After examining the source code, the biggest issue we have is escapeshellcmd, as it filters out most characters. However, as with most things, there must be a workaround. After a bit of research, we can find that the useful payload is simply this:

```
sth -or -exec cat /etc/passwd ; -quit
```

After testing it out, we see we are getting output, so we need to get to the 2nd part of this chall, bruteforcing the toupper conversion.

First, I wanted to execute ls, so we need to bruteforce a short string. I effectively printed only results which only contain printable chars, spaces and /n, after which I noticed trends and filtered by them (first that the website ends in .php, and that it's named index.php). Then, we can find out the contents of the current directory are index.php and flag. So we do "cat flag" and get a pretty long string:

Y3RMEZM4MGEXYZAYMGM0NGYYZJI5MJNINTCYMTE1ZTK1MJK0YMI2MJJMNGRLMZI4NJRJZJNJZTNHNTI0YJQWZTU2N2R9

To solve this, we need to keep into account what we know

1. the flag starts with ctf{ and ends with }

2. the flag contains only 0-9 and a-f

3. base64 effectively turns 3 characters into 4

Taking all that into account, we can make a deterministic script that decodes the touppered b64:

```python
import base64
import itertools

encoded = "Y3RMEZM4MGEXYZAYMGM0NGYYZJI5MJNINTCYMTE1ZTK1MJK0YMI2MJJMNGRLMZI4NJRJZJNJZTNHNTI0YJQWZTU2N2R9"
hex_chars = set("0123456789abcdef")
expected_length = 69

def decode_chunkwise(encoded):
    chunk_size = 4
    decoded_result = b""
    for i in range(0, len(encoded), chunk_size):
        chunk = encoded[i:i+chunk_size]
        permutations = map("".join,itertools.product(*((c.lower(), c.upper()) for c in chunk)))
        for permuted_chunk in permutations:
            try:
```

```
                test_decoded = base64.b64decode(permuted_chunk)
                if i == 0:
                    if test_decoded.decode() != "ctf":
                        continue

                elif i == chunk_size:
                    if not (test_decoded[0] == ord("{") and
                            all(chr(c) in hex_chars for c in test_decoded[1:3])):
                        continue

                elif i < len(encoded) - chunk_size:
                    if not all(chr(c) in hex_chars for c in test_decoded):
                        continue

                elif i == len(encoded) - chunk_size:
                    if not (all(chr(c) in hex_chars for c in test_decoded[:2])
and
                            test_decoded[2] == ord("}")):
                        continue
                decoded_result += test_decoded
                break
            except Exception:
                continue
        else:
            return None

    return decoded_result

result = decode_chunkwise(encoded)
print(result.decode())
```

Made with love by: AndreiCat