

Team B - CTF Documentation

Level -1, Welcome to the CTF!

Your mission - should you choose to accept it - is to navigate through a series of fun, brain-bending levels, collecting flags as you go. Most of what you need can be done right from your favorite web browser (**Chrome**, **Firefox**, even good old `curl`). You might have to put on your "attacker hat" and do a bit of sleuthing along the way.

A Few Friendly Ground Rules

- This is all about sharpening and testing your cybersecurity skills in a safe, friendly, and slightly mischievous way. Please **don't** unleash automated scanners or hacking tools—this is still running inside the production TKH/AWS environment, and we really don't want your experiment crashing anything real.
- Found a flag? Nice work! 🎉 But please keep your clever solutions to yourself for now—no write-ups or walkthroughs, as we'll be reusing these challenges in the future. We promise there'll be a group walkthrough at the end where you can flex your skills.
- **Please don't share level URLs with other teams or players.** Each level's URL actually contains the flag — so if someone clicks on, say, the Level 8 link you gave them, they'll be instantly credited with completing it.

Ready to Begin?

Somewhere on this page lies the URL to your first challenge. It's hiding in plain sight—you just have to find it.

Let the hunt begin, and most importantly... have fun!

Lets proceed to [next level](#) !

Objective: Locate the URL to begin the challenge.

- **Approach:** Right-clicked the page → [View Page Source](#).
- **Discovery:** Found a comment hidden in the HTML:

php-template

CopyEdit

-->

<!-- Always read the source, Luke! The next level is at /levels/level0

- **Action:** Manually visited </levels/level0> to proceed.

Level 0

Kudos! Welcome to Level 0

Lets go to [next level](#) !

To reach to level 1 change the URL from 0 to 1



Level 1

Good one. URL tampering continues to be a big source of web application vulnerabilities. As a developer you have probably seen systems that use auto-incrementing identifiers like this. If the target resource is not intended to be public it's important to verify the logged in user should have access to that specific resource when it is requested (ownership check). If you need to generate a random URL such as a password reset token, you can read best practices around that under {TEAM CISO: PASSWORD SECURITY BEST PRACTICES AND GUIDELINES<GO>}.

Ok, now we know you're past the basics. Let's try something harder.

Imagine this is the logon page for a new third party developed web application, but the developers didn't give you credentials. Try to login (**don't use real credentials!!!**):

Username:

Password:

I have guessed common default credentials and admin worked

- Username: admin
- Password: admin

Level 2

Congrats, you made it.

Each HTTP request includes headers that the browser doesn't show. Same with the response. Sometimes developers don't realize that they are leaking information in these headers, such as software versions.

Can you find the next level?

Level 2

Congrats, you made it.

Each HTTP request includes headers that the browser doesn't show. Same with the response. Sometimes developers don't realize that they are leaking information in these headers, such as software versions.

Can you find the next level?

| Name | Headers | Preview | Response | Initiator | Timing | Cookies |
|----------------------------|------------------|---------|---------------------------------|---|--------|---------|
| 2425c5f6fb3636de3a1c5b9... | Referer Policy | | strict-origin-when-cross-origin | | | |
| bootstrap.min.css | Response Headers | | keep-alive | | | |
| simple-sidebar.css | | | Date | Fri, 27 Jun 2025 15:35:50 GMT | | |
| jquery-3.4.1.min.js | | | Etag | W/"1932-PuMxaWLjftO0B4cV1<7Vdja1g" | | |
| bootstrap.min.js | | | Keep-Alive | timeout=5 | | |
| | | | X-Greetz-To | All CTF Haxors! | | |
| | | | X-Level-3-Key | /level3/e7ebca78265e83fa77b855d81b8ec35 | | |
| | | | X-Powered-By | Express | | |
| | Request Headers | | Accept | text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 | | |
| | | | Accept-Encoding | gzip, deflate | | |
| | | | Accept-Language | en-US,en;q=0.9 | | |
| | | | Cache-Control | max-age=0 | | |
| | | | Connection | keep-alive | | |
| | | | Cookie | _ga=GA1.1.886791606.1750872392; _gcl_au=1.1.860311155.1750872392; | | |

- **Right-click >Inspect**
- Go to > **Network tab**
- **Reload the page** while the Network tab is open.
- Click on the **first request**
- click > **Headers** tab.
- Scroll down to > **Response Header**

Level 3

Modern web applications may need to share state between the server and client. As with any information on the client, assume it can be tampered with unless you have a means to trust it (ie, signature). This level has a variable "nextLevelKey" defined on the "window" object. Figure it out and you'll be onto the next challenge.

Bonus points: You might notice that this page sets the "X-XSS-Protection" header to 0, effectively disabling the browser's built-in XSS detection. By default browsers will detect basic XSS attacks and throw up an error page. Try to use a cross site scripting (XSS) payload in the textbox below to get the key.

If you request this page with "xss=1" in the query string and then try your XSS payload you can see the browser's default behavior.

Level 3

Modern web applications may need to share state between the server and client. As with any information on the client, assume it can be tampered with unless you have a means to trust it (ie, signature). This level has a variable "nextLevelKey" defined on the "window" object. Figure it out and you'll be onto the next challenge.

Bonus points: You might notice that this page sets the "X-XSS-Protection" header to 0, effectively disabling the browser's built-in XSS detection. By default browsers will detect basic XSS attacks and throw up an error page. Try to use a cross site scripting (XSS) payload in the textbox below to get the key.

If you request this page with "xss=1" in the query string and then try your XSS payload you can see the browser's default behavior.

The screenshot shows the browser's developer tools open to the 'Console' tab. The error message 'Uncaught ReferenceError: windows is not defined' is displayed, with the offending line 'at <anonymous>:1:11' highlighted by a red box. The stack trace shows the error occurred at 'window.nextLevelKey'. The browser's status bar at the bottom indicates 'VM202:1'.

- Right-click > **Inspect**
- Go to the **Console** tab
- Type: `window.nextLevelKey`

Level 4

Since the web is stateless, implementing a multi page form can be difficult. For instance it's common for the first page of a registration process to accept some basic parameters, and ask the user to provide more information on the second page before actually creating an account. This is usually done via "hidden" form fields. In these cases it is important for the final processing page to validate all user input, not just validating it after the first submission.

This form has a hidden field called "has_accepted_eula" with a value of "yes". Change the value of "has_accepted_eula" to be something else submit the form.

Enter your name below and hit "Submit":

Level 4

Since the web is stateless, implementing a multi page form can be difficult. For instance it's common for the first page of a registration process to accept some basic parameters, and ask the user to provide more information on the second page before actually creating an account. This is usually done via "hidden" form fields. In these cases it is important for the final processing page to validate all user input, not just validating it after the first submission.

This form has a hidden field called "has_accepted_eula" with a value of "yes". Change the value of "has_accepted_eula" to be something else submit the form.

```
form 426x30 Enter your name below and hit "Submit":  
Submit
```

The screenshot shows the Google Chrome DevTools Elements tab. The page content is displayed in the main pane, and the DevTools interface is visible on the right. The code in the Elements tab shows a form with a text input and a hidden input named "has_accepted_eula" with a value of "yes". The developer is currently inspecting the hidden input field. The right pane shows the Styles, Computed, and Layout tabs, along with the Lighthouse performance audit results.

Level 4

Since the web is stateless, implementing a multi page form can be difficult. For instance it's common for the first page of a registration process to accept some basic parameters, and ask the user to provide more information on the second page before actually creating an account. This is usually done via "hidden" form fields. In these cases it is important for the final processing page to validate all user input, not just validating it after the first submission.

This form has a hidden field called "has_accepted_eula" with a value of "yes". Change the value of "has_accepted_eula" to be something else submit the form.

```
Enter your name below and hit "Submit":  
Submit
```

You got it! On to the [next level](#)

The screenshot shows the Google Chrome DevTools Elements tab. The page content is displayed in the main pane, and the DevTools interface is visible on the right. The code in the Elements tab shows a form with a text input and a hidden input named "has_accepted_eula" with a value of "nope". The developer is currently inspecting the hidden input field. The right pane shows the Styles, Computed, and Layout tabs, along with the Lighthouse performance audit results.

- **Inspect > Elements tab**
- Find the hidden input in the form:

- o <input type="hidden" name="has_accepted_eula" value="yes">
 - Double-click on the value="yes" and change it to something like:
 - o value="nope"
 - Click hit **Submit/ Enter**
-

Level 5

Most web applications rely on cookies to store session information. Cookies are sent by the server in HTTP response headers and stored on the client machines by the browser. Remember, this means that they can be tampered with!

To get the next level of this page, set your own cookie for this site called "isAuthenticated" with a value of "1" and reload this page.

- Right-click > **Inspect**
- Go to > **Application** tab
- Click > **Cookies**"
- Add a new cookie: (double click to add)
 - o **Name:** isAuthenticated
 - o **Value:** 1
- **Reload the page**

Level 5

[Elements](#)
[Console](#)
[Sources](#)
[Network](#)
[Performance](#)
[Memory](#)
[Application](#)
[Privacy and security](#)
[Lighthouse](#)
[Recorder](#)

Most web applications rely on cookies to store session information. Cookies are sent by the server in HTTP response headers and stored on the client machines by the browser. Remember, this means that they can be tampered with!

To get the next level of this page, set your own cookie for this site called "isAuthenticated" with a value of "1" and reload this page.

Cool, you bypassed our **super secure auth!**

Did you notice the "httpOnly" and "secure" bits that could be set on your cookie? You can read about what they do at {TEAM CISO:HTTPONLY ATTRIBUTE NOT SET ON SESSION COOKIE<GO>} and {TEAM CISO:SECURE ATTRIBUTE MISSING FOR SESSION COOKIE<GO>} respectively.

The next level is located at [here](#)

| Name | Value | Do... | Path | Expl... | Size | Http... | Sec... | Ses... | Part... | Cr... | Prio... |
|-----------------|-------|--------|------|---------|------|---------|--------|--------|---------|-------|---------|
| isAuthenticated | 1 | hac... | / | Ses... | 16 | | | | | | Med... |

No cookie selected
Select a cookie to preview its value

Level 6

JSON Web Tokens (JWT's) are commonly used to represent signed claims. Think of them like a passport or driver's license, in which a trusted authority gives you a document and you present it back to another party in the future to prove something.

Developers may assume that by using a JWT to store transmit information about users' identifies their application is secure. However, just like the driver's license example there are things to consider when using JWT's:

- How long should the JWT be seen as valid?
- Does the client gain any information they should not be able to see by possessing the JWT (JWT's are not encrypted)?
- Can a user brute force my JWT's secret and issue arbitrary valid tokens?
- Does the application ensure the JWT is signed by the secret only they know?

The last point is an interesting one. If the service doesn't validate the signature of a JWT, a client can basically send whatever they want in the JWT.

Here's a sample payload and the corresponding JWT:

Sample Payload: `{"username": "Steve", "email": "steve@steve.com", "isAdmin": 0}`
Sample JWT: `eyJhbGciOiJIUzI1NiIsInR5cCIkIkpXVCJ9.eyJ1c2VybmFtZSI6IlN0ZXZlIiwiZW1haWwiOiJzdGV2ZUBzdGV2ZS5jb20iLCJpc0FkbWluIjowLCJpYXQiOjE3NTEyMjM0MTB9.TycdSkg6xpQuLwrrTPd2HV0IVizlbu6rgPrh4fLMGy`

Read up on JWT's and how they are encoded (not encrypted), then submit a new JWT with the "isAdmin" value set to "1":

Submit

Objective: Elevate privileges via JWT token.

Action:

- Used [jwt.io](#) to decode original JWT.
- Discovered the secret: `a-string-secret-at-least-256-bits-long`
Recreated JWT with:

```
{  
  "username": "Steve"  
,  
  "email": "steve@steve.com"  
,  
  "isAdmin": 1  
}
```

- Signed with correct secret.

Result: Authenticated as admin → Reached Level 7.

Level 7

Impressive!

To get the next flag, request this page but with the URL parameter "flag=true". The catch is that the route will only give you the flag if the request comes from localhost so you'll have to trick the application into thinking you're the localhost. You will **NOT** need ssh access to the server to get the flag.

This may sound easy, but can't be done natively in Chrome. On the "Networks" tab in Chrome's inspector you can copy requests as curl which may make things easier.

Objective: Trick the server into thinking the request came from localhost.

Approach:

- Appended ?flag=true to the URL.

Used curl to spoof the IP:

```
curl -H "X-Forwarded-For: 127.0.0.1"  
"http://hackathon.theknowledgehouse.org/levels/015c8d5e17455064e50c407  
cf8ec7e1e?flag=true"
```

- Outcome:** Server believed the request came from 127.0.0.1 and responded with the flag/next level.
-

Level 8

Lets try some very elementary binary reverse engineering for a change now!

We've got a 64-bit linux executable binary for you to [download](#). This binary works on 64-bit linux (ubuntu/debian) machines. This program requires a password to access to the 'admin' functionality. Your mission is to reverse engineer the binary and find the password.



A screenshot of a terminal window titled "Terminal — 67x11". The command entered is `./ctfconsole -p mysecretpass`. The output shows an error message: "Incorrect admin password, try again".

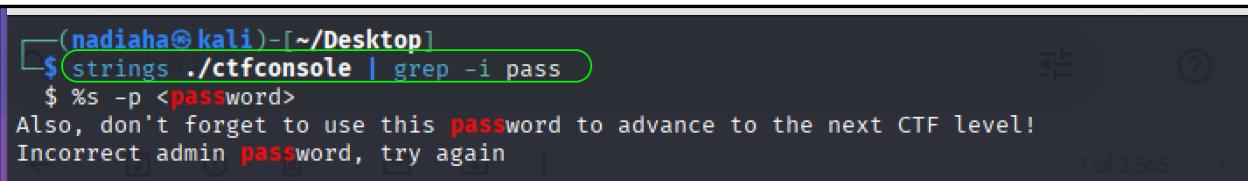
A few notes:

- Please do not brute force password on TKH's shared infrastructure. This challenge doesn't require brute forcing passwords.

Enter the password that works for the binary above:

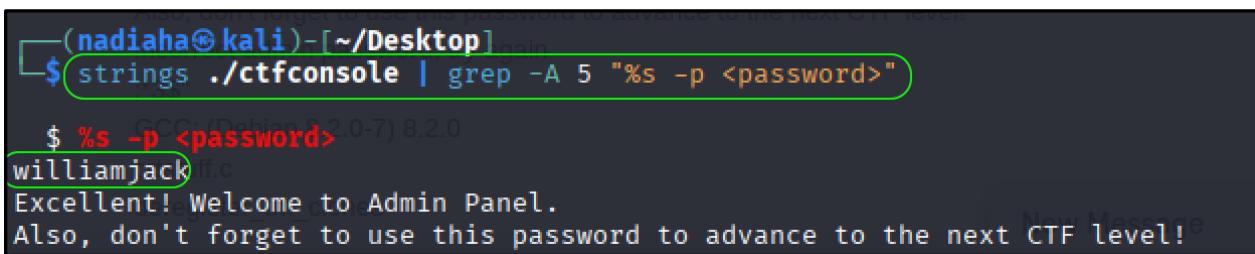
Submit

- Cd to where the file is downloaded and execute the following command:



A screenshot of a terminal window titled "(nadiaha㉿kali)-[~/Desktop]". The command entered is `strings ./ctfconsole | grep -i pass`. The output shows the password "williamjack" highlighted in red. A note at the bottom says "Also, don't forget to use this password to advance to the next CTF level!".

- Output: “ %s -p <password>”
- Then execute the following command:



A screenshot of a terminal window titled "(nadiaha㉿kali)-[~/Desktop]". The command entered is `strings ./ctfconsole | grep -A 5 "%s -p <password>"`. The output shows the password "williamjack" highlighted in red. A note at the bottom says "Excellent! Welcome to Admin Panel." and "Also, don't forget to use this password to advance to the next CTF level!".

- Password : williamjack

Level 9

Client side hashing is often considered as a good defense in depth measure. However, just like any other cryptographic primitive, if used incorrectly, client side hashing might not buy you additional security. Arguably, if used incorrectly, it could even make the application insecure. Lets see how client side logic that involves hashing makes the design insecure in this case!

Tip: Lists of common passwords, such as this one, are commonly available on internet. Could this be useful for this exercise?

Enter admin secret to advance to next level:

Submit

Level 9

Client side hashing is often considered as a good defense in depth measure. However, just like any other cryptographic primitive, if used incorrectly, client side hashing might not buy you additional security. Arguably, if used incorrectly, it could even make the application insecure. Lets see how client side logic that involves hashing makes the design insecure in this case!

Tip: Lists of common passwords, such as this one, are commonly available on internet. Could this be useful for this exercise?

Enter admin secret to advance to next level:

Submit

```
a1508d0522781e8...c9b004e77b23bf X
106 </div>
107 </div>
108 <!-- /#sidebar-wrapper -->
109 <!-- Page Content -->
110 <div id="page-content-wrapper">
111
112
113 <nav class="navbar navbar-expand-lg navbar-light bg-light border-bottom">
114   <div class="collapse navbar-collapse" id="navabarSupportedContent">
115     <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
116       <li class="nav-item active">
117         <a class="nav-link" href="/">Home <span class="sr-only">(current)</span></a>
118       </li>
119     </ul>
120   </div>
121
122   <div class="container-fluid">
123     <h1 class="mt-4">Level 9</h1>
124
125   <script src="/vendor/core.min.js"></script>
126   <script src="/vendor/sha256.min.js"></script>
127   <script>
128     function check_sekret(a){
129       sekret = document.getElementById("sekret").value
130       hash = CryptoJS.SHA256(sekret)
131       if(hash!= "b03136e9801c99d70cb80a2c5a654f572dfd9a8fb6e6dde9fcbcd16e2bc61a6"){
132         alert("Incorrect admin key. Please contact the Management Committee to get your admin credentials")
133         e.preventDefault();
134         return false;
135       }
136     }
137   </script>
138 </script>
139 
```

- Right-click > **Inspect**
- Go to > **source** tab
- Inspect HTML/JS > find the Hash
- Take the hash Value and **Extract it**
“b03136e9801c99d70cb80a2c5a654f572dfd9a8fb6e6dde9fcbcd16e2bc61a”
- Use tool to reverse the hash: <https://crackstation.net/>

CrackStation • Defuse.ca • Twitter

CrackStation • Password Hashing Security • Defuse Security

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
b03136e9801c99d70cb80a2c5a654f572dfd9a8fb6e6dde9fcbcd16e2bc61a6
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (`sha1(shai_bin)`), QubesV3.1BackupDefaults

| Hash | Type | Result |
|---|--------|----------|
| b03136e9801c99d70cb80a2c5a654f572dfd9a8fb6e6dde9fcbcd16e2bc61a6 | sha256 | delpiero |

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

- Hash result: delpiero
-

Level 10

Open redirect vulnerabilities usually appear when an application tries to return the user to the previous page they were on. You have probably seen this type of pattern in URL's before:

<https://www.somesite.com/login?redirect=/profile>

This is a handy way for the browser to return the user to profile page after a successful logon. But what would happen if the redirect could lead the user anywhere? The user would successfully logon to the real site, be redirected to a phishing site, think they typed their password wrong, and probably re-enter credentials on the phishing site! For instance:

<https://www.somesite.com/login?redirect=https%3A%2F%2Fwww.myphishingsite.com>

Web applications try to prevent this scenario by restricting redirects to relative paths, so they can be assured you never end up on an external site. However, this isn't always so easy.

To get to the next level, append a "redirect" parameter to this URL and bypass the check to have the user land on "www.myphishingsite.com". You can read more about open redirect vulnerabilities at {TEAM CISO:OPEN REDIRECT<GO>}.

The screenshot shows a web browser window with the URL <http://hackathon.theknowledgehouse.org/levels//802b6ddfabde64a6f05b8d6776e7e9cb> highlighted with a red box. The page content includes a section titled "Level 10" and several paragraphs of text about open redirect vulnerabilities, including URLs and examples.

Level 10

Open redirect vulnerabilities usually appear when an application tries to return the user to the previous page they were on. You have probably seen this type of pattern in URL's before:

<https://www.somesite.com/login?redirect=/profile>

This is a handy way for the browser to return the user to profile page after a successful logon. But what would happen if the redirect could lead the user anywhere? The user would successfully logon to the real site, be redirected to a phishing site, think they typed their password wrong, and probably re-enter credentials on the phishing site!. For instance:

<https://www.somesite.com/login?redirect=https%3A%2F%2Fwww.myphishingsite.com>

Web applications try to prevent this scenario by restricting redirects to relative paths, so they can be assured you never end up on an external site. However, this isn't always so easy.

To get to the next level, append a "redirect" parameter to this URL and bypass the check to have the user land on "www.myphishingsite.com". You can read more about open redirect vulnerabilities at {TEAM CISO:OPEN REDIRECT<GO>}.

- Take the URL
- “<http://hackathon.theknowledgehouse.org/levels//802b6ddfabde64a6f05b8d6776e7e9cb>”
- and add “?redirect=//www.myphishingsite.com” at the end

The screenshot shows a web browser window with the URL <http://hackathon.theknowledgehouse.org/levels//802b6ddfabde64a6f05b8d6776e7e9cb?redirect=//www.myphishingsite.com> highlighted with a red box. The page content includes a section titled "Level 10" and several paragraphs of text about open redirect vulnerabilities, including URLs and examples. A green success message is displayed at the bottom.

Level 10

Open redirect vulnerabilities usually appear when an application tries to return the user to the previous page they were on. You have probably seen this type of pattern in URL's before:

<https://www.somesite.com/login?redirect=/profile>

This is a handy way for the browser to return the user to profile page after a successful logon. But what would happen if the redirect could lead the user anywhere? The user would successfully logon to the real site, be redirected to a phishing site, think they typed their password wrong, and probably re-enter credentials on the phishing site!. For instance:

<https://www.somesite.com/login?redirect=https%3A%2F%2Fwww.myphishingsite.com>

Web applications try to prevent this scenario by restricting redirects to relative paths, so they can be assured you never end up on an external site. However, this isn't always so easy.

To get to the next level, append a "redirect" parameter to this URL and bypass the check to have the user land on "www.myphishingsite.com". You can read more about open redirect vulnerabilities at {TEAM CISO:OPEN REDIRECT<GO>}.

Correct! You bypassed the filter using a schemaless URL. Browsers will do all sorts of hacky things to account for user errors, including turning backslashes into slashes.

[On to the next level](#)

Level 11

This level has a bug that causes a server-side error. It's a common practice to include full stack traces in a dev environment, but they commonly leak sensitive information.

Try to find the bug and cause a stack trace.

The screenshot shows a web browser window with the URL <http://hackathon.theknowledgehouse.org/levels//dfd2efdec05aa823566fac1f1a6409d9?q=abc123>. The page content is identical to the one above, displaying the challenge details and the instruction to find a bug causing a stack trace.

- Change the end of the URL and add “?q[0]=abc” at the end

[http://hackathon.theknowledgehouse.org/levels//dfd2efdec05aa823566fac1f1a6409d9
?q\[0\]=abc](http://hackathon.theknowledgehouse.org/levels//dfd2efdec05aa823566fac1f1a6409d9?q[0]=abc)

The screenshot shows a web browser window with the URL [http://hackathon.theknowledgehouse.org/levels//dfd2efdec05aa823566fac1f1a6409d9?q\[0\]=abc](http://hackathon.theknowledgehouse.org/levels//dfd2efdec05aa823566fac1f1a6409d9?q[0]=abc). The page displays a stack trace error message:

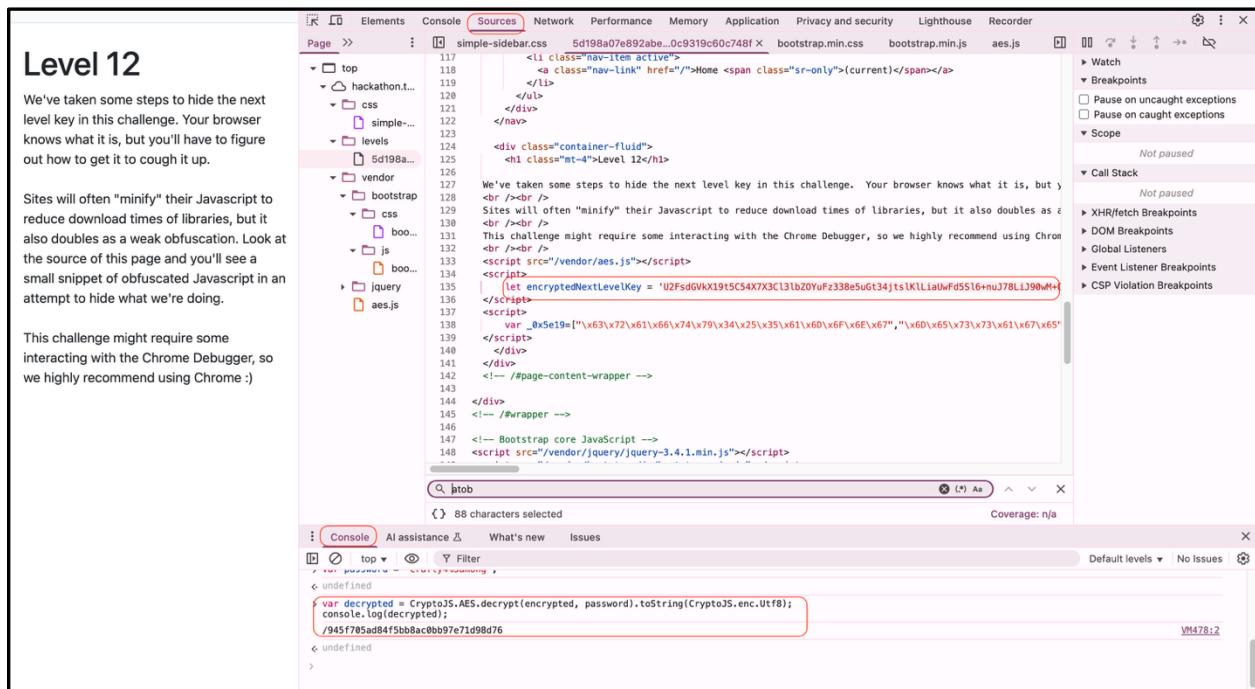
```
TypeError: Cannot create property 'path' on string 'Passing an object might crash the application! Go to the next level at /levels/5d198a07e892abe6ad0c9319c60c748f'
at rethrow (/home/ctf/ctf/src/node_modules/ejs/lib/ejs.js:349:12)
at eval (/home/ctf/ctf/src/views/level11.ejs:66:3)
at level11 (/home/ctf/ctf/src/node_modules/ejs/lib/ejs.js:682:17)
at tryHandleCache (/home/ctf/ctf/src/node_modules/ejs/lib/ejs.js:254:36)
at exports.renderFile [as engine] (/home/ctf/ctf/src/node_modules/ejs/lib/ejs.js:485:10)
at View.render (/home/ctf/ctf/src/node_modules/express/lib/view.js:135:8)
at tryRender (/home/ctf/ctf/src/node_modules/express/lib/application.js:657:10)
at Function.render (/home/ctf/ctf/src/node_modules/express/lib/application.js:609:3)
at ServerResponse.render (/home/ctf/ctf/src/node_modules/express/lib/response.js:1049:7)
at /home/ctf/ctf/src/routes/LevelRouter.js:27:8
```

Level 12

We've taken some steps to hide the next level key in this challenge. Your browser knows what it is, but you'll have to figure out how to get it to cough it up.

Sites will often "minify" their Javascript to reduce download times of libraries, but it also doubles as a weak obfuscation. Look at the source of this page and you'll see a small snippet of obfuscated Javascript in an attempt to hide what we're doing.

This challenge might require some interacting with the Chrome Debugger, so we highly recommend using Chrome :)



Goal is to Decrypt the encrypted AES string to reveal the URL for the next level

- Right-click > **Inspect**.
- Go to > Sources Tab > Look for Js files
- Type in the console:

```
var decrypted = CryptoJS.AES.decrypt(encrypted, password).toString(CryptoJS.enc.Utf8);
console.log(decrypted);
```

Level 13

Moving right along...

This page downloads an image, but never actually puts it in the DOM. Can you find it? Input the contents displayed on the image:

Secret:

Submit

Level 13

Moving right along...

This page downloads an image, but never actually puts it in the DOM. Can you find it?
Input the contents displayed on the image:

Secret:

Submit

The screenshot shows the Network tab of the browser developer tools. A single request is listed, highlighted with a red box. The request URL is <http://hackathon.theknowledgehouse.org/vendor/cc7ba8ace7d53450e05e4bac51cae9e2.png>. The status code is 304 Not Modified. The response headers include Accept-Ranges, Cache-Control (public, max-age=0), Connection (keep-alive), Date (Sat, 28 Jun 2025 17:06:44 GMT), Etag (W/"1064-19766515258"), Keep-Alive (timeout=5), Last-Modified (Thu, 12 Jun 2025 22:44:55 GMT), X-Powered-By (Express), and a Content-Type header (image/avif,image/webp,image/apng,image/svg+xml,image/*/*;q=0.8). The request headers show Accept (image/avif,image/webp,image/apng,image/svg+xml,image/*/*;q=0.8) and Accept-Encoding (gzip, deflate).



- **Right-click > Inspect.**
 - Go to > **Network** tab & reload the page
 - Look for an **image file (.png)**
 - Right click > Copy URL > paste it into a new browser tab
 - The secret is “MEME 149”
-

The screenshot shows a website for the TKH Cybersecurity Hackathon 2025. On the left, there's a sidebar with links for "About", "Set a Team Name", and "Leaderboard". The main content area is titled "Level 14 Congratulations!" and includes a message of congratulations, a reference link, and a note from the organizers.

TKH Cybersecurity
Hackathon 2025

About
Set a Team Name
Leaderboard

Home

Level 14

Congratulations!

You have successfully completed the CTF! Hopefully you've learned some cool tricks that will help you grow as a developer.

To prove your skills, you can reference [/93807da7c92bd76716eb5d3c4433a37f](#).

We hope you enjoyed,
Steve, Shivam, and Peter

The screenshot shows the same website after completing Level 15. The main content area now says "Ok, there's really no more levels. :)"

TKH Cybersecurity
Hackathon 2025

About
Set a Team Name
Leaderboard

Home

Level 15

Ok, there's really no more levels. :)