Files

main

Go to file

1337UP_2022
UTCTF_2022
VishwaCTF_2022
Cryptography/John_the_Rocker
files
idrsa.id_rsa.docx
idrsa.id_rsa.txt
img
README.md
Forensic
So_Forgetful
files
Ghost.pcap
img
README.md

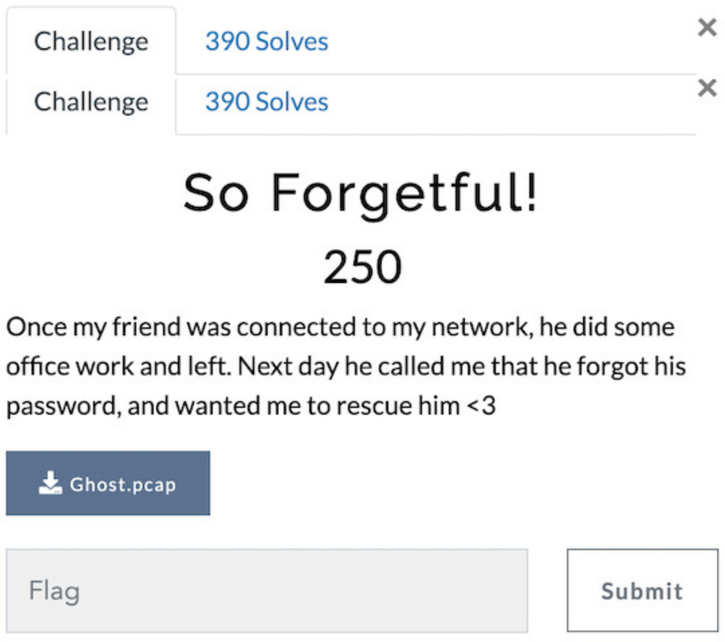
LambdaMamba Added writeup for VishwaCTF So Forgetful 8e4fe15 · 2 years ago History

Name	Last commit message	Last commit date
..		
files	Added writeup for VishwaCTF So Forgetful	2 years ago
img	Added writeup for VishwaCTF So Forgetful	2 years ago
README.md	Added writeup for VishwaCTF So Forgetful	2 years ago

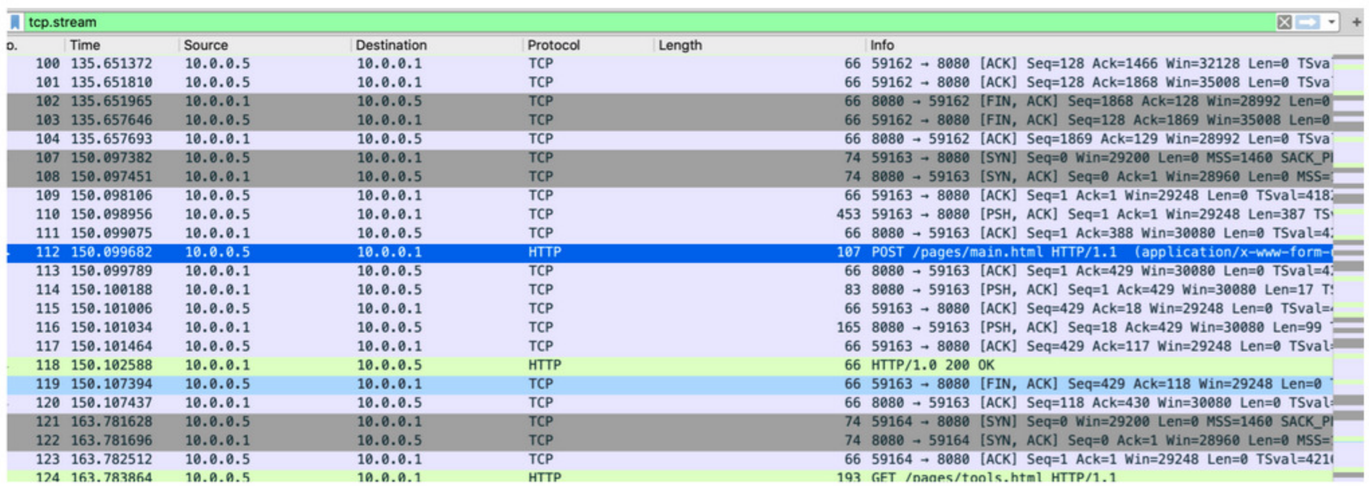
README.md

So Forgetful! (Category: Forensic)

The challenge is the following,

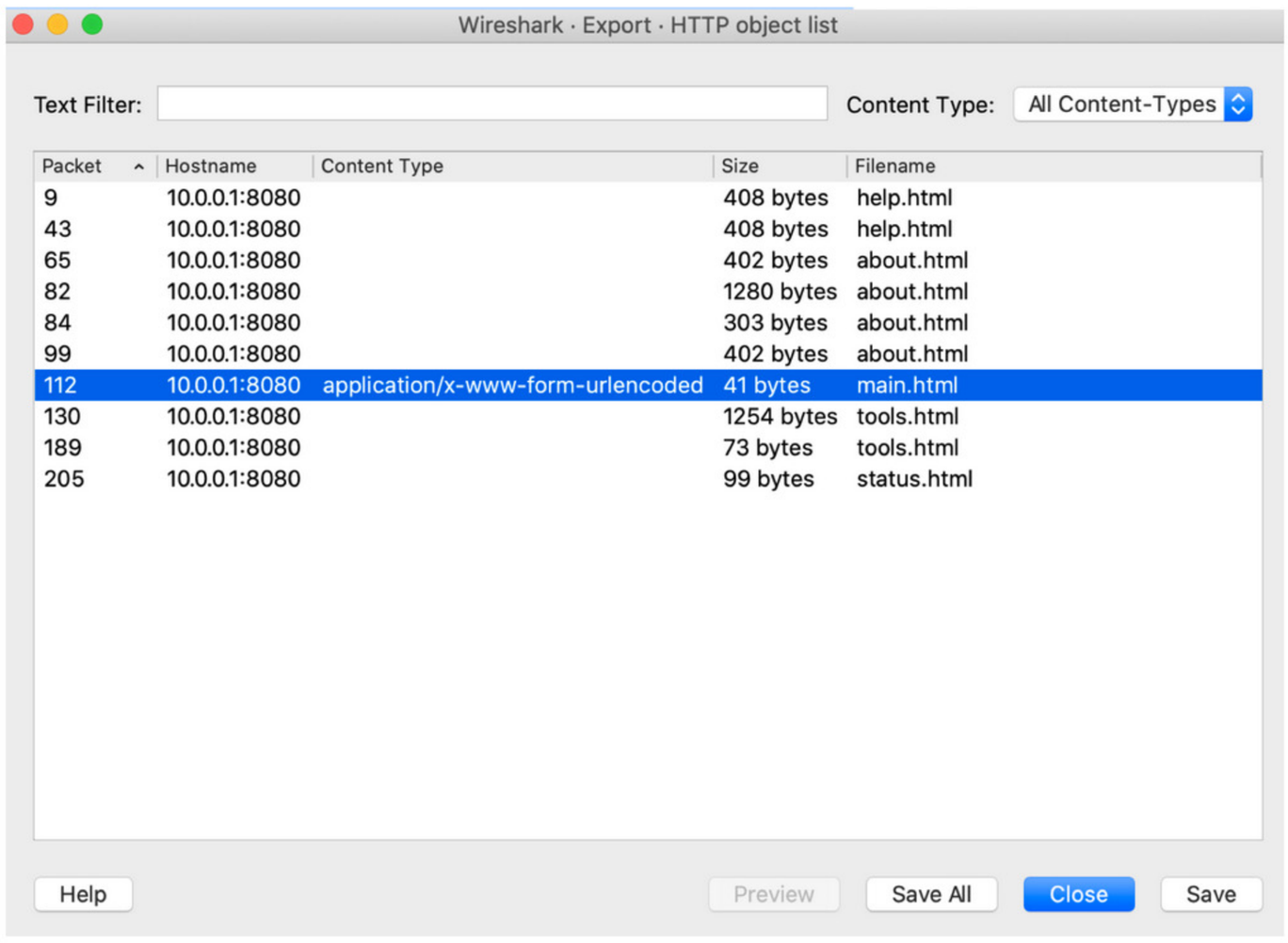


And we are given the file Ghost.pcap. I used `tcp.stream` in the filters, and looked at the packets. Packet 112, which uses `HTTP` caught my attention because it uses `POST`. I looked into that packet and saw the following,




```
0000  08 00 27 3d 47 5d 08 00  27 38 2c 5c 08 00 45 00   ··'=G]·· '8,\··E·
0010  00 5d ec 95 40 00 40 06  3a 00 0a 00 00 05 0a 00   ·]··@·@· :·······
0020  00 01 e7 1b 1f 90 e9 d7  8e c2 65 ad ce d8 80 18   ········ ··e·····
0030  01 c9 d2 3f 00 00 01 01  08 0a 00 06 61 ad 00 06   ···?···· ····a···
0040  7f d8 75 73 65 72 69 64  3d 73 70 69 76 65 79 70   ··userid =spiveyp
0050  26 70 73 77 72 64 3d 53  30 34 78 57 6a 5a 51 57   &pswrd=S 04xWjZQW
0060  46 5a 35 4f 51 25 33 44  25 33 44                  FZ5OQ%3D %3D
```

I went to `Export Objects > HTTP` and selected packet 112,



And I saved it as a .txt text file, which contained,

```
userid=spiveyp&pswrd=S04xWjZQWFZ5OQ%3D%3D
```

I couldn't find anything information related to the website where these credentials were used, and the challenge didn't specify any server to log in to. Therefore, I assumed that the password `S04xWjZQWFZ5OQ%3D%3D` itself would be the flag (after processing it more). The `%3D%3D` at the end of the flag looked like Base64 in URL,

> **The URL applications** [edit]
>
> Base64 encoding can be helpful when fairly lengthy identifying information is used in an HTTP environment. For example, a database persistence framework for Java objects might use Base64 encoding to encode a relatively large unique id (generally 128-bit UUIDs) into a string for use as an HTTP parameter in HTTP forms or HTTP GET URLs. Also, many applications need to encode binary data in a way that is convenient for inclusion in URLs, including in hidden web form fields, and Base64 is a convenient encoding to render them in a compact way.
>
> Using standard Base64 in URL requires encoding of '`+`', '`/`' and '`=`' characters into special percent-encoded hexadecimal sequences ('`+`' becomes '`%2B`', '`/`' becomes '`%2F`' and '`=`' becomes '`%3D`'), which makes the string unnecessarily longer.
>
> For this reason, **modified Base64 for URL** variants exist (such as **base64url** in RFC 4648), where the '`+`' and '`/`' characters of standard Base64 are respectively replaced by '`-`' and '`_`', so that using URL encoders/decoders is no longer necessary and has no effect on the length of the encoded value, leaving the same encoded form intact for use in relational databases, web forms, and object identifiers in general. A popular site to make use of such is YouTube.[11] Some variants allow or require omitting the padding '`=`' signs to avoid them being confused with field separators, or require that any such padding be percent-encoded. Some libraries[which?] will encode '`=`' to '`.`', potentially exposing applications to relative path attacks when a folder name is encoded from user data.

So I converted it to the standard Base64 format by replacing `%3d` with `=`, to give,

```
S04xWjZQWFZ5OQ==
```

Finally, I went to CyberChef to convert it from Base64, which gave,

```
KN1Z6PXVy9
```

Therefore, I assumed that this would be the flag,

```
vishaCTF{KN1Z6PXVy9}
```

And submitting it confirmed that this was the flag.