Logotipo

Descrição gerada automaticamente com confiança baixa

Relatório de CTF

Sequel Dump – Hackfinity Battle

|  |  |
| --- | --- |
| **Informações do documento** | |
| **Referência** | Sequel Dump – Leonardo Moura |
| **N° Revisão** | 1 |
| **Data de publicação** | 28/03/2025 |
| **Link** | <https://tryhackme.com/room/HackfinityBattle> |

|  |  |  |
| --- | --- | --- |
| **Redação** | Leonardo Henrique “Quazmo” Dias Moura | Estudante |
| **Revisão** | Leonardo Henrique “Quazmo” Dias Moura | Orientador |
| **Aprovação** | Leonardo Henrique “Quazmo” Dias Moura | Diretor |

|  |  |  |
| --- | --- | --- |
| **Histórico de revisões** | | |
| **N°** | **Entregas** | **Descrição** |
| **0** | 28/03/2025 | Produção |
| **1** | 28/03/2025 | Revisão |
| **2** | 28/03/2025 | Aprovação |

|  |  |
| --- | --- |
| **Informações do CTF** | |
| **Nível de Dificuldade** | Difícil |
| **Tipo de acesso** | Gratuito |
| **Conceitos envolvidos** | Wireshark, SQLmap (indiretamente) |
| **Plataforma** | Hackfinity Battle |
| **Área** | Blue |

**Sumário**

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

[SQLi e sqlmap 3](#_heading=h.30j0zll)

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

[Encontrando padrões 3](#_heading=h.3znysh7)

[Investigação](#_heading=h.2et92p0) 4

[Trabalho manual](#_heading=h.tyjcwt) 5

[Conclusão](#_heading=h.1t3h5sf) 5

**Contextualização**

Foi capturado um ataque automatizado de SQLi utilizando o sqlmap em um arquivo PCAP, onde os analistas suspeitam que houve exfiltração de dados de um banco de dados possivelmente sensíveis.

**SQLi e sqlmap**

Para a realização desse CTF, conhecimento prévio sobre o funcionamento do sqlmap é necessário. O sqlmap é uma automação de um ataque SQLi, sua maior utilidade é rapidamente realizar um ataque blind SQLi e obter informações. Um ataque blind SQLi é um SQL em que o site não retorna nenhuma mensagem de erro que dê informações sobre o retorno da query SQL injetada.

Sem o uso da mensagem de erro, o blind SQLi utiliza indicadores de que a query funcionou ou errou como resposta do site ou tempo de resposta. A falta de retorno das tabelas torna necessário que cada query seja especificamente montada para confirmar partes da estrutura da tabela.

**Desenvolvimento**

**Encontrando padrões**

Quando aberto, o PCAP apresenta mais de 50 mil pacotes, então analisá-los um a um não é uma opção viável. Dito isso, vamos encontrar alguns padrões sobre o ataque para refinar nossa investigação. Vamos começar filtrando somente pela comunicação HTTP:

Uma imagem contendo Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Para cada requisição HTTP, as respostas parecem sempre vir em dois tamanhos: 262 e 425. A respostas tem o seguinte conteúdo:

- Resposta 1 (262 length): <h2>Search Results:</h2>No results found.

- Resposta 2 (425 length): <h2>Search Results:</h2><p><strong>Void:</strong> The cryptography expert who deciphers the toughest encryptions, searching for vulnerabilities in Void’s encoded fortress.</p>

A segunda resposta aparenta ser nossa confirmação de que a query funcionou, o que nos mostra que o ataque é um blind SQLi feito pelo sqlmap. Vamos agora olhar para o corpo das requisições. Elas têm queries SQL com codificação de URL, e elas podem nos trazer algumas dicas sobre os componentes da tabela. Observando uma com mais cuidado:

1 AND ORD(MID((SELECT IFNULL(CAST(name AS NCHAR),0x20) FROM profile\_db.profiles ORDER BY id LIMIT 0,1),1,1))>85

Essa query essencialmente pega o primeiro caractere da coluna “name”, transforma esse caractere em código ASCII, e checa o valor do seu código ASCII é maior que 85. O sqlmap faz uma série de comparações checando se os valores ASCII dos caracteres são maiores do que alguns números, mudando os valores cuidadosamente até encontrar o código ASCII exato. Esse foi o método para exfiltração de dados.

Investigando algumas outras queries, sabemos da existência das colunas “name”, “id” e “description”. Então, nosso passo a passo a partir de agora é descobrir qual campo da tabela contém a flag, caractere por caractere.

**Investigação**

Teremos que manipular a query para encontrar os valores que nos interessam e codificá-la em URL para filtrar no Wireshark. Vamos olhar para a query utilizada mais cedo:

1 AND ORD(MID((SELECT IFNULL(CAST(name AS NCHAR),0x20) FROM profile\_db.profiles ORDER BY id LIMIT 0,1),1,1))>85

Estamos pegando o primeiro caractere (1,1) da primeira linha (0,1) do nome. Vou grifar o que deveremos manipular para ficar mais fácil de entender. Para achar o próximo caractere vamos manipular o que está grifado em verde, e para achar a próxima linha vamos manipular o que está grifado em amarelo.

1 AND ORD(MID((SELECT IFNULL(CAST(name AS NCHAR),0x20) FROM profile\_db.profiles ORDER BY id LIMIT 0,1),1,1))>85

Agora que sabemos o que manipular, temos que criar nosso filtro. Vamos utilizar o operador “contains” e entregar somente uma porção da query codificada com tudo antes do símbolo de maior, porque queremos encontrar todas as comparações feitas com um caractere para defini-lo. Além disso, só nos interessa o tráfego http, então podemos formular a query desta forma:

http.request.uri contains "ORD%28MID%28%28SELECT%20IFNULL%28CAST%28%60name%60%20AS%20NCHAR%29%2C0x20%29%20FROM%20profile\_db.%60profiles%60%20ORDER%20BY%20id%20LIMIT%200%2C1%29%2C2%2C1%29%29"

O conteúdo entre aspas deve ser modificado cada vez que mudarmos de caractere. O que é exibido no Wireshark é o seguinte:

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Tela de computador

O conteúdo gerado por IA pode estar incorreto.

Agora, é só escolher a opção de “Follow HTTP Stream” para visualizar a resposta e realizar a seguinte análise por caractere: Qual o número mais alto da comparação que retornou com sucesso e qual o número mais baixo da comparação que retornou com erro?

Por exemplo: Na pesquisa do primeiro caractere do primeiro nome, a comparação se o código é maior que 85 retornou com sucesso, mas falhou na comparação se é maior que 86. Como o operador utilizado é “>”, sabemos que não pode ser maior que 86 e menor que 85, portanto o caractere tem código ASCII 86 que equivale a “V”.

**Trabalho manual**

Não vou descrever todo o trabalho manual que eu fiz, mas vou compartilhar os resultados obtidos. Eu conclui que a coluna “id” não traz nenhum valor de grande importância, os valores de “description” trazem textos muito grandes e analisa-los um por um é uma grande perda de tempo, então a coluna “name” provavelmente será o nosso indicador mais definitivo.

A coluna “name” tem 7 linhas, e na segunda linha o usuário Zero-day aparenta ser promissor mas não traz nada de relevante. Estava procurando por uma linha que começa-se com “C” de Cipher, o vilão da competição, mas acabei encontrando na última coluna um nome que não começava com letra maiúscula. Continuei investigando até coletar “supers” que eu assumi que seria “supersecret”. Na descrição dessa linha, fui montando os caracteres até encontrar “Here’s the flag: THM{XXXXXXXXXXXXXX}”. O filtro utilizado para encontrar o primeiro caractere dessa descrição está a seguir:

http.request.uri contains "ORD%28MID%28%28SELECT%20IFNULL%28CAST%28%60description%60%20AS%20NCHAR%29%2C0x20%29%20FROM%20profile\_db.%60profiles%60%20ORDER%20BY%20id%20LIMIT%206%2C1%29%2C1%2C1%29%29"

**Conclusão**

É um CTF que requer um conhecimento teórico muito bom e é bem trabalhoso (demorei 6h30 para achar a flag), mas é extremamente satisfatório quando a flag é encontrada. Não sei se há um jeito de automatizar a procura, mas provavelmente deveria tentar isso antes de gastar tanto tempo fazendo cada análise. Se a teoria é muito difícil para você, tente saber mais blind SQLi!