Disciplina: Segurança Cibernética

Crypto Delphi (UTCTF 2021)

Grupo
Arthur Hugo Barros Gaia
Felipe Ivo da Silva
Nathalia Cristina Santos
Thiago Zucarelli Crestani
Wilson de Camargo Vieira

SUMÁRIO

Introdução

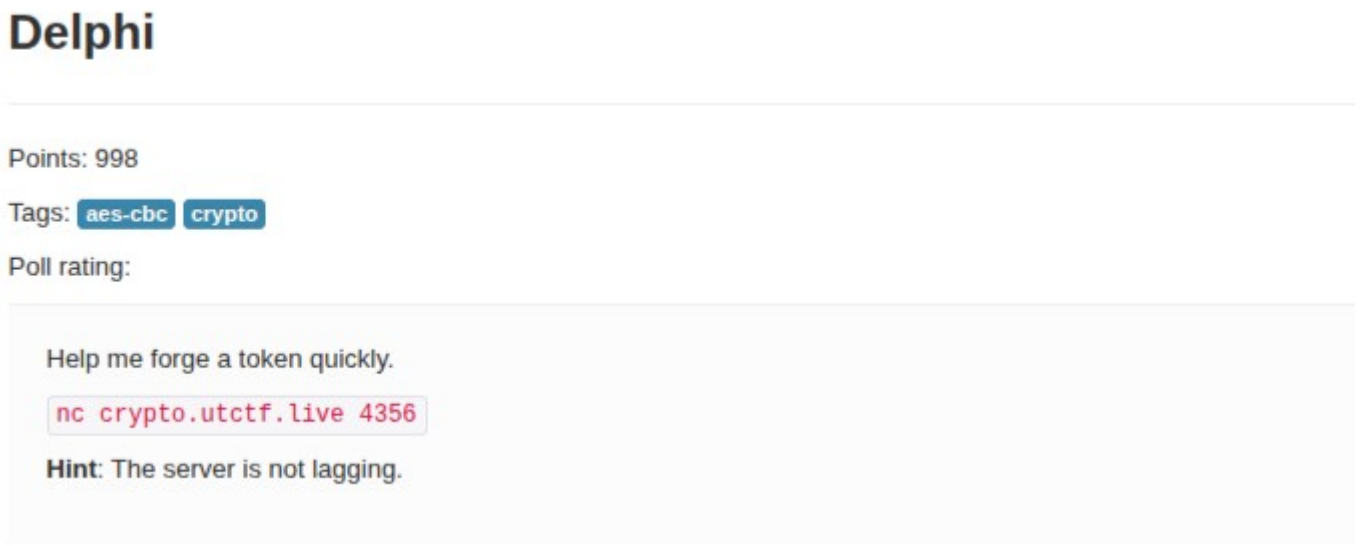
DELPHI

Desafio do UTCTF de 2021

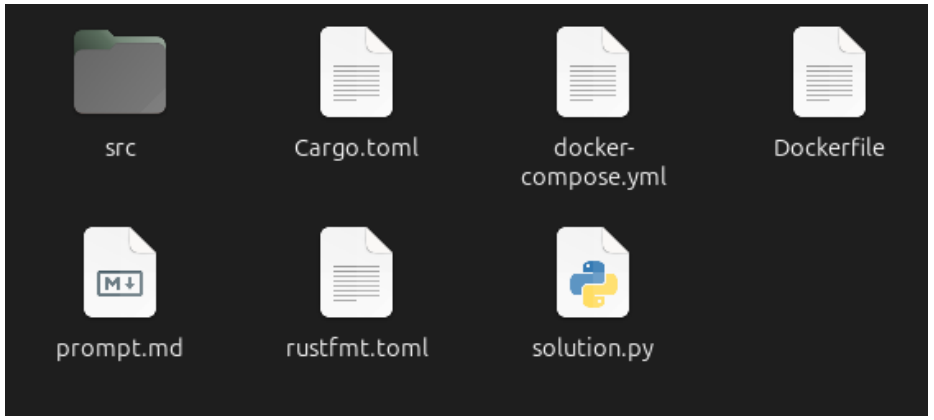
- O servidor pede que você **forje um token**: “cripte o challenge com **AES-256-CBC** + **PKCS7**; os primeiros 16 bytes são o **IV**; envie em **hex**”.

Isso é um **padding oracle** em **AES-CBC**:

- Mostra como **um detalhe de implementação quebra a segurança** de um esquema sólido (AES-CBC).



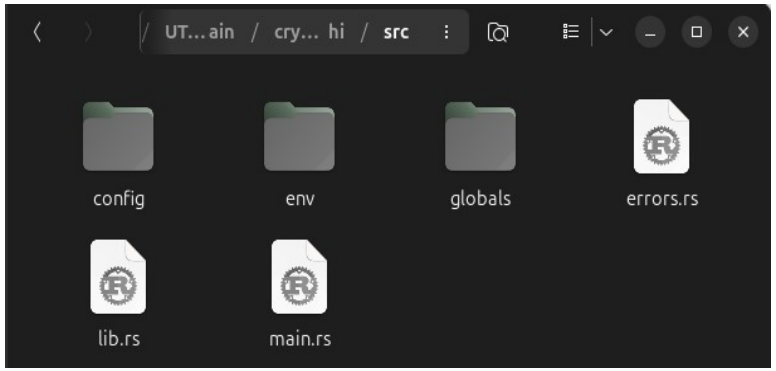
Clique para adicionar o título




```
nati@nati-IdeaPad-3-15IML05:~$ nc crypto.utctf.live 4356
nc: getaddrinfo for host "crypto.utctf.live" port 4356: No address associated with
hostname
```

```
0.569 warning: the cargo feature `strip` has been stabilized in the 1.58 release and is no lon
ger necessary to be listed in the manifest
0.569   See https://doc.rust-lang.org/nightly/cargo/reference/profiles.html#strip-option for m
ore information about using this feature.
0.726    Compiling delphi v0.1.0 (/delphi)
0.825 error[E0635]: unknown feature `num_as_ne_bytes`
0.825  --> src/lib.rs:2:12
0.825   |
0.825 2 | #![feature(num_as_ne_bytes)]
0.825   |            ^^^^^^^^^^^^^^^
0.825
1.223 For more information about this error, try `rustc --explain E0635`.
```

- nc crypto.utctf.live 4356 = tenta abrir uma conexão TCP com o host crypto.utctf.live na porta 4356.
- "No address associated with hostname" significa que **o DNS não conseguiu resolver** esse nome para um IP.
- nc -v localhost 4356 pontei para **um serviço local** (dentro do container) na porta 4356



#![feature(num_as_ne_bytes)]

Clique para adicionar o título

```
nati@nati-IdeaPad-3-15IML05:~/Downloads/UTCTF-21-main/crypto-delphi$ sudo docker compose up
WARN[0000] /home/nati/Downloads/UTCTF-21-main/crypto-delphi/docker-compose.yml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 2/2
 ✔ Network crypto-delphi_default  Creat...                                    0.1s
 ✔ Container delphi               Created                                     0.1s
Attaching to delphi
```

```
nati@nati-IdeaPad-3-15IML05:~$ nc -v localhost 4356
Connection to localhost (127.0.0.1) 4356 port [tcp/*] succeeded!
Welcome to the secure flag vault.
Only authorized users can retrieve the flag.
Just encrypt the following challenge in bytes with the secret key.
Use AES-256-CBC with PKCS7 padding. The first 16 bytes should be the IV.
Submit it in hex encoded form.
For security reasons, only 12289 tries are permitted.
Challenge: cbd6f9640187673b8e7dc42e31c72e9e75b9e4ff5642939d2a58443f5951bcde

Uh-oh. Someone is trying to shut down the system.
The connection will close in 29 seconds.

29 seconds remain. 12289 tries left.
Please submit authorization token.
Decryption failed.

0 seconds remain. 12288 tries left.
Please submit authorization token.
Connection has been closed.
0
00
```

FLAG ENCONTRADA!!

```
Authorization verified. The flag is utflag{oracle_padded_oops}
```

Clique para adicionar o título

Write-ups e fontes utilizadas:

https://github.com/utisss/UTCTF-21/tree/main/crypto-delphi
https://github.com/cscosu/ctf-writeups/tree/master/2021/utctf/Delphi

Conceitos criptográficos relevantes

- AES-CBC (Cipher Block Chaining) — cifra blocos de 16 bytes; cada bloco de plaintext é combinado (XOR) com o bloco anterior antes de ser cifrado. Na decriptação: $P_i = D_k(C_i) \oplus C_{i-1}$. O primeiro bloco usa um IV aleatório, que deve ser imprevisível para garantir segurança.

- PKCS#7 (padding) — completa o último bloco com N bytes de valor 0xN; ex.: se faltam 4 bytes → 0x04 0x04 0x04 0x04. A validação retorna verdadeiro/falso conforme a consistência do preenchimento.

- Padding Oracle — ocorre quando o servidor revela, direta (mensagem de erro) ou indiretamente (tempo de resposta, comportamento), se o padding foi válido. Esse “oracle” fornece um bit de informação por tentativa, suficiente para reconstruir o plaintext bloco a bloco.

- Referência teórica — ataque descrito por Vaudenay (2002), amplamente reproduzido em writeups e exercícios práticos.

Como o serviço vulnerável se comporta

- Entrada e formato — o serviço recebe ciphertext (IV + blocos), normalmente enviado em hex ou base64.
- Feedback distinguível — o servidor responde de forma distinta quando o padding é inválido vs quando a decriptação/padding é válida; essa distinção é o sinal que cria o oracle.
- Efeito prático (alto nível) — ao manipular bytes do bloco anterior e observar o feedback, um adversário infere bytes intermediários e reconstrói P_i do último byte para o primeiro.
- Custo teórico — no pior caso até 256 tentativas por byte (≈4096 por bloco de 16 bytes); em médias práticas o número tende a ser menor, mas ainda significativo.
- Sinal de exploração (teoria de detecção) — padrões de requisições onde apenas alguns bytes mudam repetidamente e elevado volume por origem são indicadores típicos que permitem detecção.
- Observação: se o serviço faz validação de autenticação (MAC) antes da remoção do padding (encrypt-then-MAC), o oracle clássico desaparece.
- Variante: quando mensagens são genéricas, diferenças de timing ou outros side-channels ainda podem vazar informação (timing-oracle).

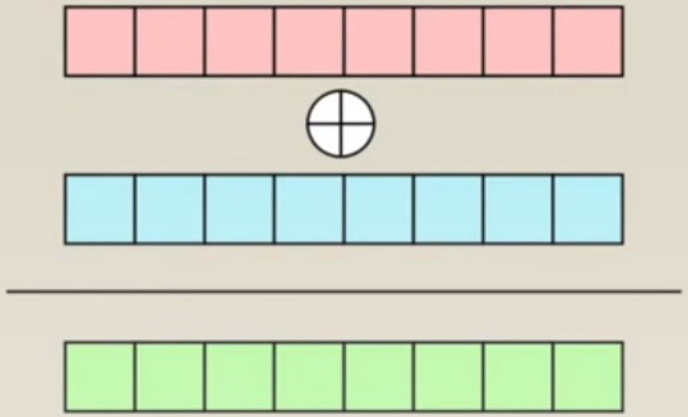
Natureza da Operação XOR

XOR Operation

- XOR operation is done byte-by-byte, bit-by-bit
  - Changing 1 byte will only affect the byte in its "lane"

- XOR operation has this peculiar property:
  - XOR any 2, and the result is the remaining 1
  - $A \oplus B = C$
  - $A \oplus C = B$
  - $B \oplus C = A$

Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



Complexidade

Modelo básico por byte / por bloco

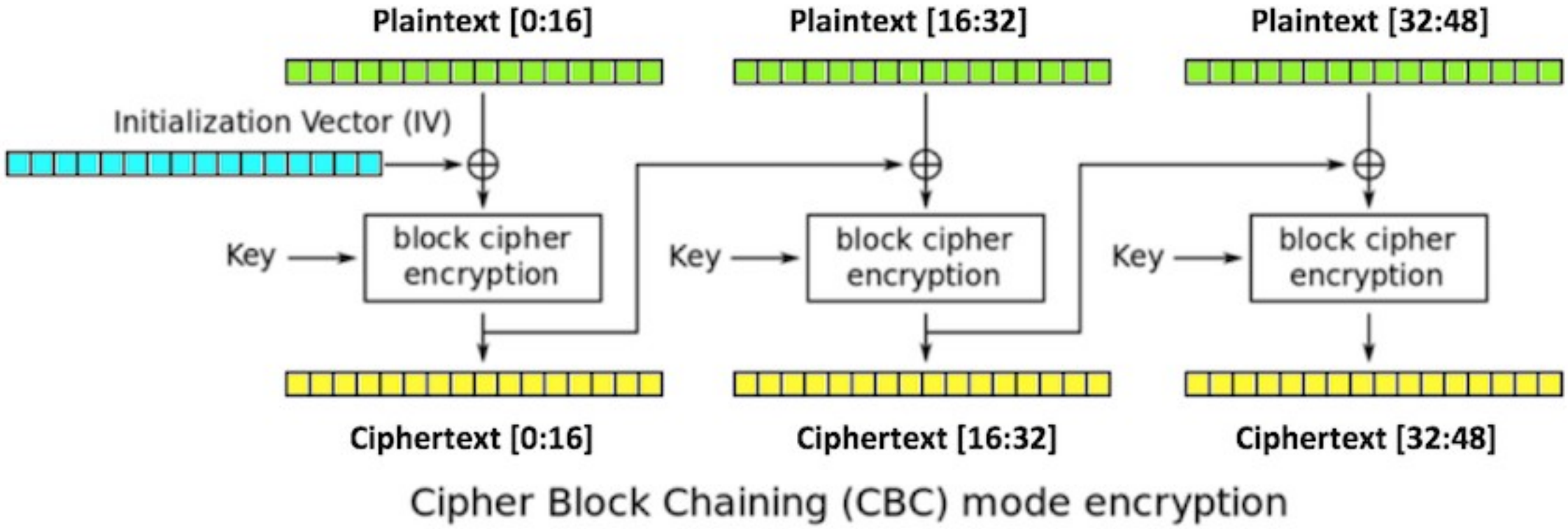
Bloco AES-CBC = 16 bytes.
Para cada byte alvo, o atacante testa valores 0..255 até encontrar o que produz padding válido.

Pior caso (determinístico): 256 testes por byte.
Por bloco: $256 \times 16 = 4096$ requisições.

Esperança (modelo uniforme): se há exatamente 1 valor correto e os testes são uniformes, expectativa de testes por byte = $\frac{256+1}{2} = 128.5 \approx 128.$
Por bloco (esperança): $128.5 \times 16 \approx 2056$ requisições.
Expectativa por byte = $E[X] = \sum_{k=1}^{256} k \cdot P(\mathit{first\ success\ at}\ k)$.
Se cada tentativa independente e sucesso único, $E[X] = (256 + 1)/2.$

Observação prática: a expectativa assume que existe exatamente 1 valor que valida o padding por byte. Em alguns casos ambíguos (ex.: coincidência com padding real), heurísticas adicionam testes extras; isso eleva a média.

Exemplo de Ciphertext de 48 Bytes



Cipher Block Chaining (CBC) mode encryption

Requests esperadas

B = número de blocos a decifrar.
E_b =expectativa de tentativas por bloco (inclui ambiguidade), aproximadamente $E_b \approx 16 \times 128.5 \times m_{amb}$.
p_{err} =probabilidade de falha por requisição (rede/timeouts); retries multiplicam custo por fator $\frac{1}{1-p_{err}}$.
R =overhead de conexão por request (ex.: handshake por conexão, se nova conexão por request).
Então:

$$\text{Requests esperadas} \approx \mathrm{B} \times \mathrm{E_b} \times \frac{1}{1 - P_{err}}$$

Se cada requisição cria nova sessão e reconnect custa 0_{conn} extra (em requisições equivalentes): multiplicador $1 + 0_{conn}$

Exemplo numérico (sem valores absolutos): para 2 blocos e $m_{amb} = 1.2$,$p_{err} = 0.05$:
$E_b \approx 16 \times 128.5 \times 1.2 \approx 2467$.
Requests totais $\approx 2 \times 2467/(1 - 0.05) \approx 5194$).

Confidencialidade

Descrição: capacidade do atacante de recuperar plaintext a partir de ciphertexts sem conhecer a chave.

Escopo do vazamento
Dados diretamente expostos: flags, tokens JWT/semelhantes, cookies de sessão, segredos armazenados no ciphertext, PII cifrada.

Dados parcialmente expostos: se só alguns blocos do plaintext contém segredos, atacante pode focar nesses blocos.

Severidade (fatores que aumentam impacto)

Sensibilidade do plaintext: sesssion tokens / chaves → impacto crítico; flag CTF → baixo-moderado (CTF apenas).

Quantidade de dados cifrados reutilizados: tokens reutilizados por muitos usuários/tempo ampliam impacto (ex.: cookie de sessão reutilizável).

Disponibilidade de many ciphertexts: se atacante pode obter muitos ciphertexts de diferentes usuários, escala a exploração (mais vítimas).

Persistence: se ciphertexts duram (tokens long-lived), tempo para explorar pode ser maior e mais valioso.

Integridade

Descrição: possibilidade de um atacante manipular ciphertexts para induzir o servidor a produzir plaintexts escolhidos ou alterar significado.

Integridade direta (forjar plaintexts)
Em CBC sem autenticação, modificar bytes de C_{i-1} altera P_i de forma controlada (XOR com Dk(Ci)). Isso permite **flipping** de bits em posições específicas do plaintext do bloco seguinte.
Padding oracle atacante normalmente busca decifrar; contudo, com conhecimento de Dk(Ci) (intermediate), atacante pode **construir** novos ciphertexts que, ao serem decifrados, resultam em plaintexts escolhidos (até certo ponto). Em geral:

- Forjar mensagens arbitrárias requer mais controle (e depende de validações do aplicativo).
- Pode ser usado para manipular tokens sem autenticação adicional.

Cenários práticos de ataque à integridade
Token forging: se token assinado/integridade for apenas por obscuridade e o servidor aceita plaintexts formados por dados do ciphertext, atacante pode injetar campos (p.ex., privilégio=admin).
Command injection indireta: se o plaintext for interpretado (p.ex., como comando ou SQL), manipular plaintext pode levar a RCE/SQLi.

Disponibilidade

Descrição: efeito do ataque sobre a capacidade do serviço de continuar funcionando; inclui DoS acidental/induzido.

Impacto direto do padding oracle attack

O ataque gera **alto volume de requisições** em curto período. Dependendo da infraestrutura:

- Pode causar saturação de CPU ou filas, especialmente se cada request envolve operações de criptografia (AES).
- Pode aumentar I/O de logs.

No entanto, ataque não é um DoS por design; objetivo é extrair informação, não derrubar.

Cenários e consequências

Rate-limits e bloqueios: o serviço pode impor bloqueios levando à indisponibilidade legítima (falsos-positivos) para usuários.

Resource exhaustion: se o endpoint de decryption é custoso (HSM, operações pesadas), muitos requests podem degradar serviço.

Impacto indireto: equipe de segurança reagindo (bloqueios/rotinas) pode causar interrupções operacionais.

Recursos Computacionais e infraestrutura

CPU/memória: desprezível para a lógica de brute-force; controlador de threads e I/O são o limitador.

Rede / throughput: taxa de requisições por segundo (RPS) é o fator crítico. Ex.: para RPS de 10 req/s, 5k requisições → ~500 s de tempo de rede (mais latência).

IP/identidade: se houver bloqueios por origem, o atacante pode precisar de proxies/rotacionamento — custo operacional e financeiro (proxies, VPS, botnet).

Paralelização: possível paralelizar testes por byte (apenas parcialmente) e por alvos distintos; paralelizar por bloco é limitado (dependência entre bytes) mas se houver múltiplos ciphertexts independentes, paralelização é eficaz.

Armazenamento de logs/estatísticas: pequeno — para auditoria de tentativas.

Seja L= latência média round-trip por requisição (segundos).
Seja R= requests esperadas.
Tempo aproximado (rede-dominante) = $T \approx R \times L$.
Se conexões/setup adicionam t_conn por request, soma-se esse tempo: $T \approx R \times (L + t\_conn)$.

Definições: B = blocos a decifrar; m_amb= multiplicador por ambiguidade; p_err = taxa de falha/retry.
Tradução em tempo: T ≈ R × L (L = latência RTT média). Exemplo: 5k reqs × 0,2 s ≈ 1.000 s (~16–17 min).
Fatores que elevam custo: reconnects por request, rate-limits, bloqueios por IP.

Estratégia de exploração

1. Capturar o ciphertext alvo (IV + blocos C0,C1,...).
2. Para cada bloco `C_i` que queremos decifrar (i ≥ 1), manipular bytes de `C_{i-1}` e enviar `(IV, ..., C_{i-1}' , C_i, ...)` ao servidor.
3. Explorar o oracle: ajustar `C_{i-1}'` até que o servidor reporte *padding válido* - isso revela informação sobre `D_k(C_i)`.
4. Repetir byte a byte (do último para o primeiro) para reconstruir `P_i = D_k(C_i) ⊕ C_{i-1}`.
5. Repetir para todos os blocos relevantes até recuperar todo o plaintext (flag). (Procedimento clássico; descrição técnica e matemática no paper de Vaudenay e em tutoriais práticos.)

Mitigações concretas (mapeadas ao OWASP A02 — *Cryptographic Failures*)

OWASP recomenda evitar cenários em que falhas criptográficas vazem dados e priorizar modos autenticados (AEAD). Pontos práticos:

- **Use AEAD em vez de CBC+MAC manualNão revele mensagens de erro detalhadas relacionadas à criptografia**
- **Autenticação de *ciphertext* antes da decriptação**.
- **Tempo e medidas *anti-timming/side-channels***
- **Revisão de protocolos proprietários e uso de bibliotecas padrão**

Detecção e monitoramento (práticas recomendadas)

- **Logar** tentativas de ciphertexts inválidos repetidas vezes de um mesmo IP/user (padrão de scanner).
- **Alertas:** volume anômalo de requisições que resultam em “padding error” ou “decryption failure”.
- **Instrumentação:** bloquear/ratelimit por origem após N falhas por X tempo; usar WAF com regras de assinatura para ataques de padding oracle conhecidos (também monitorar padrões de requests que mudam apenas alguns bytes do ciphertext).
- **Honeypot:** prover um token/ciphertext de teste para observar exploração automatizada e capturar endereços IP/behavior.

Lições aprendidas e recomendações operacionais

- Pequenos vazamentos informacionais (mensagens de erro) podem quebrar segurança aparentemente forte — princípio do *fail securely*.
- Migrar serviços críticos para **AEAD** e bibliotecas modernas; rever todas as rotas que manipulam dados cifrados (sessões, tokens, dados armazenados no cliente).
- Testes automatizados (fuzzing) e checklists de segurança (OWASP WSTG) devem incluir checagem de padding oracle. **OWASP+1**

Referência

Writeup / repositório UTCTF Delphi — utisss/UTCTF-21/crypto-delphi. ctftime.org

Writeup alternativo: cscosu/ctf-writeups/2021/utctf/Delphi. ctftime.org

Vaudenay, S. — Security Flaws Induced by CBC Padding (Eurocrypt 2002). iacr.org

Padding oracle attack — resumo e matemática (Wikipedia). Wikipedia

NCC Group / Cryptopals tutorial — Exploiting CBC Padding Oracles (prático). nccgroup.com

OWASP — A02: Cryptographic Failures (Top 10 — 2021). OWASP+1

OWASP WSTG — Testing for Padding Oracle (guia prático de testes). OWASP

Rizzo & Duong — Practical Padding Oracle Attacks (WOOT/2010) — estudo sobre variantes práticas e otimizações. ACM Digital Library

Muito obrigado!