Logotipo

Descrição gerada automaticamente com confiança baixa

Relatório de CTF

Breaking Crypto the Simple Way – tryhackme

|  |  |
| --- | --- |
| **Informações do documento** | |
| **Referência** | Breakingcryptothesimpleway – tryhackme and l000g1c |
| **N° Revisão** | 1 |
| **Data de publicação** | 05/04/2025 |
| **Link** | https://tryhackme.com/room/breakingcryptothesimpleway |

|  |  |  |
| --- | --- | --- |
| **Redação** | Murilo Orsi Marchezzane | Estudante |
| **Revisão** | Nome do revisor | Orientador |
| **Aprovação** | Nome do aprovador | Diretor |

|  |  |  |
| --- | --- | --- |
| **Histórico de revisões** | | |
| **N°** | **Entregas** | **Descrição** |
| **0** | 06/09/2025 | Produção |
| **1** | DD/MM/AAAA | Revisão |
| **2** | DD/MM/AAAA | Aprovação |

|  |  |
| --- | --- |
| **Informações do CTF** | |
| **Nível de Dificuldade** | Facil |
| **Tipo de acesso** | Gratuito |
| **Conceitos envolvidos** | WPScan, msfconsole, costume SUID |
| **Plataforma** | Criptography |
| **Área** | Red |

**Sumário**

[Contextualização 3](#_heading=h.gjdgxs)

[Desenvolvimento 3](#_heading=h.1fob9te)

[Task 1: Introduction 3](#_heading=h.3znysh7)

[Task 2: Brute-forcing Keys 4](#_heading=h.3znysh7)

[Task 3: Breaking Hashes 5](#_heading=h.3znysh7)

[Task 4: Exposed Keys](#_heading=h.2et92p0) 7

[Task 5: Bit Flipping Attacks 8](#_heading=h.tyjcwt)

[Conclusão 11](#_heading=h.1t3h5sf)

[Referências](#_heading=h.4d34og8) 12

**Contextualização**

A criptografia desempenha um papel essencial na proteção de informações sensíveis, garantindo confidencialidade, integridade e autenticidade dos dados. No entanto, sua eficácia depende diretamente da forma como é implementada. Pequenos erros de configuração ou uso incorreto de algoritmos podem comprometer totalmente a segurança, transformando-se em oportunidades fáceis para atacantes em testes de invasão ou em cenários reais.

Erros comuns incluem o uso de chaves de criptografia fracas, geradores de números aleatórios previsíveis ou até mesmo a exposição de segredos, como chaves de API, em código cliente. Nesses casos, não é o algoritmo em si que falha, mas sim sua aplicação inadequada.

Compreender por que essas falhas ocorrem e os impactos que geram é fundamental para qualquer profissional de segurança. Além de reconhecer e explorar vulnerabilidades, como chaves frágeis e algoritmos mal configurados, é igualmente importante saber como evitá-las, fortalecendo a proteção dos sistemas criptográficos contra potenciais ataques.

**Desenvolvimento**

**Task 1: Introduction:**

A criptografia, um dos pilares da cibersegurança moderna, é essencial para a proteção de informações confidenciais. No entanto, sua eficácia não reside apenas na robustez dos algoritmos, mas principalmente em sua implementação correta. Um simples erro, como o uso de uma chave fraca ou o manuseio inadequado de segredos, pode comprometer todo o sistema e expor dados a ataques.

**A vulnerabilidade na implementação**

Falhas de configuração criptográfica são mais comuns do que se imagina. Em avaliações de segurança (testes de penetração) e incidentes de segurança reais, essas falhas frequentemente se manifestam como vulnerabilidades de fácil exploração que podem levar a comprometimentos significativos. Este cenário reforça uma conclusão crucial: o problema, na maioria das vezes, não está no algoritmo em si, mas na forma como ele é aplicado.

**Objetivo do módulo de aprendizado**

O propósito deste módulo do TryHackMe é capacitar o profissional a identificar, analisar e compreender a relevância dessas falhas de implementação. A partir disso, o aprendizado foca tanto na mitigação dessas vulnerabilidades quanto em sua exploração controlada, permitindo um aprofundamento técnico e prático na segurança criptográfica.

## **Task 2: Brute-forcing Keys**

A criptografia baseia-se no princípio de que as chaves usadas para codificar dados são computacionalmente impossíveis de adivinhar. Uma chave considerada "forte" é aquela que possui um alto nível de entropia (imprevisibilidade) e um comprimento adequado, tornando ataques de força bruta inviáveis.

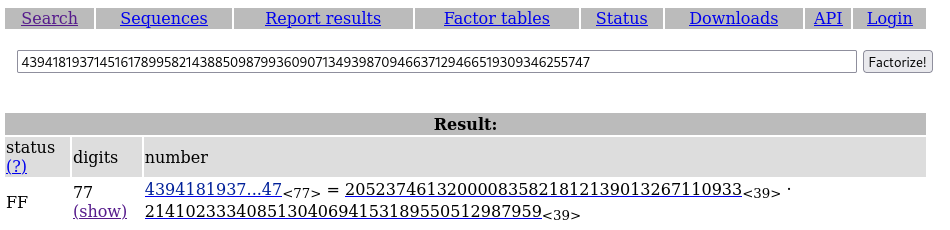
**A Força da Entropia e do Comprimento**

Para ilustrar, considere uma chave de 128 bits. Ela tem 2^128 combinações possíveis, um número tão vasto que a tentativa de decifrá-la usando o hardware atual levaria centenas de anos. A força de uma chave reside, portanto, na sua imprevisibilidade e na sua extensão, que juntas criam uma barreira intransponível para a computação convencional.

**Exercicio:**

Para começarmos, devemos fatorar o n = 43941819371451617899582143885098799360907134939870946637129466519309346255747, por meio de um site chamado factordb.com.

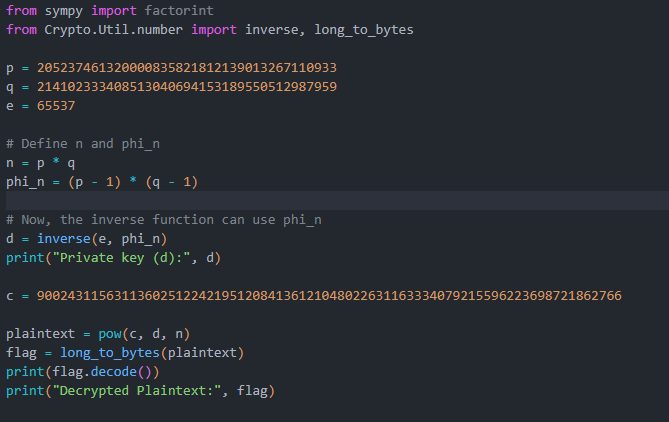
Ao fazer isso temos este resultado:



p = [205237461320000835821812139013267110933](https://factordb.com/index.php?id=1100000007830224022)

q = [214102333408513040694153189550512987959](https://factordb.com/index.php?id=1100000007830224021)

Com todos os dados disponibilizado, fazemos o incremento em um código python:



Que assim podemos resolver a primeira pergunta.

**Pergunta 1- What is the flag?**

R: THM{Psssss\_4nd\_Qsssssss}



## **Task 3: Breaking Hashes:**

**O que é Hashing?**

O hashing é um processo criptográfico que converte um dado de entrada (como uma senha ou uma mensagem) em uma sequência de caracteres de tamanho fixo, chamada de hash. A principal característica desse processo é ser unidirecional, ou seja, é computacionalmente inviável reverter o hash para recuperar a informação original.

**Aplicações do Hashing**

O hashing é empregado em diversas áreas da segurança digital, sendo fundamental para:

* Armazenamento de Senhas: Em vez de guardar senhas em texto puro, os sistemas armazenam apenas seus hashes. No momento do login, a senha digitada é convertida em um hash e comparada com o valor armazenado, garantindo que a senha original nunca seja exposta.
* Integridade de Dados: Ao gerar um hash de um arquivo ou mensagem, é possível verificar se ele foi alterado durante a transmissão. Se o hash da mensagem recebida for diferente do hash original, significa que os dados foram corrompidos ou adulterados.
* Autenticação de Mensagens (HMAC): Quando combinado com uma chave secreta, o hashing (HMAC) garante que uma mensagem não foi apenas preservada em sua integridade, mas também foi enviada por uma fonte autêntica.

**Exercicio:**

Para começarmos, precisamos salvar o código hash e a mensagem em um arquivo, que denominei digest.txt:



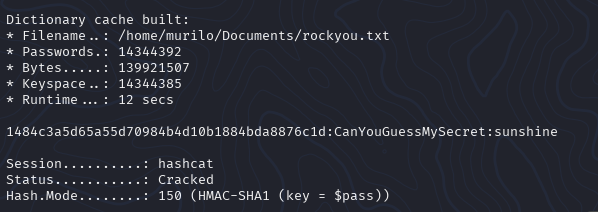
Após isso, rodamos o Hashcat com a lista RockYou.txt:



Oque nos mostra uma série de informações junto com a resposta.

**Pergunta 2 - What is the secret used to encrypt the message?**

R: sunshine



## **Task 4: Exposed Keys:**

**Riscos da Exposição de Chaves Criptográficas em Código do Lado do Cliente**

A exposição de chaves criptográficas em código executado no lado do cliente é um erro frequente e grave. Quando as chaves são embutidas em linguagens como JavaScript, que rodam diretamente no navegador do usuário, qualquer pessoa com acesso ao aplicativo pode facilmente extraí-las e usá-las de forma maliciosa. Tal prática compromete a própria finalidade da criptografia e da autenticação, uma vez que o atacante obtém acesso direto ao mecanismo de proteção dos dados.

**Principais Riscos Envolvidos**

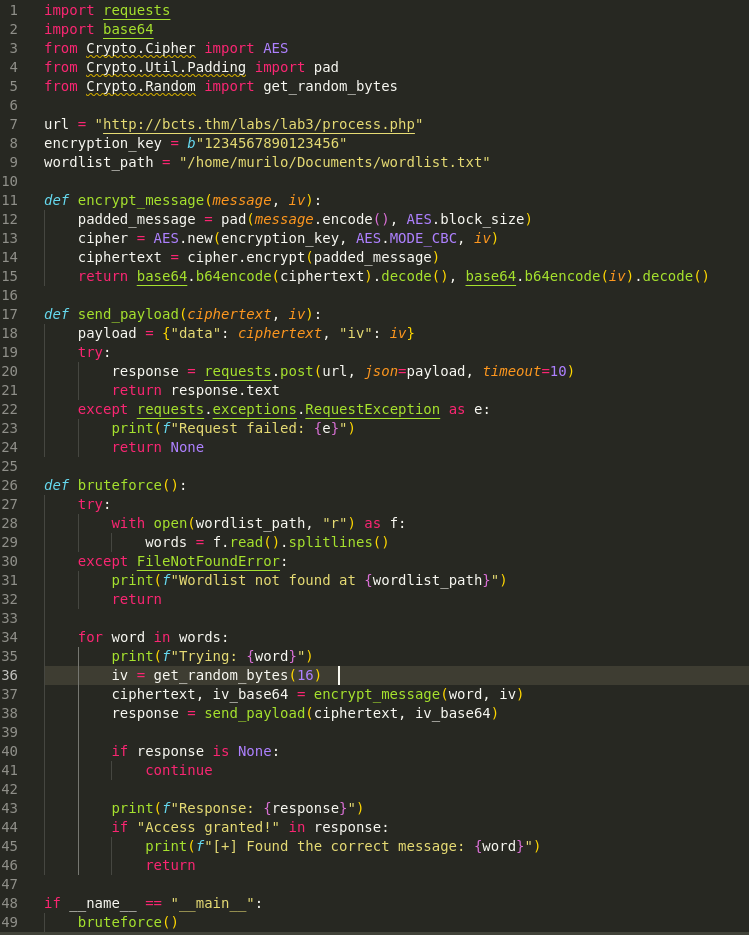
A inclusão indevida de chaves no front-end acarreta riscos significativos, tais como:

* Acesso Não Autorizado: Chaves expostas podem ser utilizadas para decifrar dados confidenciais ou para interagir com APIs de back-end como se fossem um usuário autenticado.
* Adulteração de Dados: Um atacante pode empregar as chaves para gerar dados assinados ou modificar mensagens criptografadas, burlando as verificações de integridade.
* Abuso de API: Chaves de API que são codificadas de forma fixa podem permitir que atacantes acessem endpoints de API com privilégios, sem a devida autorização.

**Exercicio:**

Primeiro precisamos criar um lista de palavras contendo mensagens possíveis está disponível no servidor, em: <http://bcts.thm/labs/lab3/wordlist.txt>

Então para rodar o código de maneira efetiva, utilizamos o código abaixo baseado em python:



Após o código ser rodado encontramos a flag correta.

**Pergunta 3 - What is the flag?**

R: THM{3nD\_2\_3nd\_is\_n0t\_c0mpl1c4ted}



## **Task 5: Bit Flipping Attacks:**

**O que é Criptografia Não Autenticada?**

A criptografia não autenticada refere-se a um tipo de criptografia que não inclui um mecanismo para verificar a integridade ou autenticidade dos dados criptografados (ciphertext). Isso implica que um atacante pode modificar os dados em trânsito e o sistema receptor os aceitará e processará sem detectar qualquer adulteração.

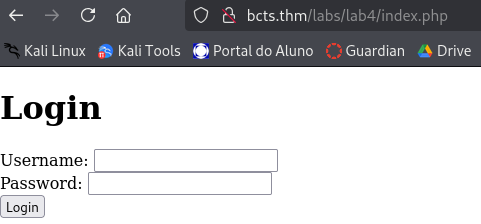
Quando uma aplicação decifra um texto manipulado sem verificar sua integridade, um atacante pode alterar o texto original (plaintext) de forma controlada e previsível. Este é o princípio por trás de ataques como o bit-flipping.

Um exemplo clássico é o algoritmo AES operando no modo CBC (Cipher Block Chaining) sem uma tag de autenticação. Embora o AES-CBC criptografe os dados de forma segura, ele não garante a integridade deles. Dessa forma, se um atacante conseguir modificar o texto criptografado, ele poderá manipular determinados bits do texto original sem comprometer a criptografia em si.

Isso possibilita ataques de inversão de bits (bit-flipping), nos quais o atacante altera o ciphertext de maneira a resultar em modificações específicas no plaintext, permitindo, por exemplo, mudar valores numéricos ou estados lógicos (como true ou false) de uma mensagem.

**Exercicio:**

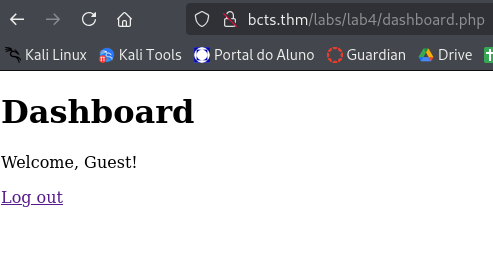
E para terminarmos, para começarmos este exercício, precisamos ir para “http://bcts.thm/labs/lab4/”.



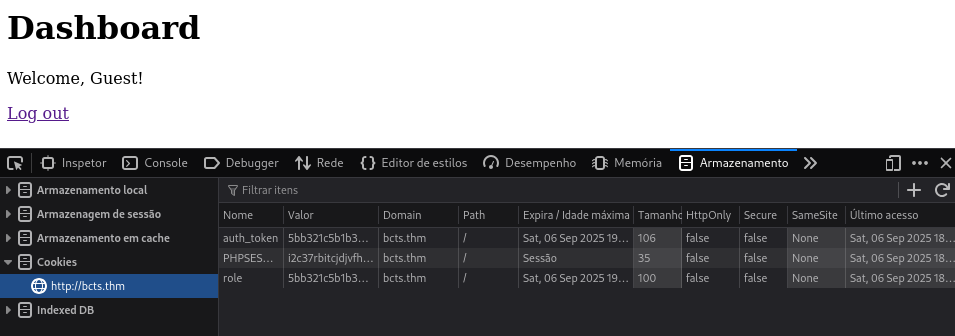
Após isso utilizamos o usuário e a senha abaixo e conseguiremos logar:

Username: guest

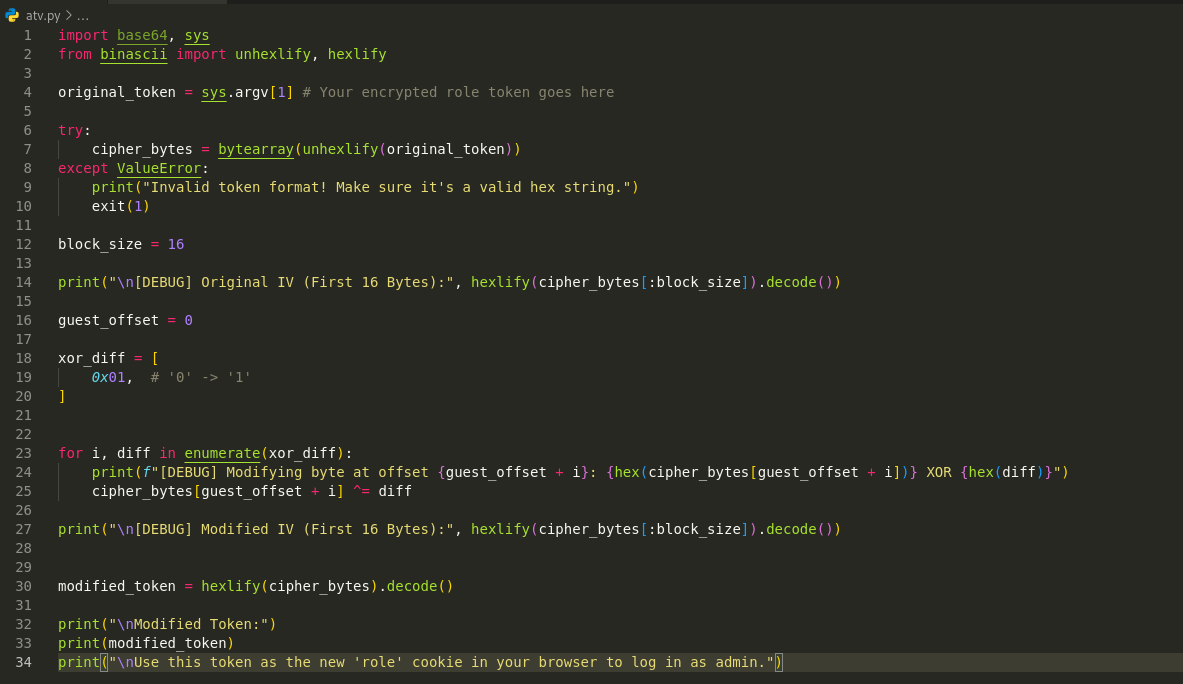
Senha: guest



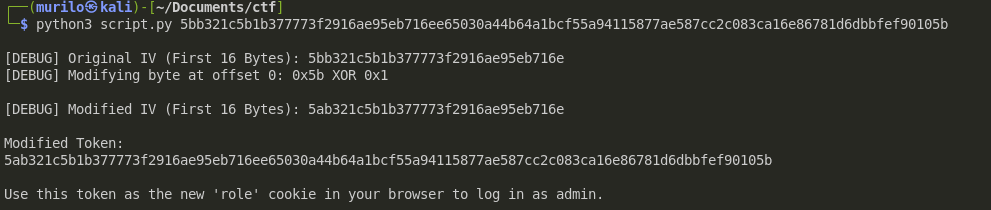
Agora iremos inspecionar a página, pelo F12 e checamos os cookies:



Pegamos o valor da role, para colocarmos no código abaixo:

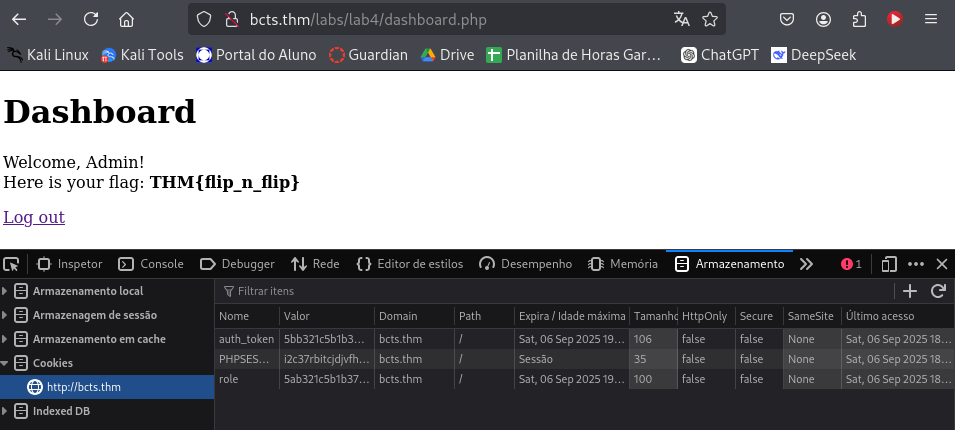


Para então recebermos o nosso token modificado e então assim conseguir entrar no modo admin.



**Pergunta 4 - What is the flag?**

R: THM{flip\_n\_flip}



**Conclusão**

O presente material aborda alguns dos equívocos mais comuns na aplicação da criptografia e como eles podem ser explorados por atacantes. A seguir, um breve resumo dos tópicos discutidos:

* Quebra de Chaves por Força Bruta: Quando as chaves de criptografia são curtas ou previsíveis (por exemplo, baseadas em datas ou senhas simples), ferramentas como o Hashcat ou John the Ripper podem decifrá-las rapidamente. A falta de entropia na geração das chaves as torna vulneráveis a este tipo de ataque.
* Violação de Hashes: Algoritmos de hashing considerados obsoletos, como o MD5 e o SHA-1, não são mais seguros. Eles podem ser comprometidos por ataques de colisão ou por meio de tabelas de rainbow tables, especialmente quando não é utilizado um salt para aumentar a complexidade da hash.
* Exposição de Chaves em Código do Lado do Cliente (Front-end): A prática de inserir chaves secretas ou tokens de API diretamente no código de uma aplicação web é um erro crítico. Essa falha de segurança permite que qualquer pessoa acesse e utilize de forma indevida esses segredos, o que pode levar a vazamentos de dados ou a acessos não autorizados.
* Ataques de Inversão de Bits (Bit-Flipping): Quando a criptografia não é implementada corretamente (por exemplo, ao usar AES-CBC sem um mecanismo de verificação de integridade), atacantes podem manipular dados criptografados e, de forma controlada, alterar o resultado da mensagem após sua decriptação.

**Referências**

Caminho dos materiais utilizados para a solução do CTF.