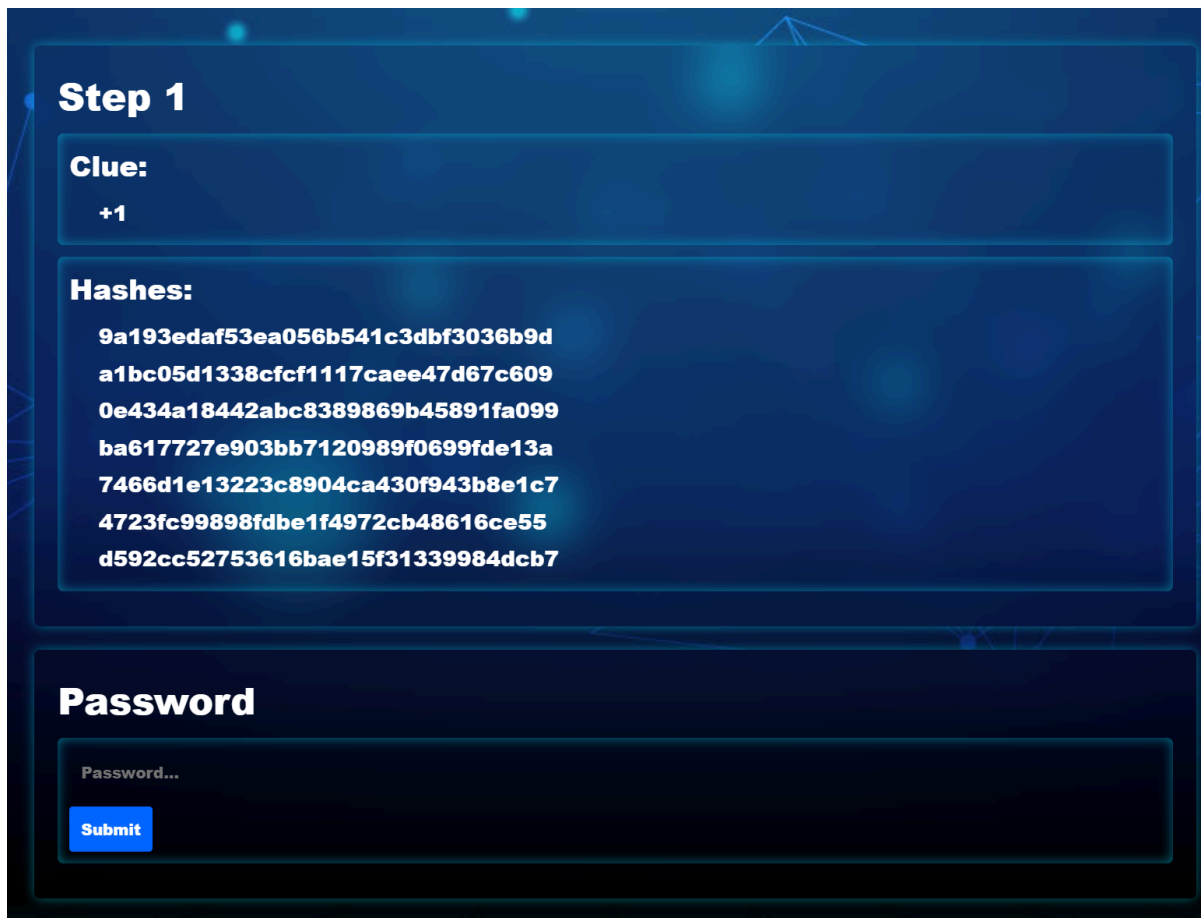


Hashes.com/Hashkiller.io

2025 Contest Writeup

This is a writeup for the contest from Hashes.com and Hashkiller.io on 2025 which started on Sunday 5 January 2025 and ended on the 7th of January (GMT)¹ with a prize reward of \$2500. This forum post redirected the users to a contest page: <https://contest.hashes.community/> which presented people with a list of hashes.



Step 1

Clue:
+1

Hashes:

```
9a193edaf53ea056b541c3dbf3036b9d
a1bc05d1338cfcf1117caee47d67c609
0e434a18442abc8389869b45891fa099
ba617727e903bb7120989f0699fde13a
7466d1e13223c8904ca430f943b8e1c7
4723fc99898fdbef14972cb48616ce55
d592cc52753616bae15f31339984dcb7
```

Password

Password...

Submit

This is everything that was provided to users to get started with the contest. With that in mind Vavaldi, Penguinkeeper, and Shooter3k (3 veteran members of HashMob) set out to tackle these challenges. The rest of the document will explain what we discovered about the individual steps and how we solved it. This might also include some additional findings or issues we ran into.

This document contains spoilers and solutions. If you are interested in completing this challenge yourself we recommend you stop reading now.

1

<https://forum.hashkiller.io/index.php?threads/completed-hashes-com-hashkiller-io-2025-contest-2-500.72373/>

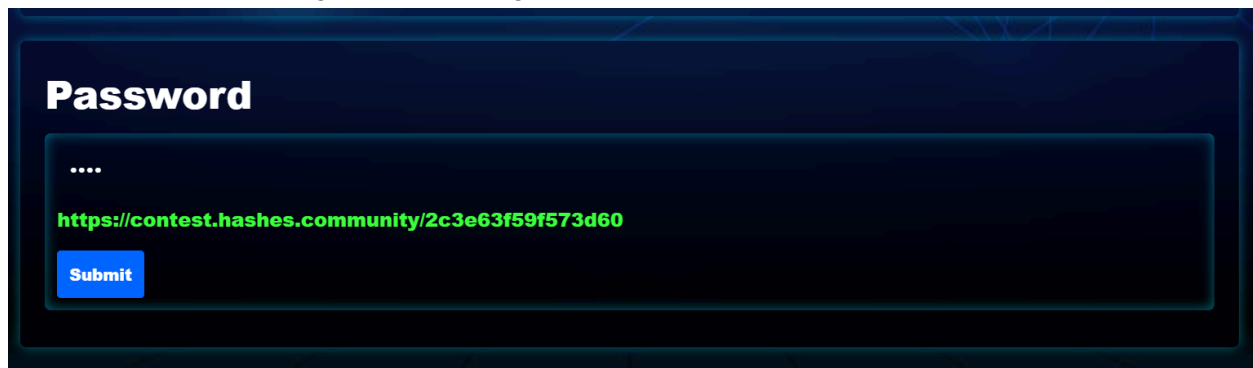
Step 1

The first step started off fairly easy to solve but hard to submit. Using mdxfind we quickly discovered that the hashes were simple MD5 x2² hashes. However, the solution didn't work initially. After trying some alternatives we reached out to the creator and after a fix was applied the solution started working.

When placing the hashes in the right order it reads:

```
9a193edaf53ea056b541c3dbf3036b9d:https
a1bc05d1338cfcf1117caee47d67c609:hashes
0e434a18442abc8389869b45891fa099:com
ba617727e903bb7120989f0699fde13a:tools
7466d1e13223c8904ca430f943b8e1c7:base64
4723fc99898fdba1f4972cb48616ce55:encode
D592cc52753616bae15f31339984dcb7:173
```

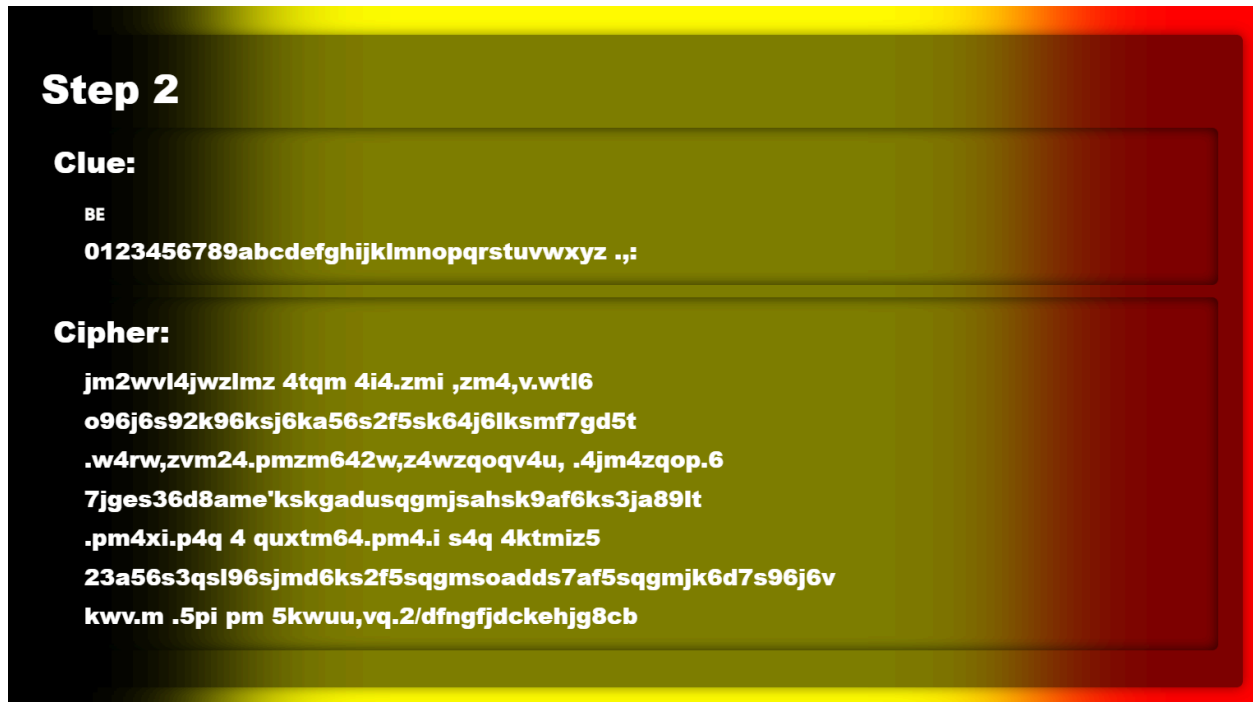
This is a broken-down URL to <https://hashes.com/tools/base64encode> allowing you to base64 encode data. When encoding the data: 173 it returns the value: "MTcz". When entering that as the password it displayed a URL to the next page. To not make it too easy for anyone wishing to follow this we won't disclose any other URL's, but will include plains. We encourage everyone interested in participating in the challenge to complete these themselves.



² md5(md5(\$pass))

Step 2

The second challenge presented the users with a large belgium flag in the background and a clue: "BE".



Step 2

Clue:

BE

0123456789abcdefghijklmnopqrstuvwxyz .,::

Cipher:

jm2wvl4jwzlmz 4tqm 4i4.zmi ,zm4,v.wtl6
o96j6s92k96ksj6ka56s2f5sk64j6lksmf7gd5t
.w4rw,zvm24.pmzm642w,z4wzqoqv4u, .4jm4zqop.6
7jges36d8ame'kskgadusqgmjsahsk9af6ks3ja89lt
.pm4xi.p4q 4 quxtm64.pm4.i s4q 4ktmiz5
23a56s3qsl96sjmd6ks2f5sqgmsoads7af5sqgmjk6d7s96j6v
kwv.m .5pi pm 5kwuu,vq.2/dfngfjdckehjg8cb

Although not visible in the front-end this also included a phone number emoticon. This could be observed by inspecting the elements of the HTML page via F12 (chrome) or 'right click' => 'inspect element'.

```
<h1>Step 2</h1>
<div class="inset">
  <h2>Clue:</h2>
  <h3> == $0
    <span style="display: none;">☎️</span>
    " BE"
  </h3>
  <h3>0123456789abcdefghijklmnopqrstuvwxyz .,::</h3>
</div>
```

The phone number clue refers to the international prefix code used by phone numbers. For the United States this is +1, for Belgium this is +32. Additionally you'll notice that it refers to the text as "Cipher" an indication that this is also different from the Hashes we were presented with before, and as many will spot it is not a hexadecimal range or a common hash format.

The hint: “0123456789abcdefghijklmnopqrstuvwxyz .,:” possibly refers to a substitution cipher and +32 refers to a transposition of some sort. So some of the first things tried were Caesar’s cipher and Atbash as they are commonly used transposition ciphers. This didn’t lead to anything legible. However, by taking the hints’ alphabet and transposing it 32 characters to the left (or right) lead to a readable text. Meaning the character 0, 32x to the left is W and ‘0’ 32x to the right is 8 and so on.

0123456789abcdefghijklmnopqrstuvwxyz .,: :	Original
wxyz .,: :0123456789abcdefghijklmnopqrstu	32 to the left
89abcdefghijklmnopqrstuvwxyz .,: :01234567	32 to the right

To the left

[https://gchq.github.io/CyberChef/#recipe=Substitute\('0123456789abcdefghijklmnopqrstuvwxyz %20.,:','wxyz%20.,:0123456789abcdefghijklmnopqrstuv',false\)&input=am0yd3ZsNGp3emxteiA0dHFtIDRpNC56bWkgLHptNCx2Lnd0bDYNCm85Nmo2czkyazk2a3NqNmthNTZzMmY1c2s2NGo2bGtzWY3Z2Q1dA0KLnc0cncsenZtMjQucG16bTY0MncsejR3enFvcXY0dSwgljRqbTR6cW9wLjYncjdgZ2VzMzZkOGFtZSdrc2tnYWR1c3FnbWpzYWhzazlhZjZrczNqYTg5bHQNCi5wbTR4aS5wNHEgNCBxdXh0bTY0LnBtNC5pIHMOcSA0a3RtaXo1DQoyM2E1NnMzcXNsOTZzam1kNmtzMmY1c3FnbXNvYWRkcZdhZjVzcWdtams2ZDdzOTZqNnYnYncmt3di5tlC41cGkgcG0gNWt3dXUzdnEuMi9kZm5nZmpkY2tlaGpnOGNi&ieol=CRLF&oeol=CRLF](https://gchq.github.io/CyberChef/#recipe=Substitute('0123456789abcdefghijklmnopqrstuvwxyz%20.,:','wxyz%20.,:0123456789abcdefghijklmnopqrstuv',false)&input=am0yd3ZsNGp3emxteiA0dHFtIDRpNC56bWkgLHptNCx2Lnd0bDYNCm85Nmo2czkyazk2a3NqNmthNTZzMmY1c2s2NGo2bGtzWY3Z2Q1dA0KLnc0cncsenZtMjQucG16bTY0MncsejR3enFvcXY0dSwgljRqbTR6cW9wLjYncjdgZ2VzMzZkOGFtZSdrc2tnYWR1c3FnbWpzYWhzazlhZjZrczNqYTg5bHQNCi5wbTR4aS5wNHEgNCBxdXh0bTY0LnBtNC5pIHMOcSA0a3RtaXo1DQoyM2E1NnMzcXNsOTZzam1kNmtzMmY1c3FnbXNvYWRkcZdhZjVzcWdtams2ZDdzOTZqNnYnYncmt3di5tlC41cGkgcG0gNWt3dXUzdnEuMi9kZm5nZmpkY2tlaGpnOGNi&ieol=CRLF&oeol=CRLF)

```
beyond borders lies a treasure untold,
g1,b,k1yc1,ckb,c2.,ky7.kc, b,dcke7:85.1
to journey there, your origin must be right,
:b86kz,502e6'ckc825mki8ebk29kc127,ckzb201d1
the path is simple, the task is clear.
yz2.,kzikd1,kbe5,cky7.ki8ekg255k:27.ki8ebc,5:k1,b,n
contest.hashes.community/57f8
```

To the right

[https://gchq.github.io/CyberChef/#recipe=Substitute\('0123456789abcdefghijklmnopqrstuvwxyz %20.,:','89abcdefghijklmnopqrstuvwxyz%20.,:01234567',false\)&input=am0yd3ZsNGp3emxteiA0dHFtIDRpNC56bWkgLHptNCx2Lnd0bDYNCm85Nmo2czkyazk2a3NqNmthNTZzMmY1c2s2NGo2bGtzWY3Z2Q1dA0KLnc0cncsenZtMjQucG16bTY0MncsejR3enFvcXY0dSwgljRqbTR6cW9wLjYncjdgZ2VzMzZkOGFtZSdrc2tnYWR1c3FnbWpzYWhzazlhZjZrczNqYTg5bHQNCi5wbTR4aS5wNHEgNCBxdXh0bTY0LnBtNC5pIHMOcSA0a3RtaXo1DQoyM2E1NnMzcXNsOTZzam1kNmtzMmY1c3FnbXNvYWRkcZdhZjVzcWdtams2ZDdzOTZqNnYnYncmt3di5tlC41cGkgcG0gNWt3dXUzdnEuMi9kZm5nZmpkY2tlaGpnOGNi&ieol=CRLF&oeol=CRLF](https://gchq.github.io/CyberChef/#recipe=Substitute('0123456789abcdefghijklmnopqrstuvwxyz%20.,:','89abcdefghijklmnopqrstuvwxyz%20.,:01234567',false)&input=am0yd3ZsNGp3emxteiA0dHFtIDRpNC56bWkgLHptNCx2Lnd0bDYNCm85Nmo2czkyazk2a3NqNmthNTZzMmY1c2s2NGo2bGtzWY3Z2Q1dA0KLnc0cncsenZtMjQucG16bTY0MncsejR3enFvcXY0dSwgljRqbTR6cW9wLjYncjdgZ2VzMzZkOGFtZSdrc2tnYWR1c3FnbWpzYWhzazlhZjZrczNqYTg5bHQNCi5wbTR4aS5wNHEgNCBxdXh0bTY0LnBtNC5pIHMOcSA0a3RtaXo1DQoyM2E1NnMzcXNsOTZzam1kNmtzMmY1c3FnbXNvYWRkcZdhZjVzcWdtams2ZDdzOTZqNnYnYncmt3di5tlC41cGkgcG0gNWt3dXUzdnEuMi9kZm5nZmpkY2tlaGpnOGNi&ieol=CRLF&oeol=CRLF)

mtzMmY1c3FnbXNvYWRkczdhZjVzcWdtams2ZDdzOTZqNnYNCmt3di5tlC41cGkgcG0gNWt3dXUsdnEuMi9kZm5nZmpkY2tlaGpnOGNi&ieol=CRLF&oeol=CRLF

```
rua0:tcr03tu34c.yu4cq53uq463uc6:50.te
where hashes reside and secrets unfold.
50cz063:uac5xu3ueca063c03ywy:c,645cruc3ywx5e
from belgium's soil, your ip shines bright.
5xuc1q5xcy4c4y,1.uec5xuc5q4 cy4cs.uq3d
abide by the rules and you will find yourself here:
s0:5u45dxq4xu4ds0,,6:y5a/1nvonrlksmprogkj
```

Smashing them together line by line results in the total plain:

```
beyond borders lies a treasure untold,
where hashes reside and secrets unfold.
to journey there, your origin must be right,
from belgium's soil, your ip shines
the path is simple, the task is clear.
abide by the rules and you will find yourself
contest.hashes.community/xxxxxxxxxxxxxxxx
```

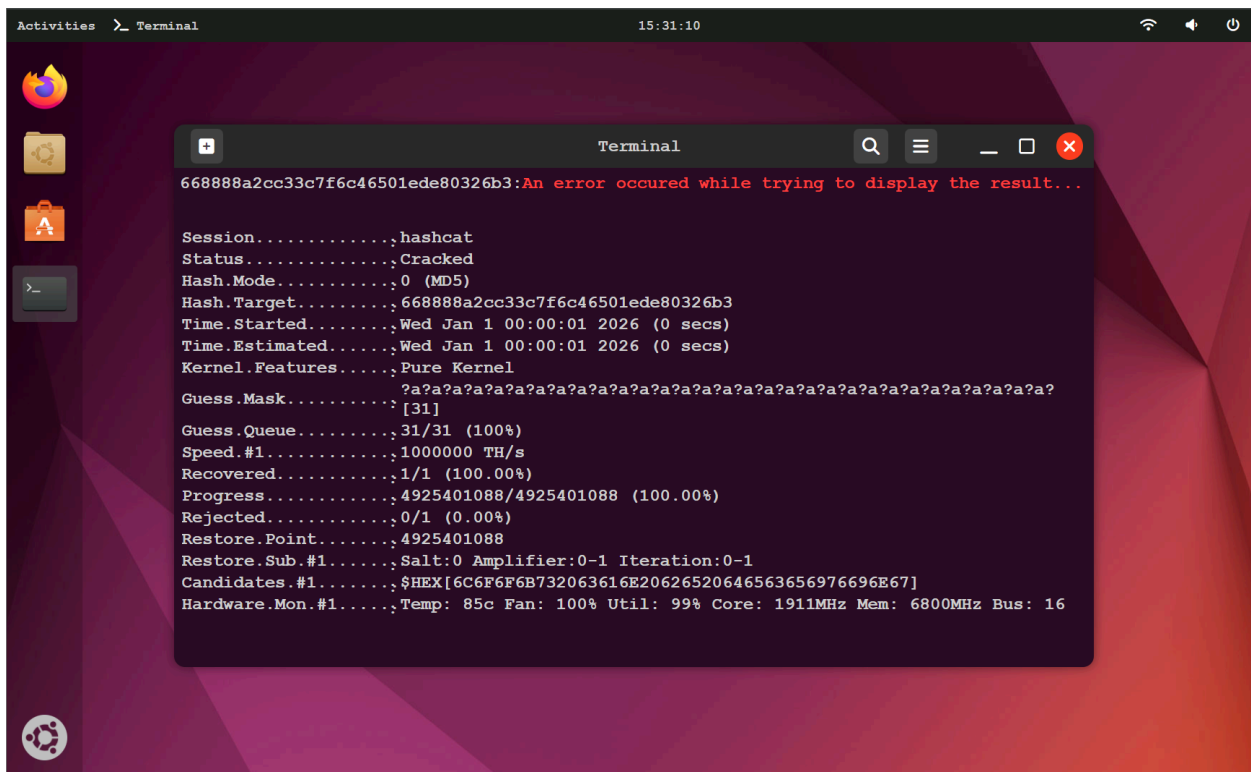
Besides a link to the next step it also includes a piece of text that could provide a hint to step 3.

Step 3

Sorry, you have been blocked

You are unable to access hashes.community

Step 3 links to a cloudflare block page. Initially considering this to be a potential issue with the link we later considered that this might be because it wants us to use the hint from the last step to place our 'origin' from 'belgian soil'. to journey there, your origin must be right, from belgium's soil, your ip shines. Using a VPN we could change our origin IP to be from Belgian origin. Allowing us to access the 3rd challenge.



```
668888a2cc33c7f6c46501ede80326b3:An error occured while trying to display the result...

Session.....,hashcat
Status.....,Cracked
Hash.Mode.....,0 (MD5)
Hash.Target.....,668888a2cc33c7f6c46501ede80326b3
Time.Started.....,Wed Jan 1 00:00:01 2026 (0 secs)
Time.Estimated.....,Wed Jan 1 00:00:01 2026 (0 secs)
Kernel.Features.....,Pure Kernel
Guess.Mask.....,?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?
[31]
Guess.Queue.....,31/31 (100%)
Speed.#1.....,1000000 TH/s
Recovered.....,1/1 (100.00%)
Progress.....,4925401088/4925401088 (100.00%)
Rejected.....,0/1 (0.00%)
Restore.Point.....,4925401088
Restore.Sub.#1.....,Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1.....,$HEX[6C6F6F6B732063616E20626520646563656976696E67]
Hardware.Mon.#1.....,Temp: 85c Fan: 100% Util: 99% Core: 1911MHz Mem: 6800MHz Bus: 16
```

This provides a hashcat terminal in an ubuntu instance - or so it seems at first. But clicking on things quickly shows this is a javascript replication of the Ubuntu desktop and it's not actually capable of much. But what grabs the attention is a hash at the top with an error message. The Hashcat status is 'cracked' and those with a sharp eye might spot that the Candidates.#1 is a printable ascii. When we decode this it reads the message: "looks can be deceiving". This does not compute to the desired hash.

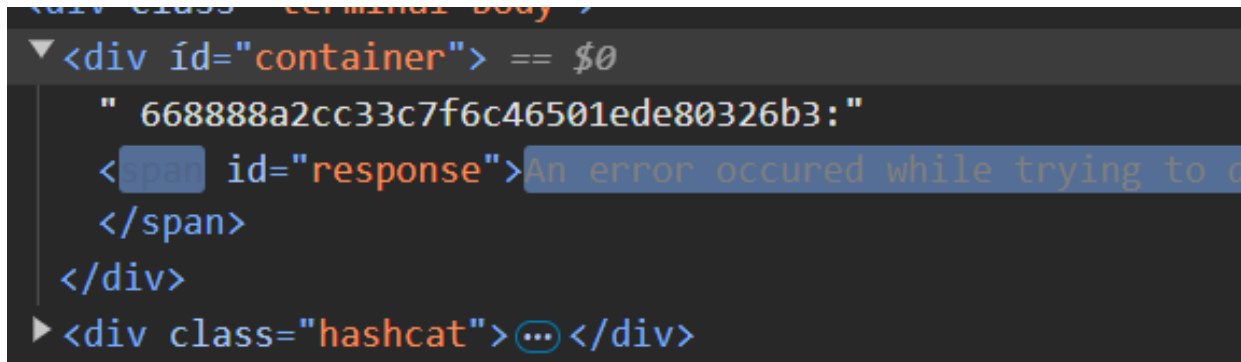
Inspecting the source-code of the website again we can spot a large chunk of encoded and obfuscated JavaScript. Now the solution to this step knows two answers / approaches.

```
</div>
</main>
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
<script>
a4947c1bd300dd40f59a5539c1e503af = Function("return " + String.fromCharCode(119, 105, 110, 100, 111, 119))();
a4635c85b1a7c650fcb753267b1ef = a4947c1bd300dd40f59a5539c1e503af[String.fromCharCode(97, 116, 111, 98)];
c2v85m50zXj2YmW = "c2v85m50zXj2YmW";
feed43774b83928a7789a86e67c0cb1 = a4947c1bd300dd40f59a5539c1e503af[a4635c85b1a7c650fcb753267b1ef](c2v85m50zXj2YmW7);function a366d0811cd002c7b26f6235414f55dac5d607f6529e3c808a6ae5dc19bb(ae80fe111d378d79f9a3e036bf61b04e48a351c66a1864ee5108004cca9d2f82248f5943e585eeef.replace(/=/g, ""));if (ae80fe111d378d79f9a3e036bf61b04e48a351c66a1864ee5108004cca9d2f82248f5943e585eeef.charAt(0da87cf1d326856640c3fe4ad4af8b9146f682242a5e337f6ae98323960c4fa3199846cd88c25d724dc34899766e9408a746604701aeb359493fb25 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce = "ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz8123456789+/-");
b9f44c19d72798170a9cb05f76b267d982761c3481592ee914d41b9d6da28f6ae41dcd729c567894d5d13ae597e4c8c916becb78a7268a751f207f916186 = "";
bc7fab4bf06c185948d69435876d0a19244324c0e08c635c8ad75c65c9e1e3d2f15a0f48f8ee1e = 0;let YmRYVjJQz9kZmW = "Z0VjZDRlVjJQz9kZmW50", EC98923F353EBB344FD348AFA3C9C7D = a4947c1bd300dd40f59a5539c1e503af[a4635c85b1a7c650fcb753267b1ef](c2v85m50zXj2YmW7);function a366d0811cd002c7b26f6235414f55dac5d607f6529e3c808a6ae5dc19bb(ae80fe111d378d79f9a3e036bf61b04e48a351c66a1864ee5108004cca9d2f82248f5943e585eeef.replace(/=/g, ""));if (ae80fe111d378d79f9a3e036bf61b04e48a351c66a1864ee5108004cca9d2f82248f5943e585eeef.charAt(0da87cf1d326856640c3fe4ad4af8b9146f682242a5e337f6ae98323960c4fa3199846cd88c25d724dc34899766e9408a746604701aeb359493fb25 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce = "ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz8123456789+/-");
c71aac6a95c1a806e6a00739e6ec7d = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce.indexof(ae80fe111d378d79f9a3e036bf61b04e48a351c66a1864ee5108004cca9d2f82248f5943e585eeef.charAt(0da87cf1d326856640c3fe4ad4af8b9146f682242a5e337f6ae98323960c4fa3199846cd88c25d724dc34899766e9408a746604701aeb359493fb25 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce = "ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz8123456789+/-");
e8d6a6839c85284121f0867993002 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce.indexof(ae80fe111d378d79f9a3e036bf61b04e48a351c66a1864ee5108004cca9d2f82248f5943e585eeef.charAt(0da87cf1d326856640c3fe4ad4af8b9146f682242a5e337f6ae98323960c4fa3199846cd88c25d724dc34899766e9408a746604701aeb359493fb25 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce = "ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz8123456789+/-");
b9f44c19d72798170a9cb05f76b267d982761c3481592ee914d41b9d6da28f6ae41dcd729c567894d5d13ae597e4c8c916becb78a7268a751f207f916186 + String.fromCharCode((73f3f808f9c87f093e2a869415489d << 2) | (cda87cf1d326856640c3fe4ad4af8b9146f682242a5e337f6ae98323960c4fa3199846cd88c25d724dc34899766e9408a746604701aeb359493fb25 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce = "ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz8123456789+/-");
b9f44c19d72798170a9cb05f76b267d982761c3481592ee914d41b9d6da28f6ae41dcd729c567894d5d13ae597e4c8c916becb78a7268a751f207f916186 + String.fromCharCode((c71aac6a95c1a806e6a00739e6ec7d & 3) << 6) | e8d6a6839c85284121f0867993002 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce.indexof(ae80fe111d378d79f9a3e036bf61b04e48a351c66a1864ee5108004cca9d2f82248f5943e585eeef.charAt(0da87cf1d326856640c3fe4ad4af8b9146f682242a5e337f6ae98323960c4fa3199846cd88c25d724dc34899766e9408a746604701aeb359493fb25 = df07f5eeds31c7703f5839c782f77e36e6dbb7b2c4d41226a2b56a1555e19769eb97cfff5ce = "ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz8123456789+/-");
feed43774b83928a7789a86e67c0cb1(()) => (const e1742a5dd2ca412b5f02489f8c674377 = Function("return " + String.fromCharCode(100, 111, 99, 117, 109, 101, 110, 116))());
a9e7f59a5b979b9eb278d98f074157f1f51f840a3f1ac3566f6f849f176664cd6736b5b50c = e1742a5dd2ca412b5f02489f8c674377.getElementById("container");
b49c11f60b2e8d8f6a008cad04f82250a2d65753db24b2e7ad30 = a1742a5dd2ca412b5f02489f8c674377.getElementById("response");
c897b1850d9b95e835102cd0f83995c2816a1041382fca77bc6ad6e7e34d41ff8be6ccfe12ac26a155f77e259743a = "aR0cHm6l9j3b2";
sed04bc4d23dd72db971692f94d3af77bcf6e55ada6501d94a3323e99460350e = "50ZXN0Lmhhc2hl";
c897b1850d9b95e835102cd0f83995c2816a1041382fca77bc6ad6e7e34d41ff8be6ccfe12ac26a155f77e259743a = "aR0cHm6l9j3b2";
sed04bc4d23dd72db971692f94d3af77bcf6e55ada6501d94a3323e99460350e = "50ZXN0Lmhhc2hl";
f54f3adac4128e4f199b5b7e1a7cdf45299a4c95052aa36732668ea = "cysj2b1tdw5pdh";
zHm07dHm9gND0300f300902f300023c500R0 = c897b1850d9b95e835102cd0f83995c2816a1041382fca77bc6ad6e7e34d41ff8be6ccfe12ac26a155f77e259743a + "kwZyYyYTBINDU4";
sed04bc4d23dd72db971692f94d3af77bcf6e55ada6501d94a3323e99460350e = "Nzc3ODA0NAA=";
f54f3adac4128e4f199b5b7e1a7cdf45299a4c95052aa36732668ea +
e0f52c4b191d114b50828c298357c3f6 +
b30e0a61c16bc747846d6728b33f4a;
zHm07dHm9gND0300f300902f300023c500R0 = c897b1850d9b95e835102cd0f83995c2816a1041382fca77bc6ad6e7e34d41ff8be6ccfe12ac26a155f77e259743a +
sed04bc4d23dd72db971692f94d3af77bcf6e55ada6501d94a3323e99460350e +
f54f3adac4128e4f199b5b7e1a7cdf45299a4c95052aa36732668ea +
e0f52c4b191d114b50828c298357c3f6 +
b30e0a61c16bc747846d6728b33f4a;
zHm07dHm9gND0300f300902f300023c500R0 = c897b1850d9b95e835102cd0f83995c2816a1041382fca77bc6ad6e7e34d41ff8be6ccfe12ac26a155f77e259743a +
sed04bc4d23dd72db971692f94d3af77bcf6e55ada6501d94a3323e99460350e +
f54f3adac4128e4f199b5b7e1a7cdf45299a4c95052aa36732668ea +
e0f52c4b191d114b50828c298357c3f6 +
b30e0a61c16bc747846d6728b33f4a;
if (!a9e7f59a5b979b9eb278d98f074157f1f51f840a3f1ac3566f6f849f176664cd6736b5b50c) {b49c11f60b2e8d8f6a008cad04f82250a2d65753db24b2e7ad30.innerHTML = "An error occurred while trying to";
b49c11f60b2e8d8f6a008cad04f82250a2d65753db24b2e7ad30.innerHTML = "<span>{c838c1b0900e69aee66cd3c9cbdd5}</span>";} catch (error) {
b49c11f60b2e8d8f6a008cad04f82250a2d65753db24b2e7ad30.innerHTML = "Error...";};};, 1000);
```

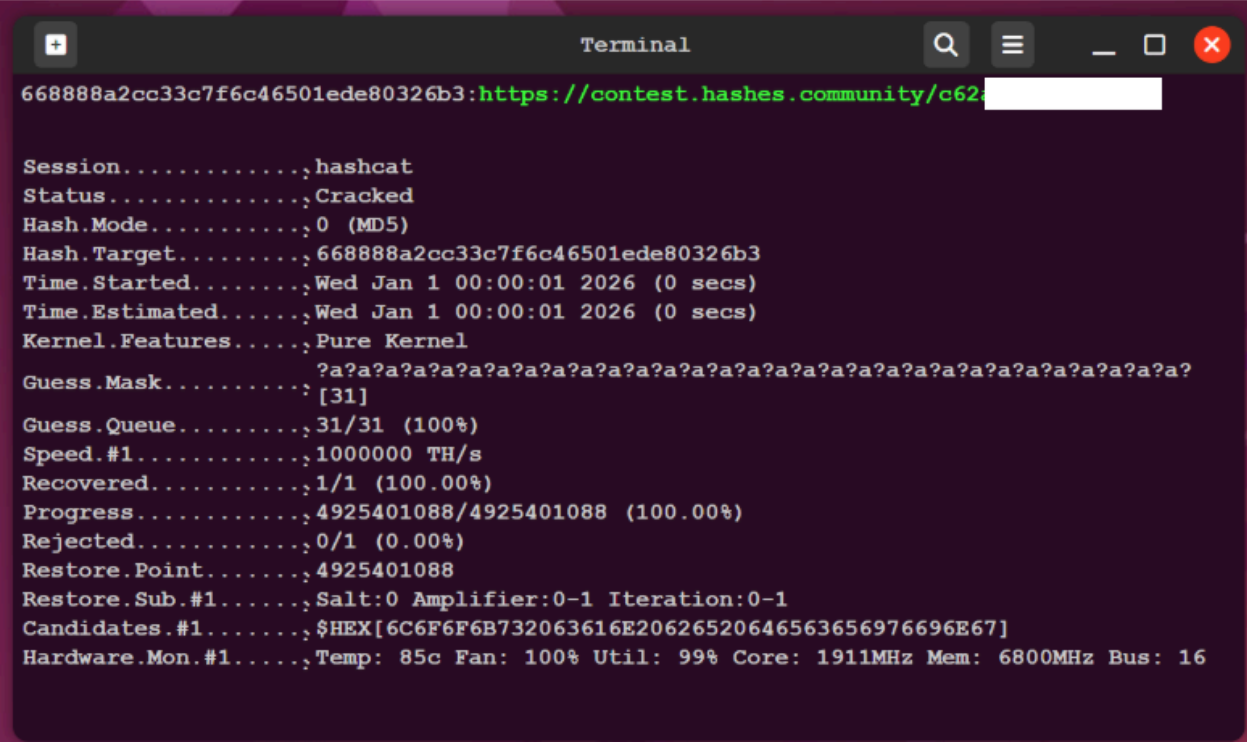
The first is to attempt to deobfuscate the JavaScript. By properly doing so you get pieces of string which include the URL of the next step. This is done through bitshifts, string replaces, and base64 encoding/decoding.

The second solution is slightly more elegant, but requires a good eye and possibly some inspection of the HTML / deobfuscated javascript source code. An observant eye might catch the "id" of the container being an í instead of an i. Replacing this doesn't perform any changes though. This is because the "container" text is multibyte as well (the "a" character specifically) and explains why the code would not function. This is of course by design as part of the CTF.

You can see the response being highlighted by the browser because an update is performed. Every second the obfuscated javascript tries to modify the text based on the container field.



As soon as the container id exists properly, the web-page updates. Displaying a link to the next location. Although this is a more elegant solution, it definitely isn't something that'll be quickly spotted and I think it's likely that relatively few people discovered this compared to the deobfuscation of the javascript.

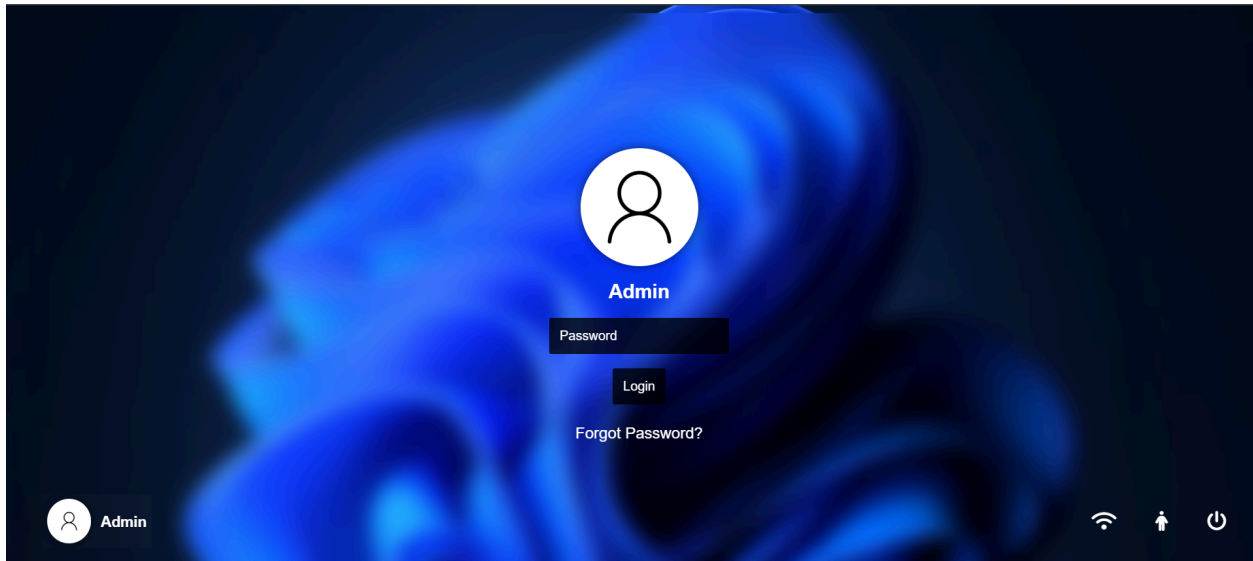


```
668888a2cc33c7f6c46501ede80326b3:https://contest.hashes.community/c62[redacted]

Session....., hashcat
Status....., Cracked
Hash.Mode....., 0 (MD5)
Hash.Target....., 668888a2cc33c7f6c46501ede80326b3
Time.Started....., Wed Jan 1 00:00:01 2026 (0 secs)
Time.Estimated....., Wed Jan 1 00:00:01 2026 (0 secs)
Kernel.Features....., Pure Kernel
Guess.Mask....., ?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?
[31]
Guess.Queue....., 31/31 (100%)
Speed.#1....., 1000000 TH/s
Recovered....., 1/1 (100.00%)
Progress....., 4925401088/4925401088 (100.00%)
Rejected....., 0/1 (0.00%)
Restore.Point....., 4925401088
Restore.Sub.#1....., Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....., $HEX[6C6F6F6B732063616E20626520646563656976696E67]
Hardware.Mon.#1....., Temp: 85c Fan: 100% Util: 99% Core: 1911MHz Mem: 6800MHz Bus: 16
```


Step 4

The fourth step presents us with a Windows Logon screen. Similar to before the only 'real' items are the password field and login button. Despite our best intentions, we were unable to shut down the machine using the power button.



One thing that ended up catching our eye with this login screen was the `<style>` tag near the top of the page. Including Hex colors that were not actively used, didn't correspond to their label, and multiple invalid colors. Providing 4 bytes of data as opposed to the usual 3 (R, G, and B). Concatenating each color + the 4 bytes for the non-rgb color yielded five 32-hex strings which we cracked as NTLM - all of which were in a hexadecimal keyspace.

```
a7d16ab30e04a3bb7db4690ab945c724:5440353  
a5078aeabeec95fac76eab27d9ba7e6d:7332D37  
4e7a2d7f4375cfc2ba25b97315e1291c:48332D5  
100f5c43598b27b074df6524aab325af:240314E  
7b11a47dfecbe3c27e5a97d4aec3325b:31333057
```

Concatenating and decoding them from hex => UTF8 resulted in a password which allowed us to successfully log in: T@573-7H3-R@1N130W.

```
<link rel="stylesheet" href="https://co
<style>
:root {
  /*Palette 1*/
  --pink: #a7d16a;
  --red: #b30e04;
  --orange: #a3bb7d;
  --yellow: #b4690a;
  --green: #b945c724;

  /*Palette 2*/
  --blue: #a5078a;
  --indigo: #eabeec;
  --violet: #95fac7;
  --purple: #6eab27;
  --turquoise: #d9ba7e6d;

  /*Palette 3*/
  --gold: #4e7a2d;
  --lime: #7f4375;
  --maroon: #cfc2ba;
  --navy: #25b973;
  --coral: #15e1291c;

  /*Palette 4*/
  --teal: #100f5c;
  --brown: #43598b;
  --white: #27b074;
  --black: #df6524;
  --sky: #aab325af;

  /*Palette 5*/
  --berry: #7b11a4;
  --grey: #7dfecb;
  --straw: #e3c27e;
  --silver: #5a97d4;
  --sapphire: #aec3325b;
}
```

Step 5

The fifth step in this saga leads us to a fake browser that has been hit with a form of SQL Injection. The password of the admin account appears exposed, but hashed. Similar to previous challenges this is not an actual vBulletin forum. We investigated the vbulletin version for exploits and made sure the server wasn't accidentally making itself vulnerable to more than might be intentional for the challenge (an RCE existed).

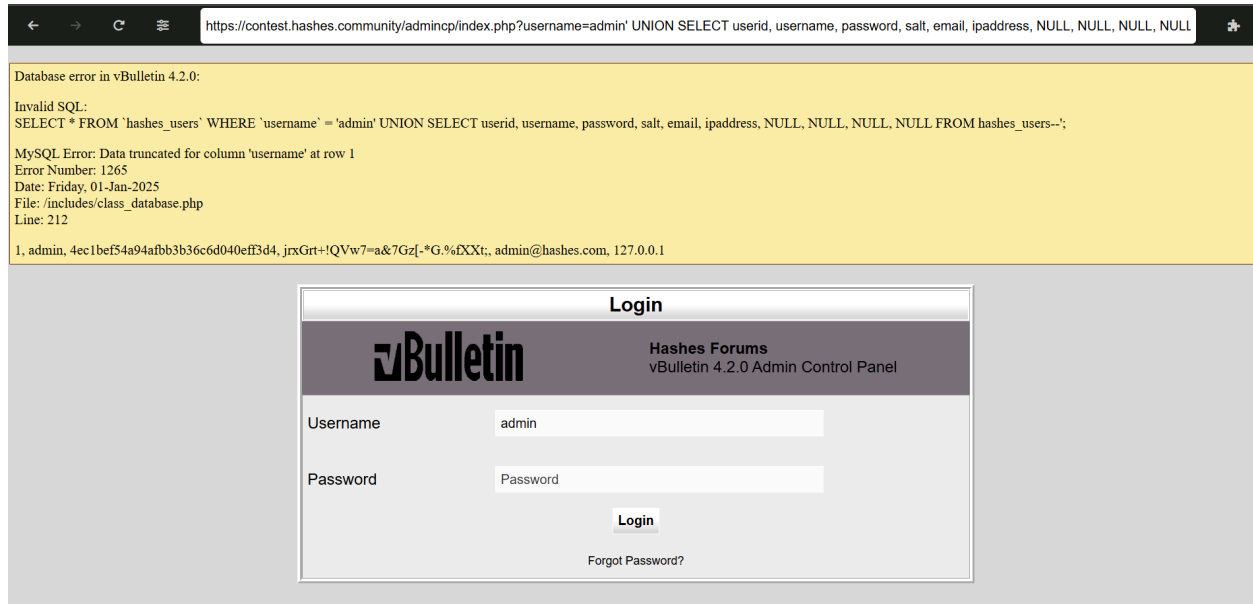
Our many years of experience working with hashes has given us a keen sense to spot what is and what isn't a valid algorithm. For example, the vbulletin hash displayed is not a normal vbulletin hash as the characters used in the salt and their distribution do not line up with how a salt is normally generated for vBulletin version $\geq 3.8.5$ (-m 2711). This was our first giveaway that the hash might potentially not be crackable - yet we still fired away just in case.

Want a personal challenge? Which of these hashes are not valid based on a quick glance?

```
$P$B3145678aIags59cHZiaQ4Phnbq7ac.  
$P$984lag476421S59wHZvyQMAr zfx58u<
```

The second giveaway was that if the 'forgot password' button was pressed, the hash would be updated to something we recognized as potentially valid.

Before password reset:

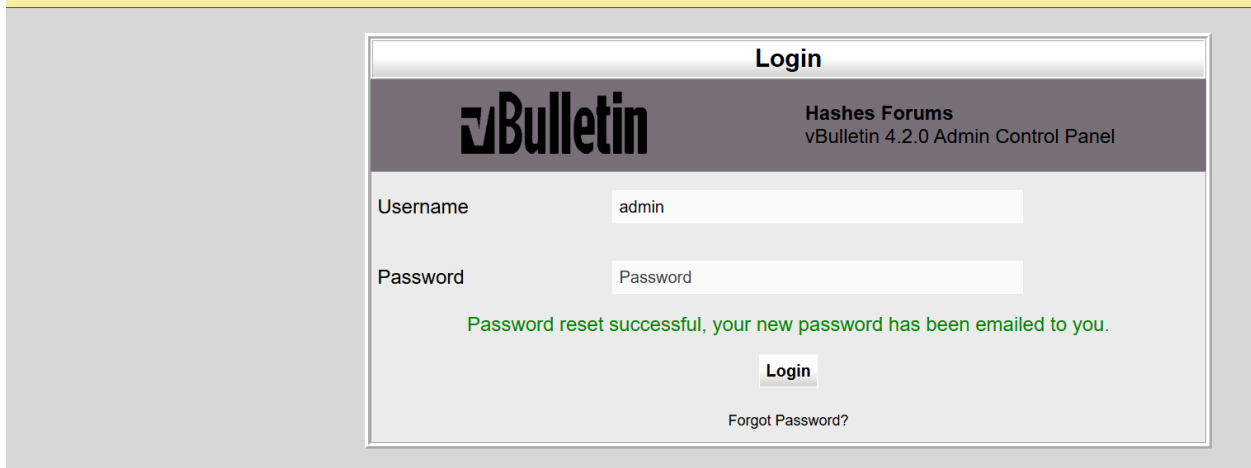


The screenshot shows a web browser window with the following content:

- Address bar: `https://contest.hashes.community/admindcp/index.php?username=admin' UNION SELECT userid, username, password, salt, email, ipaddress, NULL, NULL, NULL, NULL`
- Yellow error banner:
 - Database error in vBulletin 4.2.0:
 - Invalid SQL:
`SELECT * FROM 'hashes_users' WHERE 'username' = 'admin' UNION SELECT userid, username, password, salt, email, ipaddress, NULL, NULL, NULL, NULL FROM hashes_users--;`
 - MySQL Error: Data truncated for column 'username' at row 1
 - Error Number: 1265
 - Date: Friday, 01-Jan-2025
 - File: /includes/class_database.php
 - Line: 212
 - 1, admin, 4ec1bef54a94afb3b36c6d040eff3d4, jrxGrt+!QVw7=a&7Gz[-*G.%EXXt;, admin@hashes.com, 127.0.0.1
- Login form:
 - Header: **Login**
 - Logo: **vBulletin**
 - Text: **Hashes Forums**
vBulletin 4.2.0 Admin Control Panel
 - Username field:
 - Password field:
 - Buttons: **Login** and **Forgot Password?**

After password reset:

```
I, admin, 1665755dd738e948ad62a8a9fd423574, 9v?jY1:*NzBxKR327xG*!mh)YWke&T, admin@hashes.com, 127.0.0.1
```



The next part was a bit of a gamble but we decided to discard the first hash, with the principle that the 'reset password' was likely the right route. Especially with how we considered the first salt to be potentially invalid or unnatural.

We cracked this hash by looking how this might work in code. We pressed a forgot password button which 'emailed us our password'. This means it generated a new password randomly, and since the hash updated it might have reset it. Now all we need to do is guess the random password. We did this by looking up the logic vbulletin uses to reset the password.

The code fetches a random character X-times from a list of characters that is just shy of the full alphabet, missing letters such as "O" and "i".

```
function fetch_random_password($length = 8)
{
    $password_characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
    $total_password_characters = strlen($password_characters) - 1;

    $digit = vbrand(0, $length - 1);

    $newpassword = '';
    for ($i = 0; $i < $length; $i++)
    {
        if ($i == $digit)
        {
            $newpassword .= chr(vbrand(48, 57));
            continue;
        }
    }
}
```

```
        $newpassword .= $password_characters{vbrand(0,
$total_password_characters)};
    }
    return $newpassword;
}
```

This leads us to build a mask attack like so:

```
?d?1?1?1?1?1?1?1?1
?1?d?1?1?1?1?1?1?1
?1?1?d?1?1?1?1?1?1
?1?1?1?d?1?1?1?1?1
?1?1?1?1?d?1?1?1?1
?1?1?1?1?1?d?1?1?1
?1?1?1?1?1?1?d?1?1
?1?1?1?1?1?1?1?d?1
?1?1?1?1?1?1?1?1?d
```

With the ?1 being the definition of the keyspace. "ABCDEFGHJKLMNPQRSTUVWXYZabcdefghjkmnpqrstuvwxyz". Which will result in commands like so:

```
hashcat.exe -m2711 -1 ABCDEFGHJKLMNPQRSTUVWXYZabcdefghjkmnpqrstuvwxyz
hash.txt ?d?1?1?1?1?1?1
```

Eventually we cracked the password: "GrwJqR9h".

```
1665755dd738e948ad62a8a9fd423574:9v?jY1;*NzBxKR327xG*!mh)YWke&T:GrwJq
R9h
```

Step 6

Step 6 was a challenge where a fake Adminer login screen was presented to us.

Language:

Adminer 4.8.1

Login

System	<input type="text" value="MySQL"/>
Server	<input type="text" value="localhost"/>
Username	<input type="text"/>
Password	<input type="password"/>
Database	<input type="text"/>

Permanent Login

We first looked at the source code but didn't immediately see anything strange, so we attempted default credentials for Adminer and even attempted to look for potential vulnerabilities in Adminer, just in case there was any exploitable information there. After a few hours, we noticed the strangely large integrity tag on the import for the Javascript library "jquery" and decided to investigate further, as none of the other steps had this oddity. It was initially ignored, due to the integrity tag being perfectly standard in a HTML page. It was only after we had already noticed the oddity, that we realised the "i" in the "integrity" tag wasn't a normal "i" (\x69), but was instead a multibyte "i" (\xd1 \x96), which was completely invisible just from looking at the code, but a quick grep solidified that the integrity tag data was the hint we had to pursue.

```
$ curl https://contest.hashes.community/$URLSUFFIX | grep -P "[^\x00-\x7F]"
integrity= ...
```

The data part of the integrity tag appeared to be a normal sha256, base64 checksum, but when decoded, it had the "PNG" magic bytes of a PNG image. This was immediately suspicious, so we decoded the data from base64 to a raw file and opened it, sure enough, it was a tiny, 1x1, transparent pixel with the metadata:

```
jEWgj1FGxG9qkv31Vva7d5hm6ZtGhKbNkBWRR4LhiCYbqRhta76FJhCTW1XTzmhAVsTkxUX5Gv
46T74Ko18PeTxhfA8gYftN6Z8iyza7jNqxf5rmuGCgxysPnNcFxrZRpvkZysnF9AnDwrgJr2kcQ
zrJwDfK3m4xBC2HFqs7pytFotsigE7ksVCikxjXD9QuQJowQXCLkT2vEaaPoYcP8koWsmWGyB1
```

QWVCqz3DkQ29uYuVp1CtUdpyy2W2bbiL9ees1VLBzHzsM1zkgLJL8pBwwiY9mMnt5XjnP4q7cj
prFPtPGrhavUfmzqNmp5YCQH5L

This, we assumed to be yet more base64 but after decoding, it was just gibberish, we attempted to make a hashlist out of this, separating it into various sizes but to no avail. After a while of staring at it, we noticed that it didn't have the usual special characters of base64 - no = padding or "/" charset. This wasn't base64, but was base58, we decoded this and it became a simple hashlist for us to crack - what we're best at:

```
8022e5f626ed56c5998e462da731302ac0455fb74d5b4c0409c1938b7360acaa  
4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2  
*B5D43CAF943292E6F969AF02D1A7819F33F9B68B  
49402232bda4a590381634a60e34bee683a256db10210eb8bf66d69e8368637c
```

These hashes were fairly simple to identify, hashes 1,2 and 4 were 64hex so it was assumed to be SHA256 (but we kept our minds open for other algorithms), and hash 3 was a tell-tale MySQL4.1 / 5 hash, with the full capitalisation and the asterisk signature at the start. We quickly realised that these were finally the credentials for the adminer login page - hash 2 fell first, "root" - the username. We then realised that there is likely a domain or IP in there, for the "server" field, so that soon fell too, with hash 1 resolving to "216.58.214.163". Shortly after was the final sha256 hash, hash 4 - "hashes_database", we only needed the password. After an hour of scratching our heads, we realised that the strange hexadecimal URLs haven't been used yet and manually attempted the step 6 URL suffix, "9cfe.." and sure enough, it matched the remaining hash perfectly. We now had all of the fields we needed for the login page and were greeted with the next step's URL.

Language: English

Adminer 4.8.1

Login

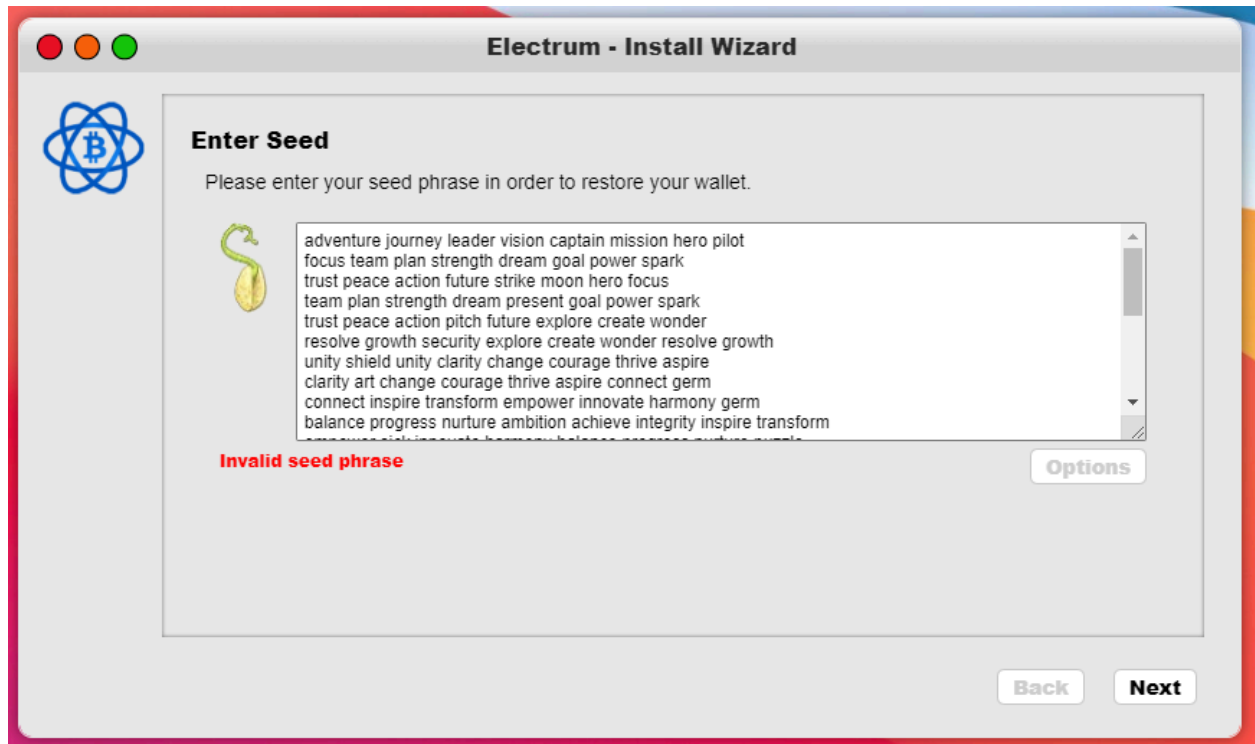
<https://contest.hashes.community/436>

System	MySQL
Server	216.58.214.163
Username	root
Password
Database	hashes_database

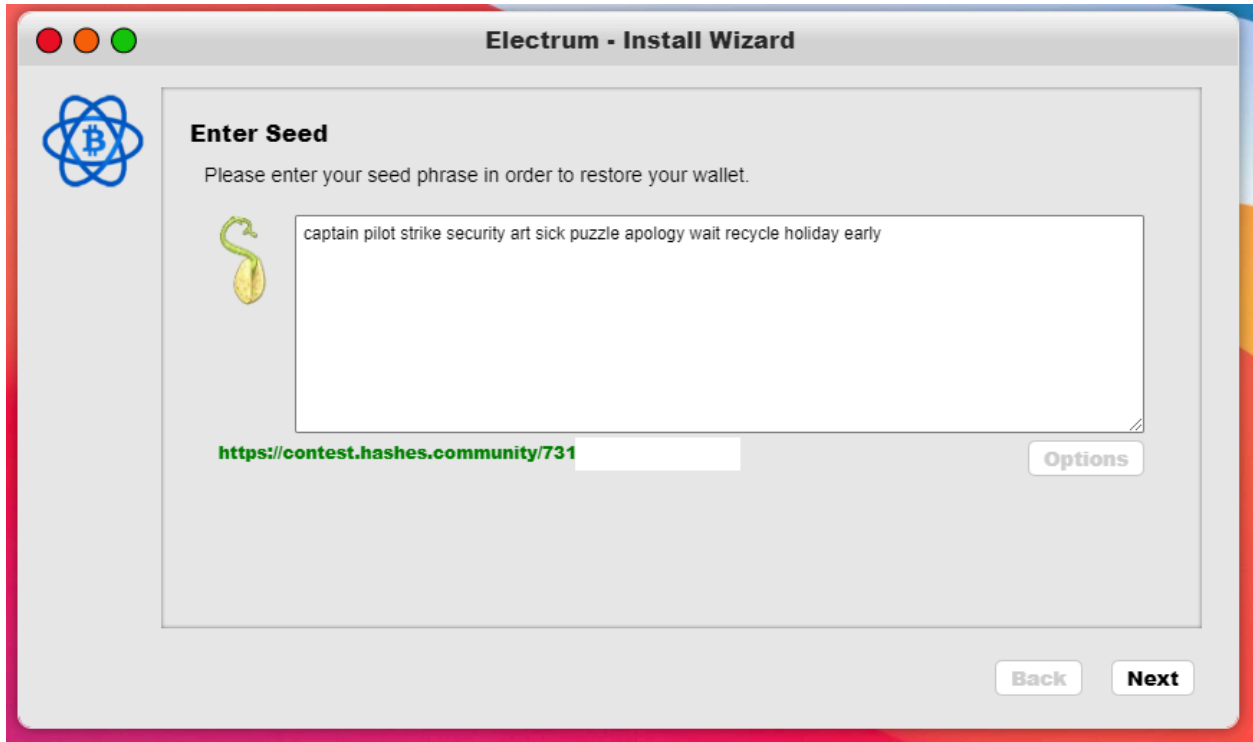
Permanent Login

Step 7

In Step 7, we were presented with a fake electrum wallet seed phrase wallet restoration prompt with the included text being way too many words to be BIP39-compatible.



Our initial approach was to check the frequency of each of the words, creating a histogram. Here we noticed there were duplicates and that exactly 12 words were included that did not have duplicates - rather interesting as most BIP39 seed phrases were 12 words long, but we didn't know what order they were in and we didn't want to flood the Hashes.com server with 12! (479,001,600) requests, so we simply tried these 12 in the order they were already in, with the placeholder text, as there was only 1 unique solution for this and sure enough - we were given Step 8's URL.



Step 8

The 8th challenge was to crack 15 different passwords encrypted with Joomla MD5 (-m400). Each user had metadata associated with them, simulating a real database.

Download

USERNAME	FIRST NAME	LAST NAME	EMAIL ADDRESS	DOB	CITY	STATE	ZIP	COUNTRY	OCCUPATION	PASSWORD
john doe90	john	doe	john.doe90@hashes.com	1990-05-16	Springfield	IL	62701	USA	Software Developer	\$P\$BtzNkKeirwzgpu/h6lakYfe5Z.le1p/
sconnor1984	sarah	connor	sarah.connor1984@hashes.com	1984-11-13	Boston	MA	02108	USA	Marketing Specialist	\$P\$Bu8xmuKPTvNhCaZ2irb78j/CeVIRU80
mjordan63	mike	jordan	mjordan63@hashes.com	1963-02-17	Chicago	IL	60614	USA	Sports Coach	\$P\$B7Xe3c/mVQ41Kwz6fdtVp2x9N2QqOE/
emilyrose92	emily	rose	emily.rose92@hashes.com	1992-09-21	Seattle	WA	98104	USA	Graphic Designer	\$P\$BKInC8zqVK9dgnYiPaspWmWb7mJhmv/
dking76	david	king	david.king76@hashes.com	1976-03-09	Denver	CO	80203	USA	Tech Engineer	\$P\$BcqoNw3JwMkrvFcANao/DqdP9EM5Bn/
jessbrown88	jessica	brown	jess.brown88@hashes.com	1988-07-22	Austin	TX	73301	USA	Fitness Trainer	\$P\$BjleeA8ctrvCnojgwW40bG5B/B6J711
robertjames70	robert	james	robert.james70@hashes.com	1970-12-15	Phoenix	AZ	85001	USA	Historian	\$P\$BrtwfbUe3Wz298FOffTueXggJY1uw/
samanthac95	samantha	carter	samantha.carter95@hashes.com	1995-04-12	Atlanta	GA	30301	USA	Creative Director	\$P\$BJT4YcY48qnaqm9NQxhcri/OJwWSF.
chris85evans	chris	evans	chris.evans85@hashes.com	1985-08-05	Orlando	FL	32801	USA	Film Critic	\$P\$Bz3by09o669E/vj9mieS3ZGBfkyG.
llewis81	laura	lewis	laura.lewis81@hashes.com	1981-06-18	San Diego	CA	92101	USA	Environmental Scientist	\$P\$BAH93rk6f.UO2qm9gca37IO4bxH7G1
mattdavis93	matt	davis	matt.davis93@hashes.com	1993-03-30	Miami	FL	33101	USA	Tech Geek	\$P\$B/24kckw5e7TAG4ax28BYAcoppJgB1
annasmith87	anna	smith	anna.smith87@hashes.com	1987-11-02	Houston	TX	77001	USA	Baker	\$P\$BilykrStaNcJgfoE.3jNZ6ofzA9LB0
jwhite80	jason	white	jason.white80@hashes.com	1980-01-19	Dallas	TX	75201	USA	Music Producer	\$P\$BndKmN6Pia3Aa71O.hCWUAAmfP5bi0
lmill94	lisa	milller	lisa.milller94@hashes.com	1994-07-26	Nashville	TN	37201	USA	Bookworm	\$P\$BIZHEMwMxWYp8fqQLkrtYfZNd0
pharris72	paul	harris	paul.harris72@hashes.com	1972-02-14	San Francisco	CA	94101	USA	Traveler	\$P\$B4ECfw280DLLGw9GE0C83J9Q7.Y7at0

By extracting all the data in CSV columns and manipulating them we created combinations of their username with rules, their dates of birth and combined them with each other to get large and complex passwords. We used tools like combinator.exe from hctools and a custom tool

Vavaldi developed at one point for another HashMob user to create combinations between metadata values.

```
$P$B/24kdckw5e7TAg4ax28BYAcqppJgB1:1993-03-30MiamiFL
$P$B4ECFw280DLGw6GE0C63J9Q7.Y7at0:USATravelerpharris72
$P$B7Xe3c/mVQ41Kwz6fdIVp2x9N2QqOE/:jordan
$P$BAH93rk6f..UO2qm9gca37iO4txH7G1:laura.lewis811981-06-18San Diego
$P$BcqnW3JwMkrvFcANao/DqdP9EM5Bn/:1976-03-09
$P$BJ/eeA6ctxvCnojgwW4ObG5B/B6J711:Austin
$P$BJT4YaCY48qnaqm9NQxhcri/OJwWSF.:samanthacartersamantha.carter95
$P$BKINc8zqVK9dgNyiPsaspWmWb7mJhm/:emily.rose92
$P$BlbykrStaNcJgfoE.3.jNZ6ofza9LB0:HoustonTX77001
$P$BndKmN6Pia3Aa71O.hCWUAAmfPt5bi0:TX75201USA
$P$BrTwbUe3W8Zj86F0fTueXqg.jY1uw/:robertjames70robertjames
$P$BtZHEMwfMixWyp6fqQLkrNylfZN/db0:37201USABookworm
$P$BtzNkKeirwzgpu/h6IakYfe5Z.Ielp/:johndoe90
$P$Bu8xmuKPTvNhCaZ6irb78/jCeVIRU80:sarah
$P$Bz3by09o669E/vfj9mieS3ZGBfklyG.:evanschris.evans851985-08-05
```

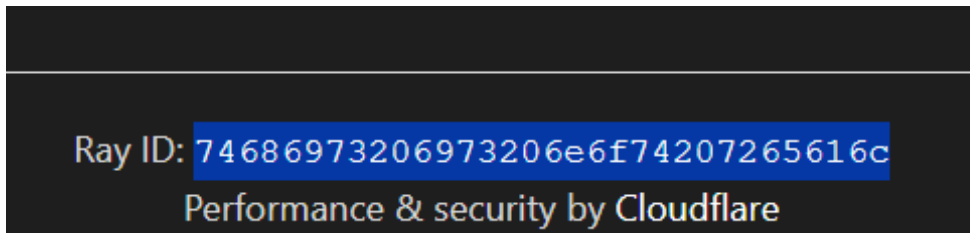
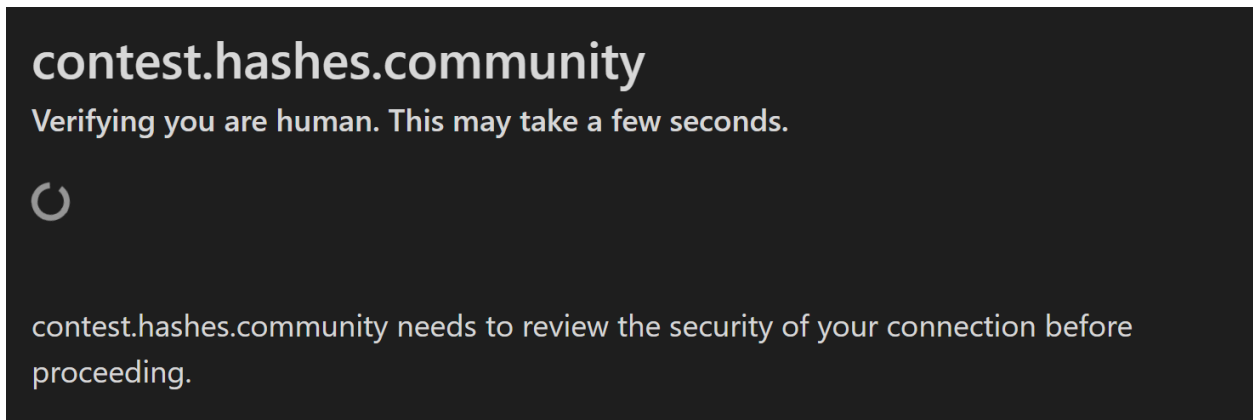
Entering all these in their correct field returns the next URL

johndoe90's password <input type="password"/>	sconnor1984's password <input type="password"/>	mjordan63's password <input type="password"/>
emilyrose92's password <input type="password"/>	dking76's password <input type="password"/>	jessbrown88's password <input type="password"/>
robertjames70's password <input type="password"/>	samanthac95's password <input type="password"/>	chris85evans's password <input type="password"/>
llewis81's password <input type="password"/>	mattdavis93's password <input type="password"/>	annasmith87's password <input type="password"/>
jwhite80's password <input type="password"/>	lmill94's password <input type="password"/>	pharris72's password <input type="password"/>

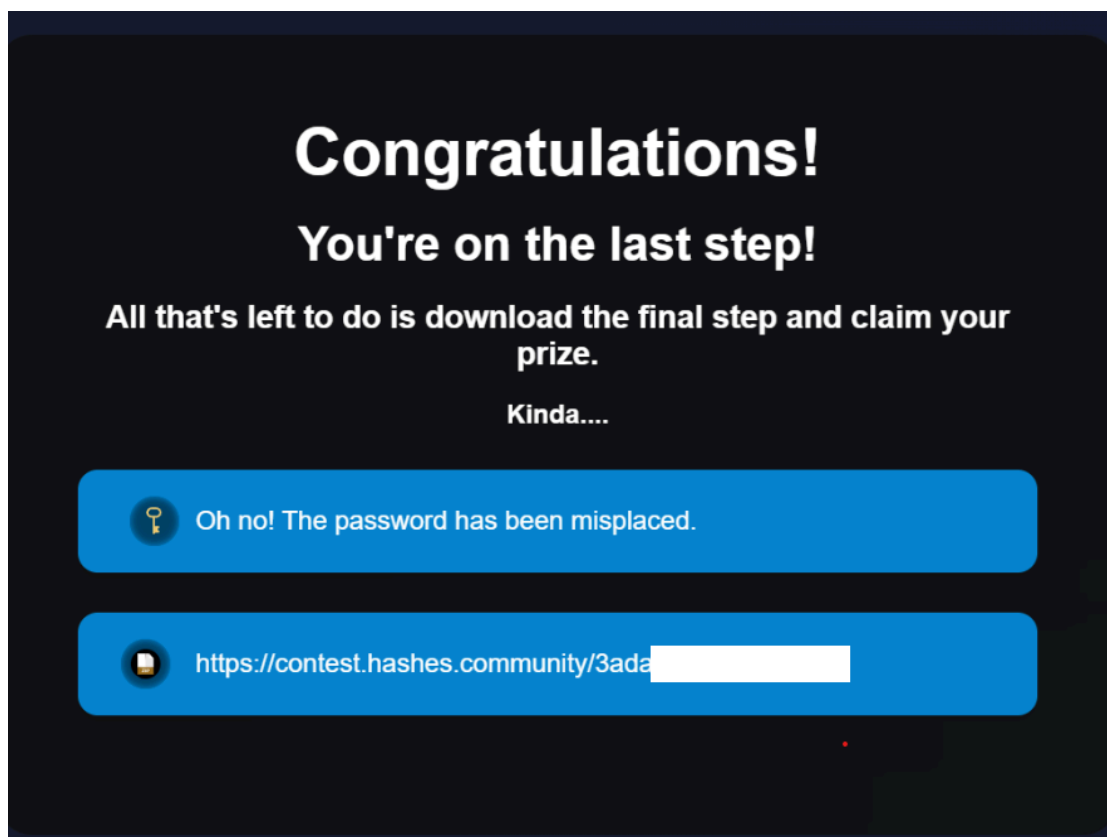
<https://contest.hashes.community/41d0ebf2cac066be>

Step 9

This step involved a repeating cloudflare loading page that kept loading over and over. One thing that stood out here was that the Ray ID was longer than usual, almost looking like a hash. This was confirmed when we recognized the bytes being in ASCII keyspace and decoding the value "74686973206973206e6f74207265616c" to get the text: "this is not real".



This leads us to inspect the source code a little closer. In doing so we discover a trigger in the cx() function that only happens if a certain localStorage value is set, disabling the loading page and revealing another website. By setting the item "uam" in local storage with the value "off" the new page is revealed.



Step 10

The last step of the puzzle leads to a .zip file download. The file is protected by a password and as we start cracking away at the hashes we tried to extract with zip2john we spot that the hint “Oh no! The password has been misplaced.” has a key as logo. We decide to investigate the SVG’s further and discover that there’s an image hidden in the <https://contest.hashes.community/key.svg>, key.svg file with 0 opacity. When setting the opacity to 1 it reveals itself.



It's a small Ethereum logo at the end of the key.

Hashes.com / HashKiller.io Contest 2025 writeup by Vavaldi, Penguinkeeper, Shooter3k

Looking inside the XML of the SVG file we observe the previously hidden logo has words like “blockHeight” which seem out of place for an SVG object about Ethereum.

```
<g xmlns="http://www.w3.org/2000/svg" fill-opacity="1"
transform="scale(0.1) translate(2500,3600)" type="gama"
blockHeight="21539263" dominant-baseline="auto" x="30" y="145"
txn="0.00023524" tabindex="2">
```

Although it proves a bit of a challenge to find the right transaction with the correct gas fee we finally locate it. A transaction processed at block height of 21539263 and a Gas fee of 0.000235249474719 which matches the txn and is on the Ethereum blockchain.

<https://www.blockchain.com/explorer/transactions/eth/0xba8b6f3d8e866e4959ac289f3ada300c5092bab6d571a5adc3f6aab8dadb7615>

In the meanwhile we struggled with collisions in the zip archive, with hashcat providing us a plethora of “valid” passwords that decrypt data for one file and causes corruption in the process. Not giving us the ‘correct file’. We tried a LOT of different encryptions and techniques in the hopes of making sense out of the corrupted data. This took up a lot of our time at this stage of the contest. We then take note of the fact that “0.000235249474719” falls almost entirely within the printable ascii range, except for the 19 at the end. Decoded it reads: “#RIGG!!”.

Unfortunately this is where the contest ended for us. With us not progressing further than this potential hint. After the contest received the hint that the solution to this last problem was based on a vulnerability in the legacy zip encryption ZipCrypto / PKWARE. This known-plaintext vulnerability allowed you to easily crack open ZIP archives if you knew at least 12 bytes of one of the files and they were encrypted using the ‘store’ setting or without compression (in the case of encryption you would need to compress your target bytes first). To help us with the entire process we will make use of the BKCrack tool: <https://github.com/kimci86/bkcrack>.

The known plaintext for us is the target hash of the transaction: The ethereum wallet address. This is likely to be stored inside the wallet.json file and could be guessed by us.

```
PowerShell 7.4.6
PS C:\Users\Admin\Downloads\bkcrack-1.7.1-win64> .\bkcrack.exe -L .\backup.zip
bkcrack 1.7.1 - 2024-12-21
Archive: .\backup.zip
Index Encryption Compression CRC32 Uncompressed Packed size Name
-----
0 ZipCrypto Store a705d1f0 30 42 password.txt
1 ZipCrypto Store 959dd083 499 511 wallet.json
PS C:\Users\Admin\Downloads\bkcrack-1.7.1-win64> |
```

We notice how the compression is “Store” and the encryption is “ZipCrypto” and this tells us that the zip file is vulnerable to exploitation.

```
PS C:\Users\Admin\Downloads\bkcrack-1.7.1-win64> echo a564881458a1be24223ea2cae64f7047749fdff5 >plain.txt
PS C:\Users\Admin\Downloads\bkcrack-1.7.1-win64> .\bkcrack.exe -C '.\backup.zip' -c wallet.json -p plain.txt -o 0
bkcrack 1.7.1 - 2024-12-21
[20:07:17] Z reduction using 35 bytes of known plaintext
100.0 % (35 / 35)
[20:07:17] Attack on 220116 Z values at index 6
5.5 % (
```

Now this approach could work by slowly increasing the -o parameter one by one. But that can take a while. An alternative is to guess the entire structure, including potential markup (spaces/tabs/newlines/carriage returns). Starting off with the luckiest guess would lead you to try: “{“address”:”a564881458a1be24223ea2cae64f7047749fdff5” which would match the first 12+ bytes of the file and give you the “keys”, which can then be used to decrypt both files: wallet.dat and password.txt. Where password.txt is the password of the encrypted wallet.dat file.

```
PS C:\Users\Admin\Downloads\bkcrack-1.7.1-win64> .\bkcrack.exe -C '.\backup.zip' -c wallet.json -p plain.txt -o 0
bkcrack 1.7.1 - 2024-12-21
[20:16:13] Z reduction using 46 bytes of known plaintext
100.0 % (46 / 46)
[20:16:13] Attack on 171995 Z values at index 6
Keys: 5a49324c 014f2d86 1b052ea3
50.2 % (86387 / 171995)
Found a solution. Stopping.
You may resume the attack with the option: --continue-attack 86387
[20:18:07] Keys
5a49324c 014f2d86 1b052ea3
```

Finally the command: `bkcrack -C encrypted.zip -k 12345678 23456789 34567890 -U unlocked.zip new_password` can be used to decrypt the file and successfully recover the public and private key of the wallet that originally contained the 2.5k reward money. We took 1\$ left in the wallet and left a tiny bit in for the next team reaching the end.

Closing

The challenges were creative in nature and really challenged us in some cases. Eventually stumping us on the last one. We spent nearly 7 hours stuck on that portion of the challenge until the contest was won. Congratulations to DDNK and his teammates for winning the contest, and thanks to Hashes.com for organizing this event with a lofty prize. We look forward to seeing future contests.