

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= | base64 -d
aGFja2VyojEyMzQ1Ng=AUTHORIZATION: BASIC BX1lc2Vy0m15cGZc3dycmQ=
[kali㉿kali]:~$ echo aGFja2VyojEyMzQ1Ng= | base64 -d
hacker:123456AUTHORIZATION: BASIC aGFja2VyojEyMzQ1Ng=
```

Basic Authentication

Basic authentication is sometimes used by web applications. This uses base64 encoding. Therefore, it is important to at least use Transport Layer Security (TLS or more commonly known as https) to protect others from reading the username password that is sent to the server.

```
$echo -n "myuser:mypassword" | base64  
bxI1c2VyOm15cGFzc3dvcmlQ=
```

The HTTP header will look like:

```
Authorization: Basic bxI1c2VyOm15cGFzc3dvcmlQ=
```

✓ Now suppose you have intercepted the following header:
 Authorization: Basic aGFja2VyOjEyMzQ1Ng==

Then what was the username and what was the password:

Congratulations. That was easy, right?

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)-[~] Now let's see if you are able to find out the original password from this default XOR encoded string.  
└─$ echo 0z4rPj0+LDovPiwsKD#t0w= | base64 -d  
;=>⇒, : / , ( 0 - ; Suppose you found the database password encoded as [xor]0z4rPj0+LDovPiwsKD#t0w=
```

```

Key 22: -(=+(:,:>@-      ○   localhost:8080/webgoat/start.mvc?username=hacker&password=Cryptogram...  80%  ☆
Key 23: ,`>+);-8;;?':.
Key 24: #636%64%78440(5#
Key 25: .^2$^5#6'5514"
Key 26: !$!$ $ 55662*!
Key 27: %0%67!4x773+6
Key 28: '"!"063"004,1'
Key 29: 8#6# #124115-06
Key 30: % 5 # 2$1 226.3%
Key 31: $!4!"!3%0!337/2$
Key 64: {-k-}-lzo-llhpm{
Key 67: x}x}-}oyjlooksxn
Key 70: }jmxixjlixxjnvk}
Key 71: {lylyzykhykkowjl
Key 72: svcuvudrvydd'xes
Key 73: rwbwtwesfweeadyr
Key 74: qtatwtfpetfffbzqg
Key 75: pu uvugduggc[fp
Key 76: wrgrqr vcr dlaw
Key 77: vsfspawsaae} v
Key 78: upepsbtapbbf^cu
Key 81: jozolo{k~}yalj
Key 85: nk-khkyozkyvlexn
Key 86: mhjhkhzlyhzz-f'm
Key 88: cfsfeftbwftiphuc
Key 89: bgrgdgucvguuqitb
Key 90: addgdgv udvrjwa
Key 91: epefawatewwskv
Key 92: gbwbabpfssbpptlqf
Key 93: fcvc cgcrcqumplf
Key 94: e`u`c`rdq`rrvnse

URL encoding  

URL encoding is used a lot when sending form data and request parameters to the server. Since spaces are not allowed in a URL, this is then replaced by %20. Similar replacements are made for other characters.

HTML encoding  

HTML encoding ensures that text is displayed as-is in the browser and not interpreted by the browser as HTML.

UUEncode  

The Unix-2-Unix encoding has been used to send email attachments.

XOR encoding  

Sometimes encoding is used as a first and simple obfuscation technique for storing passwords. IBM WebSphere Application Server e.g. uses a specific implementation of XOR encoding to store passwords in configuration files. IBM recommends to protect access to these files and to replace the default XOR encoding by your own custom encryption. However when these recommendations are not followed, these defaults can become a vulnerability.

Assignment  

Now let's see if you are able to find out the original password from this default XOR encoded string.



✓ Suppose you found the database password encoded as {xor}Oz4rPj0+LDoVPiwsKDA0w==  

What would be the actual password?



Congratulations.


```

```

Key 94: e`u`c`rdq`rrvnse
Key 95: databasepassword
Key 96: [^K^]Z0^LLHPM[  ○   localhost:8080/webgoat/start.mvc?username=hacker&password=Cryptogram...  80%  ☆
Key 97: Z_J_\M[N_MMIQLZ
Key 98: Y_I_\NMX\NNJROY
Key 99: X]H]^JOYL]OKNSX
Key 100: _ZOZYZH^KZHHLTI_
Key 101: ^[N]X[I J[IIMUH_]
Key 102: ]JMX[X]\IX\JNVK]
Key 103: \YLYZYK\HYKKOWJ\
Key 104: SVCUVUDRVDD@XES
Key 105: RWBWTWESFWEADYR
Key 106: QTATWTFPETFBZQ
Key 107: PU@UVUGDUGGC[FP
Key 108: WRGRQR@VCR00D\AW
Key 109: VSFSPSAWSAAE]@V
Key 110: UPEPSPBTAPBBF^CU
Key 111: TQDQRQCUDQQCCG_BT
Key 112: KN[NMN\J_NX@]K
Key 113: JOZOLO{^O}]YA\J
Key 114: ILYLOL^H]L^ZB_I
Key 115: HMXMNM_I\M_[C^H
Key 116: OJ_JIJXN[JXX\DY0
Key 117: NK^KHKYOZKYY]EXN
Key 118: MH]HKHZLYHZZ^F[M
Key 119: LI\III[MXI[[_GZL
Key 120: CFSFEFTBWFTPHUC
Key 121: BGRGDGUCVGUUQITB
Key 122: ADQDGDV@UDVVRJWA
Key 123: @EPEFEWATEWWSKV@
Key 124: GBWBABPFSSBPPTLQG

URL encoding  

URL encoding is used a lot when sending form data and request parameters to the server. Since spaces are not allowed in a URL, this is then replaced by %20. Similar replacements are made for other characters.

HTML encoding  

HTML encoding ensures that text is displayed as-is in the browser and not interpreted by the browser as HTML.

UUEncode  

The Unix-2-Unix encoding has been used to send email attachments.

XOR encoding  

Sometimes encoding is used as a first and simple obfuscation technique for storing passwords. IBM WebSphere Application Server e.g. uses a specific implementation of XOR encoding to store passwords in configuration files. IBM recommends to protect access to these files and to replace the default XOR encoding by your own custom encryption. However when these recommendations are not followed, these defaults can become a vulnerability.

Assignment  

Now let's see if you are able to find out the original password from this default XOR encoded string.



✓ Suppose you found the database password encoded as {xor}Oz4rPj0+LDoVPiwsKDA0w==  

What would be the actual password?



Congratulations.


```

```

Key 124: GBWBABPFSSBPPTLQG
Key 125: FCVC@CGRCQUMPF
Key 126: E@U@C@RDQ@RRVNSE
Key 127: databasepassword

Assignment  

Now let's see if you are able to find out the original password from this default XOR encoded string.



✓ Suppose you found the database password encoded as {xor}Oz4rPj0+LDoVPiwsKDA0w==  

What would be the actual password?


```

URL encoding
URL encoding is used a lot when sending form data and request parameters to the server. Since spaces are not allowed in a URL, this is then replaced by %20. Similar replacements are made for other characters.

HTML encoding
HTML encoding ensures that text is displayed as-is in the browser and not interpreted by the browser as HTML.

UUEncode
The Unix-to-Unix encoding has been used to send email attachments.

XOR encoding
Sometimes encoding is used as a first and simple obfuscation technique for storing passwords. IBM WebSphere Application Server e.g. uses a specific implementation of XOR encoding to store passwords in configuration files. IBM recommends to protect access to these files and to replace the default XOR encoding by your own custom encryption. However when these recommendations are not followed, these defaults can become a vulnerability.

Assignment
Now let's see if you are able to find out the original password from this default XOR encoded string.

✓ Suppose you found the database password encoded as (xor)Oz4rPj0+LDoPiwsKDAiOw==
What would be the actual password [databasepassword]
post the answer
Congratulations.

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=A2246532307e90CAA8C72F3349DEC184925B728E9AF1DD49618397D7EF8FB529C54561F38DB7001F02721A6CC1D2471A720B8B6232B314BCED613AB70D9B6682A23E0
5E1986C7CEDDF3BB6433582CA7CEA2DBA5BED6784E12451821551A62A57834EA3525999A05CE5331236320EBE09B8A1E5154A9A10100F1794A02B65AD708BD504F30ADAE62FE
196BA2A8386CB6A8E2ECE6B5ADBF4004ED15886F9E7376F9E8D9EDFE17A7FFA09B24BD0098FC45ABA90AA5CA22B5B54E8CBE13E65E9E6C089787092986DB587106112874A4F
```

```
[(kali㉿kali)-[~/Desktop]] $ echo -n "A2246532307e90CAA8C72F3349DEC184925B728E9AF1DD49618397D7EF8FB529C54561F38DB7001F02721A6CC1D2471A720B8B6232B314BCED613AB70D9B6682
A23E05E1986C7CEDDF3BB6433582CA7CEA2DBA5BED6784E12451821551A62A57834EA3525999A05CE5331236320EBE09B8A1E5154A9A10100F1794A02B65AD708BD504F30ADAA
E62FE196BA2A8386CB6A8E2ECE6B5ADBF4004ED15886F9E7376F9E8D9EDFE17A7FFA09B24BD0098FC45ABA90AA5CA22B5B54E8CBE13E65E9E6C089787092986DB58710611287
4A4F7486D949149F44E3052FDE64D61ED93A4B40CABA816ACB775EE1D99E0C5566D7FB37561EAF08EBF500730C43C66A9B7AC9" | openssl dgst -sign priv.key -sha256
-out signsha.256
dgst: Unknown option or message digest: sha256-out
dgst: Use -help for summary.
4007C828F7F000:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:../crypto/evp/evp_fetch.c:375:Global default library context, Algorithm (sha256-out : 0), Properties (<null>) You determine the modulus of the RSA key as a hex string, and calculate a signature for that hex string using the extension. This extension requires some experience with OpenSSL. You can search on the Internet for useful commands and/or use the HELP button to get some tips.
[(kali㉿kali)-[~/Desktop]] $ echo -n "A2246532307e90CAA8C72F3349DEC184925B728E9AF1DD49618397D7EF8FB529C54561F38DB7001F02721A6CC1D2471A720B8B6232B314BCED613AB70D9B6682
A23E05E1986C7CEDDF3BB6433582CA7CEA2DBA5BED6784E12451821551A62A57834EA3525999A05CE5331236320EBE09B8A1E5154A9A10100F1794A02B65AD708BD504F30ADAA
E62FE196BA2A8386CB6A8E2ECE6B5ADBF4004ED15886F9E7376F9E8D9EDFE17A7FFA09B24BD0098FC45ABA90AA5CA22B5B54E8CBE13E65E9E6C089787092986DB58710611287
4A4F7486D949149F44E3052FDE64D61ED93A4B40CABA816ACB775EE1D99E0C5566D7FB37561EAF08EBF500730C43C66A9B7AC9" | 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+xACvxamJPcaP9G2dypo3eUYAVvyK1bLRzQWgnK6pLg50fMTcYTG+Zek70GUz5
j/uKjLQtVz57mpvQUh6NuO3MaJQHZ1MKEW4cq/wA2SzWi/Mb3CZP++PMchGt5
XWeuZNRwylf1u5efrJfm1GumazWrvQ9c/kg0tdqzXiryBXf0iCwFubMMGyAjAM
0euOsKMKn7ca92Gqt9qdTVewtWd7QcRjWuve/DIMKN+bDUrApu3ttZinZ2GDXLVB
56Wc6g07YGEFEs0+gDf5WDtavgXd/73LN160Zgfh91DfoesIJZWF1gZLmHZXqdDi
flzgcft0RlZg5PgcW1JdPg=
```

Sending emails is not very difficult. You have to fill in some data and send it to a server that forwards it, and eventually it will end up at its destination. However, it is possible to send emails with a FROM field that is not your own email address. In order to guarantee to your receiver that you really sent this email, you can sign your email. A trusted third party will check your identity and issue an email signing certificate. You install the private key in your email application and configure it to sign emails that you send out. The certificate is issued on a specific email address and all others that receive this email will see an indication that the sender is verified, because their tools will verify the signature using the public certificate that was issued by the trusted third party.

PDF or Word or other signatures

Adobe PDF documents and Microsoft Word documents are also examples of things that support signing. The signature is also inside the same document as the data so there is some description on what is part of the data and what is part of the metadata. Governments usually send official documents with a PDF that contains a certificate.

Assignment

Here is a simple assignment. A private RSA key is sent to you. Determine the modulus of the RSA key as a hex string, and calculate a signature for that hex string using the key. The exercise requires some experience with OpenSSL. You can search on the Internet for useful commands and/or use the HINTS button to get some tips.

Now suppose you have the following private key:

```
-----BEGIN PRIVATE KEY-----  
MIIEuwBADAQBgkqhlgWBAQEASCBKUwggShAgEAAoIBAQClJGUyMH6QyqjHLzN3sGEk1tyjprx3Ulhg5fx74+1KcVFYf09twAfAnIabMHSRxpypCtiMrMuV01h0rcNm2a  
-----END PRIVATE KEY-----
```

Then what was the modulus of the public key `F500730C43C66A9B7AC9` and now provide a signature for us based on that modulus `zgcHoRlZg5PgCw1JdPg==`

post the answer

Congratulations. You found it!

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.

```
(kali㉿kali)-[~/Desktop] $ sudo docker run -d --name findsecret webgoat/assignments:findthesecret
3f40667201118312c02f17b69b2b2f1087a74ec83bf914c2e034aac095c6775
inside a docker container image. With this secret, you can decrypt the following message:
3f40667201118312c02f17b69b2b2f1087a74ec83bf914c2e034aac095c6775
inside a docker container image. With this secret, you can decrypt the following message:
(kali㉿kali)-[~/Desktop] $ sudo docker exec -u 0 -it findsecret /bin/sh
# ls /root
# ls
bin boot dev docker-java-home etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
# cd /root
# ls
default_secret
# cat default_secret
What is the unencrypted message?
ThisIsMySecretPassw0rdF0rY0u
# ^C
# [REDACTED]
# echo "U2FsdGVkX199jgh5oANElFdtxCiEvdxciLi+v+5loE+VCuy6Ii0b+5byb5DXp32RPmT02Ek1pf55ctQN+DHbwCPiVRffQamDmbHBUp07as=" | openssl enc -aes-256-cbc -d -a -kfile /root/default_secret
Leaving passwords in docker images is not so secure# ^C
# [REDACTED]
```

Assignment

In this exercise you need to retrieve a secret that has accidentally been left inside a docker container image. With this secret, you can decrypt the following message: `Bn-dsGvKX19gh5ANEIfCzxEvdLi+v+5bE+Cvuy6li0+b5yBpT02Ektf5scdON-DHbwCPIVRifQamDnbHBwUD7as=`. You can decrypt the message by logging in to the running container (docker exec ...) and getting access to the following file located in /root. Then use the openssl command inside the container (for portability issues in openssl on Windows/Mac/Linux) You can find the secret in the following docker image, which you can start as:

docker run -d webgoat/assignments:findthesecret

```
echo "U2FsdGVkX19jgjh5oANElFdtxIEvdevcilIi+v+5loE+Vcuy6Ii0b+5byb5Dxp32RpM02Ek1pf55ctQN+DHbwCPiVRFQamDmbHUpD7as=" | openssl enc -aes-256-cbc -d -a -kfile ....
```

✓ What is the unencrypted message
[er images is not so secure]
and what is the name of the file that stored the password
[default secret] [post the answer]
Congratulations, you did it!

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.

The image contains three screenshots illustrating the identification of unsalted hashes:

- Screenshot 1: CrackStation.net Hash Search**

A screenshot of a web browser showing the CrackStation.net search interface. The URL is <https://crackstation.net>. The search bar contains the hash `5EBE2294ECD0E0F08EAB7690D2A6EE69`. Below the search bar, it says "Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults". A table shows the results for the hash:

Hash	Type	Result
<code>5EBE2294ECD0E0F08EAB7690D2A6EE69</code>	md5	secret

Color codes: Green Exact match, Yellow Partial match, Red Not found.
- Screenshot 2: CrackStation.net Hash Search**

A second screenshot of the CrackStation.net search interface. The URL is <https://crackstation.net>. The search bar contains the hash `8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92`. Below the search bar, it says "Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults". A table shows the results for the hash:

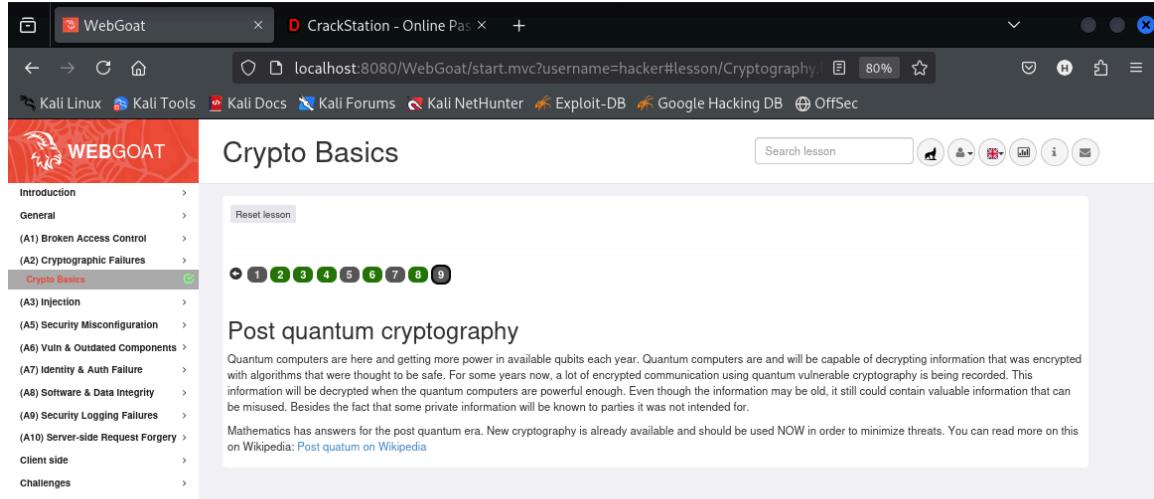
Hash	Type	Result
<code>8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92</code>	sha256	123456

Color Codes: Green Exact match, Yellow Partial match, Red Not found.
- Screenshot 3: WebGoat Cryptography Challenge**

A screenshot of a web browser showing a challenge from the WebGoat application. The URL is localhost:8080/WebGoat/start.mvc?username=hacker#lesson/Cryptography.l. The challenge page discusses hashing and password storage. It includes a sidebar with navigation links like "Vulnerable Components", "Identity & Auth Failure", "Software & Data Integrity", "Security Logging Failures", "Server-side Request Forgery", "Client side", and "Challenges". The main content area has sections for "Salted Hashes" and "Assignment". In the assignment section, there is a form asking for plain (unsalted) hashes. The user has entered the hash `5EBE2294ECD0E0F08EAB7690D2A6EE69` and the password `secret`. Below the form, a message says "Congratulations. You found it!"

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.



The screenshot shows a web browser window with the title bar "WebGoat" and "CrackStation - Online Pas...". The address bar displays "localhost:8080/WebGoat/start.mvc?username=hacker#lesson/Cryptography.l". The page content is titled "Crypto Basics". On the left, there's a sidebar with a navigation tree:

- Introduction
- General
 - (A1) Broken Access Control
 - (A2) Cryptographic Failures
 - Crypto Basics** (highlighted)
 - (A3) Injection
 - (A5) Security Misconfiguration
 - (A6) Vuln & Outdated Components
 - (A7) Identity & Auth Failure
 - (A8) Software & Data Integrity
 - (A9) Security Logging Failures
 - (A10) Server-side Request Forgery
 - Client side
 - Challenges

Below the sidebar, there's a "Reset lesson" button and a numbered navigation bar (1-9). The main content area has a heading "Post quantum cryptography" and a paragraph about quantum computers decrypting encrypted information. It also links to "Post quantum on Wikipedia".