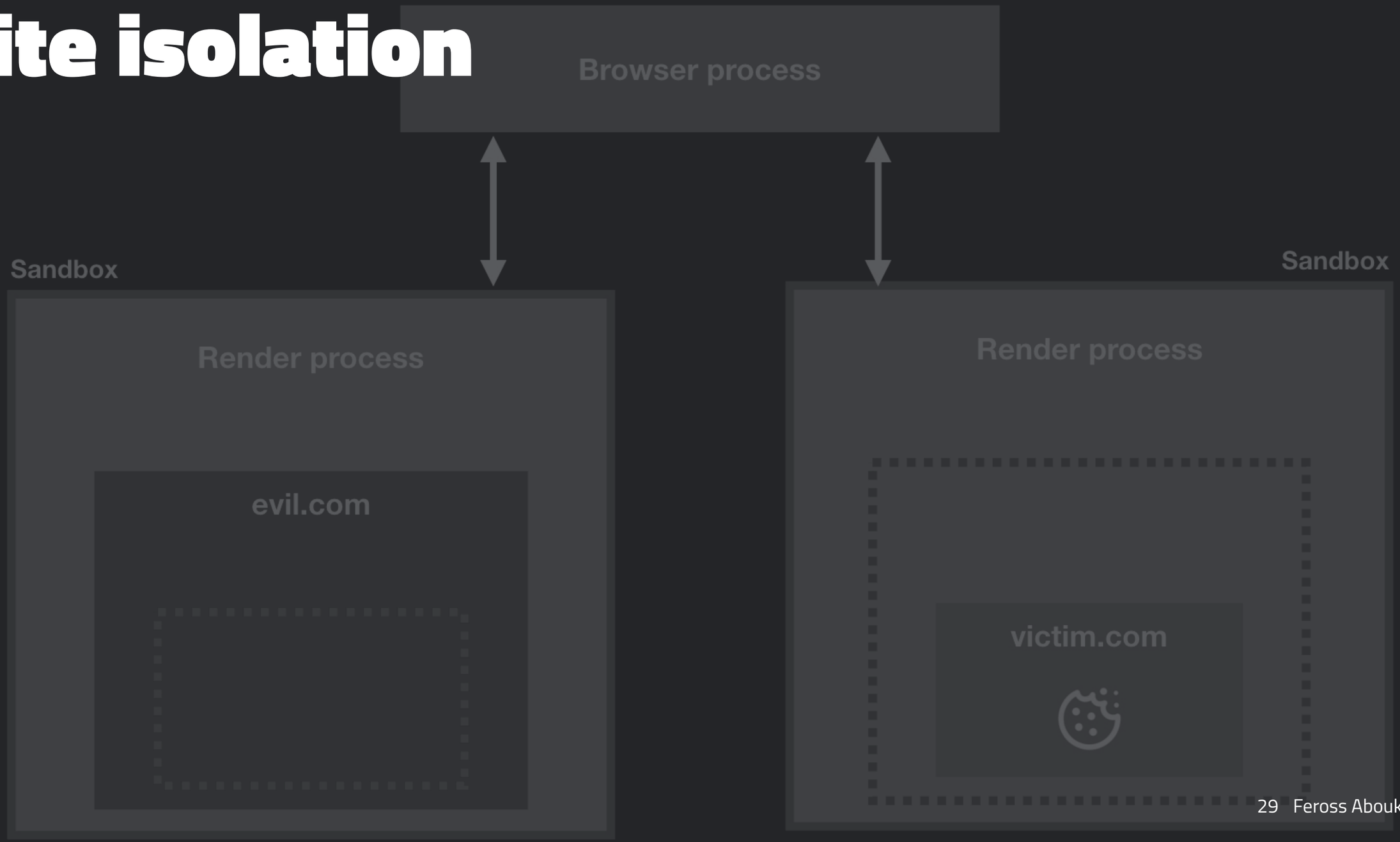
Site isolation

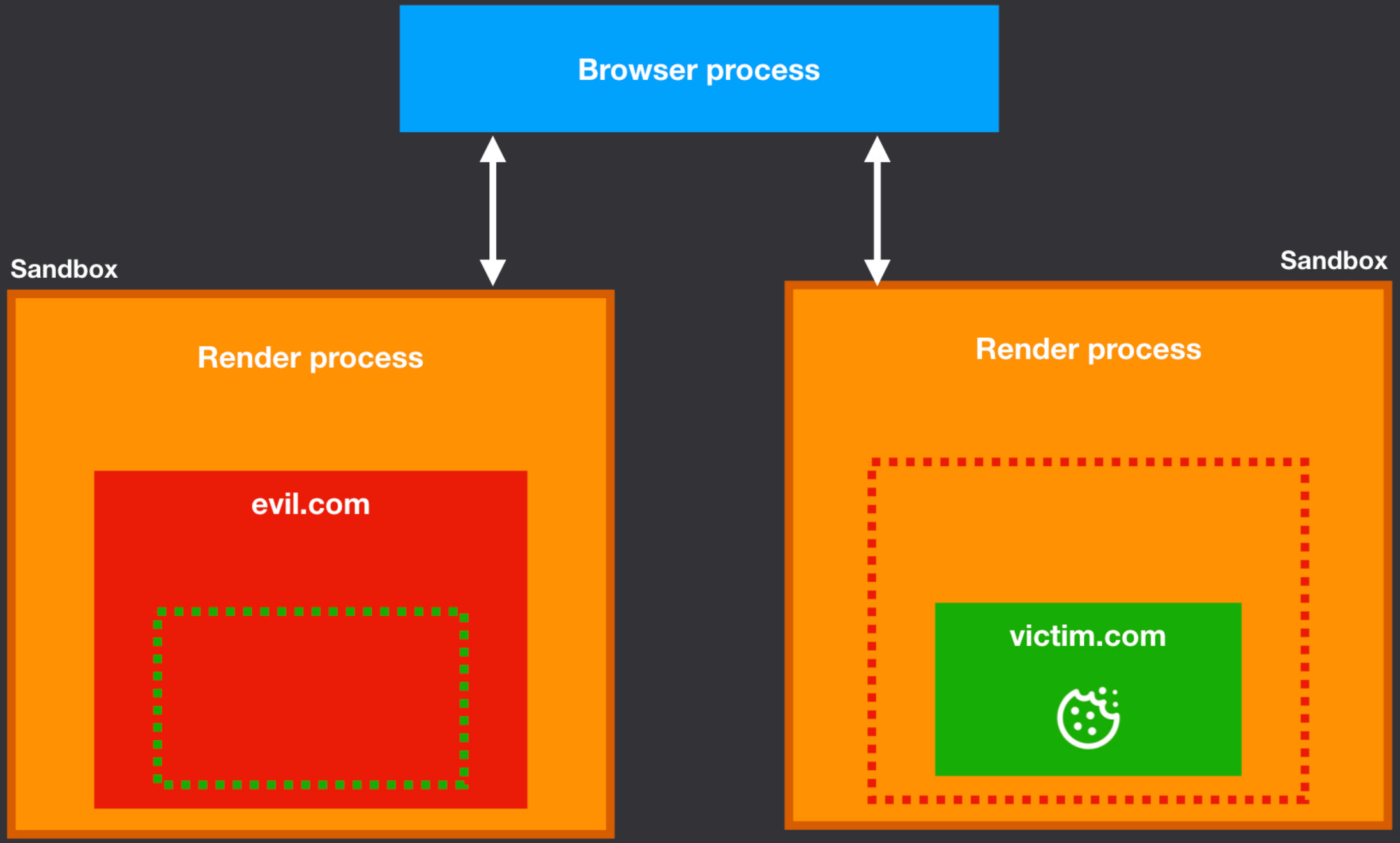
- **Idea:** Isolate sites from each other (not just tabs)
  - Each process should have content from one site (not origin)
  - Then the browser process can restrict each renderer process to access e.g. only the cookies for the specific site
- Helps defend against speculative side channel attacks (e.g., Spectre) and UXSS (universal XSS), and fully compromised renderer processes

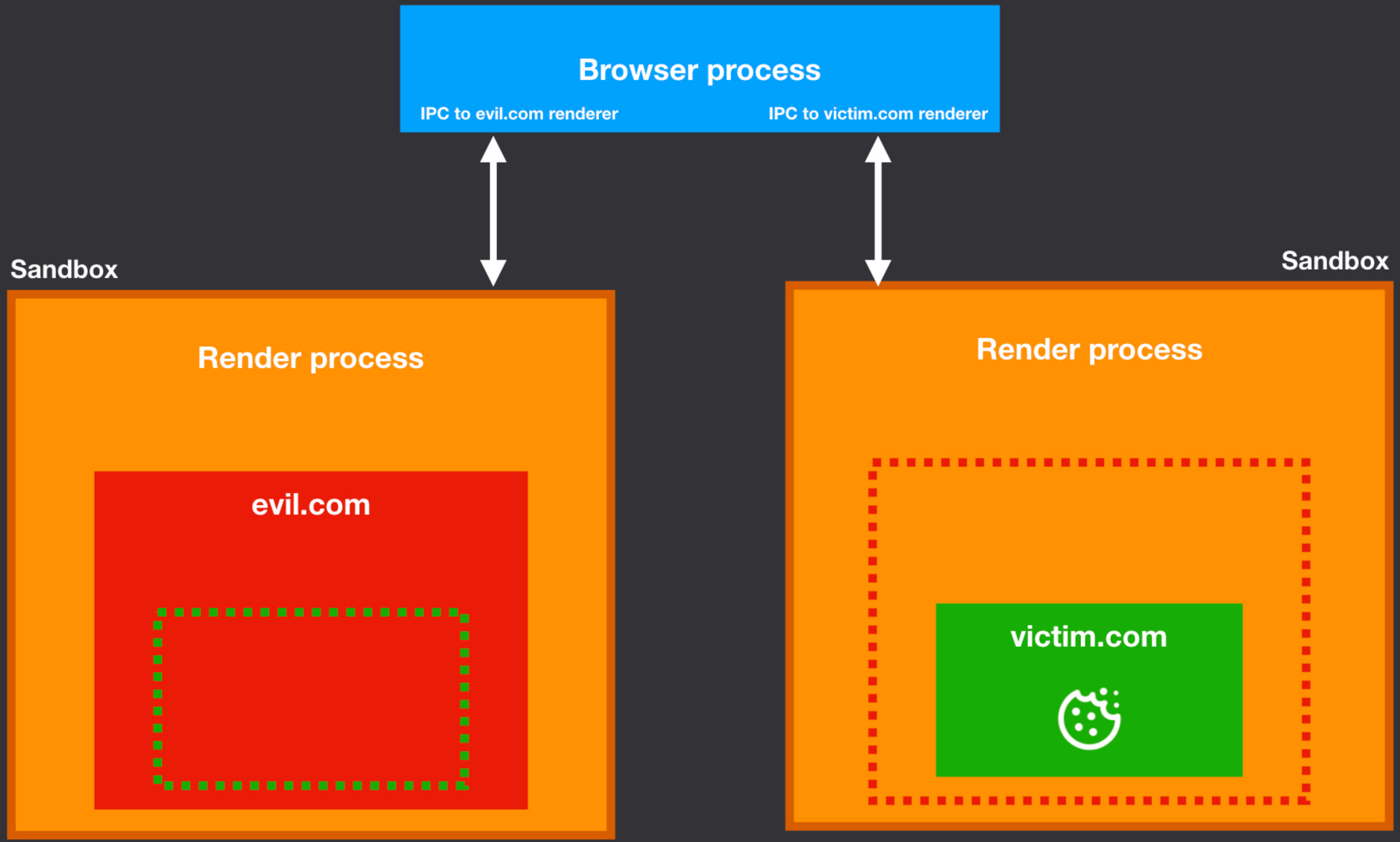
Site isolation goals

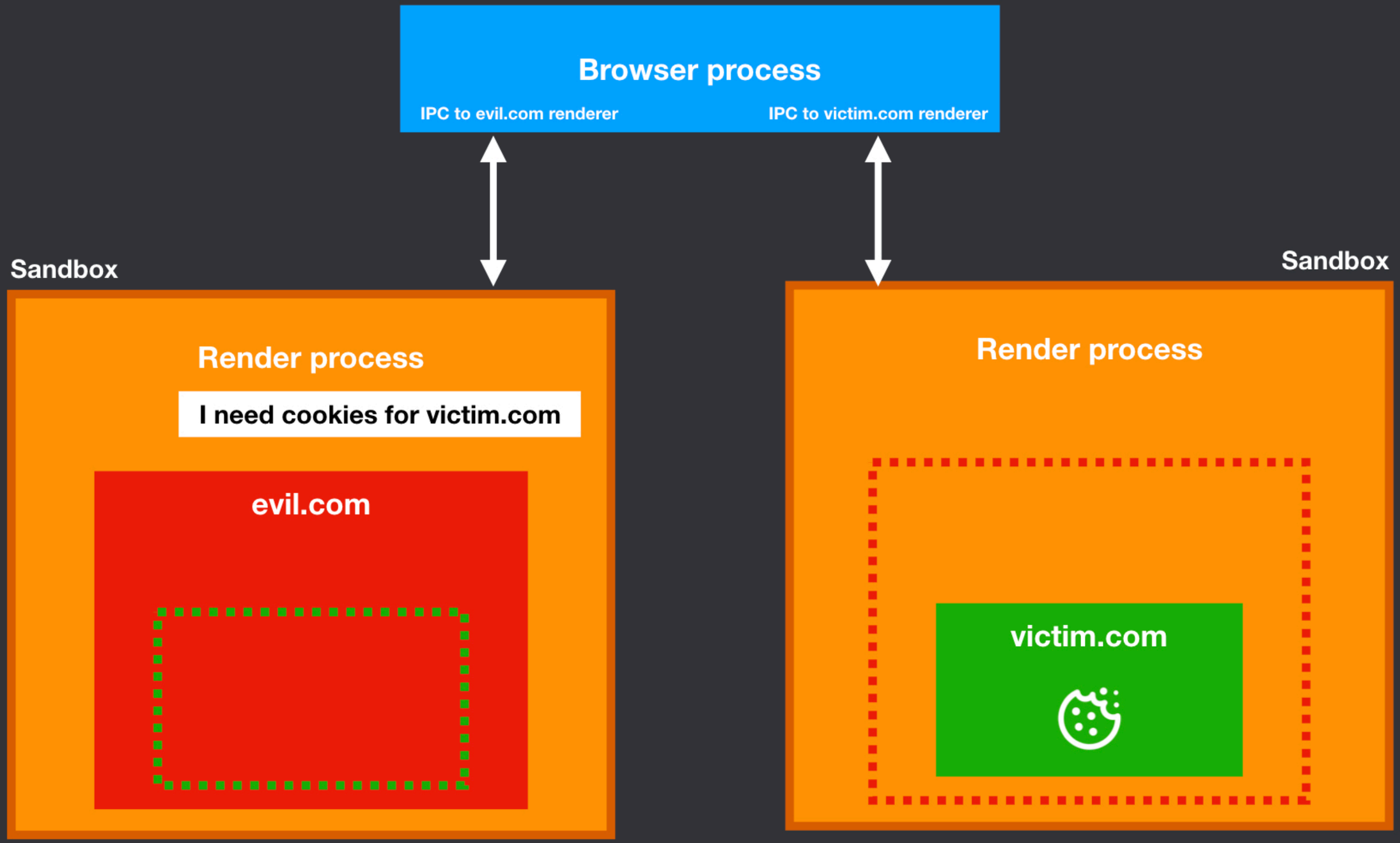
- **Stealing cross-site cookies and HTML5 stored data.** We can prevent a renderer process from receiving cookies or stored data from sites other than its own.
- **Stealing cross-site HTML, XML, and JSON data.** Using content type and content sniffing, we can prevent a renderer process from loading most sensitive cross-site data. We cannot block all cross-site resources, however, because images, scripts, and other opaque files are permitted across sites.
- **Stealing saved passwords.** We can prevent a renderer process from receiving saved passwords from sites other than its own.
- **Abusing permissions granted to another site.** We can prevent a renderer process from using permissions such as geolocation that the user has granted to other sites.
- **Compromising X-Frame-Options.** We can prevent a renderer process from loading cross-site pages in iframes. This allows the browser process to decide if a given site can be loaded in an iframe or not based on X-Frame-Options or CSP frame-ancestors headers.
- **Accessing cross-site DOM elements via UXSS bugs.** An attacker exploiting a universal cross-site scripting bug in the renderer process will not be able to access DOM elements of cross-site pages, which will not live in the same renderer process.

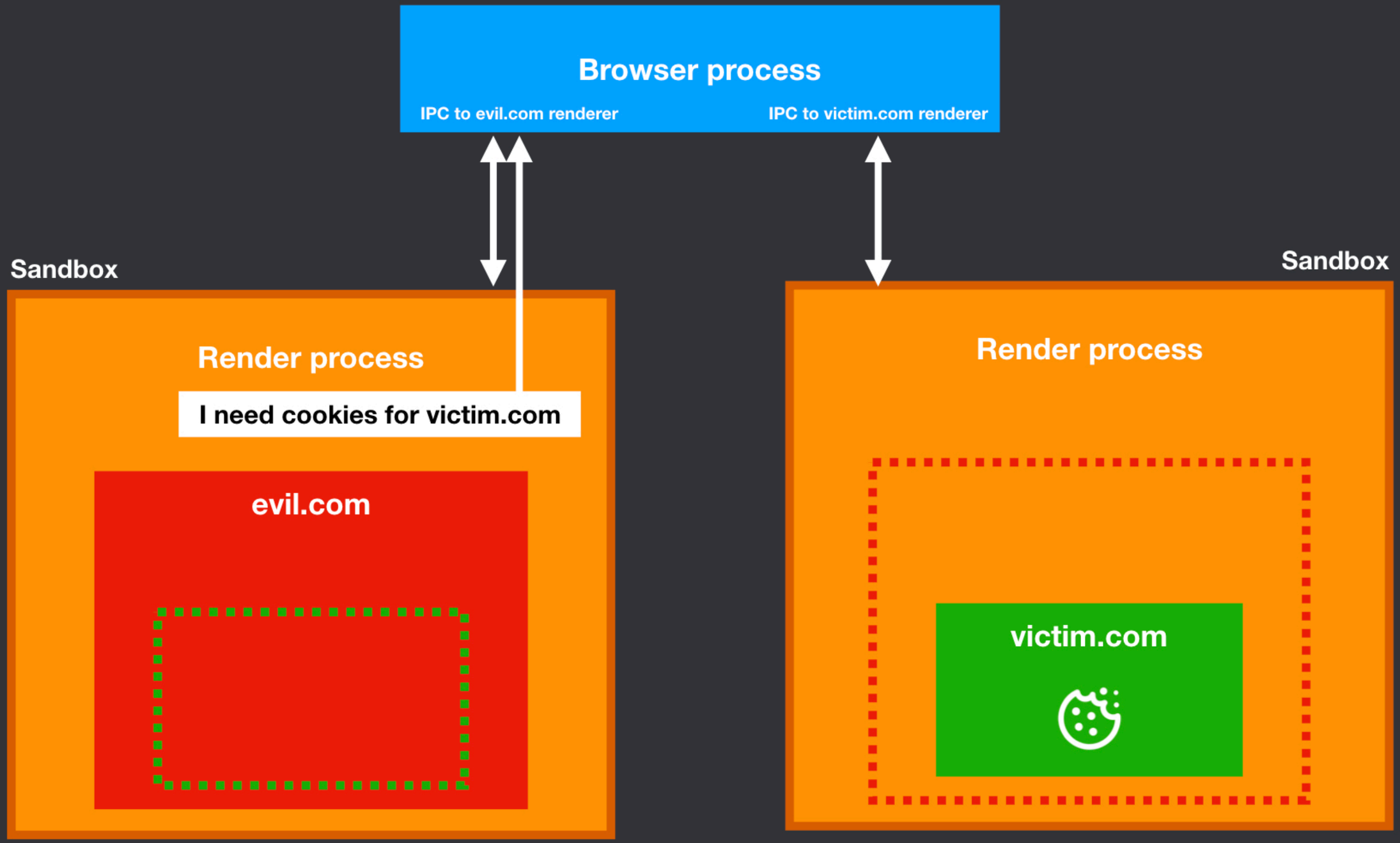
Site isolation

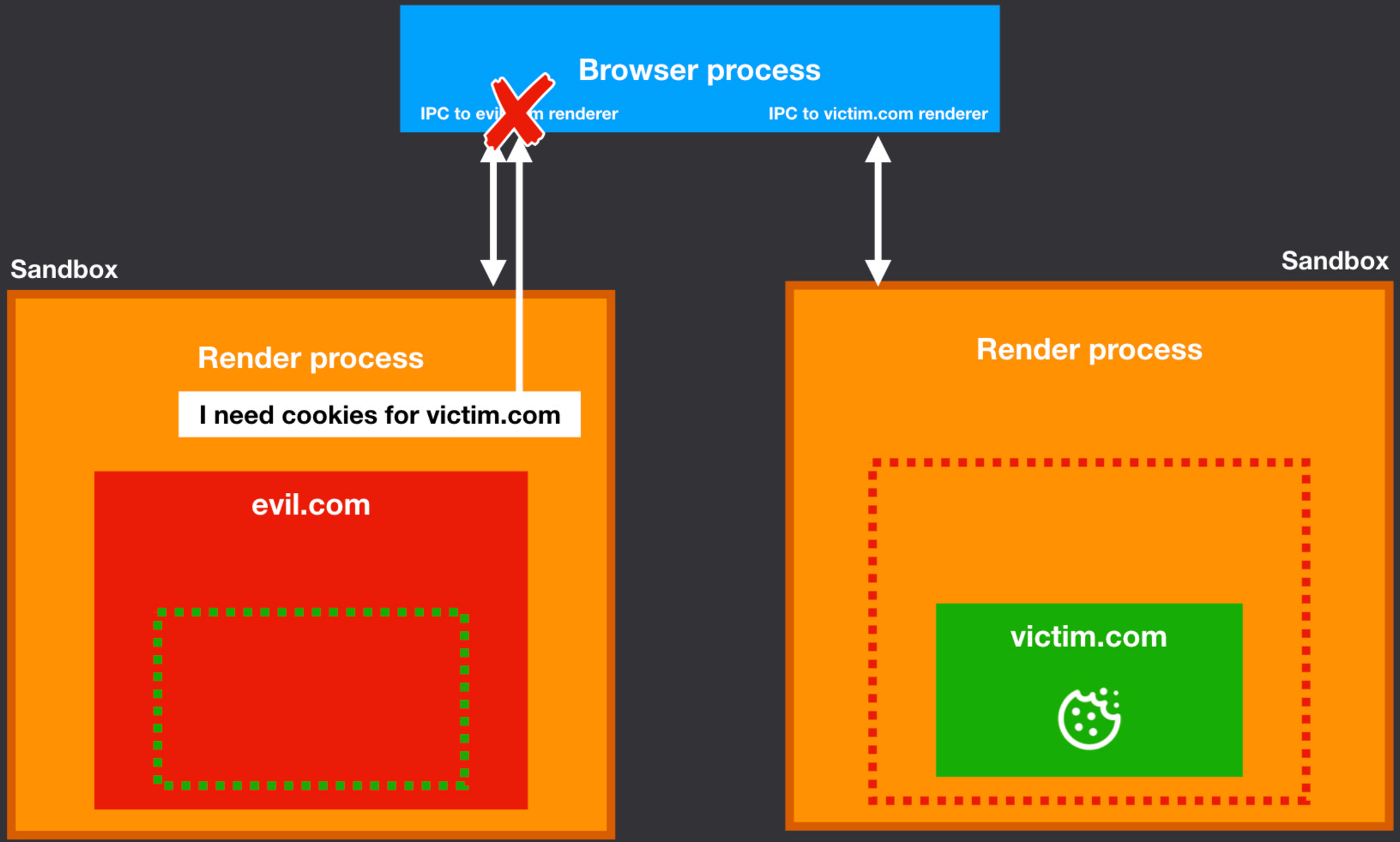


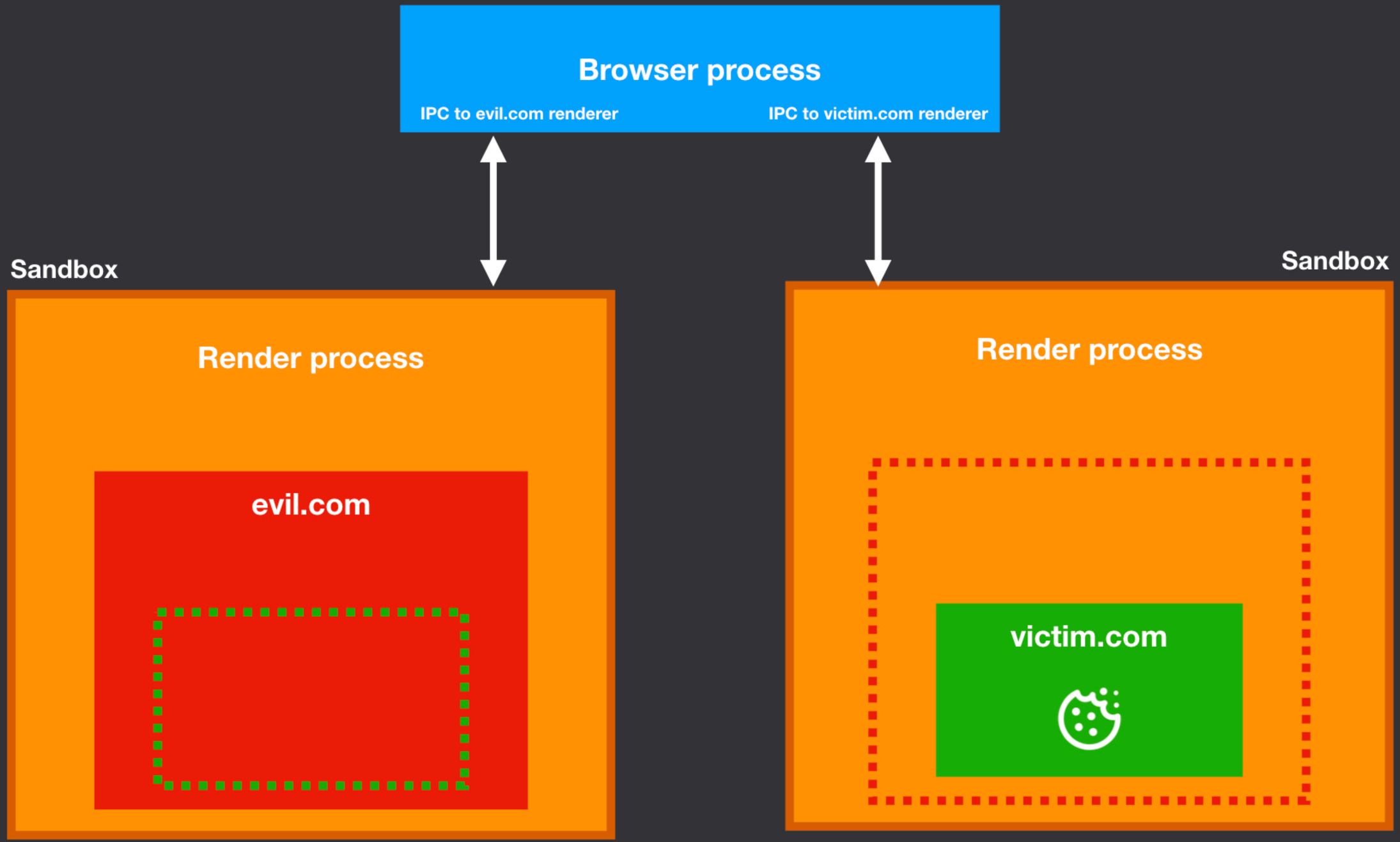

Browser process
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com

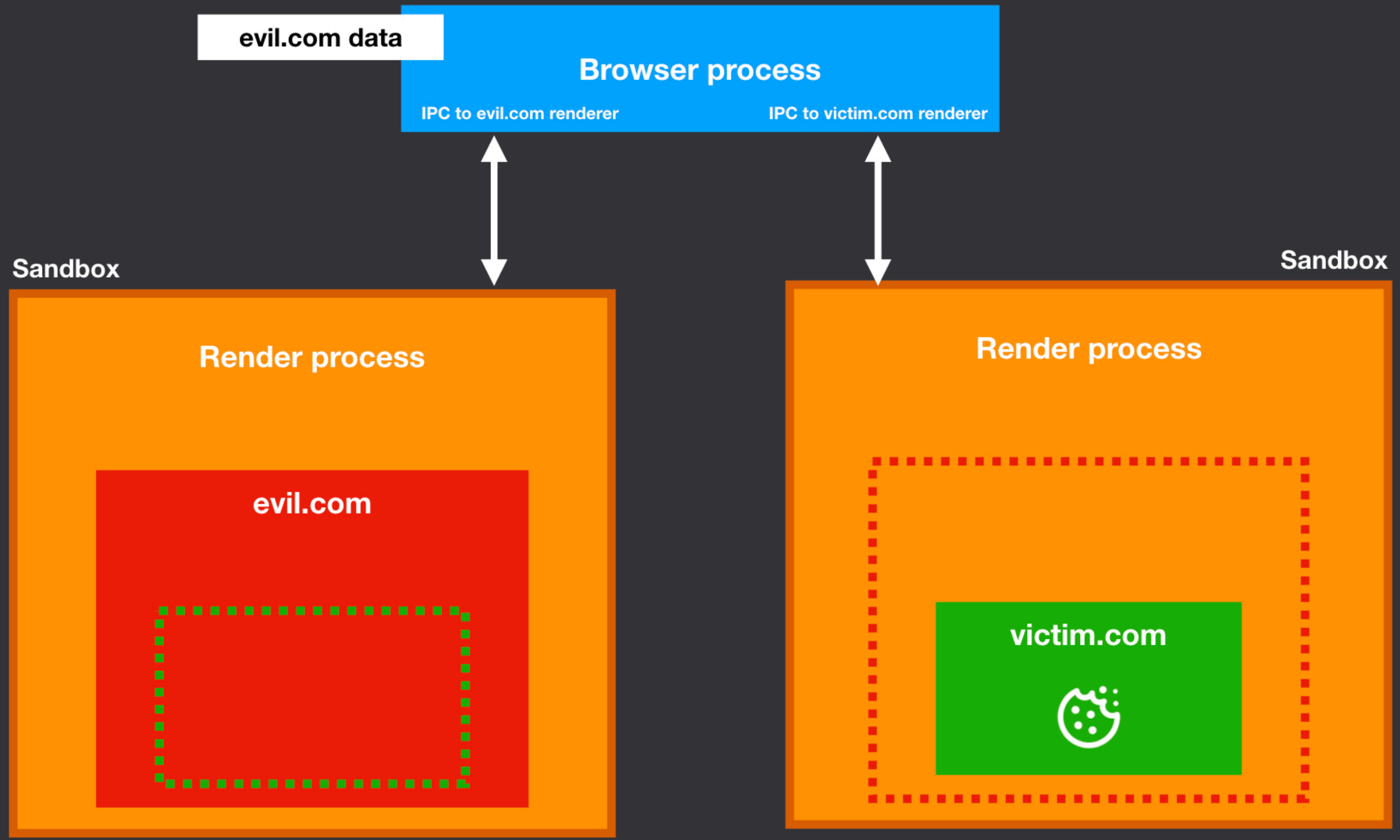

Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com

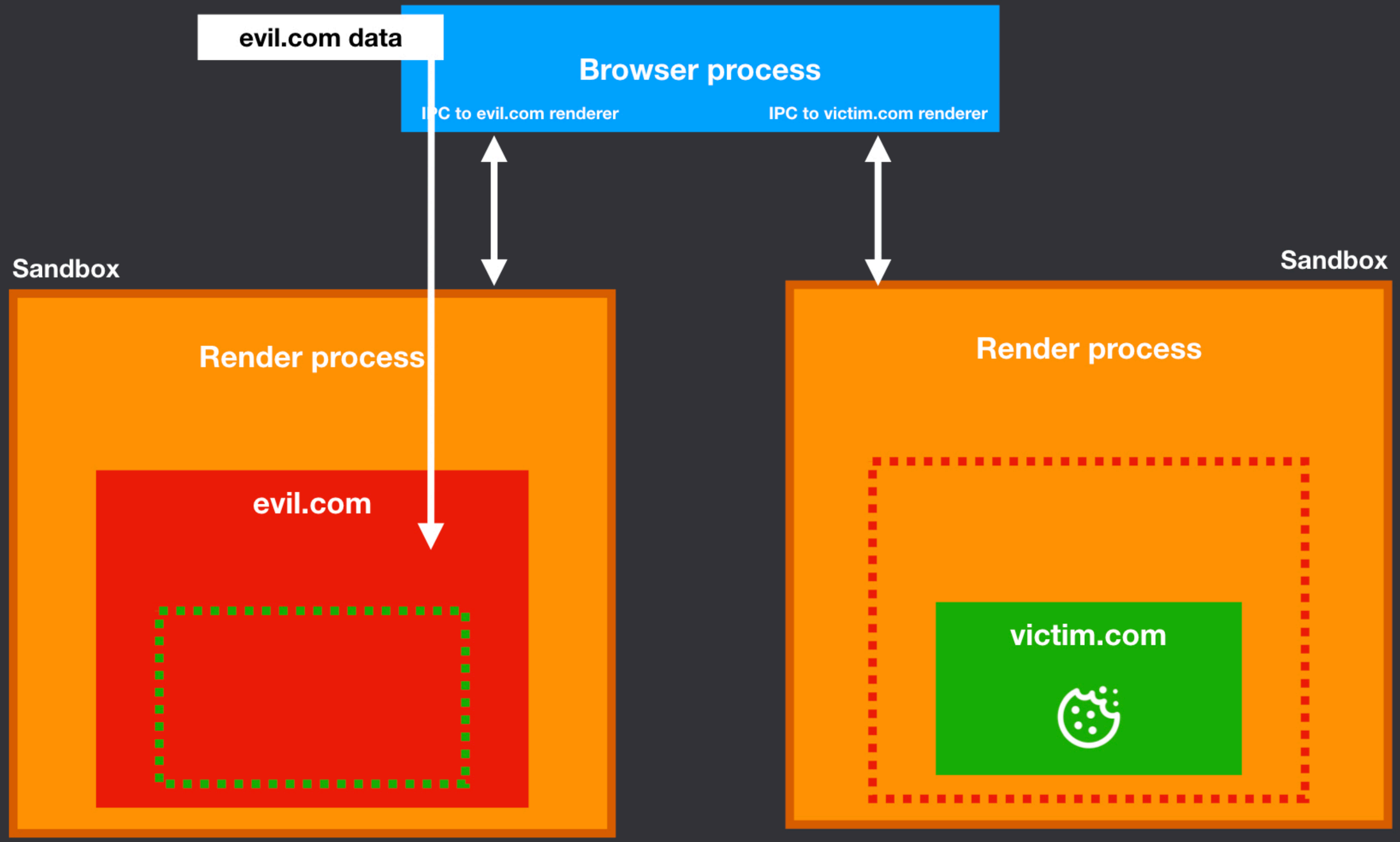

Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
I need cookies for victim.com
evil.com
Sandbox
Render process
victim.com

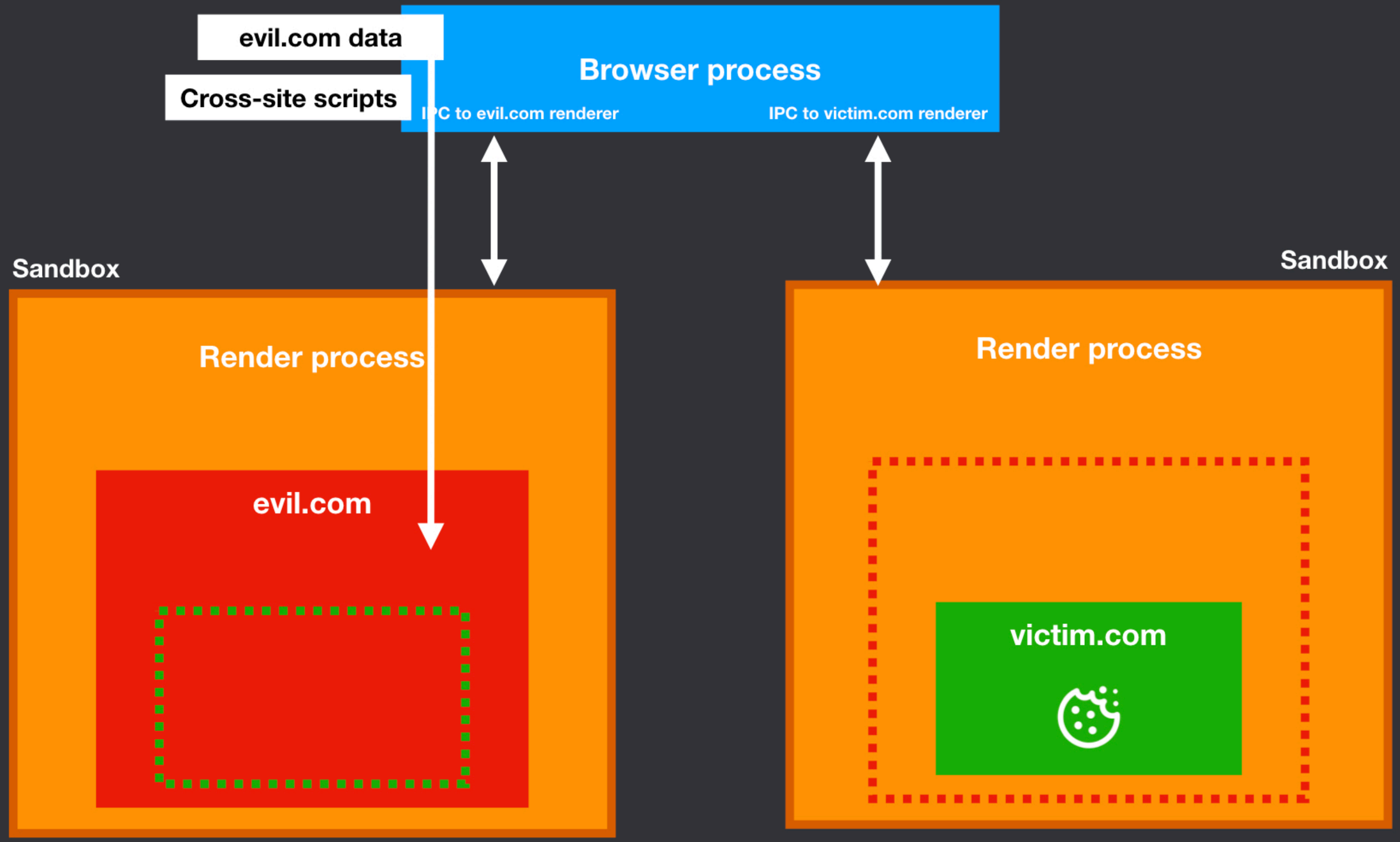

Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
I need cookies for victim.com
evil.com
Sandbox
Render process
victim.com

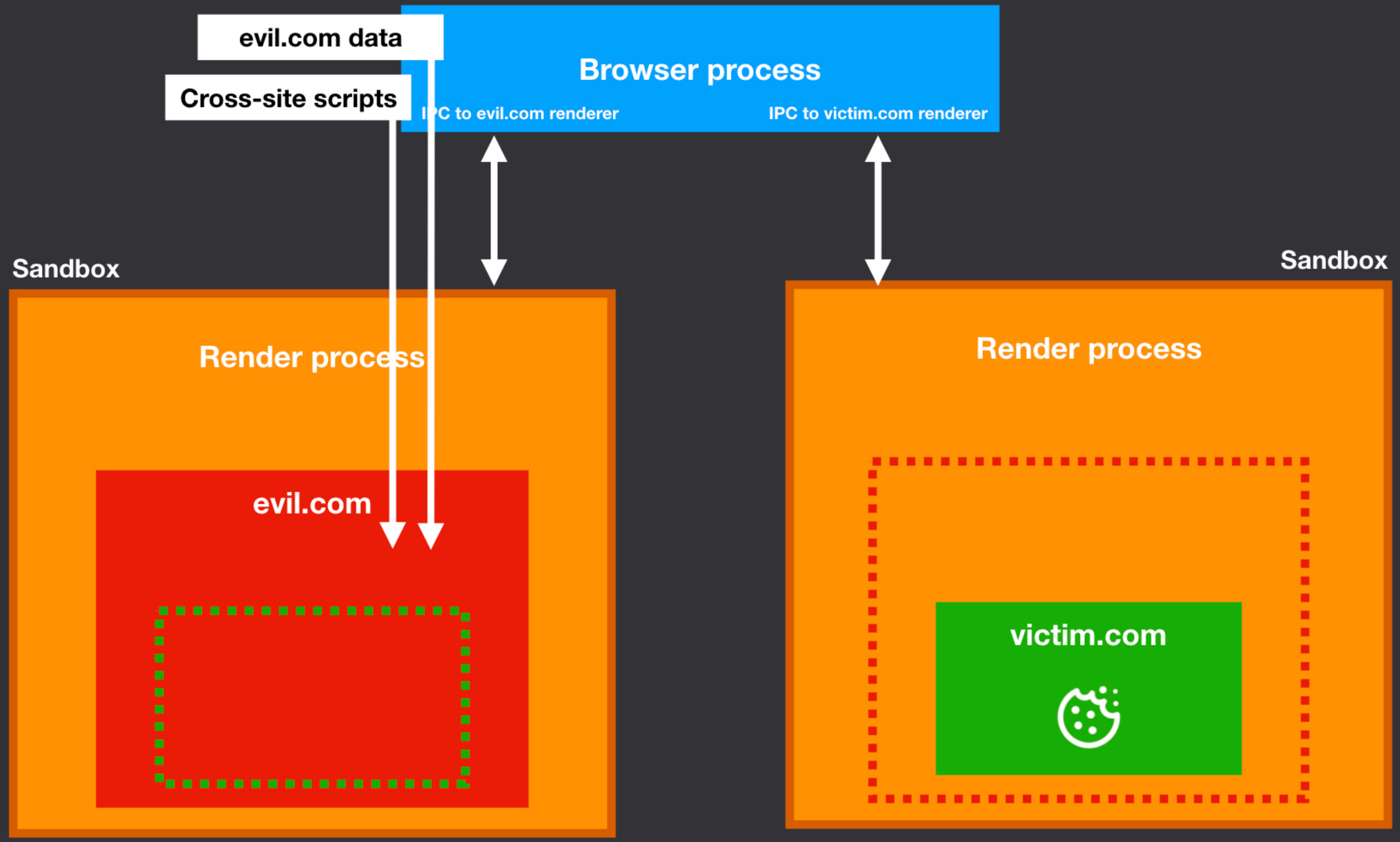

Browser process
IPC to evi m renderer
IPC to victim.com renderer
Sandbox
Render process
I need cookies for victim.com
evil.com
Sandbox
Render process
victim.com

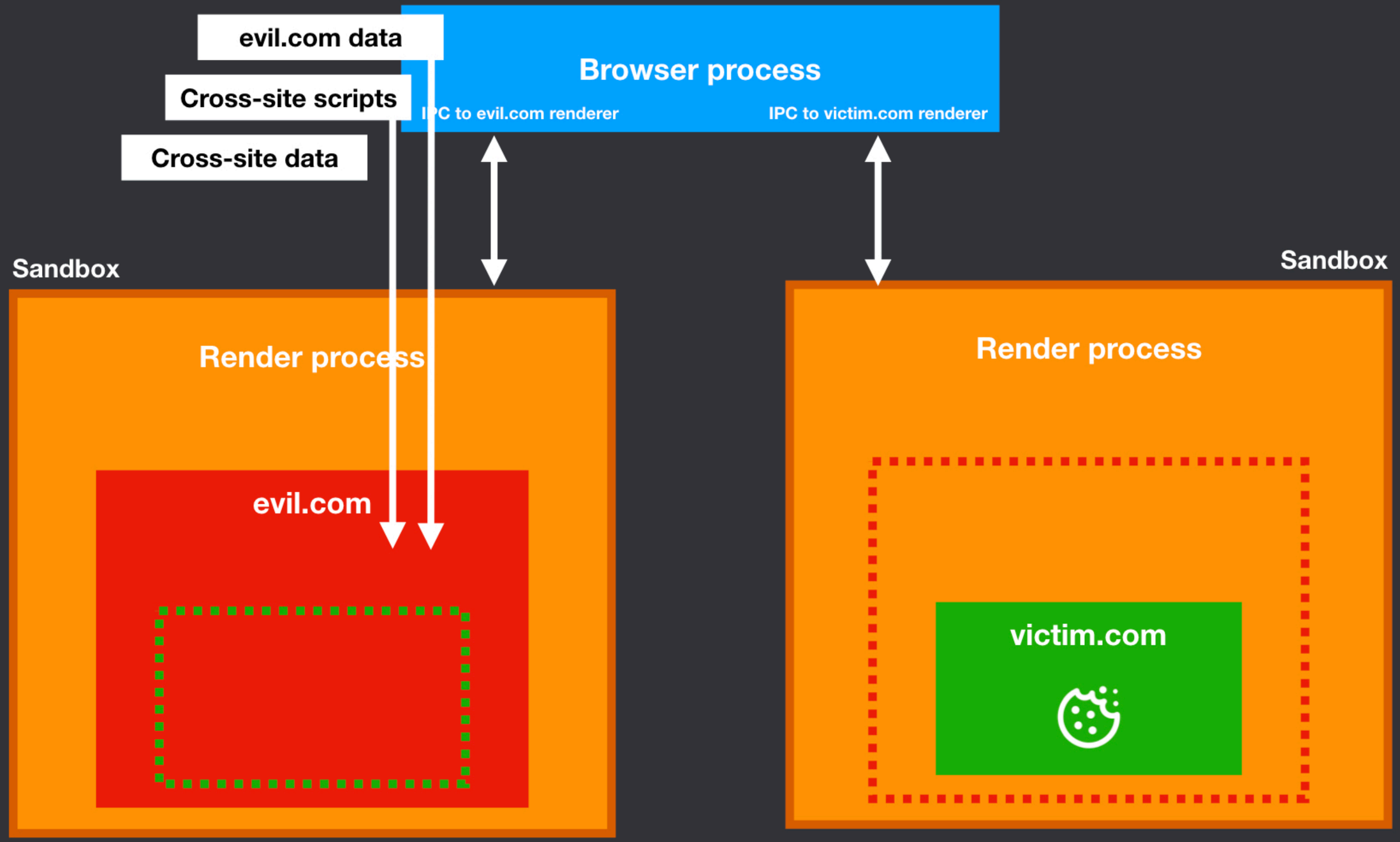

Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com

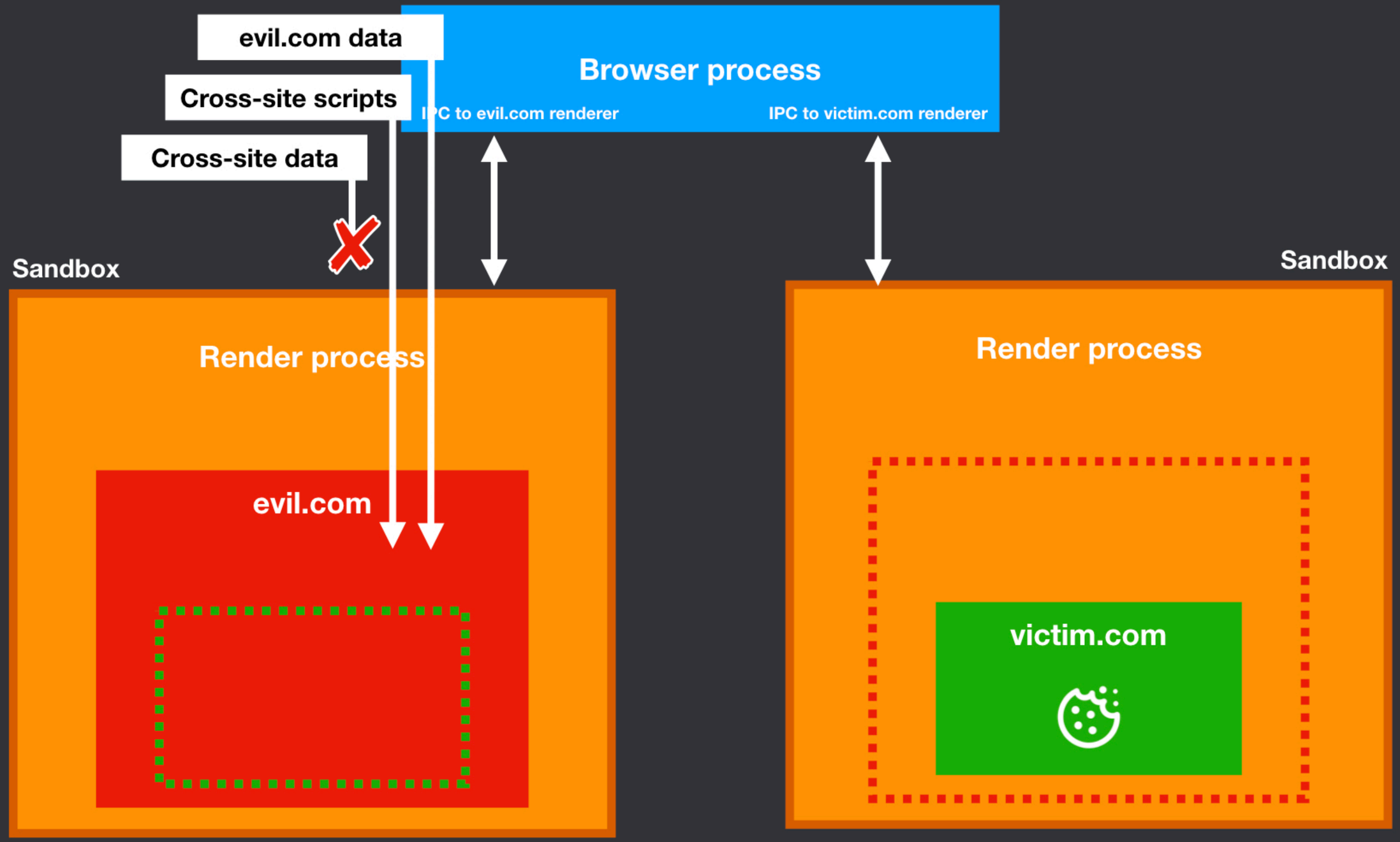

evil.com data
Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com

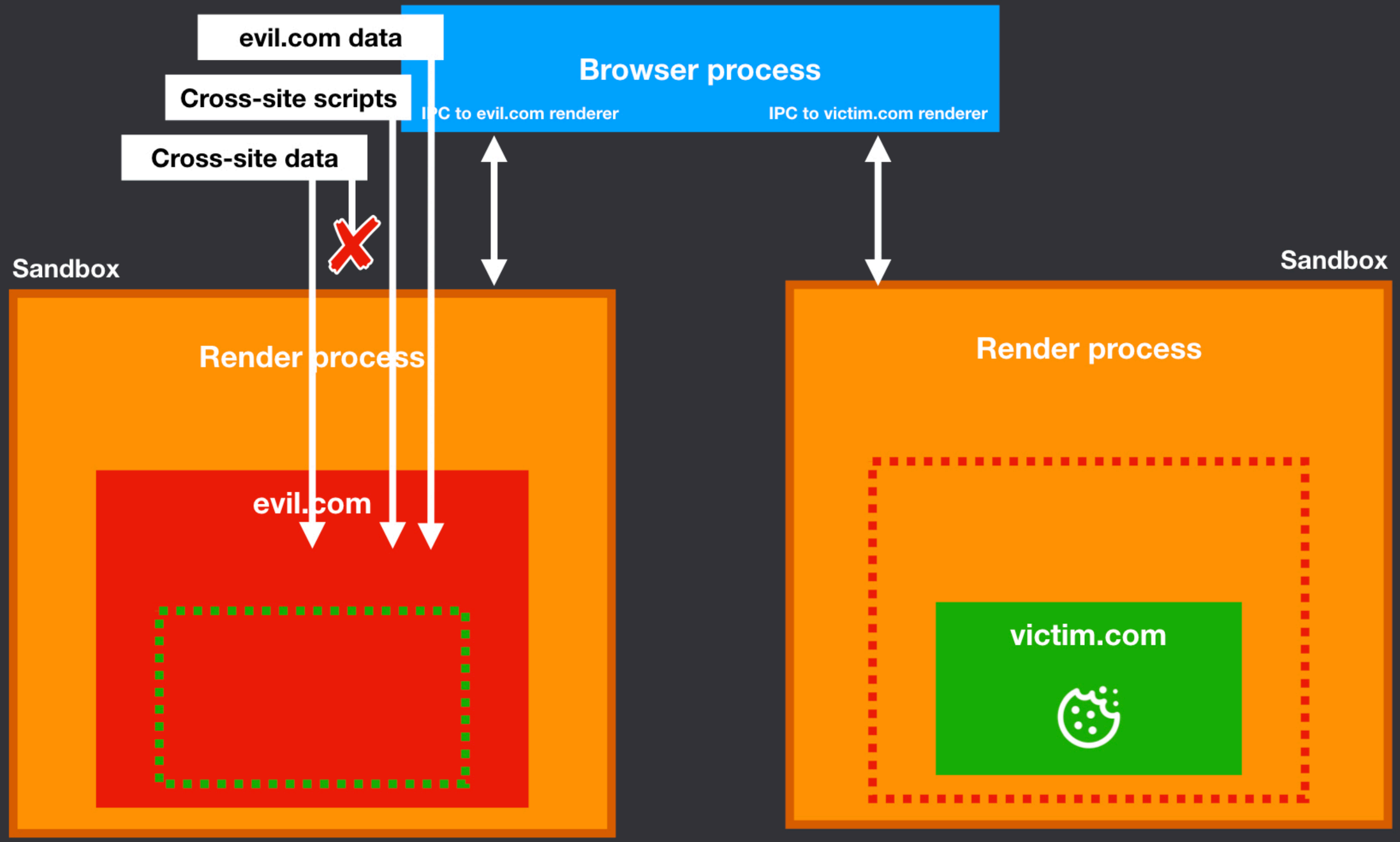

evil.com data
Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com


evil.com data
Cross-site scripts
Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com


evil.com data
Cross-site scripts
Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com


evil.com data
Cross-site scripts
Cross-site data
Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com


evil.com data
Cross-site scripts
Cross-site data
Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com


evil.com data
Cross-site scripts
Cross-site data
Browser process
IPC to evil.com renderer
IPC to victim.com renderer
Sandbox
Render process
evil.com
Sandbox
Render process
victim.com

Cross Origin Read Blocking (CORB)

- To ensure sensitive resources (e.g. pages or JSON files with user-specific information, or pages with CSRF tokens) will not leak to other origins, take the following steps to distinguish them from resources that are allowed to be embedded by any site (e.g., images, JavaScript libraries):
- Served resources with a correct **`Content-Type`** response header as well as a **`X-Content-Type-Options: nosniff`** response header:
  - HTML: text/html
  - XML: text/xml, application/xml, *+xml
  - JSON: text/json, application/json, *+json

Content sniffing ...good?

- Chrome will also do its best to protect responses labeled with a HTML, XML, or JSON content type and without a "nosniff" header, but this has limitations.
- Many JavaScript files on the web are unfortunately labeled using some of these content types, and if Chrome blocked access to them, existing websites would break.
- Thus, when the "nosniff" header is not present, Chrome first looks at the start of the file to try to confirm whether it is HTML, XML, or JSON, before deciding whether to protect it.
- If it cannot confirm this, it allows the response to be received by the cross-site page's process.
- This is a best-effort approach which adds some limited protection while preserving compatibility with existing sites

Cross Origin Read Blocking (CORB)

evil.com

```
fetch('https://evil.com')

<img src='https://victim.com/image.jpg'>

<img src='https://victim.com/secret.html'>
```

evil.com

```
fetch('https://evil.com')

<img src='https://victim.com/image.jpg'>

<img src='https://victim.com/secret.html'>
```

evil.com

```
fetch('https://evil.com')

<img src='https://victim.com/image.jpg'>

<img src='https://victim.com/secret.html'>
```

evil.com data


evil.com
fetch('https://evil.com')
<img src='https://victim.com/image.jpg'>
<img src='https://victim.com/secret.html'>
evil.com data


evil.com
fetch('https://evil.com')
<img src='https://victim.com/image.jpg'>
<img src='https://victim.com/secret.html'>
evil.com data
Cross-site images or scripts


evil.com
fetch('https://evil.com')
<img src='https://victim.com/image.jpg'>
<img src='https://victim.com/secret.html'>
evil.com data
Cross-site images or scripts


evil.com
fetch('https://evil.com')
<img src='https://victim.com/image.jpg'>
<img src='https://victim.com/secret.html'>
evil.com data
Cross-site images or scripts
Cross-site HTML, XML, or JSON


evil.com
fetch('https://evil.com')
<img src='https://victim.com/image.jpg'>
<img src='https://victim.com/secret.html'>
evil.com data
Cross-site images or scripts
Cross-site HTML, XML, or JSON

Blocked if:

- Content-Type is HTML and "nosniff" header present
- Content-Type is HTML and content sniffs as HTML

Allowed if:

- Content-Type is HTML and content sniffs as non-HTML

Secure coding practices

JavaScript has rough edges

Problem: == uses Abstract Equality Comparison Algorithm

```
function isZero (arg) {
  return arg == 0
}
```

Problem: == uses Abstract Equality Comparison Algorithm

```
function isZero (arg) {
  return arg == 0
}

isZero(0) // true
isZero('0') // true
isZero(false) // true
```

Solution: Always use ===

```
function isZero (arg) {
  return arg === 0
}
```

Solution: Always use ===

```
function isZero (arg) {
  return arg === 0
}

isZero(0) // true
isZero('0') // false
isZero(false) // false
```

Problem: Duplicate function arguments are allowed

```
function foo (a, b, a) {
  console.log(`value of the second a is ${a}`)
}
```

Solution: Use strict mode or a linter

```
'use strict'
function foo (a, b, a) { // SyntaxError: Duplicate parameter
                         // name not allowed in this context
  console.log(`value of the second a is ${a}`)
}
```

Use strict mode

- Opt in to a restricted variant of JavaScript
- Not merely a subset – strict mode intentionally has different semantics from normal code
  - Eliminates some silent errors by changing them to throw errors
  - Fixes mistakes that make it difficult for JavaScript engines to perform optimizations

Use a linter

- ESLint is the most popular JavaScript linter
  - Comes with hundreds of configurable rules and a plugin system
- StandardJS provides a pre-defined ESLint configuration
  - Add a linter to your JavaScript project quickly

```
$ standard
Error: Use JavaScript Standard Style
  lib/torrent.js:950:11: Expected '===' and instead saw '=='.

$ standard --fix
```

Problem: Duplicate keys in object literals are allowed

```
let foo = {
  bar: 'baz',
  bar: 'qux'
}
```

Solution: Use a linter

```
let foo = {
  bar: 'baz',
  bar: 'qux'
}
```

```
$ standard
standard: Use JavaScript Standard Style
  /Users/feross/websec/script.js:5:3: Duplicate key 'bar'.
```

- Not prevented by strict mode because runtime error is still possible with computed properties

```
let foo = {
  bar: 'baz',
  [someVariable] : 'qux'
}
```

Problem: Using Object.prototype builtins directly

```
const obj = { foo: 1, bar: 2, hasOwnProperty: 3 }
obj.hasOwnProperty('bar') // TypeError: obj.hasOwnProperty is not a function

const obj = Object.create(null)
obj.hasOwnProperty('bar') // TypeError: obj.hasOwnProperty is not a function
```

Solution: Use a linter

```
const obj = { foo: 1, bar: 2, hasOwnProperty: 3 }
obj.hasOwnProperty('bar') // TypeError: obj.hasOwnProperty is not a function

const obj2 = { foo: 1, bar: 2, hasOwnProperty: 3 }
Object.prototype.hasOwnProperty.call(obj2, 'bar') // true
```

```
standard: Use JavaScript Standard Style
  /Users/feross/websec/script.js:2:5: Do not access Object.prototype method 'hasOwnProperty' from target object.
```

Problem: Automatic semicolon insertion

```
let foo = bar
(1 || 2).baz()

let hello = 'world'
[1, 2, 3].forEach(addNumber)

let x = foo
/regex/g.test(bar)

function foo () {
  return
    {
      apples: 1,
      bananas: 2
    }
}
```

Problem: Automatic semicolon insertion

```
let foo = bar
(1 || 2).baz()

let hello = 'world'
[1, 2, 3].forEach(addNumber)

let x = foo
/regex/g.test(bar)

function foo () {
  return
    {
      apples: 1,
      bananas: 2
    }
}
```

```
let foo = bar;
(1 || 2).baz();

let hello = 'world';
[1, 2, 3].forEach(addNumber);

let x = foo;
/regex/g.test(bar);

function foo () {
  return
    {
      apples: 1,
      bananas: 2
    };
}
```

Problem: Automatic semicolon insertion

```
let foo = bar
(1 || 2).baz()

let hello = 'world'
[1, 2, 3].forEach(addNumber)

let x = foo
/regex/g.test(bar)

function foo () {
  return
    {
      apples: 1
    }
}
```

```
let foo = bar;
(1 || 2).baz();

let hello = 'world';
[1, 2, 3].forEach(addNumber);

let x = foo;
/regex/g.test(bar);

function foo () {
  return
    {
      apples: 1
    };
}
```

```
// Leading semicolon
let foo = bar
;(1 || 2).baz()

// Much better (boring code)
let num = 1 || 2
num.baz()

let nums = [1, 2, 3]
nums.forEach(addNumber)

let result = /regex/g.test(bar)

function foo () {
  return {
    apples: 1
  }
}
```

Solution: Use a linter

```
function foo () {
  return
    {
      apples: 1
    }
}

function foo2 () {
  return
    {
      apples: 1
    };
}
```

```
$ standard
standard: Use JavaScript Standard Style
  /Users/feross/websec/script.js:3:5: Unreachable code
  /Users/feross/websec/script.js:10:5: Unreachable code
```

Solution: Use a linter

```
let hello = 'world'
[1, 2, 3].forEach(addNumber)
```

```
$ standard
standard: Use JavaScript Standard Style
  /Users/feross/websec/script.js:2:1: Unexpected newline between object and [ of property access.
```

Problem: Leading zero in numbers is treated as octal

```
let num = 010 // 8
let num = 011 // 9
let num = 099 // 99
```

Solution: Use strict mode or linter

```
let num = 010
```

```
$ standard
standard: Use JavaScript Standard Style
  /Users/feross/websec/script.js:1:11: Parsing error: Invalid number
```

Problem: Variables default to global without `var`, `let`, or `const`

```
function foo () {
   x = 5
   y = 10
   return x + y
}

foo()
console.log(`${x} ${y}`) // Prints '5 10'
```

Solution: Use strict mode or a linter

```
'use strict'
function foo () {
  x = 5
  y = 10
  return x + y
}

foo() // ReferenceError: x is not defined
console.log(`${x} ${y}`)
```

Problem: Assigning to non-writable globals fails silently

```
undefined = true
console.log('hey') // Prints 'hey'
```

Solution: Use strict mode or a linter

```
undefined = true // TypeError: Cannot assign to read only property
                 // 'undefined' of object '#<Object>'
```

Problem: Setting properties on primitive values fails silently

```
true.foo = 42
;(0).foo = 42
```

Solution: Use strict mode or a linter

```
'use strict'
true.foo = 42 // TypeError: Cannot create property 'foo' on boolean 'true'
;(0).foo = 42
```

You need a linter!

```
npm install standard
standard

npm install eslint
eslint --init # Follow the wizard to create a configuration
```

Don't be too clever for your own good!

- Code is for humans, not for computers

Fancy logic operator short-circuiting

```
const foo = true
const bar = () => {
  console.log('Hey!')
}

foo && bar() // Prints 'Hey!'
```

Fancy logic operator short-circuiting

```
const foo = true
const bar = () => {
  console.log('Hey!')
}

foo && bar()
```

```
const foo = true
const bar = () => {
  console.log('Hey!')
}

if (foo) bar()
```

Prefer readability over brevity

```
const obj = {
   ...condition && { prop: value }
}
```

Prefer readability over brevity

```
const obj = {
  ...(condition && { prop: value })
}
```

Prefer readability over brevity

```
const obj = {
  ...(condition && { prop: value })
}
```

```
const obj = {
  ...false
}
```

Prefer readability over brevity

```
const obj = {
  ...(condition && { prop: value })
}

const obj = {
  ...false
}

const obj = {
  ...{ prop: value }
}
```

Prefer readability over brevity

```
const obj = {
  ...condition &&
    { prop: value }
}
```

```
const obj = {}
if (condition) {
  obj[prop] = value
}
```

Don't write clever code

```
// Programmer is flexing
usernames.map(Function.prototype.call, String.prototype.trim)

// Programmer is self-confident as evidenced by clear, simple code
usernames.map(str => str.trim())
```

Don't be overly clever

- Readability is more important than brevity
- Don't write code which requires the reader to understand esoteric language features or obscure edge cases
- Just like in writing, using flowery words where simpler ones would suffice may confuse or alienate readers

Untested code is broken code

Open source ecosystem

Open source supply chain risk

- Accidents
- Lack of resources (undermaintained and unmaintained packages)
- Malice

```
349  349    rm -rf /etc/alternatives/xorg_extra_modules
350  350    rm -rf /etc/alternatives/xorg_extra_modules-bumblebee
```

underyx on Apr 15, 2016 • edited

Doing well so far, nice script 👍

👍 34 😄 5

Reply...

```
351    rm -rf /usr/lib/nvidia-current/xorg/xorg
```

Hezion on Jun 16, 2011

Great Success!!

👍 217 👎 6 😄 82 🎉 70 😕 10 ❤️ 25 🚀 10

nddrylliog on Jun 16, 2011

Hahahaha epic fail :D

0xd34df00d on Jun 16, 2011

Bleeding edge really bleeding for someone now!

😄 43 🎉 9

danielgtaylor on Jun 16, 2011

Oh man hope nobody got bit by this one - always let someone else test first! :-P

Feross Aboukhadijeh

BIZ & IT —

Tech giants, chastened by Heartbleed, finally agree to fund OpenSSL

IBM, Intel, Microsoft, Facebook, Google, and others pledge millions to open source.

JON BRODKIN - 4/24/2014, 5:00 AM

Aurich Lawson / Thinkstock

The important role OpenSSL plays in securing the Internet has never been matched by the financial resources devoted to maintaining it.

The open source cryptographic software library secures hundreds of thousands of Web servers and many products sold by multi-billion-dollar companies, but it operates on a shoestring

The plight of the maintainer

- **`OpenSSL`** - "OpenSSL typically receives about $2,000 in donations a year and has just one employee who works full time on the open source code."
- **`curl`** - "According to Stenberg, cURL is included in billions of smartphones, “several hundred million” TVs, and at least 100 million smart cars, every iPhone ever produced, and almost every other modern connected device and service you touch every day. The scale of its use is staggering considering that Stenberg does the lion’s share of the work maintaining it, with assistance from a community of volunteers"

statement on event-stream compromise

readme.md Raw

Hey everyone - this is not just a one off thing, there are likely to be *many* other modules in your dependency trees that are now a burden to their authors. I didn't create this code for altruistic motivations, I created it *for fun*. I was learning, and learning is fun. I gave it away because it was easy to do so, and because sharing helps learning too. I think most of the small modules on npm were created for reasons like this. However, that was a long time ago. I've since moved on from this module and moved on from that thing too and in the process of moving on from that as well. I've written way better modules than this, the internet just hasn't fully caught up.

@broros

> otherwise why would he hand over a popular package to a stranger?

If it's not fun anymore, you get literally nothing from maintaining a popular package.

One time, I was working as a dishwasher in a resturant, and I made the mistake of being too competent, and I got promoted to cook. This was only a 50 cents an hour pay rise, but massively more responsibility. It didn't really feel worth it. Writing a popular module like this is like that times a million, and the pay rise is zero.

I've shared publish rights with other people before. Of course, If I had realized they had a malicious intent I wouldn't have, but at the time it looked like someone who was actually trying to help me. Since the early days of node/npm, sharing commit access/publish rights, with other contributors was a widespread community practice. felixge.de/2013/03/11/the-pull-request-hack.html open source is driven by sharing! It's great! it worked really well before bitcoin got popular.

So right now, we are in a weird valley where you have a bunch of dependencies that are "maintained" by someone who's lost interest, or is even starting to burnout, *and that they no longer use themselves*. You can easily share the code, but no one wants to share the responsibility for maintaining that code. Like a module is like a piece of digital property, a right that can be transferred, but you don't get any benefit owning it, like being able to sell or rent it, however you still retain the responsibility.

I see two strong solutions to this problem...

1. Pay the maintainers!! Only depend on modules that you know are definitely maintained!
2. When you depend on something, you should take part in maintaining it.

Aug 1, 2019

Google and Mozilla are failing to support browser extension developers

It is a regular occurrence to hear about open source developers selling their browser extensions, only for their users to be exploited later on by the new owners.

Purchase offers for browser extensions usually range between $0.1 and $0.3 per user, depending on factors such as the geographical distribution of users, and monetization offers are also frequent.

Accepting such an offer may significantly improve one's life, and it can be potentially life-changing. It feels like a just reward for all those years of free labor, and it's uplifting for someone to value your work and propose to buy your project.

We are witnessing the failure of browser vendors to recognize the value of our labor and the important role it plays in a healthy browser ecosystem.

Mozilla has deprioritized the placement of the donation button during the redesign of Firefox Add-ons. The button was pushed below the fold, previously it was featured prominently near the install button. Requesting donations at the end of the install flow has also been deprecated.

The Chrome Web Store and the Microsoft Store do not offer features for supporting extension developers.

Browser vendors must recognize that developers who feel valued and are compensated by the communities they contribute to are less likely to give up on their projects and abandon their users.

Conclusion

CS 253 Key ideas

- ***Think like an attacker!***
- Never trust user input - *always sanitize it, at time of use*
- Use defense-in-depth – *provide redundancy in case security controls fail*
- Salt and hash user passwords – *'just use* **`bcrypt`***!'*
- Beware ambient authority – *use* **`SameSite`** *cookies!*
- Don't write clever code – *explicit code is safer than magical code*
- Dangerous code should look dangerous – *make it stand out*
- You can never be too paranoid – *practice constant vigilance*

What to take next?

- **Winter**
  - **CS 255** – Introduction to Cryptography
- **Spring**
  - **CS 155** – Computer and Network Security
  - **CS 355** – Topics in Cryptography
- **Next year**
  - **CS 251** – Cryptocurrencies and Blockchain Technologies (Fall)
  - **CS 190** – Software Design Studio (Winter)


THANK YOU


END

Credits:

https://unit42.paloaltonetworks.com/google-chrome-exploitation-case-study/

https://developers.google.com/web/updates/2018/09/inside-browser-part1

https://www.researchgate.net/figure/Modular-browser-architecture*fig1*224163004

https://feross.org/chrome/

https://chromium.googlesource.com/chromium/src/+/master/docs/security/rule-of-2.md