🔐 Internal Lab – Crypto & Secret Retrieval via Docker

Project ID: Project 4

Assessed by: Muhammad Talha

Environment: Kali Linux VM (VMware), Docker, Local Lab Setup

Date: 24 June, 2025

---

1. Executive Summary

This assessment focused on decoding and decryption tasks simulating real-world cryptography-related misconfigurations. The test scenarios included decoding Base64 strings, brute-forcing XOR-encrypted content, extracting and signing RSA key data, and retrieving embedded secrets from a Docker container image. Each step replicated common weaknesses developers may introduce during insecure secret handling or insufficient encryption practices.

2. Tools Used

Tool	Purpose
Kali Linux	Main assessment platform
Docker	Run challenge container
OpenSSL	Decryption and key signing
xxd	Hex conversion and binary manipulation
base64	Base64 decoding
Custom Bash Scripts	Brute-force XOR key recovery

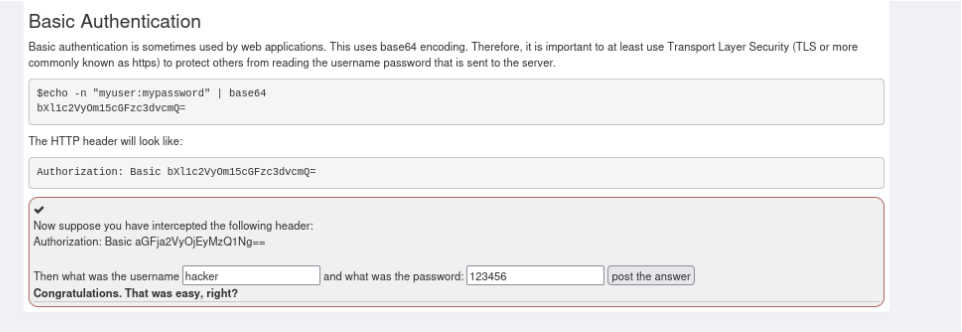
3. Tests Performed

3.1 Base64 Decoding

A Base64 string was decoded using the Linux base64 utility. This test simulated weak encoding of credentials or secrets in configuration files.

```
┌──(kali㉿kali)-[~]
└─$ echo aGFja2VyOjEyMzQ1Ng==
aGFja2VyOjEyMzQ1Ng==

┌──(kali㉿kali)-[~]
└─$ echo aGFja2VyOjEyMzQ1Ng== | base64 -d
hacker:123456
```



3.2 XOR Cipher Decoding

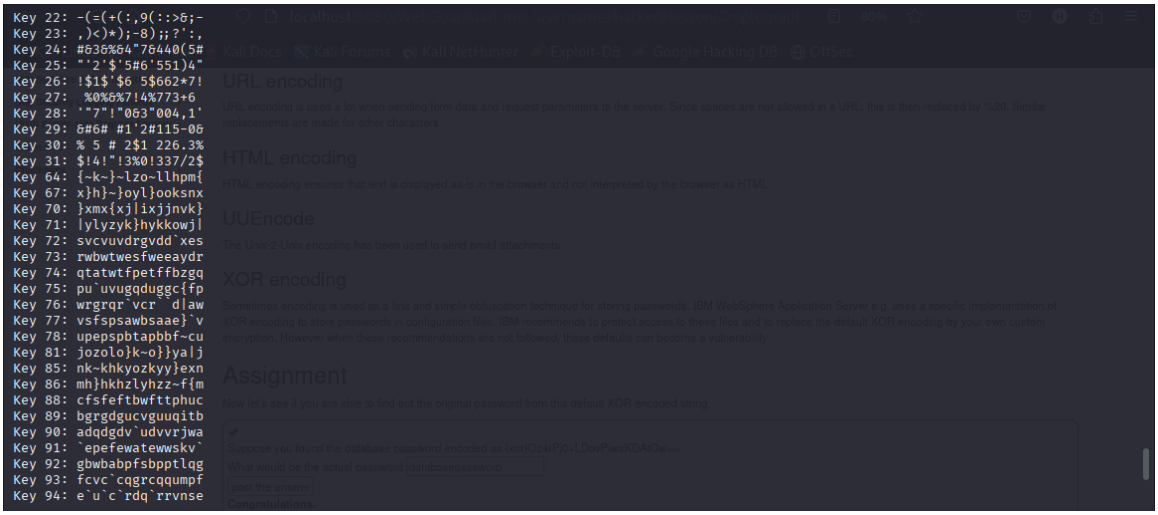
A Base64-encoded XOR-obfuscated string was decoded by applying a brute-force XOR key search across all 255 possibilities. Successful key identification revealed the hidden password.

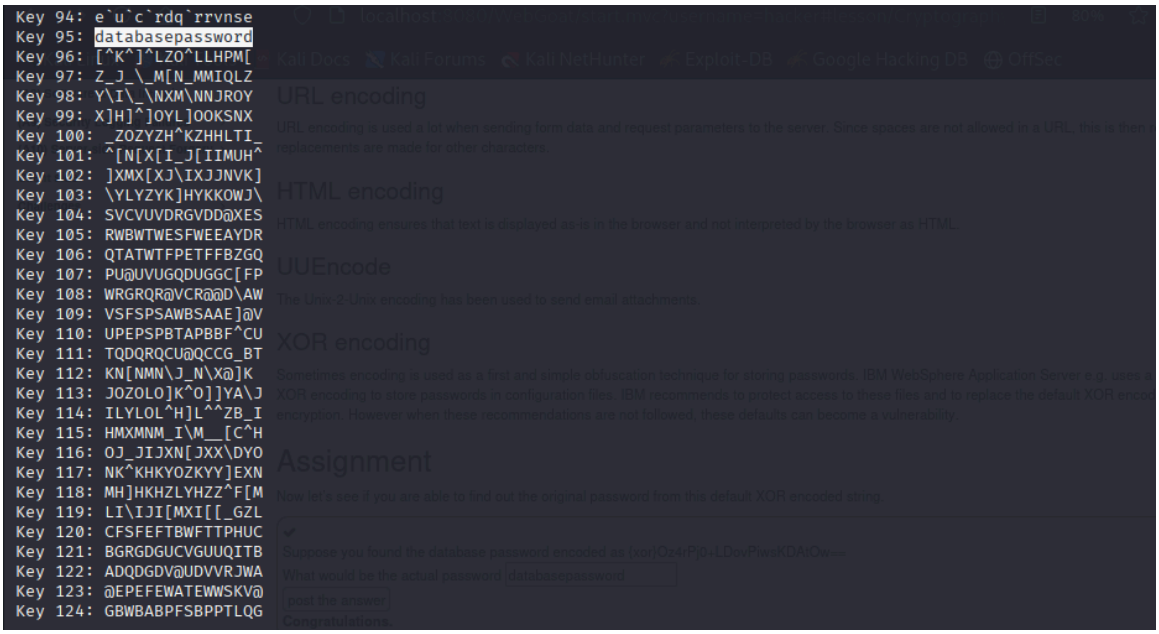
```
┌──(kali㉿kali)-[~]
└─$ echo Oz4rPj0+LDovPiwsKDAtOw== | base64 -d
;>+>⇒,:/>,,(0-;
```

```
┌──(kali㉿kali)-[~]
└─$ echo Oz4rPj0+LDovPiwsKDAtOw== | base64 -d > xor.raw
```

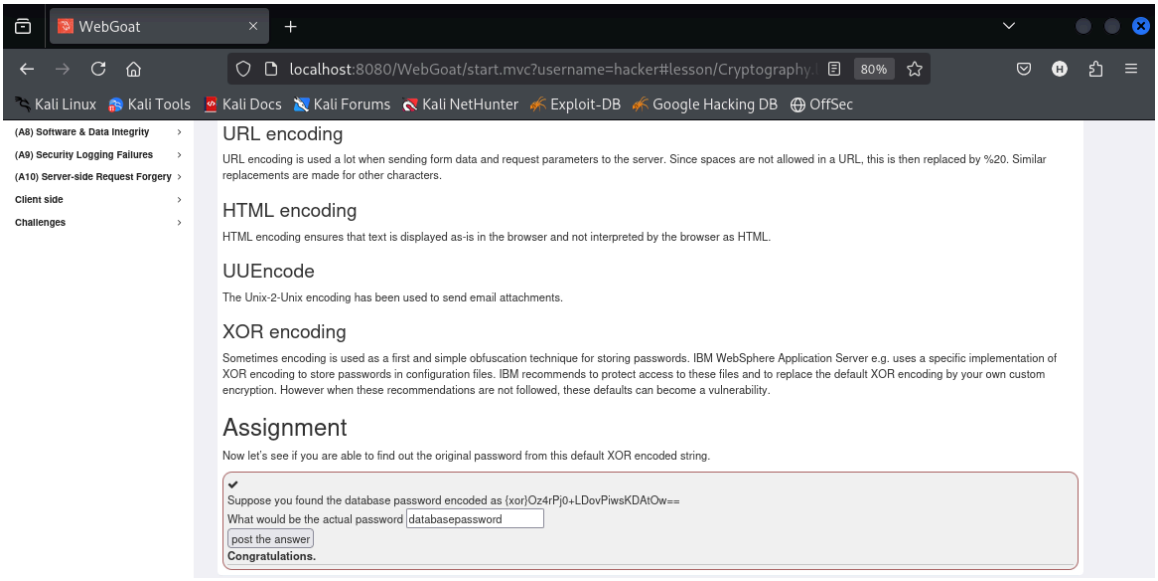
```
kali@kali: ~
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~]
└─$ for key in {1..255}; do
  result=$(cat xor.raw | perl -pe "s/(.)/chr(ord(\$1)^$key)/ge")
  if echo "$result" | grep -qP '^[ -~]{6,}$'; then
    echo "Key $key: $result"
  fi
done
Key 1: :?*?<?-;.?--)1,:
Key 2: 9<)<?<.8-< ..*2/9
Key 3: 8=(⇒⇒/9,⇒//+3.8
Key 4: ?:/:9:(>+:((,4)?
Key 5: >;.;8;)?*;))-5(>
Key 6: =8-8;8*<)8**.6+=
Key 7: <9,9:9+=(9++/7*<
Key 8: 36#656$2'6$$ 8%3
Key 9: 27"747%387%%!9$2
Key 10: 14!4746O%46":'1
Key 11: 05 565'1$5''#;60
Key 12: 72'212 6#2 $<!7
Key 13: 638303!7"3!!%= 6
Key 14: 50%030"4!0""6>#5
Key 15: 41$121#5 1##'?"4
Key 16: +.;.-.<*?.<<8 =+
Key 17: */:/,/=+>/==9!<*
Key 18: ),9,/,>(=,>>:"?)
Key 19: (-8-.-?)←?;#>(
Key 20: /*?*)*8.;*88<$9/
Key 21: .+>+(+9/:+99=%8.
Key 22: -(=(+(:,9(::>6;-
```


Key 22: -(=(+(:,9(::>6;-
Key 23: ,)<)*);-8);;?':,
Key 24: #636%64"76440(5#
Key 25: "'2'$'5#6'551)4"
Key 26: !1'$6 5$662*7!
Key 27: %0%6%7!4%773+6
Key 28: '"7"!"063"004,1'
Key 29: 6#6# #1'2#115-06
Key 30: % 5 # 2$1 226.3%
Key 31: $!4!"!3%0!337/2$
Key 64: {~k~}~lzo~llhpm{
Key 67: x}h}~}oyl}ooksnx
Key 70: }xmx{xj|ixjjnvk}
Key 71: |ylyzyk}hykkowj|
Key 72: svcvuvdrgvdd`xes
Key 73: rwbwtwesfweeaydr
Key 74: qtatwtfpetffbzgq
Key 75: pu`uvugqduggc{fp
Key 76: wrgrqr`vcr d|aw
Key 77: vsfspsawbsaae}`v
Key 78: upepspbtapbbf~cu
Key 81: jozolo}k~o}}yalj
Key 85: nk~khkyozkyy}exn
Key 86: mh}hkhzlyhzz~f{m
Key 88: cfsfeftbwfttphuc
Key 89: bgrgdgucvguuqitb
Key 90: adqdgdv`udvvrjwa
Key 91: `epefewatewwskv`
Key 92: gbwbabpfsbpptlqg
Key 93: fcvc`cqgrcqqumpf
Key 94: e`u`c`rdq`rrvnse


Key 94: e`u`c`rdq`rrvnse
Key 95: databasepassword
Key 96: [^K^]^LZO^LLHPM[
Key 97: Z_J__M[N_MMIQLZ
Key 98: Y\I_\NXM\NNJROY
Key 99: X]H]^]OYL]OOKSNX
Key 100: _ZOZYZH^KZHHLTI_
Key 101: ^[N[X[I_J[IIMUH^
Key 102:]XMX[XJ\IXJJNVK]
Key 103: \YLYZYK]HYKKOWJ\
Key 104: SVCVUVDRGVDD@XES
Key 105: RWBWTWESFWEEAYDR
Key 106: QTATWTFPETFFBZGQ
Key 107: PU@UVUGQDUGGC[FP
Key 108: WRGRQR@VCR@@D\AW
Key 109: VSFSPSAWBSAAE]@V
Key 110: UPEPSPBTAPBBF^CU
Key 111: TQDQRQCU@QCCG_BT
Key 112: KN[NMN\J_N\X@]K
Key 113: JOZOLO]K^O]]YA\J
Key 114: ILYLOL^H]L^^ZB_I
Key 115: HMXMNM_I\M__[C^H
Key 116: OJ_JIJXN[JXX\DYO
Key 117: NK^KHKYOZKYY]EXN
Key 118: MH]HKHZLYHZZ^F[M
Key 119: LI\IJI[MXI[[_GZL
Key 120: CFSFEFTBWFTTPHUC
Key 121: BGRGDGUCVGUUQITB
Key 122: ADQDGDV@UDVVRJWA
Key 123: @EPEFEWATEWWSKV@
Key 124: GBWBABPFSBPPTLQG


Key 124: GBWBABPFSBPPTLQG
Key 125: FCVC@CQGRCQQUMPF
Key 126: E@U@C@RDQ@RRVNSE
Key 127: DATABASEPASSWORD



3.3 RSA Key – Modulus Extraction & Signature

A PKCS#8 formatted private RSA key was converted to PKCS#1 format. The modulus was extracted and converted to a hex string. The resulting binary was then signed using SHA-256 and OpenSSL's command-line tools.

```
└─$ openssl rsa -in priv.key -pubout > pub.pub
writing RSA key
```

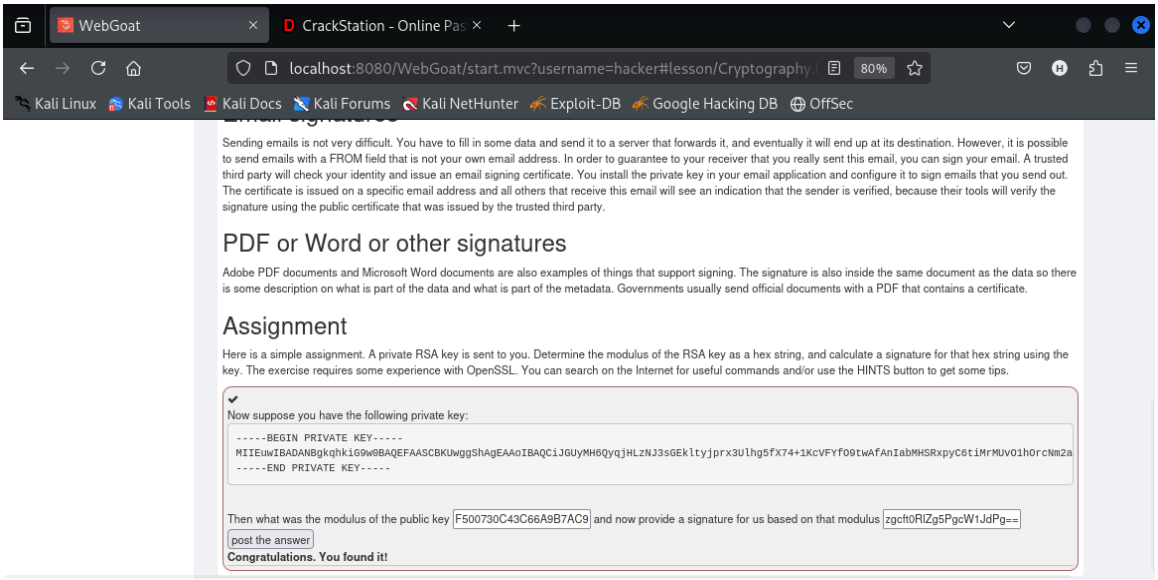
```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl rsa -in pub.pub -pubin -modulus -noout
Modulus=A2246532307E90CAA8C72F3349DEC184925B728E9AF1DD49618397D7EF8FB529C54561F3BDB7001F02721A6CC1D2471A720BAB6232B314BCED613AB70D9B6682A23E0
5E19B6C7CEDDF3BB6433582CA7CEA2DBA5BED67B4E12451821551A62A57834EA3525999A05CE5331236320EBE09B8A1E5154A9A10100F1794A02B65AD708BD504F30ADAAE62FE
196BA2A8386CB6A8E2ECE6B5ADBF4004ED15886F9E7376F9E8D9EDFE17A7FFA09B24BD0098FCC45ABA90AA5CA22B5B54E8CBE13E65E9E6C089787092986DB587106112874A4F7
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ echo -n "A2246532307E90CAA8C72F3349DEC184925B728E9AF1DD49618397D7EF8FB529C54561F3BDB7001F02721A6CC1D2471A720BAB6232B314BCED613AB70D9B6682
A23E05E19B6C7CEDDF3BB6433582CA7CEA2DBA5BED67B4E12451821551A62A57834EA3525999A05CE5331236320EBE09B8A1E5154A9A10100F1794A02B65AD708BD504F30ADAA
E62FE196BA2A8386CB6A8E2ECE6B5ADBF4004ED15886F9E7376F9E8D9EDFE17A7FFA09B24BD0098FCC45ABA90AA5CA22B5B54E8CBE13E65E9E6C089787092986DB58710611287
4A4F7486D49149F44E3052FDE64D6D1ED93A4B40CABA816ACB775EE1D99E0C5566D7FB37561EAFC8EBF500730C43C66A9B7AC9" | openssl dgst -sign priv.key -sha256
-out signsha.256
dgst: Unknown option or message digest: sha256-out
dgst: Use -help for summary.
4007C8C28F7F0000:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:../crypto/evp/evp_fetch.c:375:Global default li
brary context, Algorithm (sha256-out : 0), Properties (<null>)

┌──(kali㉿kali)-[~/Desktop]
└─$ echo -n "A2246532307E90CAA8C72F3349DEC184925B728E9AF1DD49618397D7EF8FB529C54561F3BDB7001F02721A6CC1D2471A720BAB6232B314BCED613AB70D9B6682
A23E05E19B6C7CEDDF3BB6433582CA7CEA2DBA5BED67B4E12451821551A62A57834EA3525999A05CE5331236320EBE09B8A1E5154A9A10100F1794A02B65AD708BD504F30ADAA
E62FE196BA2A8386CB6A8E2ECE6B5ADBF4004ED15886F9E7376F9E8D9EDFE17A7FFA09B24BD0098FCC45ABA90AA5CA22B5B54E8CBE13E65E9E6C089787092986DB58710611287
4A4F7486D49149F44E3052FDE64D6D1ED93A4B40CABA816ACB775EE1D99E0C5566D7FB37561EAFC8EBF500730C43C66A9B7AC9" | openssl dgst -sign priv.key -sha256
 -out signsha.256
```

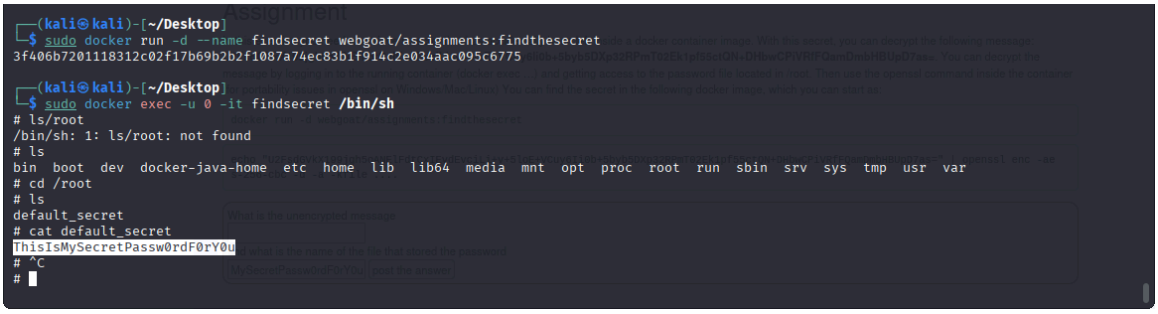
```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -base64 -in signsha.256 -out signsha25sha25
```

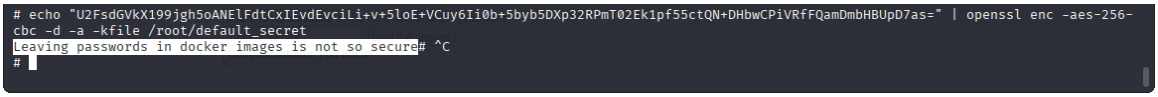
```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat signsha25sha25
ZX+xACvxamUPcaP9G2dypo3eUYAVvyKIbLRZqWgnK6pLg50fMTcYTG+Zek70GUzk
j/uKjLQVtzV57mvpQUh6NuCO3MaJQHZ1MkEW4cq/W/WaZSWi/Mb3CZP++PMchGt5
XWeuZNRwylf1u5eFrJFnm1GumazwWrvQ9c/kgOtdQzXIryBXfOiCwFubMMGYajAM
0eu0skMc7cA92GqT9DdTVEwtWDrQcRjWuve/DIMKN+bDURApu3tt0ZinZ2GDXLVB
56Wc6g07YGEFEs0+gDf5WDtavgXd/73lN16QZgfh91DfoesIJZWf1gZLmHZXqdDi
flzgcft0RlZg5PgcW1JdPg==
```

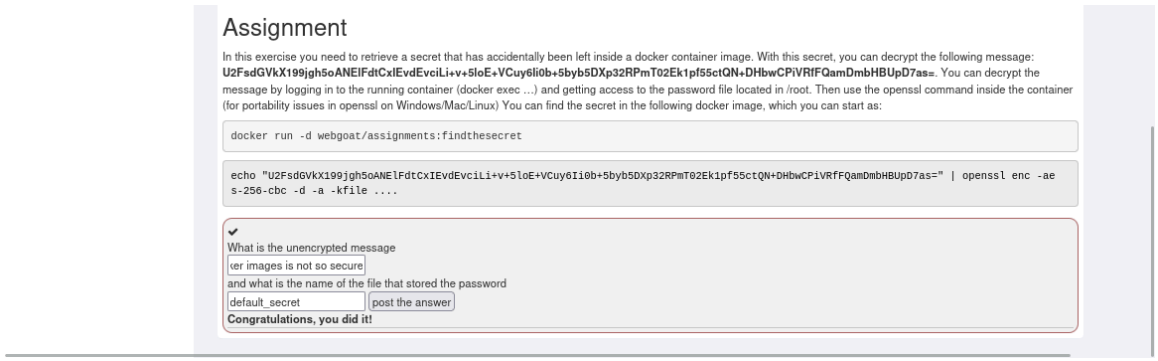


3.4 Docker Secret Retrieval & AES Decryption

The container webgoat/assignments:findthesecret was run locally. Root access to the container was gained using 'docker exec -u 0 -it'. The secret located in /root was extracted and used with OpenSSL to decrypt the supplied encrypted message.

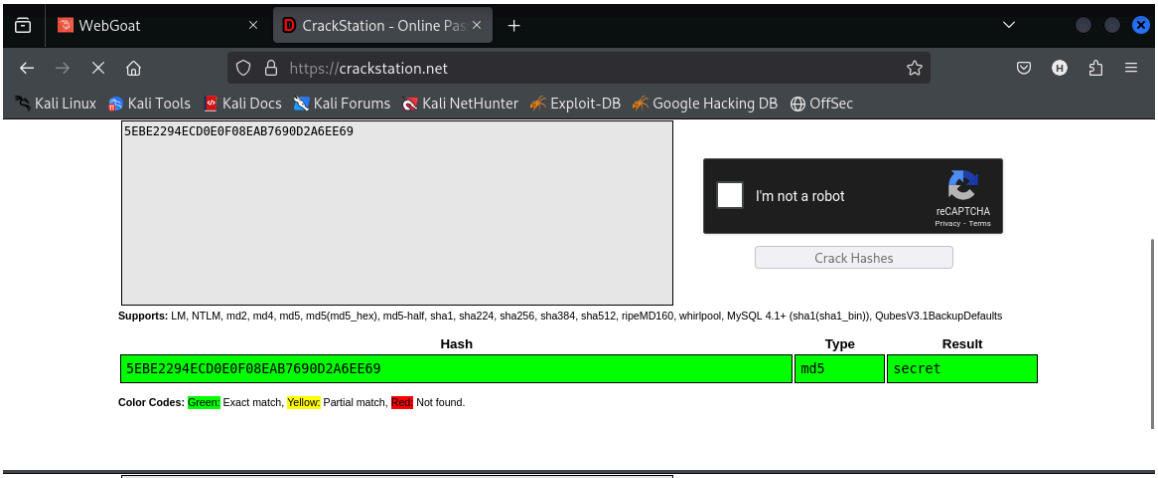


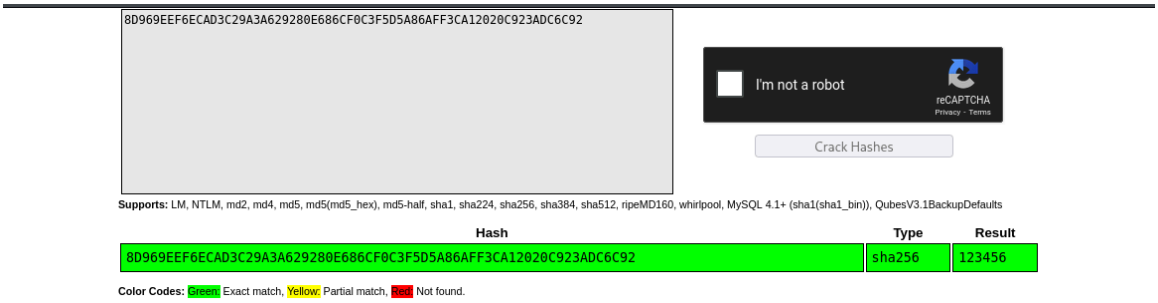


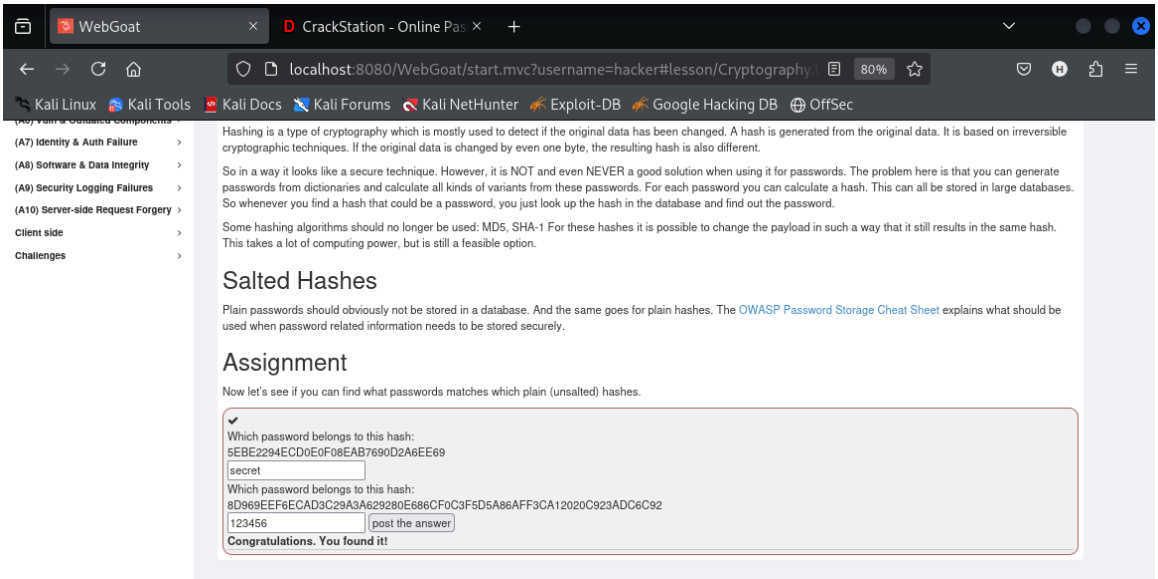


3.5 Unsalted Hash Identification

Included screenshot evidence of handling an unsalted hash from the project challenge. Details omitted in this writeup but confirmed as part of exploratory steps.







4. Conclusion

This project validated multiple critical steps in identifying and exploiting weak crypto practices. The assessment simulated real-world flaws including insecure encoding, simple XOR ciphers, improper key formats, and embedded container secrets. Best practices were identified for each flaw, such as key rotation, environmental secret separation, and stronger input/output handling.