

p4-team / ctf

Watch78

Star296

Fork63

<> Code

🔔 Issues 0

🔗 Pull requests 0

📁 Projects 0

📶 Pulse

📊 Graphs


Branch: master ▾

ctf / 2015-10-22-ekoparty / crpto_200_xorcrypter /



Create new file

Find file

History

 Pharisaeus typos

Latest commit f052ffc on Oct 24 2015

..		
 README.md	typos	2 years ago
 shiftcrypt.py	Adding hacklu and ekoparty writeups	2 years ago

 README.md

XOR Crypter (crypto 200p)

Description: The state of art on encryption, can you defeat it?
CjBPewYGc2gdD3RpMRNfdDcQX3UGGmhpBxZhYhF1fQA=

PL

ENG

Cały kod szyfrujący jest [tutaj](#). Szyfrowanie jest bardzo proste, aż dziwne że zadanie było za 200 punktów. Szyfrowanie polega na podzieleniu wejściowego tekstu na 4 bajtowe kawałki (po dodaniu paddingu jeśli to konieczne, aby rozmiar wejścia był wielokrotnością 4 bajtów), rzutowanie ich na inta a następnie wykonywana jest operacja $x \oplus x \gg 16$. Jeśli oznaczmy kolejnymi literami bajty tego inta uzyskujemy:

$$ABCD \oplus ABCD \gg 16 = ABCD \oplus 00AB = (A \oplus 0)(B \oplus 0)(C \oplus A)(D \oplus B) = AB(C \oplus A)(D \oplus B)$$

Jak widać dwa pierwsze bajty są zachowywane bez zmian a dwa pozostałe bajty są xorowane z tymi dwoma niezmienionymi. Wiemy także że xor jest operacją odwracalną i $(A \oplus B) \oplus B = A$ możemy więc odwrócić szyfrowanie dwóch ostatnich bajtów xorując je jeszcze raz z pierwszym oraz drugim bajtem (pamiętając przy tym o kolejności bajtów)

```
data = "CjBPewYGc2gdD3RpMRNfdDcQX3UGGmhpBxZhYhF1fQA="
decoded = base64.b64decode(data)
blocks = struct.unpack("I" * (len(decoded) / 4), decoded)
output = ''
for block in blocks:
    bytes = map(ord, struct.pack("I", block))
    result = [bytes[0] ^ bytes[2], bytes[1] ^ bytes[3], bytes[2], bytes[3]]
    output += "".join(map(chr, result))
print(output)
```

W wyniku czego uzyskujemy flagę: EK0{unshifting_the_unshiftable}

ENG version

Cipher code is [here](#). The cipher is actually very simple, it was very strange that the task was worth 200 point. The cipher splits the input text in 4 byte blocks (after adding padding if necessary so that the input is a multiply of 4 bytes), casting each block to integer and the performing $x \oplus x \gg 16$. If we mark each byte of the single block with consecutive alphabet letters we get:

$$ABCD \oplus ABCD \gg 16 = ABCD \oplus 00AB = (A \oplus 0)(B \oplus 0)(C \oplus A)(D \oplus B) = AB(C \oplus A)(D \oplus B)$$

As can be noticed, first two bytes are unchanged and last two are xored with those two unchanged. We also know that xor is reversible and $(A \oplus B) \oplus B = A$ so we can revert the cipher of the last two bytes by xoring them again with first and second byte (keeping in mind the byte order).

```
data = "CjBPewYGc2gdD3RpMRNfdDcQX3UGGmhpBxZhYhF1fQA="
decoded = base64.b64decode(data)
blocks = struct.unpack("I" * (len(decoded) / 4), decoded)
output = ''
for block in blocks:
    bytes = map(ord, struct.pack("I", block))
    result = [bytes[0] ^ bytes[2], bytes[1] ^ bytes[3], bytes[2], bytes[3]]
    output += "".join(map(chr, result))
print(output)
```

As a result we get: EKO{unshifting_the_unshiftable}

