

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



CSRF | TryHackMe Walkthrough



CyferNest Sec · [Follow](#)

13 min read · Jan 3, 2025



Listen



Share



More



CSRF: The Art of Sneaky Online Mischief

Welcome to the wacky world of web hacking, where even beginners — yes, the so-called *script kiddies* — can wreak havoc with a few lines of code. It's like giving toddlers access to paint but on the internet... and the “paint” can shut down businesses!

Today, we're zooming in on a particularly devious trick called **Cross-Site Request Forgery (CSRF or XSRF)**. Think of it as convincing someone to send money from their bank account while they think they're just clicking on a cat meme.

In this blog, we'll uncover how **CSRF** works, the crafty techniques hackers use to exploit it, and — most importantly — how you can slam the brakes on this digital mischief. Buckle up, because things are about to get sneakily educational! 🐱

TASK 2: Overview of CSRF Attack

Imagine someone using your credit card while you're blissfully clicking on a meme. That's **Cross-Site Request Forgery (CSRF)** in a nutshell! It's a security vulnerability where attackers trick your browser into performing unwanted actions on trusted websites where you're logged in — like transferring money or changing passwords — all without you realizing it. Thanks, browser cookies.

The 3-Act Drama of a CSRF Attack

1. **The Setup:** The attacker learns how a web application processes requests and crafts a malicious link.
2. **The Click:** You, the unsuspecting victim, click the link, and your browser (helpful as always) sends your credentials.
3. **The Grand Finale:** The web application can't tell the difference between you and the attacker, so it obliges, completing the attacker's dirty work.

Why CSRF is a Big Deal

Though it doesn't directly steal your data, **CSRF** can still cause major headaches, like:

- **Unauthorized Access:** From draining your bank account to changing your email, attackers take the wheel.
- **Trust Issues:** It abuses the trust websites have in you, leaving your digital credibility in shambles.
- **Sneaky Shenanigans:** No malware here, just regular browser behavior working against you.

Protecting against **CSRF** is like installing a lock on your cookie jar — literally. Don't let this silent trickster turn your online life into a comedy of errors. Stay safe, stay smart, and keep those cookies protected! 🍪 🛡️

Question: Which of the following is a possible effect of CSRF? Write the correct option only.

- a. Unauthorised Access
- b. Exploiting Trust
- c. Stealthy Exploitation
- d. All of the above

Answer: d

Question: Does the attacker usually know the web application requests and response format while launching a CSRF attack (yea/nay)?

Answer: yea

TASK 3: Types of CSRF Attack

CSRF attacks come in different flavors, but they all have one goal: to make your browser do the attacker's bidding without your knowledge. Let's explore the most popular versions of these digital pranks.

Traditional CSRF: The Classic Con

In its old-school form, **CSRF** focuses on tricking victims into submitting forms unknowingly. Think: transferring money or updating account details — all while sipping coffee, unaware.

How It Works:

1. You're logged into your bank.
2. An attacker sends you a malicious email.
3. You click the link, and boom — your money is en route to a stranger's account without your consent.

Visualize This:

Victim logs in  → Malicious link arrives  → Click → Auto-transfer initiated 

XMLHttpRequest CSRF: The Sneaky Asynchronous Trick

Modern web apps love **AJAX** (asynchronous JavaScript and XML) for dynamic updates, and attackers love exploiting it. This method doesn't reload pages but still sends those pesky unauthorized requests.

Example:

Imagine you're adjusting your email settings. A **CSRF** vulnerability could allow a hacker to secretly alter your email forwarding preferences, sending all your emails to their inbox.

How It Works:

1. You log into your mailbox.
2. The attacker tricks you into visiting their webpage with a malicious script.
3. The script uses **AJAX** to send a request to change your settings, cookies and all.

Flash-based CSRF: The Relic of the Past

Once upon a time, **Adobe Flash** ruled the interactive web. It also ruled the nightmares of cybersecurity experts with its vulnerabilities. **Flash-based CSRF attacks** took advantage of insecure **Flash** components to send unauthorized requests.

Why It Matters:

Flash may have gone extinct in 2020, but legacy systems still relying on it remain vulnerable. Hackers used malicious Flash files to carry out **CSRF** attacks, proving even animated content wasn't safe.

The Takeaway:

From sneaky form submissions to **stealthy AJAX** calls and outdated **Flash** tricks, **CSRF** attacks adapt to the times. The lesson? Stay vigilant, secure your applications, and remember — if it looks too good to be true, it probably comes with a hidden script. 🕵️💻

Question: What is usually the extension of a malicious flash file used during a **CSRF** attack?

Answer: .swf

Question: Which type of CSRF exploitation is carried out when operations are initiated without a complete page request-response cycle?

Answer: Asynchronous

TASK 4: Basic CSRF — Hidden Link/Image Exploitation

Sometimes, the smallest things cause the biggest problems — like a 0x0 pixel image quietly plotting chaos in your browser. This sneaky CSRF technique leverages hidden links or images to execute unauthorized actions on behalf of unsuspecting users, taking full advantage of browsers automatically sending cookies.

How It Works: A Tale of Josh and His Browser Woes

Meet Josh: A busy financial broker juggling emails and banking tabs, blissfully logged into his bank account. Enter the attacker, a crafty villain who:

1. **Learns Josh's Bank URL:** After poking around the same bank platform, the attacker finds a lack of validation on transactions.
2. **Crafts a Sneaky Email:** The email includes a seemingly harmless link with hidden intent:

A screenshot of a code editor with a light gray background. It shows a line of HTML code: `<a href="http://mybank.thm:8080/dashboard.php?to_account=GB82MYBANK5698&amount=`. The text is in a monospaced font, with the opening tag in black and the rest in red. A horizontal scrollbar is visible below the text.

```
<a href="http://mybank.thm:8080/dashboard.php?to_account=GB82MYBANK5698&amount=
```

3. **Baits the Trap:** Josh, lured by the promise of a Dubai trip, clicks the link, triggering an automatic funds transfer without his knowledge.

The Missing Piece: CSRF Token

The attack succeeds because the bank fails to validate requests. Specifically, it doesn't confirm whether the request originates from a legitimate user action.

The Fix: Fortifying Against CSRF

1. **Add CSRF Tokens:**

Unique tokens are included with every form or request, ensuring that only valid, user-initiated actions are processed.

Updated example:

```
<input type="hidden" id="csrf_token" name="csrf_token" value="<?php echo $_COOKIE
```

2. Server Validation:

The server checks every request for a **valid CSRF token**. If absent or incorrect, the request is rejected faster than a spam email.

Josh's Redemption

The bank's IT team patches the vulnerability, implementing **CSRF tokens**. Now, even if Josh clicks on another "Dubai trip" email, the malicious link fails because the required token is missing. His money — and trust in the system — are safe again.

Takeaway

Never underestimate the power of a **0x0 pixel**. For developers, implementing **CSRF** protection isn't just good practice; it's essential. For users like Josh, if it smells like spam, don't click the link — even if it promises paradise. 🌴 💻

Question: What is the flag value after a successful transfer from Josh's account?

Answer: THM{SUCCESSFUL_ATTACK}

Question: What is the flag value once the CSRF attack is detected?

Answer: THM{INVALID_CSRF_TOKEN}

Question: Does the hidden image exploitation require the `img` tag's `src` attribute to be linked toward a legitimate image file (yea/nay)?

Answer: nay

TASK 5: Double Submit Cookie Bypass

Let's dive into how a crafty hacker almost outsmarted Josh's bank with a mix of code wizardry, social engineering, and a sprinkle of greed. Spoiler: the story ends with a lesson in good token hygiene.

The Hero: CSRF Tokens

Think of **CSRF tokens** as the bouncers of the web world. Every time you want to "do the thing" (like transfer money), these **tokens** make sure it's actually you requesting it and not some sneaky outsider. They're unique, unpredictable, and, when implemented correctly, the ultimate gatekeepers.

The Plot Twist: Double Submit Cookies

Here's where things get interesting. The **Double Submit Cookies** method makes your browser send two versions of the **CSRF token** — *one in a cookie and another hidden in the form*. The server plays detective and matches these up. If they align, the party continues; if not, the request gets kicked out faster than a fake ID.

The Villain's Playbook: Hacking the System

Enter the hacker, a wannabe genius who found loopholes like a pro:

1. **Session Cookie Hijacking** — Think "Man in the Middle" meets "Spy Kids."
2. **XSS Shenanigans** — Use **Cross-Site Scripting** to nab tokens and wreak havoc.
3. **Token Guessing** — Predictable tokens are basically leaving the bank vault open.

The Big Heist

Our villain didn't stop at small-scale theft. Armed with a **decrypted CSRF token** (seriously, Josh's bank, why use an account number as a **token**?!), the hacker went for gold: Josh's password.

By crafting a fake "Suspicious Login" email that could make anyone panic, the hacker lured Josh into clicking a link to an attacker-controlled page. The fake page looked like the real deal and automatically submitted a password change request to the actual bank. The result? A successful password heist... or so the hacker thought.

The Redemption Arc

The bank's IT team finally stepped up their game. They ditched their predictable **token-generation** methods, implemented super-random **tokens**, and patched up vulnerabilities. Josh's bank account was secured, and the hacker was left empty-handed, probably sulking in a dark basement somewhere.

Moral of the Story

1. **Developers:** Stop being lazy with your **CSRF tokens**. Make them strong and unpredictable!
2. **Users:** Don't click on sketchy links, even if they promise you free donuts.
3. **Hackers:** Just... stop. Go learn **ethical hacking**. It's way cooler and legal.

With proper **CSRF token implementation**, Josh's bank now has the last laugh. Security, like comedy, is all about timing — and the bank finally nailed it. 🙌

Question: What is the decoded value of the CSRF token for Josh's account?

Answer: GB82MYBANK56997

Question: What is the updated password for Josh's account?

Answer: GB82MYBANK56997

Question: What is the flag value if someone clicks on malicious links after the IT team of MyBank has successfully employed a random token generation algorithm?

Answer: THM{SECURED_CSRF}

Question: What is the hidden field name that the MyBank IT team added to avoid CSRF attacks?

Answer: csrf_token

Question: Is it technically possible to hijack a session cookie in a web application (yea/nay)?

Answer: yea

TASK 6: Samesite Cookie Bypass

Ever wondered how cookies decide whether to play nice or lock down tight? Enter **SameSite cookies**, the guardians of your browser's snack jar. They come with rules — **strict**, **lax**, or **free-for-all (None)** — that decide when and where they're shared.

These rules are here to fend off mischief like **cross-origin** data leaks, **CSRF**, and **XSS** attacks. Let's meet the squad:

The Cookie Squad

1. **Strict:** The overprotective parent. **Cookies** stay *home* — only sent if the request originates from the **same site**. It's ultra-secure but might ruin the **cookie's** social life.
2. **Lax:** The cool neighbour. Allows **cookies** to mingle during safe top-level visits, like **GET** requests. But don't expect it to party with **POST** requests.
3. **None:** The globetrotter. These **cookies** are sent everywhere, but only if the connection's **HTTPS-secured**. Safety first, even for adventurers!

The Lax Cookie Mishap: A Comedy of Errors 🧑

Meet Josh. One day, he innocently logs into his bank. Enter our villain: an attacker armed with a lax **SameSite** cookie and a bogus survey email titled, *"Win a Ferrari!"*



The attacker's plan? Log Josh out of his account and lock him out faster than you can say "phishing." Using a sneaky **GET request** linked to the logout **cookie**, the attacker logs Josh out, leaving him sobbing on the sidelines. If only the cookie had been **Strict** — Josh's Ferrari dream might've stayed intact.

But Wait, There's More! The Lax+POST Exploit 🕵️

What happens when **cookies** get updated faster than you can refresh a TikTok feed? Chrome's "2-minute exception rule" for **cookies** without a **SameSite** attribute comes into play. Here's the twist:

- A mischievous **isBanned cookie** gets updated during login or logout.
- Within 2 minutes, any **POST** request can exploit this update.
- The attacker logs Josh out, waits for the **cookie** to update, and then sneakily modifies it via **POST**.
- Voilà! Josh gets banned. **No cookies**, no banking, and definitely no Ferrari. 🍪



Lesson Learned (The Hard Way) 📜

Dear developers, when coding cookies:

- **Set SameSite=Strict** unless you're feeling adventurous (or reckless).
- Don't leave your **cookies** unattended — they're sweeter when secure.

And for **pentesters**? Always analyze **cookies** like they owe you money. 🍪 ✨

Stay sharp, stay safe, and remember — your cookies are worth protecting!

Question: What is the logout cookie value?

Answer: 7kRt2x9LpQyW

Question: What is the flag value after successfully logging out Josh from the mybank.thm application?

Answer: THM{LOGGED_OUT}

Question: Once logged into the mybank.thm:8080 application, change the logout cookie value to hellothm, and try to click on the malicious link. What is the flag value after detecting a CSRF attack?

Answer: THM{ATTACK_DETECTED}

Question: After updating the isBanned cookie value to true through a CSRF attack, what is the flag value?

Answer: THM{USER_IS_B@NNED}

TASK 7: Few Additional Exploitation Techniques

XMLHttpRequest Exploitation and CSRF: The Sneaky Browser Hijack 🕵️

Imagine someone tricking your browser into playing puppet master on your favorite website, making changes to your account while you sit blissfully unaware. That's **Cross-Site Request Forgery (CSRF)** in a nutshell — a sneaky attack that even gets around the **Same-Origin Policy (SOP)**. Let's see how attackers use **AJAX** and **CORS mischief** to pull off these digital heists.

The AJAX Attack Playbook

Here's how an attacker might use **AJAX** to hijack your account:

1. **Target:** Password updates on mybank.thm.
2. **Weapon:** JavaScript's **XMLHttpRequest** object.
3. **Action:** Seamlessly sends a **POST** request to update your password *and* email.

The culprit code:

```
var xhr = new XMLHttpRequest();
xhr.open('POST', 'http://mybank.thm/updatepassword', true);
xhr.setRequestHeader("X-Requested-With", "XMLHttpRequest");
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xhr.onreadystatechange = function () {
  if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    alert("Action executed!");
  }
};
xhr.send('action=execute&parameter=value');
```

This stealth operation happens entirely in the background, and boom — your account is compromised without lifting a finger.

CORS and SOP Bypass: Loopholes Galore

While Same-Origin Policy (SOP) generally prevents cross-origin requests, CORS (Cross-Origin Resource Sharing) introduces exceptions. When misconfigured, these exceptions can become open doors for attackers.

Vulnerable CORS Configuration:

```
<?php
header('Access-Control-Allow-Origin: *');
// BAD: Accepts requests from *any* origin.
?>
```

Why is this bad?

- The **wildcard (*)** allows **any** website to send requests, even malicious ones.
- If combined with **Access-Control-Allow-Credentials:true** (which is forbidden by the CORS spec), it could expose sensitive data.

Referer Header Bypass: The Disappearing Trail 🕵️

Some websites rely on the **Referer** header to validate requests. It shows the **URL** of the last visited page, but attackers can:

- **Remove it** using browser extensions or privacy tools.
- **Modify it** to spoof legitimate origins.

Relying solely on the **Referer header for CSRF protection** is like *using a paper lock on your front door* — easy to bypass.

How to Stay CSRF-Free 🚫🕵️

Defend your site with these **key mitigation measures**:

1. **Anti-CSRF Tokens:** Add unique, unpredictable **tokens** to forms and validate them server-side.
2. **Verify Origin Headers:** Check both the **Origin** and **Referer** headers for incoming requests.
3. **CORS Best Practices:**
 - Never use **Access-Control-Allow-Origin: *** for authenticated requests.
 - Set **Access-Control-Allow-Credentials: true** only for trusted origins.
4. **SOP-Adhering AJAX:** Use strict rules for **cross-origin** requests and headers.
5. **Content Security Policy (CSP):** Prevent script injection and unauthorized resource loading.

Bottom Line 📜

CSRF is like a mischievous ghost in the machine — silent but dangerous. Arm your site with robust defenses, and you'll keep attackers from hijacking the browser and wrecking your users' trust. Let's send these digital pranksters packing! 🎯

Question: What is the general policy name that forbids cross-origin requests?

Answer: same-origin policy

Question: Strictly speaking, while updating an email address of Josh through the API endpoint, is Access-Control-Allow-Origin: * a vulnerable header (yea/nay)?

Answer: yea

Question: What is the header called that stores the URL of the last page the user visited before making a request?

Answer: referer

TASK 8: Defence Mechanisms

Cross-Site Request Forgery (CSRF) is a vital battlefield for *pentesters* and a challenge for *secure coders*. For *pentesters*, it's a way to *identify and exploit vulnerabilities, testing applications to their limits*. For *developers*, it's a *call to arms to fortify defenses and keep attackers at bay*. Let's dive into the dual roles:

For Pentesters and Red Teamers

CSRF Testing: The Hunt Begins

- Actively poke around applications for **CSRF** vulnerabilities.
- Attempt sneaky unauthorized actions to see if the app shrugs it off or crumbles.
- **Example:** *Can you make the app send your pet goldfish \$100? If yes, jackpot (for testing, of course)!*

Boundary Validation: Guard the Gates

- Check if user inputs are validated like a strict airport security line.
- Look for those magical **anti-CSRF tokens** — *are they present, validated, and as unpredictable as a cat's mood?*

Security Headers Analysis: Helmet Check

- Assess **headers** like **CORS** and **Referer** — **are they holding strong or as flimsy as paper?**

- Weak headers = Open doors.

Session Management Testing: Token Treasure Hunt 💎

- Examine how **session tokens** are generated, transmitted, and guarded.
- Ensure **tokens** aren't predictable or exposed like a bad poker hand.

Exploitation Scenarios: Getting Creative 🎨

- Test embedding malicious requests in sneaky places like image tags or trusted endpoints.
- **Example:** *An image that secretly logs out the user — art or attack?*

For Secure Coders 🔒

Anti-CSRF Tokens: The Magic Shield 🛡️

- Add unique, unpredictable **anti-CSRF tokens** to every form or request.
- No token? No entry — *simple as that*.

SameSite Cookie Attribute: Cookie Bodyguards 🍪

- Use **SameSite=Strict** or **SameSite=Lax** to control **cookie** behavior.
- Prevent **cookies** from wandering off to shady parties (**cross-site requests**).

Referrer Policy: Who's Calling? 📞

- Implement a strict **Referrer Policy** to ensure requests are legit and from trusted sources.

Content Security Policy (CSP): Script Police 🚔

- Define trusted content sources with **CSP** to block malicious scripts from joining the party.

Double-Submit Cookie Pattern: Double Trouble for Hackers 🙌

- Store an **anti-CSRF token** in a **cookie** and send it as a request parameter.
- The server compares the two, and if they don't match, it's a hard no.

Implement CAPTCHAs: Are You Human? 🤖

- Add CAPTCHA challenges to sensitive actions like logging in, creating accounts, or sending emails.
- Bots hate CAPTCHAs almost as much as you do.

Wrapping Up: A Balanced Battle ⚔️

For **pentesters**, CSRF is a treasure map to uncover vulnerabilities and security lapses.

For **coders**, it's a dragon that needs slaying with shields, **tokens**, and strict policies.

When both sides do their job well, the web becomes a safer place for all — except for the bad guys. 🎯

Question: Is it a good practice to keep the anti-CSRF token predictable so that secure coders can quickly implement them? (yea/nay)

Answer: nay

Conclusion: CSRF — Taming the Beast 🐉

CSRF is like the sneaky pickpocket of the web world, capable of wreaking **havoc** if left unchecked.

The battle between attackers and defenders is ongoing, but with knowledge, vigilance, and a sprinkle of CAPTCHA magic, you can stay ahead of the curve. In the end, a secure web is the ultimate win — for pentesters, developers, and every user browsing with peace of mind. 🚀

Stay Connected!

 Instagram: [Cyfer.Nest](#)

 YouTube: [CyferNest](#)

 LinkedIn: [CyferNest Sec](#)

 TikTok: [CyferNest](#)

Let's share some laughs, learn something new, and connect in the digital world! 🚀

Tryhackme

Tryhackme Walkthrough

Tryhackme Writeup

Csrf

Cybersecurity



Follow

Written by CyferNest Sec

107 Followers · 1 Following

CyferNest is your go-to hub for mastering cybersecurity, where we break down complex concepts into easy-to-understand lessons. 💻 🔒

No responses yet



What are your thoughts?

Respond

More from CyferNest Sec

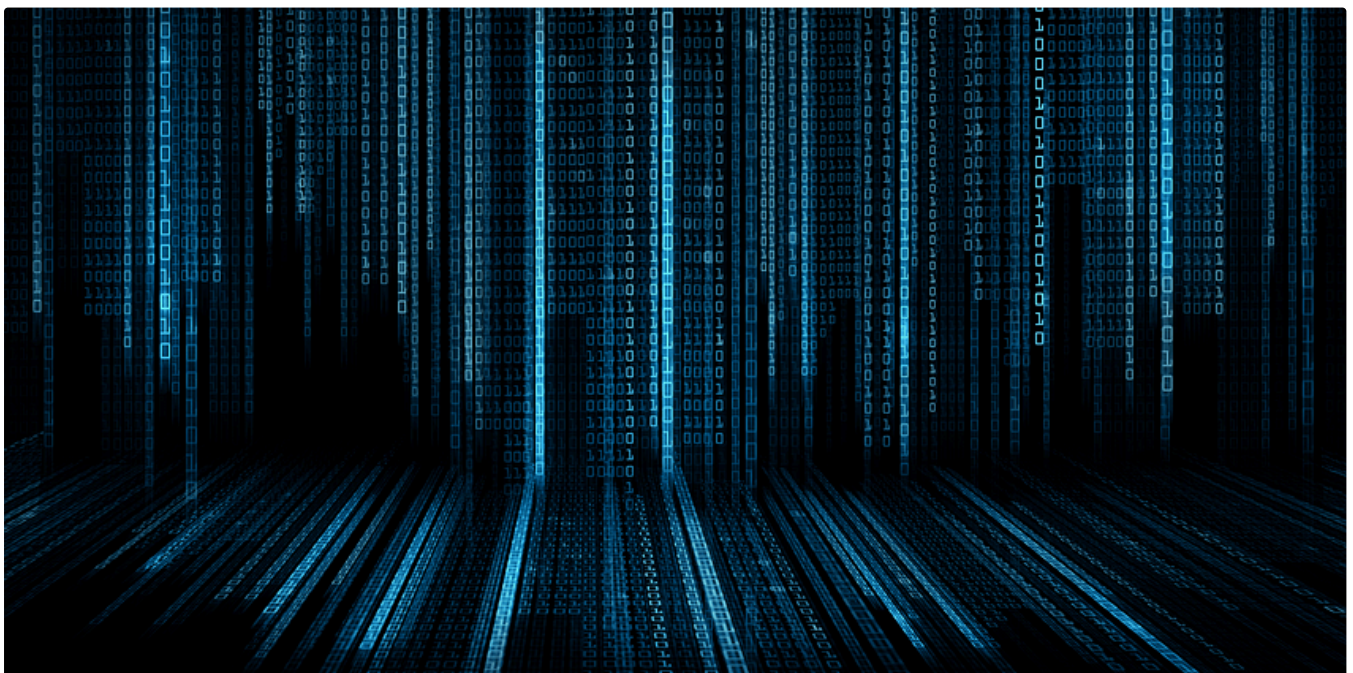


CyferNest Sec

Session Management | TryHackMe Walkthrough

TASK 2: What is Session Management?

Nov 25, 2024 🖱 103





CyferNest Sec

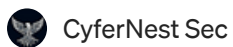
Hashing Basics | TryHackMe Walkthrough

TASK 1: Introduction

Nov 5, 2024



10



CyferNest Sec

SQL Fundamentals | TryHackMe Walkthrough

TASK 1: Introduction

Nov 10, 2024





CyferNest Sec

Basic PenTesting CTF | TryHackMe CTF Walkthrough

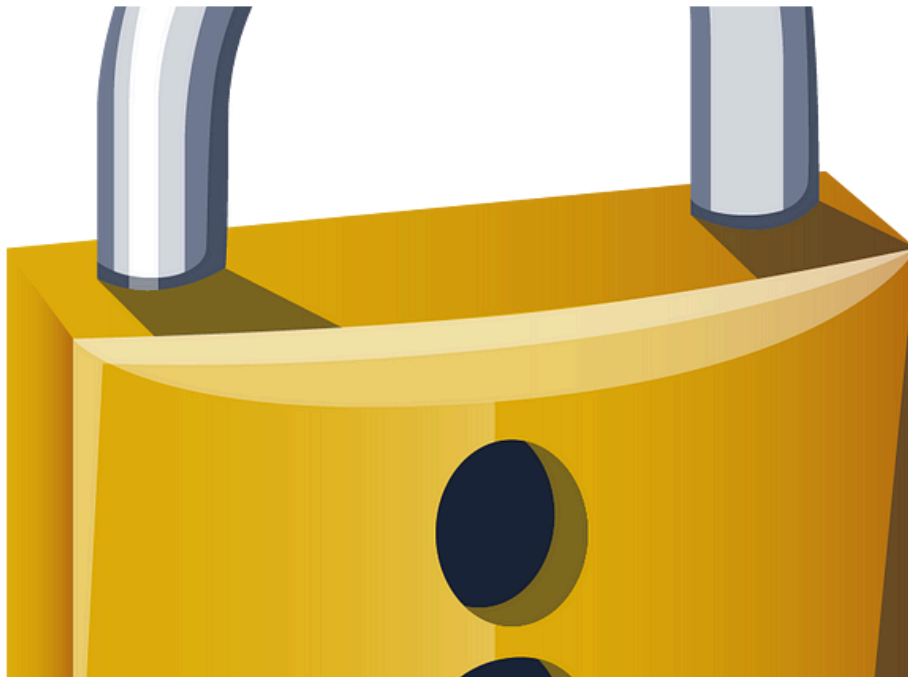
You can access the Basic PenTesting room on TryHackMe here.

★ 1d ago 🖱️ 1



See all from CyferNest Sec

Recommended from Medium



Huy Phu

TryHackMeOAuth Vulnerabilities

URL

★ Nov 10, 2024



Prototype Pollution

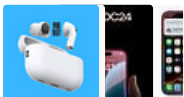
[Open in app ↗](#)**Medium** Search IBM PTC Security

Prototype Pollution

Prototype pollution occur in JavaScript environments where an attacker is able to modify the prototype of an object.

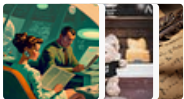
Aug 7, 2024  15

Lists



Tech & Tools

22 stories · 385 saves



Medium's Huge List of Publications Accepting Submissions

377 stories · 4373 saves



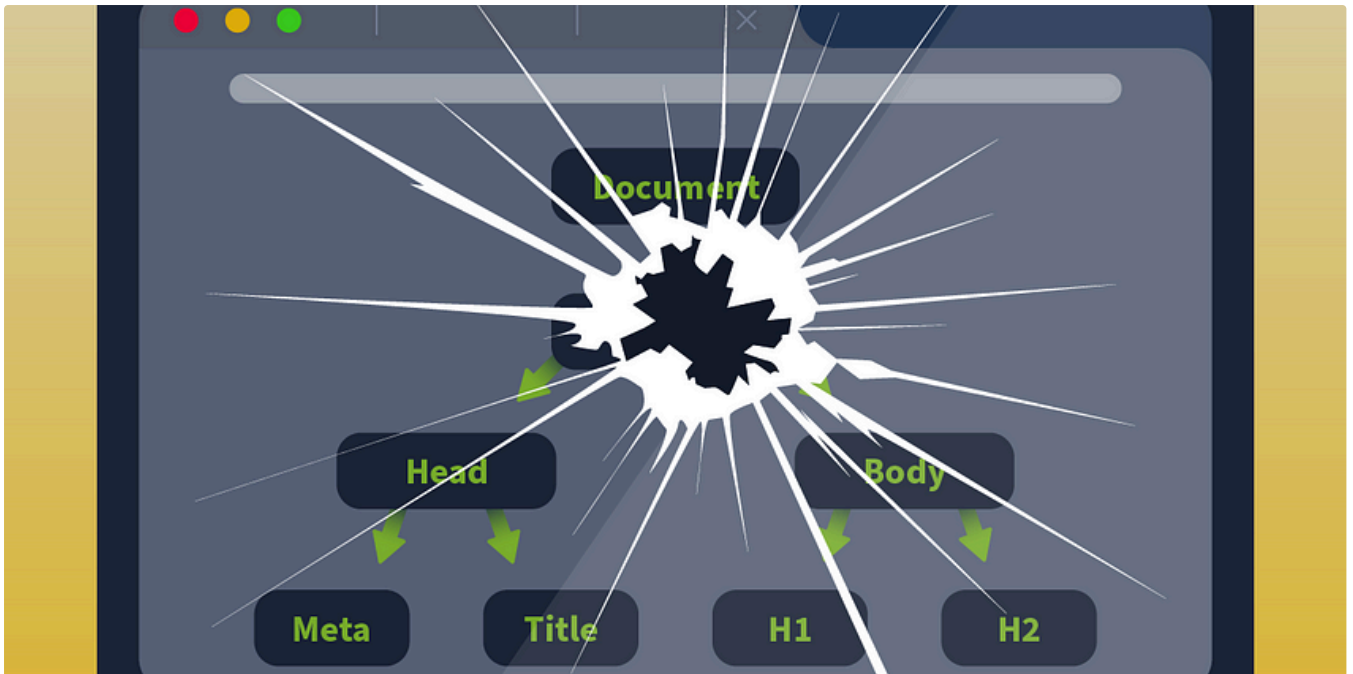
Staff picks

798 stories · 1566 saves



Natural Language Processing

1887 stories · 1536 saves




 Jawstar

DOM-Based Attacks Tryhackme Write-up

Task 1 : Introduction

★ Nov 20, 2024



 Berat Arslan

TryHackMe—Hammer Writeup

‘Hammer’ is one of the ‘Medium’ difficulty rooms in THM.

Sep 1, 2024  69  1





 It4chis3c

Day 1 of 30 Days—30 Vulnerabilities Tips & Tricks

Day 1: Mastering Reflected XSS—Essential Tricks & Techniques Based on Personal Experience and Valuable POCs [In collaboration with...

★ Aug 3, 2024 🖱 5



 Jawstar

Multi-Factor Authentication Tryhackme Answers

Task 1 : Introduction

★ Nov 21, 2024 🖱 1



See more recommendations